Distributed Fusion in Sensor Networks

Jie Gao
Computer Science Department
Stony Brook University
Papers


• [Xiao05] Lin Xiao, Stephen Boyd and Sanjay Lall, A Scheme for Robust Distributed Sensor Fusion Based on Average Consensus, IPSN'05, 2005.


• Acknowledgement: many slides/figures are borrowed from Lin Xiao.
How to diffuse information?

• One node has a piece of information that it wants to send to everyone.
  – Flood, multi-cast.
• Every node has a piece of information that it wants to send to everyone.
  – Multi-round flooding.

• How do we diffuse information in real life?
  Gossip.
Uniform gossip

- Each node $x$ randomly picks another node $y$ and send to $y$ all the information $x$ has.

- After $O(\log n)$ rounds, every node has all the information with high probability.

- Totally distributed.
- Isotropic protocol.
Other applications

• Load balancing:
  – $N$ machines with different work load.
  – Goal: balance the load.

• Diffusion-based load balancing
  – each machine picks randomly another machine $y$ and shift part of its extra load, if any, to $y$.

• Good for the case when the work load of a job is unknown until it starts.
Use distributed diffusion for computing
Parameter estimation

- We want to fit a linear model to the sensor data.
- E.g., linear fitting.
Maximum likelihood estimation

- estimate a vector of unknown parameters $\theta \in \mathbb{R}^m$ with $n$ sensors
  \[ y_i = A_i \theta + v_i, \quad i = 1, \ldots, n \]
  measurements $y_i \in \mathbb{R}^{m_i}$, noises $v_i \sim \mathcal{N}(0, \Sigma_i)$ independent

- aggregate measurement ($\sum m_i \geq m$)
  \[ y = A \theta + v = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_n \end{bmatrix} \theta + \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \]

- maximum likelihood estimate given by weighted least-squares solution
  \[ \hat{\theta}_{ML} = (A^T \Sigma^{-1} A)^{-1} A^T \Sigma^{-1} y, \quad \text{where } \Sigma = \text{Diag}(\Sigma_1, \ldots, \Sigma_n) \]
Example: target localization

- estimate target position \((\theta_1, \theta_2)\) (red point) within unit square

- 20 range sensors located at \((s_{i1}, s_{i2})\), 
  \(i = 1, \ldots, 20\) (blue spots)

- each sensor measures distance to target 
  \(r_i\), with additive noise \(v_i \sim \mathcal{N}(0, 0.1)\)

- sensor output

\[
y_i = r_i + a_i^T \begin{bmatrix} s_{i1} \\ s_{i2} \end{bmatrix} \approx a_i^T \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} + v_i
\]

\(a_i\): unit vector from sensor to target
How to estimate $\theta$?

- Gather all the information and run the centralized maximum likelihood estimate.
- Or,
- Use a distributed fusion algorithm:
  - Each sensor exchanges data with its neighbors and carries out local computation, e.g., a least-square estimate.
  - Eventually each sensor obtains a good estimation.
- Advantages:
  - Completely distributed.
  - Robust to link dynamics, only requires a mild assumption on the network connectivity.
  - No assumption on routing protocol or any global info.
Distributed average consensus

• Let’s start with a simple task.

• Goal: compute the average of the sensor readings by a distributed iterative algorithm.

• Assume sensors are synchronized. $x(t)$ is the value of sensor $x$ at time $t$. 
Algorithm

- compute average $\bar{x} = \frac{1}{n} \sum_i x_i$ (using local communication, iteration)
- each node takes a weighted average of its own and neighbors’ values:

$$x_i(t + 1) = W_{ii}x_i(t) + \sum_{j \in \mathcal{N}_i} W_{ij}x_j(t)$$

usually $W_{ij} > 0$; but (surprisingly) this is not necessary for all edges
• Write the algorithm in a matrix form.
• **W**: the weighted adjacency matrix. The value at position \((i, j)\) is \(W_{i,j}\). It is a matrix of size \(n\) by \(n\).
• **x(t)**: the sensor values at time \(t\), a vector of size \(n\).
• We know: \(x(t+1) = Wx(t)\).
• Inductively, \(x(t) = W^tx(0)\).
• We hope the iterative algorithm converge to the correct average.
Performance

• Questions:
  – Does this algorithm converge?
  – How fast does it converge?
  – How to choose the weights so that the algorithm converges quickly?
Convergence condition: intuition

- The vector \((1, 1, \ldots, 1)\) is a fixed point.
- Each row sums up to 1.
Convergence condition: intuition

• Think the value as money. The total money in the system should be kept the same.
• Mass conservation.
• $\Rightarrow$ each column sums up to 1.
Doubly stochastic matrix

• $W$ must be a **doubly stochastic matrix**: all the row sum up to 1; and all the columns sum up to 1.
Convergence condition: intuition

• The algorithm should converge to the average.
• Write the average in a matrix form.
• Average vector: \( \frac{1}{n} \mathbf{1} \mathbf{1}^T \mathbf{x}(0) \).

\[
\begin{bmatrix}
\frac{1}{n} & \frac{1}{n} & \ldots & \frac{1}{n} \\
\frac{1}{n} & \frac{1}{n} & \ldots & \frac{1}{n} \\
\ldots & \ldots & \ldots & \ldots \\
\frac{1}{n} & \frac{1}{n} & \ldots & \frac{1}{n}
\end{bmatrix}
\]

• We want \( \mathbf{W}^t \rightarrow \frac{1}{n} \mathbf{1} \mathbf{1}^T \mathbf{x}(0) \), as \( t \rightarrow \infty \).
Convergence condition

• Theorem: \( \lim_{t \to \infty} x(t) = \frac{1}{n} \sum_{i=1}^{n} x(0) \) if and only if \( W \) is a doubly stochastic matrix and the spectral radius of \((W - 11^T/n)\) is less than 1.
A detour on matrix theory
Matrix, eigenvalues, eigenvectors

- An $n$ by $n$ matrix $A$.
- Eigenvalues: $\lambda_1, \lambda_2, \ldots, \lambda_n$. (real numbers)
- Corresponding eigenvectors: $v_1, v_2, \ldots, v_n$. (non-zero vector of size $n$).
- $Av_i = \lambda_i v_i$.
- $A^2v_i = A(Av_i) = A(\lambda_i v_i) = \lambda_i (Av_i) = \lambda_i^2 v_i$.
- Inductively, $A^k v_i = \lambda_i^k v_i$. 
Spectral radius

- Spectral radius of $M$: $\rho(A) = \max |\lambda_i|$.  

- Theorem: \[ \lim_{k \to \infty} A^k = 0 \quad \text{if and only if} \quad \rho(A) < 1. \]

- Proof: (⇒) Suppose $\lambda = \rho(A)$ with eigenvector $v$. 
  \[
  0 = (\lim A^k)v = \lim A^k v = \lim \lambda^k v = (\lim \lambda^k) v.
  \]
  Since $v$ is non-zero, $\lim \lambda^k = 0$. This shows $\rho(A) < 1$.

- (⇐) This direction uses Jordan Normal Form.
Back to distributed diffusion
Convergence condition

- Theorem: \( \lim_{t \to \infty} x(t) = \frac{1}{n} \sum_{i=1}^{n} x(0) \) if and only if \( W \) is a doubly stochastic matrix and the spectral radius of \( (W - 11^T/n) \) is less than 1.

\[ \begin{array}{c|c}
\text{Row i} & W \\
\hline
\text{Column j} & \\
\end{array} \]
Proof of the convergence condition

• Sufficiency: if \( W \) is a doubly stochastic matrix and \( \rho(W - 11^T/n) < 1 \), then

\[
\lim_{t \to \infty} x(t) = \frac{1}{n} \sum_{i=1}^{n} x(0)
\]

• Proof:

1. \( W \) is doubly stochastic. Thus

\[
1^T W = 1^T, \quad W 1 = 1,
\]

2. Now we have

\[
W^t - 11^T/n = W^t (I - 11^T/n)
= W^t (I - 11^T/n)^t
= \left((W (I - 11^T/n))^t\right)^t
= (W - 11^T/n)^t,
\]

3. Since \( \rho(W - 11^T/n) < 1 \),

\[
\lim_{t \to \infty} W^t = \frac{11^T}{n}.
\]
Convergence rate

- convergence factor

\[ \rho(W - 11^T/n) = \max_{i=2,\ldots,n} |\lambda_i(W)| \]

asymptotically

\[ \|x(t) - \bar{x}\|_2 \leq \rho^t \|x(0) - \bar{x}\|_2 \]

The smaller the better.

- associated mixing time

\[ \tau = \frac{1}{\log(1/\rho)} \]

(asymptotic) number of steps for the error to decrease by factor \(e\)
Fastest iterative algorithm?

- Given a graph, find the weight function such that the iterative algorithm converges fastest.

  \[
  \begin{align*}
  \text{minimize} & \quad \rho(W - 11^T/n) \\
  \text{subject to} & \quad W \in \mathcal{S}, \quad 1^T W = 1^T, \quad W 1 = 1,
  \end{align*}
  \]

- Theorem (Xiao & Boyd 04): When the matrix $W$ is symmetric, the above optimization problem can be formulated by a semi-definite programming and can be solved efficiently.
Choosing the weight

- **constant weight on all edges, e.g., max-degree weights**
  \[ W_{ij} = \frac{1}{1 + d_{\text{max}}}, \quad \{i, j\} \in \mathcal{E} \]

  (self-weights given by \( W_{ii} = 1 - \sum_{j \in \mathcal{N}_i} W_{ij} \))

- **Metropolis weights** (only needs local information)
  \[
  W_{ij}(t) = \begin{cases} 
  \frac{1}{\max\{d_i(t), d_j(t)\}} & \text{if } \{i, j\} \in \mathcal{E}(t), \\
  1 - \sum_{\{i, k\} \in \mathcal{E}(t)} W_{ik}(t) & \text{if } i = j, \\
  0 & \text{otherwise.}
  \end{cases}
  \]
Example: weight selection

![Graphs showing different weight selection methods: max-degree weights, Metropolis weights, and fastest weights.](image)

- **max-degree weights**
  - $\rho^{md} = 0.78$
  - $\tau^{md} = 4.02$

- **Metropolis weights**
  - $\rho^{mh} = 0.77$
  - $\tau^{mh} = 3.91$

- **fastest weights**
  - $\rho^* = 0.72$
  - $\tau^* = 3.06$
Extension to changing topologies
Changing topologies

• The sensor network topology changes over time.
  – Link failure.
  – Mobility.
  – Power constraints.
  – Channel fading.

• However, the distributed fusion algorithm only assumes a mild condition on network connectivity -- the network is “connected in a long run”.
Changing topologies

- The communication graph $G(t)$ is time-varying.
- For $n$ nodes, there are only finitely many communication graphs, and finitely many weight functions.
- There are a subset of graphs that appear infinitely many times.
- If the collection of graphs that appear infinitely many times are **jointly connected**, then the algorithm converges.
Changing topologies

- We emphasize that this is a very mild condition on connectivity.
- Many links can fail permanently.
- We only require that a connected graph “survives” in the sequence of (possibly disconnected) graphs.
Choice of weights

- Maximum-degree weights

\[
W_{ij}(t) = \begin{cases} 
  \frac{1}{n} & \text{if } \{i, j\} \in \mathcal{E}(t) \\
  \frac{d_i(t)}{n} & \text{if } i = j \\
  0 & \text{otherwise}
\end{cases}
\]

- Metropolis weights (only need real-time local information)

\[
W_{ij}(t) = \begin{cases} 
  \frac{1}{1 + \text{max}\{d_i(t), d_j(t)\}} & \text{if } \{i, j\} \in \mathcal{E}(t) \\
  1 - \sum_{\{i, k\} \in \mathcal{E}(t)} W_{ik}(t) & \text{if } i = j \\
  0 & \text{otherwise}
\end{cases}
\]
Robust convergence

- **Theorem**: If the communication graphs that occur infinitely often in \( \{G(t)\}_{t=0}^{\infty} \) are jointly connected, then the iteration

  \[
x(t + 1) = W(t)x(t)
  \]

  converges to the average for any \( x(0) \in \mathbb{R}^n \), with either Metropolis weights or maximum-degree weights.

- Intuition: the weight function \( W \) (for both max degree and Metropolis) is paracontracting.

  \[
  Mx \neq x \iff \|Mx\| < \|x\|.
  \]

- It preserves the fixed-point subspace and contract all other vectors. Thus if we apply the matrix infinitely many times, the limit has to be a fixed point.
Extension to parameter estimation
Maximum likelihood estimation

- estimate a vector of unknown parameters $\theta \in \mathbb{R}^m$ with $n$ sensors
  \[ y_i = A_i \theta + v_i, \quad i = 1, \ldots, n \]
  measurements $y_i \in \mathbb{R}^{m_i}$, noises $v_i \sim \mathcal{N}(0, \Sigma_i)$ independent

- aggregate measurement ($\sum m_i \geq m$)
  \[ y = A\theta + v = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_n \end{bmatrix} \theta + \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \]

- maximum likelihood estimate given by weighted least-squares solution
  \[ \hat{\theta}_{ML} = (A^T \Sigma^{-1} A)^{-1} A^T \Sigma^{-1} y, \quad \text{where } \Sigma = \text{Diag}(\Sigma_1, \ldots, \Sigma_n) \]
Distributed parameter estimation

• A sensor node $i$ knows

$$y_i = A_i \theta + v_i, \quad i = 1, \ldots, n$$

• Goal: we want to evaluate in a distributed fashion

$$\hat{\theta}_{ML} = (A^T \Sigma^{-1} A)^{-1} A^T \Sigma^{-1} y, \quad \text{where } \Sigma = \text{Diag}(\Sigma_1, \ldots, \Sigma_n)$$

• Idea: use the average consensus algorithm.
Distributed parameter estimation

- distributed computation of

\[ \hat{\theta}_{\text{ML}} = (A^T \Sigma^{-1} A)^{-1} A^T \Sigma^{-1} y \]

define

\[ P = [A_1^T \ldots A_n^T] \begin{bmatrix} \Sigma_1 & \cdots & \Sigma_1 \\ \vdots & \ddots & \vdots \\ \Sigma_n & \cdots & \Sigma_n \end{bmatrix} \begin{bmatrix} A_1 \\ \vdots \\ A_n \end{bmatrix} = \sum_{i=1}^{n} A_i^T \Sigma_i^{-1} A_i \]

\[ q = [A_1^T \ldots A_n^T] \begin{bmatrix} \Sigma_1 & \cdots & \Sigma_1 \\ \vdots & \ddots & \vdots \\ \Sigma_n & \cdots & \Sigma_n \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^{n} A_i^T \Sigma_i^{-1} y_i \]

therefore

\[ \hat{\theta}_{\text{ML}} = P^{-1} q = \left( \sum_{i=1}^{n} A_i^T \Sigma_i^{-1} A_i \right)^{-1} \sum_{i=1}^{n} A_i^T \Sigma_i^{-1} y_i \]

- idea: distributed average consensus of matrices and vectors (entry-wise)

\[ P_i(0) = A_i^T \Sigma_i^{-1} A_i \quad (m \times m \text{ composite information matrix}) \]

\[ q_i(0) = A_i^T \Sigma_i^{-1} y_i \quad (m \times 1 \text{ composite measurement vector}) \]
• distributed average consensus (same form as in scalar case)

\[ P_i(t + 1) = W_{ii}(t)P_i(t) + \sum_{j \in N_i(t)} W_{ij}(t)P_j(t) \]

\[ q_i(t + 1) = W_{ii}(t)q_i(t) + \sum_{j \in N_i(t)} W_{ij}(t)q_j(t) \]

• use distributed average consensus to compute

\[ \lim_{t \to \infty} P_i(t) = \frac{1}{n} \sum_{i=1}^{n} A_i^T \Sigma_i^{-1} A_i = \frac{1}{n} P \]

\[ \lim_{t \to \infty} q_i(t) = \frac{1}{n} \sum_{i=1}^{n} A_i^T \Sigma_i^{-1} y_i = \frac{1}{n} q \]

• each node can eventually compute

\[ \hat{\theta}_{\text{ML}} = P_i(\infty)^{-1}q_i(\infty) = P^{-1}q \]
Intermediate estimates

- instead of waiting for convergence, use

\[ \hat{\theta}_i(t) = P_i(t)^{-1}q_i(t) \]

available at node \( i \) as soon as \( P_i(t) \) invertible
Properties

- universal data structure for storage and communication
  \[ P_i(t) \in \mathbb{R}^{m \times m}, \quad q_i(t) \in \mathbb{R}^m \]
  independent of local dimension \( m_i \) (of \( A_i \) and \( y_i \))

- isotropic protocol (diffusion)
  - taking weighted average of neighbors’ data (Metropolis weights)
  - no explicit routing/broadcasting involved
  - convergence under dynamically changing topology

- intermediate estimates \( \hat{\theta}_i(t) = P_i(t)^{-1} q_i(t) \)
  - available at each node whenever \( P_i(t) \) invertible
  - always unbiased
  - error covariances converge to global optimal
Simulation

- estimate target position \((\theta_1, \theta_2)\) (red point) within unit square

- 20 range sensors located at \((s_{i1}, s_{i2})\), \(i = 1, \ldots, 20\) (blue spots)

- each sensor measures distance to target \(r_i\), with additive noise \(v_i \sim \mathcal{N}(0, 0.1)\)

- sensor output

\[
y_i = r_i + a_i^T \begin{bmatrix} s_{i1} \\ s_{i2} \end{bmatrix} \approx a_i^T \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} + v_i
\]

\(a_i\): unit vector from sensor to target
A demo
A larger example

- 50 sensors, 200 communication links
- $\theta \in \mathbb{R}^5$, measurements $y_i = a_i^T \theta + v_i$
  $a_i$ uniform distribution on unit sphere
  $v_i \sim \mathcal{N}(0, 1)$
- convergence of mean-squares error
  - left: distributed fusion on fixed graph
  - right: links fail (iid) with prob. $3/4$
Random gossip model
Random gossip

• Completely asynchronous. No synchronized clock is needed.
• At each time, a node can only talk to one other node.
• Distributed average consensus: each node picks one node with some probability distribution and compute the average.
• Natural averaging algorithm: each node uniformly randomly picks a neighbor and compute the avg.
• Again, one can find the optimal averaging distribution by convex programming s.t. the algorithm converges fastest.
Random geometric graphs

- $G^d(n, r)$: place $n$ nodes uniformly random in a $d$-dimensional cube and connect two nodes if they are within distance $r$.

- Bad news: the natural averaging algorithm converges about the same order as the optimal one $\Rightarrow$ both are slow.

- Good news: no need to optimize. The natural averaging is a local and distributed algorithm with optimal performance.
• Preferential attachment model: a new comer connects an edge to the existing nodes with probability proportion to the degree.
• “Rich get richer”.
• The graph obtained is an expander:
  – spectral gap is a constant;
  – the second largest eigenvalue is small enough;
  – random walk mixes fast;
• Optimal averaging algorithm has an averaging time $O(\log 1/\varepsilon)$, independent of the graph size.
• Averaging on P2P network is extremely fast.
Summary

• One of the few examples that are so robust to topological changes.

• Many applications on similar problems.

• Distributed optimization.