TU Delft, Faculty of Electrical Engineering, Mathematics, and Computer Science

EE4540 Distributed Signal Processing ASSIGNMENT

Fire Detection Using Distributed Temperature Sensors

Yiting Lu(4714881)
Karishma Kumar(4737598)

April 2018

1 INTRODUCTION

Recent developments in hardware level have led to the emergence of small-size and low-power mobile devices with limited onboard data processing and wireless communication capabilities. These devices are called sensors, consisting of a radio frequency circuit, a low-power digital signal processor, a sensing unit and a battery.[1] Due to their low cost and low complexity design requirement, individual sensors can merely perform simple local computation and transmit information over a short range at low data rates. But when deployed in large numbers across a spatial domain, these primitive sensors can form an intelligent network to collect data form the physical environment with high precision.[1] Sensor networks of this type are widely used in situation awareness applications such as environmental monitoring (temperature, air condition, water and soil).

With the boom of electronic measurements, more and more people focus on using electronic equipment to guarantee personal and property security. Based on the deficiencies of conventional fire detection on real time and monitoring accuracy, the wireless sensor network technique for fire detection has been introduced.[2] In this assignment, a proper sensor network that covers all the area of the plant is designed to monitor the temperature. Distributed averaging algorithms will be used to compute the average temperature of the whole sensor network.

This report is constructed in the following order: section 2 will introduce the sensor network topology; section 3 will explain the theory of various averaging algorithms; section 4 will discuss the convergence rates of different averaging algorithms mentioned in previous sections; section 5 will test the robustness to transmission failure and lost of sensors; section 6 will give introduce the fire detection model and section 7 will give a final conclusion.

2 SENSOR NETWORK TOPOLOGY

In this case, a sensor network on the ceiling is required to cover a plant of size $30m \times 60m$. These sensors are used to provide a real time monitoring of the environmental temperature. The sensors can communicate via wireless data transmission but the transmission radius is only $3m$. To make the problem simple, we assume that the normal temperature in the plant follows a Gaussian distribution, i.e., $X \sim \mathcal{N}(20, \sigma^2)$. In the fire region, the temperature follows

another Gaussian distribution, i.e., $X \sim \mathcal{N}(80, \sigma^2)$.

The main purpose of this assignment is to detect automatically which region is on fire when a fire starts in the plant. The network which is going to be designed not only should cover the whole area of the plant using less sensors as much as possible to reduce the cost but also make sure each node should have no less than two neighbors to guarantee the robustness of the network. In case of losing a neighbor, this node still can talk with other neighbors.

Intuitively, the 4-regular grid topology is chosen because the plant is a rectangular. Fig1 shows the temperature sensor network topology. The side length of each unit grid is $3m$. Two nodes are regard to be connected if the distance between them is $3m$. The solid circle in the figure is the transmission range of this example node. Therefore, this node has four neighbors. Similarly, most nodes in this 4-regular grid topology have four neighbors, except for the boundary nodes, which have only 2 or 3 neighbors. It takes 200 sensors in total to cover the whole plant in this way.

One advantage of 4-regular grid topology is that, in average, each node has approximately 4 neighbors, which is robust to the transmission failure and topology change.

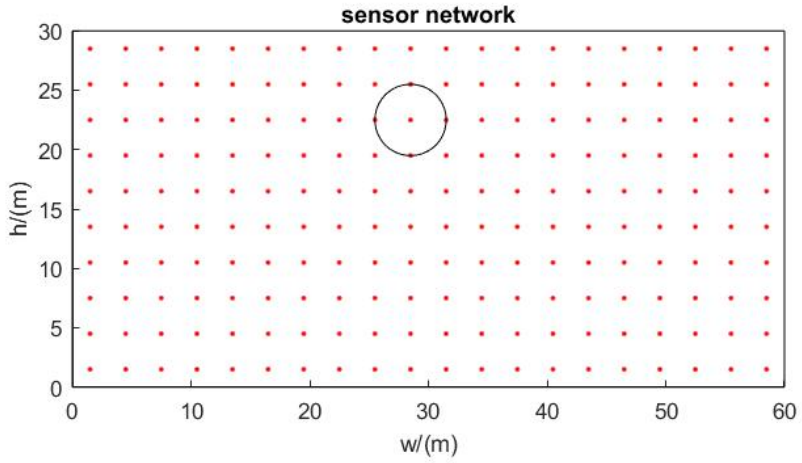
3 AVERAGING ALGORITHMS

Assume that there is no central unit to collect and process the data. All computations and transmissions are carried over the sensors in a distributed way. Such an averaging problem can be solved in three different ways, distributed averaging, gossip algorithms and distributed convex optimization. In this assignment, we pick some algorithms from each category and make a brief comparison among them.

Distributed averaging is consist of synchronous algorithm and distributed asynchronous algorithm. Synchronous algorithm requires a global clock to guarantee synchronization within the network, thus sensitive to changes in network topology. Therefore, we choose asynchronous distributed averaging to realize temperature averaging problem in our assignment, which has the advantage of less sensitive to changes in network topology and can get rid of the global clock which is costly to implement in application.

Actually asynchronous distributed averaging is still quite sensitive to changes in the network topology because at each iteration, we compute the average over the chosen node and its all neighbors. The solution is that in a given time slot, each node can communicate with only one of its neighbors, which is called gossip algorithms. Average consensus and gossip algorithms have recently received significant attention due to their simplicity and robustness for distributed information processing over networks.[3] Inspired by heat diffusion, they solve the averaging problem by iterating local averages until a desired level of convergence.[3]

Multiple gossip algorithms can be chosen to realize temperature averaging, such as randomized gossip, geographic gossip, greedy gossip with eavesdropping and broadcast weighted gossip. Geographic gossip combines gossip with geographic routing, exploiting geographic information to create a new complete communication graph as a overlay on the original graph. The new communi-



Figuur 1: 4-regular grid topology

cation graph is dense so that gossiping converges more quickly. However, The topology of such networks changes as new nodes join and old nodes leave the network, then the complete graph is not applicable any more. Algorithms for such networks need to be robust against changes in topology. So we do not discuss this algorithm here.

Besides, such an averaging problem can also be formulated as the following distributed convex optimization problem:

$$minimizing \qquad f(x) = \sum_{i=1}^{n} \frac{1}{2}(x_i - a_i)^2 \tag{1}$$

$$subject \quad to \quad x_i - x_j = 0, \quad \forall (i,j) \in E \tag{2}$$

Where x_i is measurements at each node, a_i is the sensor measurements, node j is one of the neighbors of node i. This is a consensus problem which can be solved by iteration.

Detailed theories about each algorithm above will be explained below.

3.1 Asynchronous Distributed Averaging

Asynchronous distributed averaging does not need any global knowledge of the network. It works under the control of local clocks. A node exchanges information only with neighbors.

Algorithm:

- In the k_{th} time slot, select a node i at random (uniform distribution, probability $P_i = 1/n$) and let it contact its neighboring nodes $j \in \mathcal{N}(i)$.
- At this time, all nodes set their values equal to the average of their current values.

3.2 Randomized Gossip

The main difference between randomized gossip and asynchronous distributed averaging is that in randomized gossip, in a given time slot, each node can only randomly talk with some of its neighboring nodes rather than all.

Algorithm:

- In the k_{th} time slot, select a node i at random (uniform distribution, probability $1/n$) and let it contact some neighboring node j with probability P_{ij}. (In our experiments, we randomly choose on neighboring node to do pairwise average.)
- At this time, both nodes set their values equal to the average of their current values.

3.3 Greedy Gossip with Eavesdropping

Greedy gossip with eavesdropping (GGE) exploits the broadcast nature of wireless communication networks. Unlike previous randomized gossip algorithms, which perform updates completely at random, GGE implements a greedy neighbor selection procedure. All neighbors within range of a transmission node receive the message. In addition to keep tracking of its own value, each node tracks its neighboring values by eavesdropping on their transmissions.

Algorithm:

- At the k_{th} iteration of GGE, a node i is chosen uniformly at random. Then, node i identifies a neighboring node j satisfying

$$j = \arg\max_{j \in \mathcal{N}_i} \left(\frac{1}{2}(x_i(k-1) - x_j(k-1))^2\right) \tag{3}$$

  which is to say, node i identifies a neighbor that currently has the most different value from its own.[4] This choice is possible because each node i maintains not only its own local variable, $x_i(k-1)$, but also a copy of the current values at its neighbors, $x_j(k-1)$, $\forall j \in \mathcal{N}_i$.[4] When node i has multiple neighbors whose values are all equally (and maximally) different from node i, it chooses one of these neighbors at random.[4]
- Node i and node j perform the pairwise averaging while all other node values remain the same.
- Finally, the two nodes, i and j broadcast these new values so that their neighbors have up-to-date information.

3.4 Sum-weight Averaging

Routing information back and forth might as well introduce delay issues, because a node that is engaged in a route needs to wait for update to come back before it can proceed to another round. Problem can be solved by using one-way averaging. In this algorithm, an additional vector s is introduced, which is the weighted sum of $x(k)$. At any time, the vector of estimates is $x(k) = s(k)/w(k)$, where the division is performed element-wise.

Algorithm:

- Initialization $s(0) = x(0)$ and $w(k) = \mathbf{1}$
- In the k_{th} time slot, instead of updating one vector $x(k)$ of variables, it updates a vector $s(k)$ of sums, and a vector $w(k)$ of weights.
- The updates are computed with column stochastic matrices $D(k)_{k\geq 1}$:

$$s(k) = D(k)s(k-1) \tag{4}$$
$$w(k) = D(k)w(k-1) \tag{5}$$

In general, when we average over 2 neighboring nodes i and j, we have (after re-indexing):

$$D(k) = \begin{bmatrix} \frac{1}{2} & 0 & \mathbf{0} \\ \frac{1}{2} & 1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I_{n-2}} \end{bmatrix} \in \mathbb{R}^{n\times n}, \tag{6}$$

so that the update equations are given by

$$\begin{cases} s_i(k) = \frac{1}{2}s_i(k-1) \\ s_j(k) = s_j(k-1) + \frac{1}{2}s_i(k-1) \end{cases} \tag{7}$$

and similarly for weights.

3.5 Broadcast Weighted Gossip

Broadcast weighted gossip is a variation to sum-weight gossip. It takes the advantage of the broadcast nature of the wireless channel. Information propagation is much faster while broadcasting compared to pairwise exchanges. At each global clock tick, a sensor is chosen uniformly at random and broadcast its pair of values in an appropriate way. Then the receiving sensors add the received pair of values to their current one. The update equations are given by

$$\begin{cases} s_i(k) = (1 - d_i\alpha^{-1})s_i(k-1) \\ s_j(k) = s_j(k-1) + \alpha^{-1}(k-1) \quad \forall j \in \mathcal{N}_i \end{cases} \tag{8}$$

and similarly for the weights.

3.6 Primal-dual Method of Multipliers (PDMM)

As we all know, in order to separate the node variables, alternating direction method of multipliers (ADMM) introduces auxiliary variables, e.g., edge variables z_{ij}. Intuitively, convergence rate can be improved when information flows

directly from node to node. We would like to design a distributed algorithm that does not need auxiliary variables to decouple the nodes. PDMM is a node-based optimization algorithm for solving problems where edge relationships are affine. The PDMM formulation for the consensus problem has been mentioned above. The updating scheme is as following:

$$x_i^{(k+1)} = \arg\min_{x_i}(f_i(x_i) - x_i^T(\sum_{j\in\mathcal{N}(i)} A_{ij}^T\lambda_{j|i}^{(k)}) + \sum_{j\in\mathcal{N}(i)} \frac{c}{2}\|A_{ij}x_i + A_{ji}x_j^{(k)} - b_{ij}\|^2) \quad (9)$$

$$\lambda_{i|j}^{(k+1)} = \lambda_{j|i}^{(k)} + c(A_{ij}x_i^{(k+1)} + A_{ji}x_j^{(k)} - b_{ij}), \forall j \in \mathcal{N}(i), \quad (10)$$

where $c \in (-L/2, 0)$. In our case, through solving the PDMM minimizing problem, we conclude that

$$x_i^{(k+1)} := \frac{a_i + \sum_{j\in\mathcal{N}(i)}(A_{ij}\lambda_{j|i}^{(k)} + cx_j^{(k)})}{1 + cd_i} \quad (11)$$

where d_i denotes the degree of node i. Similarly we find that

$$\lambda_{i|j}^{(k+1)} = \lambda_{j|i}^{(k)} + cA_{ij}(x_i^{k+1} - x_i^k), \forall (i,j) \in E, \quad (12)$$

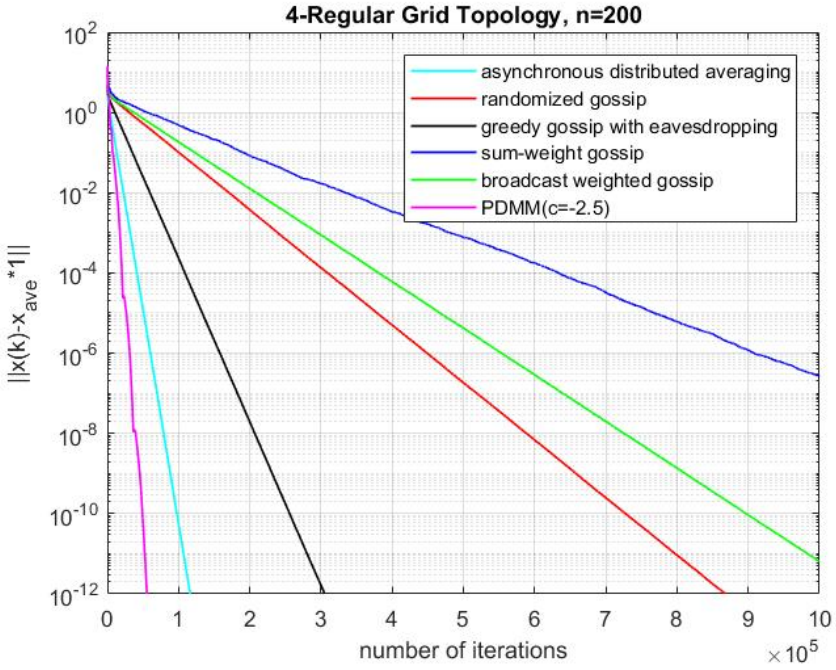
where $A_{ij} = 1$ if $i < j$ and $A_{ij} = -1$ otherwise.

4 CONVERGENCE ANALYSIS

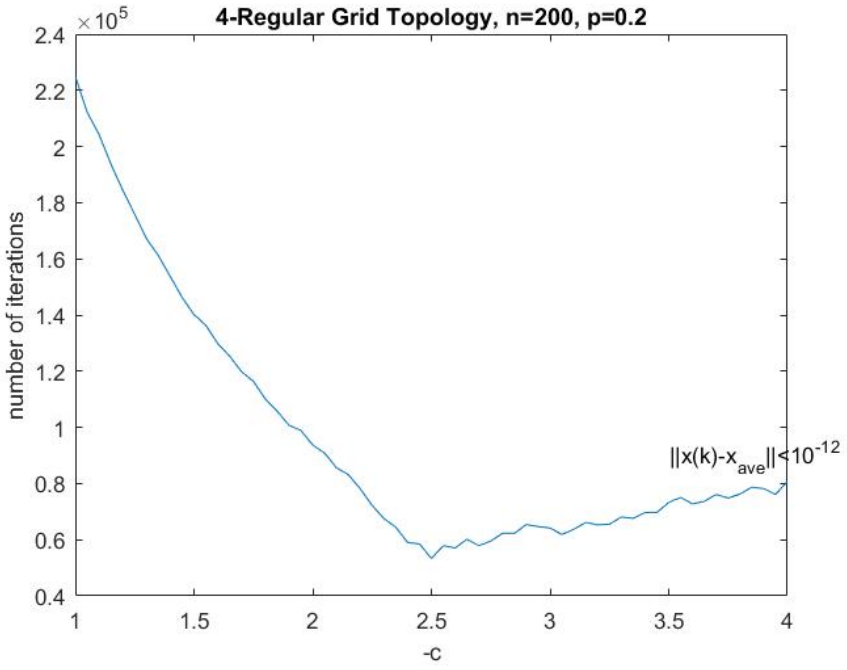
When analyzing the convergence rates of above six averaging algorithms, we set the temperature variance σ equal to 1. The maximum number of iterations k_{max} is 1×10^6 and the error tolerance ϵ is 1×10^{-12}. Fig2 illustrates the corresponding experimental results. The vertical axis is the $L2$ norm error and the horizontal axis is the number of iterations k. All the algorithms converge at a linear rate.

As is shown in graph, pairwise averaging algorithms such as randomized gossip, GGE and sum-weight gossip have a slower convergence compared with asynchronous distributed averaging algorithm which takes the average over the chosen node and all its neighboring nodes. It does make sense because information exchange in pairwise algorithms are slower than that in asynchronous distributed averaging algorithm. What's more, one-way algorithms such as sum-weight gossip and broadcast weighted gossip are much slower than others where routing information is bidirectional. Also averaging algorithms like broadcast weighted gossip and GGD take the advantage of the broadcast nature of wireless communication network are faster than their corresponding variations, sum-weight gossip and randomized gossip where information only exchanged with a randomly chosen neighboring node. In a word, the amount of information exchanged in every iteration actually dominates the convergence speed.

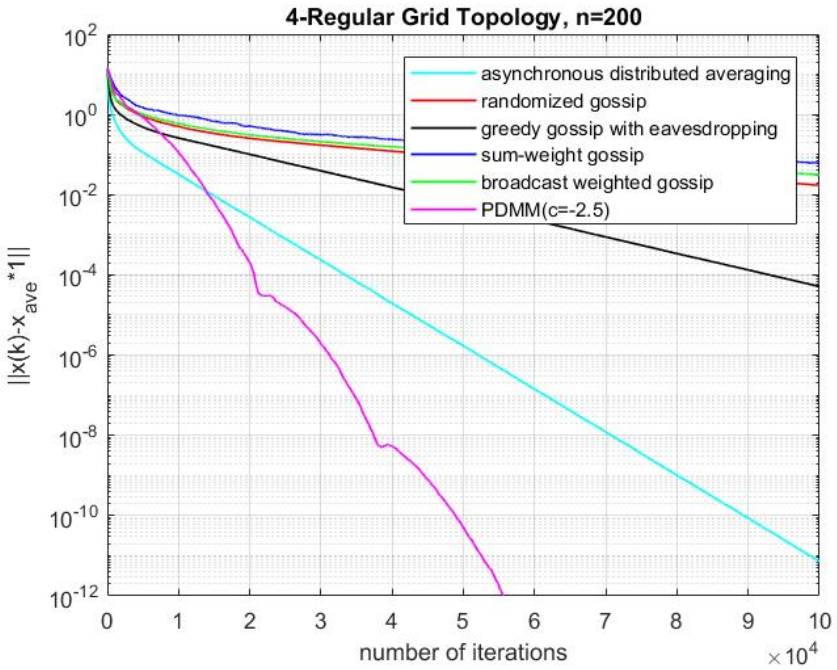
The choice of c in PDMM is very crucial, Fig3 plots the effect of c value on number of iterations. The curve attains the minimum at around 2.5, thus we choose $c = -2.5$ in our experiment. PDMM has the fastest convergence due to the high efficiency of gradient descent method. Every iteration is in the right direction to get close to the average temperature. If we take a closer look at the initial iteration stage in Fig4, convergence speed of PDMM is not that fast because every iteration, only node i update its temperature value.



Figuur 2: Convergence Rate



Figuur 3: Choice of c



Figuur 4: Convergence Rate (Zoom-in)

5 ROBUSTNESS TEST

Transmission failure and topology change are two common problems in wireless sensor networks. Therefore, a good averaging algorithm not only needs to have a fast convergence rate to alarm people timely, but also to be robust to these two problems. In this section, we will test the performance of the above six averaging algorithms in the context of transmission failure and lost of nodes respectively and make a brief comparison among them.

5.1 Transmission Failure

Assume that if a transmission between a sensor and its neighbor fails, the averaging between two nodes will not be carried out. All transmissions are independent from each other. A transmission failure between two nodes has no affect on other transmissions. The degree of each node remains the same.

In bidirectional averaging algorithms, routing information forth and back might both introduce transmission failures. For simplicity, in asynchronous distributed averaging and randomized gossip, we consider two kinds of transmission failure, if neighboring node j fails to transmit current value to node i, node j does not take part into averaging in this time slot; if node i receives information from node j, then value of node j will not be updated in this time slot. Similarly in PDMM, if node j fails to transmit its node value, it does not take part into averaging in this time slot. In GGE, we both consider the affect of transmission failure on searching for the most different value and the updating of neighboring

node. In one-way gossip like sum-weight gossip and broadcast gossip, we only consider the updating failure on weight vector and sum vector of neighboring nodes.

In our simulation, the probability of a transmission failure p is set to 0.1, 0.2 and 0.3 respectively. Fig567 show the effect of transmission failure, which are obtained after 20 repeats. All the algorithms still converges though at a much higher error. Transmission failure form neighboring nodes to node i only decrease the convergence speed but miss updating at neighboring nodes seems results in uncompensated error accumulation in the network. PDMM is the most sensitive algorithm to transmission failure, which had the best performance working in an ideal sensor network, but now is the worst one probably due to node-specific message $\lambda_{i|j}^{(k+1)}$. So the gradient is not in the correct direction and it seems PDMM is stuck with a local optimal solution, the averaging error does not go down anymore and the curve is very jittery. Asynchronous distributed averaging has the best performance when considering transmission failures in the sensor network.

5.2 Lost of Nodes

The topology of networks changes continuously as new nodes join and old nodes leave the network. To test the performance of the above six averaging algorithms when nodes out of work in our sensor network, we remove 5%, and 20% nodes from the network respectively and test the robustness of the algorithms.

Fig89 show the experimental results. All the averaging are less sensitive to lost of nodes compared with transmission failures. However, the speed of convergence decrease a bit mainly because noise drawn from same distribution has larger influence to networks with less sensors. The good news is that all the algorithms still converge at a linear rate and high precision.

6 FIRE DETECTION MODEL

Fire detection is a simple binary detection problem: determine whether a certain signal that is embedded in noise is present or not. In this case, two hypothesis can be described in the following way:

$$\begin{cases} \mathcal{H}_0 & x(i) \sim \mathcal{N}(20, \sigma^2), \quad i \in [1, 200] \\ \mathcal{H}_1 & x(i) \sim \mathcal{N}(80, \sigma^2), \quad i \in [1, 200] \end{cases} \tag{13}$$

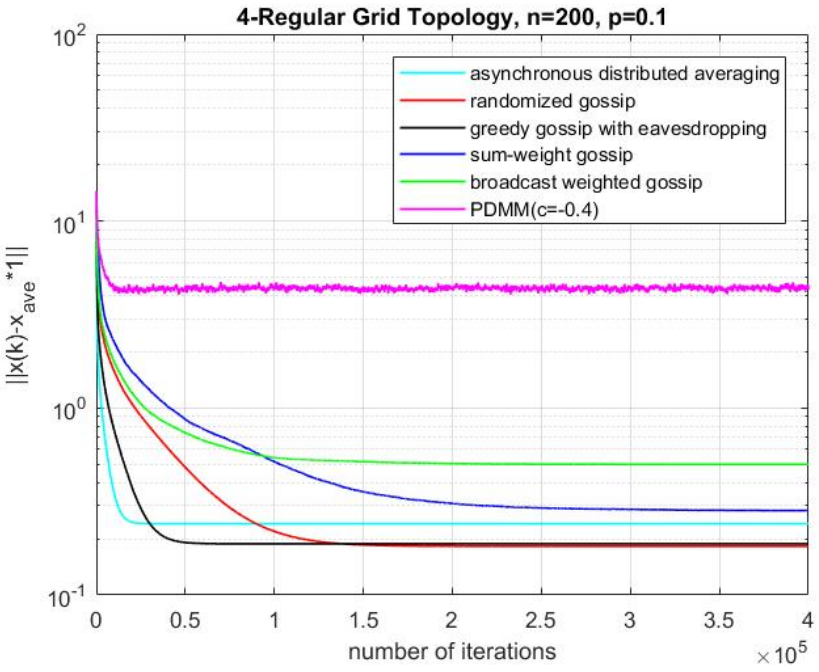
The probability density function of $x(i)$ under each hypothesis is as follows:

$$p(x(i); \mathcal{H}_0) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{1}{2\sigma^2}(x(i) - 20)^2), \quad i \in [1, 200] \tag{14}$$

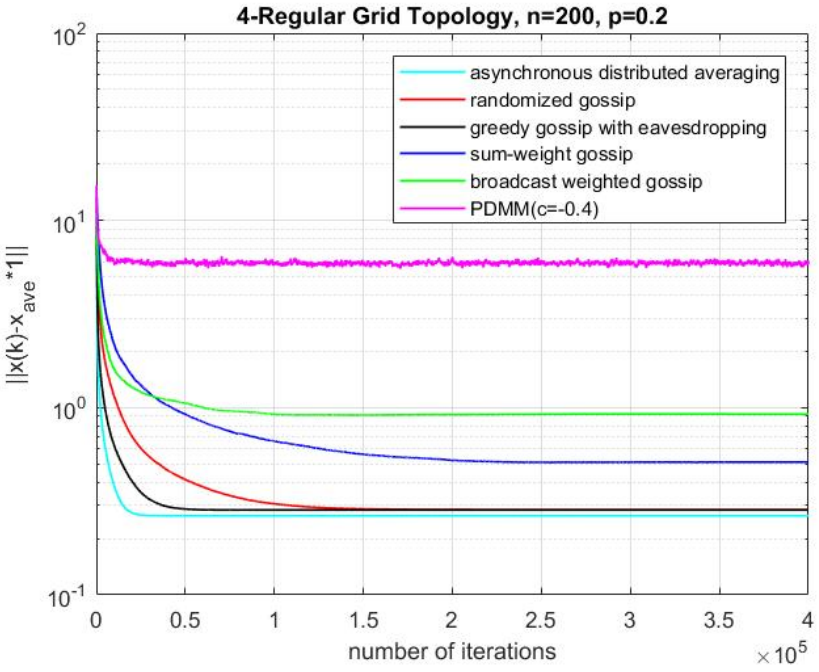
$$p(x(i); \mathcal{H}_1) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{1}{2\sigma^2}(x(i) - 80)^2), \quad i \in [1, 200] \tag{15}$$

Deciding between $\mathcal{H}_0$ and $\mathcal{H}_1$, we are essentially asking whether $x(i)$ has been generated according to the pdf $p(x(i); \mathcal{H}_0)$ or the pdf $p(x(i); \mathcal{H}_1)$. The Neyman-Pearson (NP) detector decides $\mathcal{H}_1$ if

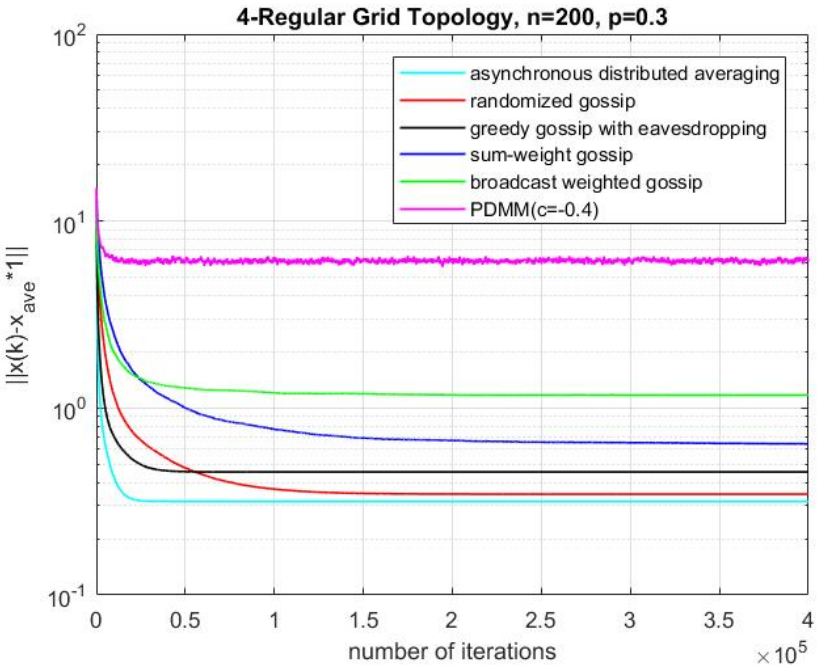
$$\frac{\frac{1}{(2\pi\sigma^2)^{\frac{2}{N}}} exp[-\frac{1}{2\sigma^2}\sum_{n=0}^{N-1}(x_i[n] - 80)^2]}{\frac{1}{(2\pi\sigma^2)^{\frac{2}{N}}} exp[-\frac{1}{2\sigma^2}\sum_{n=0}^{N-1}(x_i[n] - 20)^2]} > \gamma \tag{16}$$



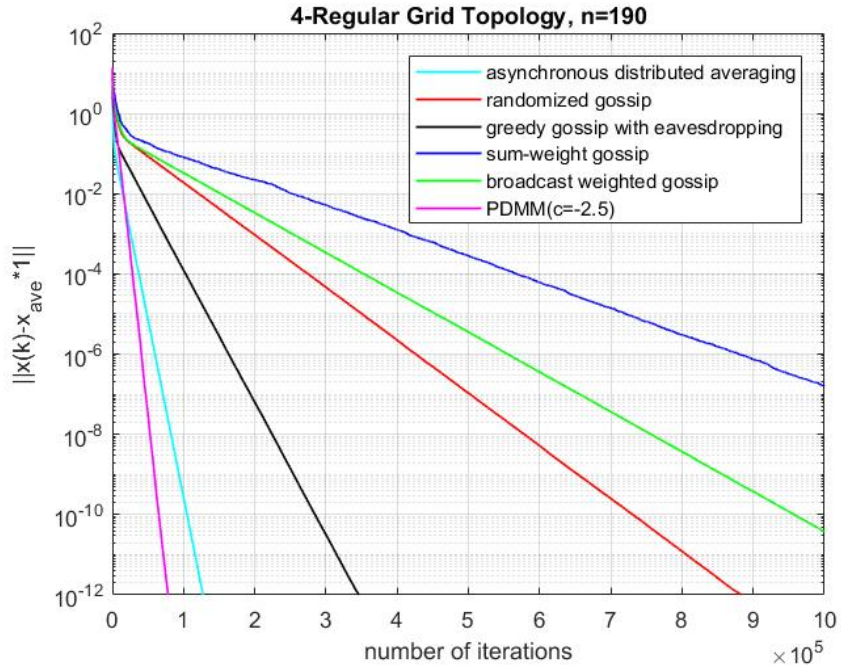
Figuur 5: Convergence Rate (p=0.1)



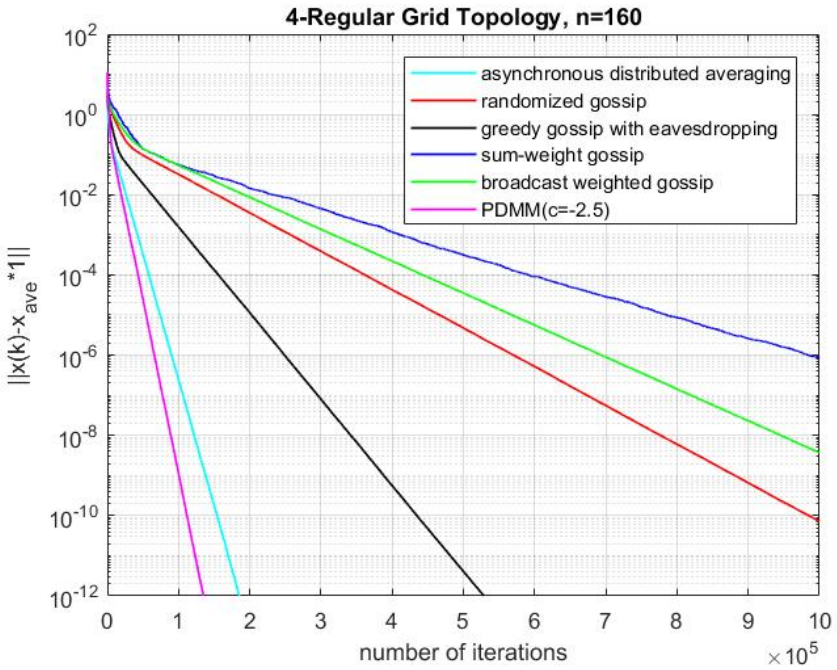
Figuur 6: Convergence Rate (p=0.2)



Figuur 7: Convergence Rate (p=0.3)



Figuur 8: Convergence Rate (remove 5% nodes)



Figuur 9: Convergence Rate (remove 20% nodes)

Let $\gamma = 1$, we can obtain that

$$T = \frac{1}{N}\sum_{n=0}^{N-1} x_i(n) > 50. \tag{17}$$

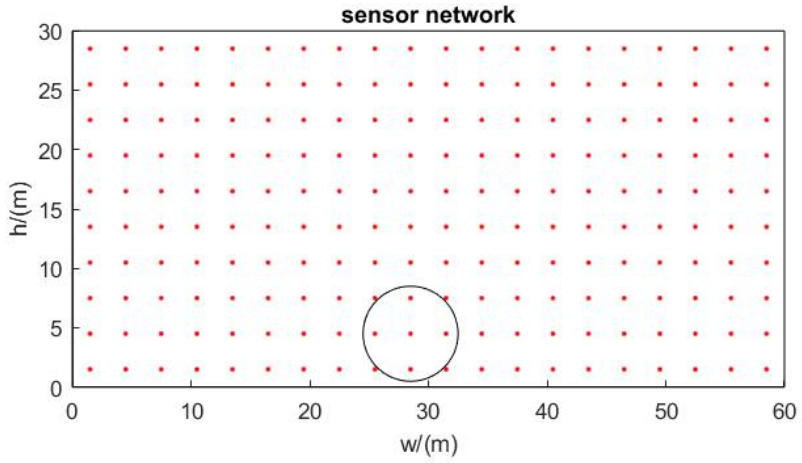
The NP detector compares sample mean $\overline{x} = \frac{1}{N}\sum_{n=0}^{N-1} x_i(n)$ threshold γ. The test statistic $T(x) = \overline{x}$ is Gaussian under each hypothesis and its distribution is as follows

$$T(x) \sim \begin{cases} \mathcal{N}(20, \frac{\sigma^2}{N}) & under \quad \mathcal{H}_0 \\ \mathcal{N}(80, \frac{\sigma^2}{N}) & under \quad \mathcal{H}_1 \end{cases} \tag{18}$$

Suppose a fire occurs at the center of the circle in 10, fire region is within the circle, thus the measured temperature of sensor 10, 29, 30, 31, 50 are drawn from a Gaussian distribution, $\mathcal{N}(80, \sigma^2)$. As equation18 shows, if the variance is very large, we can do averaging over N measurements to decrease the variance of test statistic $T(x)$. Then the overlapping area under two pdfs is decreased, so is the detection error. The corresponding results are shown in Fig11. We take average over $N = 10$ sensor measurements and set $\sigma = 1, 10, 20$ respectively. The detection results are all correct, which means our detection model is robust.

7 CONCLUSION

In this assignment, a fire detection problem is proposed and solved by distributed averaging algorithms. Also the robustness of the chosen algorithms are



Figuur 10: fire region

discussed.

The sensor network is build in 4-regular grid topology where each node has approximately 4 neighboring nodes, which is robust to transmission failures and topology changes.

Six algorithms, asynchronous distributed averaging, randomized gossip, GGE, sum-weight gossip, broadcast weighted gossip and PDMM are discussed in this report. All algorithms are robust to lost of nodes but have much higher averaging error when transmission failure occurs. In an ideal sensor network, PDMM has the best performance but it is extremely sensitive to transmission failures. Asynchronous distributed averaging has comparably good and stable performance both on convergence speed and averaging accuracy due to the larger amount of information exchanged in the network. one-way gossip like sum-weight gossip and broadcast gossip which are more suitable in dynamic sensor networks show low efficiency in our case.

Timely fire alarm can prevent loss of life and property. We would like to choose robust and high efficiency algorithms at the cost of routing overhead, thus asynchronous distributed averaging is recommended based on our experimental results. But in practical, routing information forth and back might as well introduce delay issues. If routing delay is very severe in a sensor network, then maybe GGE and randomized gossip will be more suitable.

In the fire region detection part, every single node average its own $N = 10$ continuous measurements and compare this $\overline{x}$ with a threshold $\gamma = 50$. If $\overline{x}$ is larger than γ, then this node is regard within the fire region. This is a decentralized decision algorithm and very robust to noise variance. We can increase

```
>> fire_detection

sigma =

     1


fire_regin =

    10    29    30    31    50
>> fire_detection

sigma =

    10


fire_regin =

    10    29    30    31    50

>> fire_detection

sigma =

    20


fire_regin =

    10    29    30    31    50
```

Figuur 11: detection Result

N to decrease the probability of both false alarm and miss detection.

Referenties

[1] Zhi-Quan Luo, M. Gastpar, Juan Liu, and A. Swami. Distributed signal processing in sensor networks [from the guest editors]. *IEEE Signal Processing Magazine*, 23(4):14–15, 2006.

[2] J. Zhang, W. Li, Z. Yin, S. Liu, and X. Guo. Forest fire detection system based on wireless sensor network. In *2009 4th IEEE Conference on Industrial Electronics and Applications*, pages 520–523, 2009.

[3] P. Denantes, F. Benezit, P. Thiran, and M. Vetterli. Which distributed averaging algorithm should i choose for my sensor network? In *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*, 2008.

[4] Deniz Üstebay, Boris N. Oreshkin, Mark Coates, and Michael Rabbat. Greedy gossip with eavesdropping. *CoRR*, abs/0909.1830, 2009.

8 APPENDIX

8.1 Matlab Code for Sensor Network topology

8.1.1 Network Visualization

```
%% generate the coordinates
x=1.5:3:60;
x=repmat(x,[1,10]);
y1=ones(1,20)*1.5;
y2=y1+3;
y3=y2+3;
y4=y3+3;
y5=y4+3;
y6=y5+3;
y7=y6+3;
y8=y7+3;
y9=y8+3;
y10=y9+3;
y=[y1, y2, y3, y4, y5, y6, y7,y8,y9,y10];
%% scatter the sensors
scatter(x,y,'.r');
hold on
% draw the transmission region
for i=30
    rectangle('Position',[x(i)-4,y(i)-4,8,8],'Curvature',[1,1]);
end
hold on
%% draw the plant
rectangle('Position',[0,0,60,30])
axis equal
title('sensor_network')
xlabel('w/(m)')
ylabel('h/(m)')
axis([0 60 0 30])
```

8.1.2 Graph creation

```
function [A,G]=Adjacency_matrix(m,n)
    size=(m+2)*(n+2);
    A=zeros(size,size);
    B=zeros(0,1);
    for i=2+m:size-m-3
            if rem(i,22)==1
                continue
            elseif rem(i,22)==0
                continue
            else
                A(i,[i-1,i+1,i+22,i-22])=1;
                B(end+1)=i;
```

```
            end
        end
        A=A(B,B);
        G=graph(A);
        neighbors(G,22)
end
```

8.2 Convergence Test

8.2.1 Main File

```
temp=20;
sigma=1;
kmax=1e6;
c=2.5;
x_initial=normrnd(temp ,sigma, [200,1]);
x_ave=mean(x_initial);
[A,G]=Adjacency_matrix(20,10);
%% asynchronous_averaging
[err2, x2]=asynchronous_averaging(kmax, G, x_initial, x_ave);
l2=length(err2);
%% randomized_gossip
[err1, x1]=randomized_gossip(kmax, G, x_initial, x_ave);
l1=length(err1);
%% greedy_gossip
[err3, x3]=greedy_gossip(kmax, G, x_initial, x_ave);
l3=length(err3);
%% broadcast_gossip
[err4, x4]=broadcast_gossip(kmax, G, x_initial, x_ave);
l4=length(err4);
%% PDMM
[err5, k]=PDMM(kmax, G, x_initial, x_ave, A, c);
l5=length(err5);
%% sum-weight_gossip
[err6, x6]=sum_weight_gossip(kmax, G, x_initial, x_ave);
l6=length(err6);
%% experimental results
figure(1);
plot (1:l2, err2,'c', 1:l1, err1, 'r', 1:l3, err3, 'k', 1:l6, err6,
'b', 1:l4, err4, 'g', 1:l5, err5, 'm', 'LineWidth', 1);
xlabel ('number of iterations');
ylabel ('||x(k)-x_{ave}*1||');
legend('asynchronous distributed averaging', 'randomized gossip',
'greedy gossip with eavesdropping', 'sum-weight gossip', 'broadcast
weighted gossip', 'PDMM(c=-2.5)');
title('4-Regular Grid Topology, n=200');
grid on;
set(gca,'yscale','log');
```

8.2.2 Asynchronous Distributed Averaging

```
function [err, x] = asynchronous_averaging(kmax, G, x, x_ave,N)
% iterate until convergence
k=0;
err=inf;
d=degree(G);
i=zeros(0,1);
while(err(end)>1e-12) && (k<kmax)
    %select node randmly
    i(end+1)=randi(N);
    nei=neighbors(G, i(end));

    %update node
    x([i(end); nei])=sum(x([i(end);nei]))/(d(i(end))+1);

    %compute the iteration error
    k=k+1;
    err(k)=norm(x-x_ave);
end
```

8.2.3 Randomized Gossip

```
function [err, x]=randomized_gossip(kmax, G, x, x_ave,N)
%iterate utill convergence
k=0;
err=inf;
d=degree(G);
i=zeros(0,1);
while(err(end)>1e-12) && (k<kmax)

   %select nodes randomly
   i(end+1)=randi(N);
   nei=neighbors(G,i(end));
   j=nei(randi(d(i(end))));


   %update nodes
   x([i(end) j])=.5*sum(x([i(end) j]));

   %compute estimation error
   k=k+1;
   err(k)=norm(x-x_ave);
end
```

8.2.4 GGE

```
function [err, x]=greedy_gossip(kmax, G, x, x_ave,N)
%iterate utill convergence
k=0;
err=inf;
d=degree(G);
i=zeros(0,1);
```

```
while(err(end)>1e-12) && (k<kmax)

    %select nodes randomly
    i(end+1)=randi(N);
    nei=neighbors(G,i(end));
    diff=abs(x(nei)-x(i(end)));
    [Y I]=max(diff);
    j=nei(I);

    %update nodes
    x([i(end) j])=.5*sum(x([i(end) j]));

    %compute estimation error
    k=k+1;
    err(k)=norm(x-x_ave);
end
```

8.2.5 Sum-weight Gossip

```
function [err, x]=sum_weight_gossip(kmax, G, s, x_ave, N)
% iterate until convergence
k=0;
err=inf;
d=degree(G);
i=zeros(0,1);
w=ones(N,1);%initialize the weight vector
x=zeros(N,1);
while(err(end)>1e-12) && (k<kmax)
    %select node randmly
    i(end+1)=randi(N);
    nei=neighbors(G, i(end));
     j=nei(randi(d(i(end))));

    %update sum vector
    s(i(end))=s(i(end))/2;
    s(j)=s(j)+s(i(end));

    %update the weight vector
    w(i(end))=w(i(end))/2;
    w(j)=w(j)+w(i(end));

    %update the node value
    x=s./w;

    %compute the iteration error
    k=k+1;
    err(k)=norm(x-x_ave);
end
```

8.2.6 Broadcast Weighted Gossip

```
function [err, x] = broadcast_gossip(kmax, G, s, x_ave,N)
% iterate until convergence
k=0;
err=inf;
d=degree(G);
i=zeros(0,1);
w=ones(N,1);%initialize the weight vector
x=zeros(N,1);
while(err(end)>1e-12) && (k<kmax)
    %select node randmly
    i(end+1)=randi(N);
    nei=neighbors(G, i(end));

    %update sum vector
    s(i(end))=s(i(end))/(d(i(end))+1);
    s(nei)=s(nei)+s(i(end));

    %update the weight vector
    w(i(end))=w(i(end))/(d(i(end))+1);
    w(nei)=w(nei)+w(i(end));

    %update the node value
    x=s./w;

    %compute the iteration error
    k=k+1;
    err(k)=norm(x-x_ave);
end
```

8.2.7 PDMM

```
%% choice of c
temp=20;
sigma=1;
kmax=1e6;
[A,G]=Adjacency_matrix(20,10);
N=190;
nd=randi(200,[10 1]); %consider 5% failure
G=rmnode(G,nd);
x_initial=normrnd(temp ,sigma, [N,1]);
x_ave=mean(x_initial);
s=zeros(0,1);
% threshold=0.05;
for c=0:0.5:4
    [err, k]=PDMM(kmax, G, x_initial, x_ave, A, c, N);
     s(end+1)=k;
end
%%
 c=0:0.5:4
plot(c,s)
```

```
title('4-Regular Grid Topology, n=200, p=0.2');
xlabel('-c');
ylabel('number of iterations');
text(3.5, 90000,'||x(k)-x_{ave}||<10^{-12}');
function [err, k]=PDMM(kmax, G, x, x_ave, A, c,N)
%compute A matrix baed on adjacency matrix
for i=1:N
    for j=1:N
        if i<j
            A(i,j)=-A(i,j);
        else
            continue
        end
    end
end

%parameter setting
d=degree(G);
lambda=1*ones(N,N);
i=zeros(0,1);
k=0;
err=inf;
x_initial=x;

%iteration
while(err(end)>1e-12) && (k<kmax)
    %select a node randomly
    i(end+1)=randi(N);
    nei=neighbors(G, i(end));
    A_nei=A(i(end), nei);
    lam=lambda(nei, i(end));

    %update the node value
    x(i(end))=
    (x_initial(i(end))+A_nei*lam+c*abs(A_nei)*x(nei))/(1+c*d(i(end)));
    lambda(i(end),nei)=
    lambda(nei,i(end))'-c.*A_nei.*(x(i(end))-x(nei))';

    %compute the iteration error
    k=k+1;
    err(k)=norm(x-x_ave);

end
```

8.3 In Case of Transmission Failure

8.3.1 Main File

```
close all
clc
```

```
clear all
temp=20;
sigma=1;
kmax=3e5;
c=0.4;
threshold=0.3;
x_initial=normrnd(temp ,sigma, [200,1]);
x_ave=mean(x_initial);
[A,G]=Adjacency_matrix(20,10);
for n=1:20
%% asynchronous_averaging
[err2(n,:), x2]=tr_asynchronous_averaging(kmax, G, x_initial, x_ave,
threshold);
l2=length(err2);
%% randomized_gossip
[err1(n,:), x1]=tr_randomized_gossip(kmax, G, x_initial, x_ave,
threshold);
l1=length(err1);
%% greedy_gossip
[err3(n,:), x3]=tr_greedy_gossip(kmax, G, x_initial, x_ave,
threshold);
l3=length(err3);
%% broadcast_gossip
[err4(n,:), x4]=tr_broadcast_gossip(kmax, G, x_initial, x_ave,
threshold);
l4=length(err4);
%% PDMM
[err5(n,:), k]=tr_PDMM(kmax, G, x_initial, x_ave, A, c, threshold);

l5=length(err5);
%% sum-weight_gossip
[err6(n,:), x6]=tr_sum_weight_gossip(kmax, G, x_initial, x_ave,

threshold);
l6=length(err6);
%%
n=n+1;
end
err1=mean(err1,1);
err2=mean(err2,1);
err3=mean(err3,1);
err4=mean(err4,1);
err5=mean(err5,1);
err6=mean(err6,1);
%% experimental results
figure(1);
plot(1:l2, err2,'c', 1:l1, err1, 'r', 1:l3, err3, 'k', 1:l6, err6,
'b', 1:l4, err4, 'g', 1:l5, err5, 'm', 'LineWidth', 1);
xlabel('number_of_iterations');
ylabel('||x(k)-x_{ave}*1||');
```

```
legend('asynchronous distributed averaging', 'randomized gossip',
'greedy gossip with eavesdropping', 'sum-weight gossip', 'broadcast
weighted gossip', 'PDMM(c=-0.4)');
title('4-Regular Grid Topology, n=200, p=0.3');
grid on;
set(gca,'yscale','log');
```

8.3.2 Asynchronous Distributed Averaging

```
function [err, x] = tr_asynchronous_averaging(kmax, G, x, x_ave, threshold)
% iterate until convergence
k=0;
err=inf;
d=degree(G);
i=zeros(0,1);
while(err(end)>1e-12) && (k<kmax)
    %select node randmly
    i(end+1)=randi(200);
    nei=neighbors(G, i(end));
    trf=rand([d(i(end)), 1])>threshold;% transmission failure from node j
    nei=nei(find(trf==1));
    dd=length(nei);
    %update node
     x(i(end))=sum(x([i(end);nei]))/(dd+1);
    if dd~=0
        trf=rand([dd, 1])>threshold;%transmission failure from noe i
        nei=nei(find(trf==1));
        dd=length(nei);
        if dd~=0
            x([nei])=x(i(end));
        end
    end
    %compute the iteration error
    k=k+1;
    err(k)=norm(x-x_ave);
end
```

8.3.3 Randomized Gossip

```
function [err, x]=tr_randomized_gossip(kmax, G, x, x_ave, threshold)
%iterate utill convergence
k=0;
err=inf;
d=degree(G);
i=zeros(0,1);
while(err(end)>1e-12) && (k<kmax)

   %select nodes randomly
   i(end+1)=randi(200);
   nei=neighbors(G,i(end));
   j=nei(randi(d(i(end))));
```

```
    trf=rand(1)>threshold;
    if trf==1

        %update nodes
        x(i(end))=.5*sum(x([i(end) j]));
         trf=rand(1)>threshold;
         if trf==1
             x(j)=x(i(end));
         end
    end


    %compute estimation error
    k=k+1;
    err(k)=norm(x-x_ave);
end
```

8.3.4 GGE

```
function [err, x]=tr_greedy_gossip(kmax, G, x, x_ave, threshold)
%iterate utill convergence
k=0;
err=inf;
d=degree(G);
i=zeros(0,1);
while(err(end)>1e-12) && (k<kmax)

    %select nodes randomly
    i(end+1)=randi(200);
    nei=neighbors(G,i(end));
    trf=rand([d(i(end)), 1])>threshold;%model the transmission failure from node
    nei=nei(find(trf==1));
    dd=length(nei);
    if dd~=0
        diff=abs(x(nei)-x(i(end)));
        [Y I]=max(diff);
        j=nei(I);

        %update nodes
        x(i(end))=.5*sum(x([i(end) j]));
        trf=rand(1)>threshold;%transmission failure form node i
        if trf==1
            x(j)=x(i(end));
        end
    end
    %compute estimation error
    k=k+1;
    err(k)=norm(x-x_ave);
end
```

8.3.5 Sum-weight Gossip

```
function [err, x]=tr_sum_weight_gossip(kmax, G, s, x_ave, threshold)
% iterate until convergence
k=0;
err=inf;
d=degree(G);
i=zeros(0,1);
w=ones(200,1);%initialize the weight vector
x=zeros(200,1);
while(err(end)>1e-12) && (k<kmax)
    %select node randmly
    i(end+1)=randi(200);
    nei=neighbors(G, i(end));
     j=nei(randi(d(i(end))));
    trf=rand(1)>threshold;
    if trf==1
        %update sum vector
        s(i(end))=s(i(end))/2;


        %update the weight vector
        w(i(end))=w(i(end))/2;
        trf=rand(1)>threshold;
        if trf==1
            s(j)=s(j)+s(i(end));
            w(j)=w(j)+w(i(end));
        end
    end

        %update the node value
        x=s./w;

    %compute the iteration error
    k=k+1;
    err(k)=norm(x-x_ave);
end
```

8.3.6 Broadcast Weighted Gossip

```
function [err, x] = tr_broadcast_gossip(kmax, G, s, x_ave, threshold)
% iterate until convergence
k=0;
err=inf;
d=degree(G);
i=zeros(0,1);
w=ones(200,1);%initialize the weight vector
x=zeros(200,1);
while(err(end)>1e-12) && (k<kmax)
    %select node randmly
```

```
    i(end+1)=randi(200);
    nei=neighbors(G, i(end));
    trf=rand([d(i(end)), 1])>threshold;% transmission failure form node j
    nei=nei(find(trf==1));
    dd=length(nei);

    %update sum vector
    s(i(end))=s(i(end))/(d(i(end))+1);

    %update the weight vector
    w(i(end))=w(i(end))/(d(i(end))+1);

    %if transmissions all fail, no neighbouring nodes update w and s
    if dd~=0
        trf=rand([dd, 1])>threshold;
        nei=nei(find(trf==1));
        dd=length(nei);
        if dd~=0
            s(nei)=s(nei)+s(i(end));
            w(nei)=w(nei)+w(i(end));
        end
    end

    %update the node value
    x=s./w;

    %compute the iteration error
    k=k+1;
    err(k)=norm(x-x_ave);
end
```

8.3.7 PDMM

```
function [err, k]=tr_PDMM(kmax, G, x, x_ave, A, c, threshold)
%compute A matrix baed on adjacency matrix
for i=1:200
    for j=1:200
        if i<j
            A(i,j)=-A(i,j);
        else
            continue
        end
    end
end

%parameter setting
d=degree(G);
lambda=1*ones(200,200);
i=zeros(0,1);
k=0;
```

```
err=inf;
x_initial=x;


%iteration
while(err(end)>1e-12) && (k<kmax)
    %select a node randomly
    i(end+1)=randi(200);
    nei=neighbors(G, i(end));
    trf=rand([d(i(end)), 1])>threshold;%model the transmission failure
    nei=nei(find(trf==1));
    dd=length(nei);
    if dd~=0
        A_nei=A(i(end), nei);
        lam=lambda(nei, i(end));

        %update the node value
        x(i(end))=(x_initial(i(end))+A_nei*lam+c*abs(A_nei)*x(nei))/(1+c*dd);
        lambda(i(end),nei)=lambda(nei,i(end))'-c.*A_nei.*(x(i(end))-x(nei))';
    end
    %compute the iteration error
    k=k+1;
    err(k)=norm(x-x_ave);

end
```

8.4 In Case of Lost of Node

8.4.1 Main File

```
clear all
clc
close all
temp=20;
sigma=1;
kmax=1e6;
c=2.5;
[A,G]=Adjacency_matrix(20,10);
N=150;
nd=randi(200,[20 1]); %consider 20% failure
G=rmnode(G,nd);
nd=randi(180,[20 1]);
G=rmnode(G,nd);
nd=randi(160,[10 1]);
G=rmnode(G,nd);
A = adjacency(G);
x_initial=normrnd(temp ,sigma, [N,1]);
x_ave=mean(x_initial);
%% asynchronous_averaging
[err2, x2]=asynchronous_averaging(kmax, G, x_initial, x_ave, N);
```

```
l2=length(err2);
%% randomized_gossip
[err1, x1]=randomized_gossip(kmax, G, x_initial, x_ave,N);
l1=length(err1);
%% greedy_gossip
[err3, x3]=greedy_gossip(kmax, G, x_initial, x_ave,N);
l3=length(err3);
%% broadcast_gossip
[err4, x4]=broadcast_gossip(kmax, G, x_initial, x_ave,N);
l4=length(err4);
%% PDMM
[err5, k]=PDMM(kmax, G, x_initial, x_ave, A, c,N);
l5=length(err5);
%% sum-weight_gossip
[err6, x6]=sum_weight_gossip(kmax, G, x_initial, x_ave,N);
l6=length(err6);
%% experimental results
figure(1);
plot (1:l2, err2,'c', 1:l1, err1,  'r', 1:l3, err3, 'k', 1:l6, err6, 'b', 1:l4,
xlabel ('number of iterations');
ylabel ('||x(k)-x_{ave}*1||');
legend('asynchronous distributed averaging', 'randomized gossip',
'greedy gossip with eavesdropping', 'sum-weight gossip', 'broadcast weighted go
% axis([0 5e5 1e-2 1e0]);
title('4-Regular Grid Topology, n=150');
grid on;
set(gca,'yscale','log');
```

8.5 Fire Detection

```
temp=20;
sigma=20
for N=1:10
    x(N,:)=normrnd(temp ,sigma, [200,1]);
end
%% nodes within fire region
F=[10,29,30,31,50];
for N=1:10
    x(N, [F])=x(N,[F])+60;
end
%% decision rule
T=mean(x,1);
fire_regin=find(T>50)
```