Gossip Algorithms

Devavrat Shah

MIT

Motivation

- Ad-hoc networks
  - Not deliberately designed with an "infrastructure"
- Some examples
  - Sensor networks
    - formed by randomly deployed sensors in a geographic area
    - for sensing and monitoring enviroment, surveillance, etc.
  - Peer-to-peer networks
    - formed by computers over Internet or overlay networks
    - for sharing information, computation, etc.
  - Mobile networks
    - formed by automobiles, radio devices
    - for infrastructure-less communication
- In such networks, nodes need to collect, process and communicate information over wireless channel

Characteristics of Ad hoc Networks

- Nodes in such networks
  - Do not have access to addressing, routing information, etc.
  - Possibly have severely limited resources (like energy, computation)
    - nodes may hibernate or leave the network or die
  - Do not know the global network topology,
    - only have access to local information, i.e. neighbors
- Algorithms deployed in such networks need to be
  - Completely distributed
  - Robust against node failures or changes in topology
  - Simple enough so as to be implementable
  - Wireless communication friendly
- Traditional algorithms generally do not meet mentioned constraints

  $\rightarrow$ Randomized gossip algorithms are very well suited

Randomized Gossip Algorithm

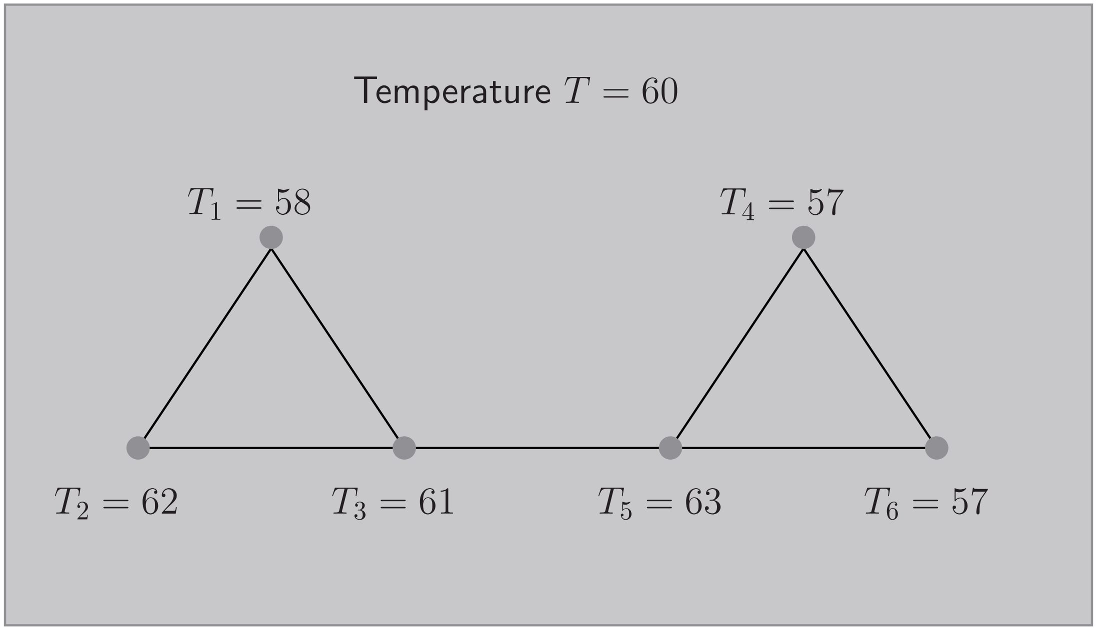
- Characteristics
  - Pair-wise operations
    - $\longrightarrow$ satisfies wireless constraint
  - Iterative and distributed
  - Use of local information only
  - Randomized

- Gossip algorithm
  - By design, *redundant* and hence not optimal

  $\longrightarrow$ robust against node failures and changes in topology

- In this talk, we will consider gossip algorithms for averaging
  - Compute average of values given at nodes of the network

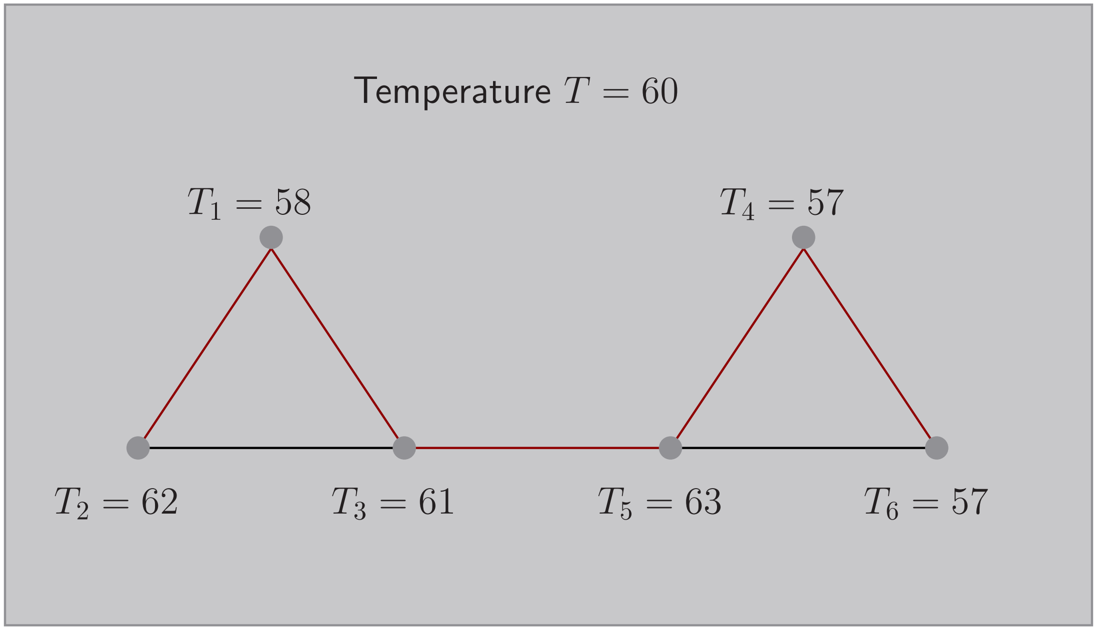
(Toy) Example



- Given a network of temperature sensors
  - To sense ambient temperature T
- Sensors have noisy reading, $T_i = T + \eta_i$
  - An unbiased MMSE: $\hat{T} = \frac{1}{6}\sum_i T_i$

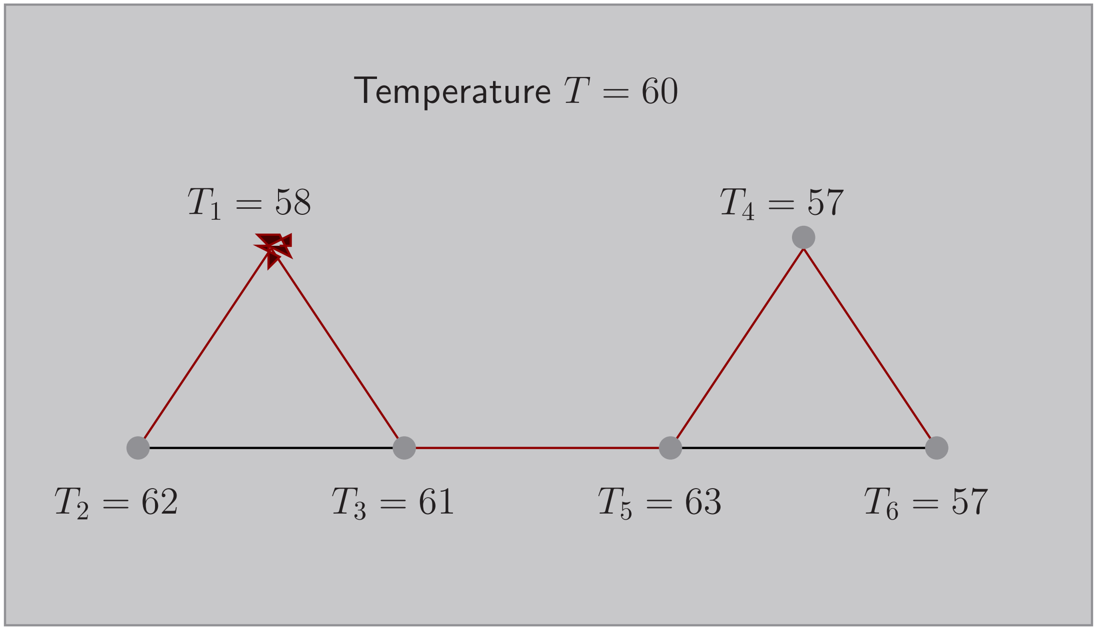
$\longrightarrow$ Compute average at each sensor in a distributed manner

(Toy) Example: A Traditional Algorithm



- First, nodes form an "infrastructure" in a distributed fashion
  - A spanning tree
- Use spanning tree to exchange values in an orderly fashion
  - To compute average
- This is simple and distributed, however...

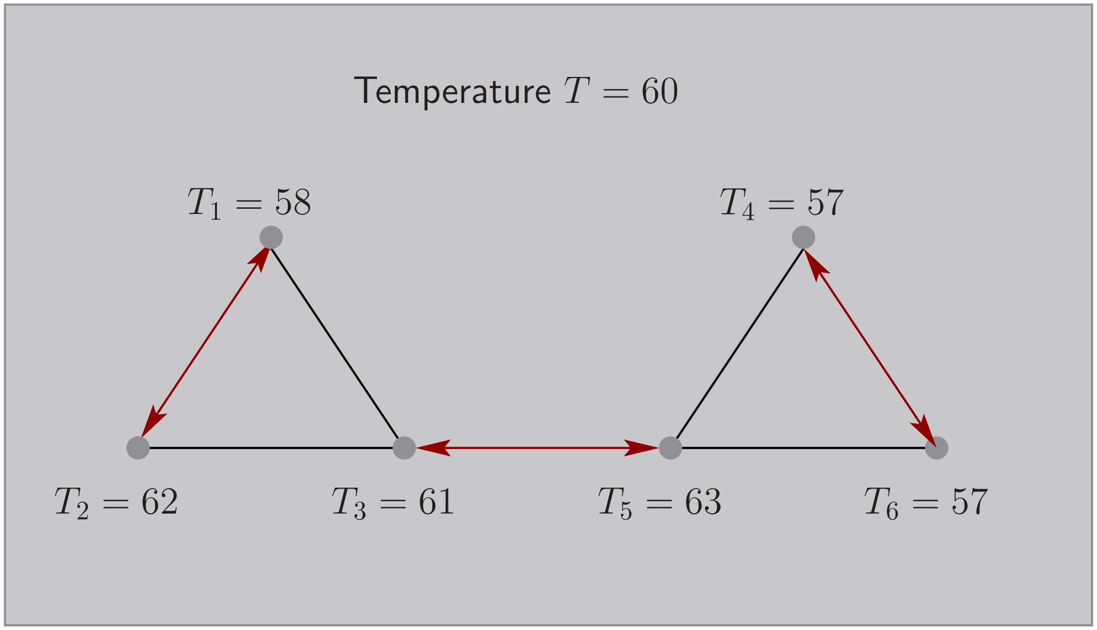
(Toy) Example: A Traditional Algorithm



- A node may fail or die
  - Requires re-computation of infrastructure

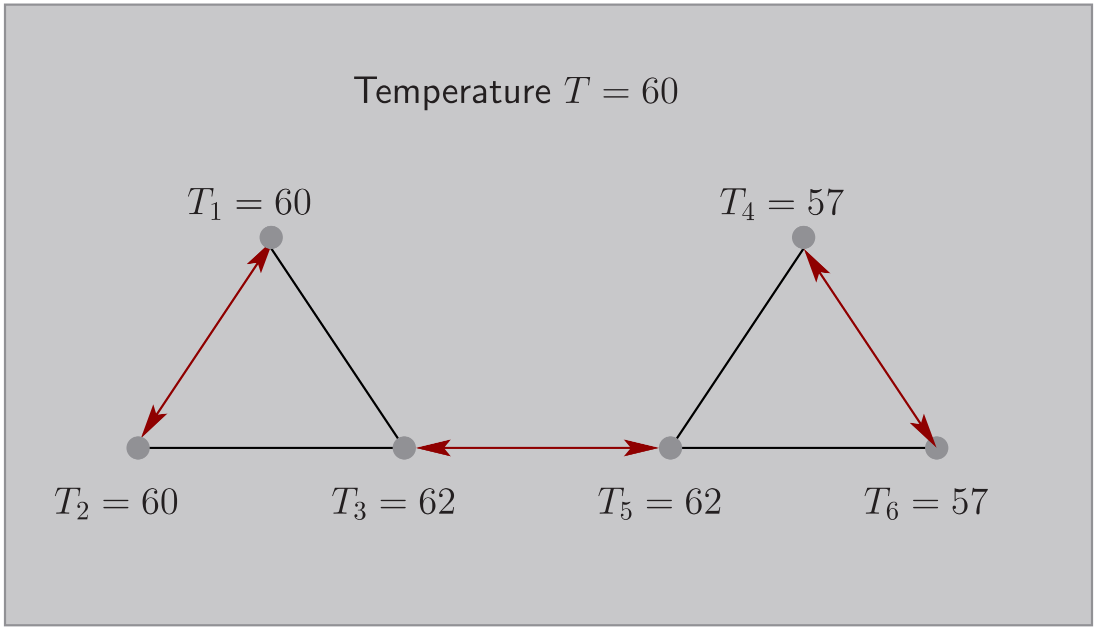
  $\longrightarrow$ Algorithms are not robust !

(Toy) Example: Gossip Algorithm



- Every time-step,
  - A node contacts one of its neighbor at random and forms a pair
  - Paired nodes average their current estimates

(Toy) Example: Gossip Algorithm



- Every time-step,
  - A node contacts one of its neighbor at random and forms a pairs
  - Paired nodes average their current estimates
- Estimate of each node converges to average
- Next, lets look at another example where averaging via gossip is useful

(Not So Toy) Example

- In many networking scenarios, scheduling is essential
    - To resolve contention of resources
        - such as bandwidth, hardware, etc.
    - Between different *tasks* or *flows* or ...

- Some well-known examples of scheduling
    - In Input-Queued switches to resolve contention in accessing cross-bar fabric
    - In Wireless networks to resolve contention in accessing bandwidth
    - In Routing to resolve contention in accessing network links
    - .....

Scheduling Algorithms

- The network performance
    - In terms of *throughput* and *delay*
    - Is strongly affected by the scheduling algorithm

- Ideally, one would like to have scheduling algorithm
    - That guarantees (close to) optimal performance, and
    - Is easy to implement, i.e.
        - requires few simple operations
        - distributed and robust against changes in network

- Design of implementable high-performance scheduling algorithms has been of central interest for more than a decade

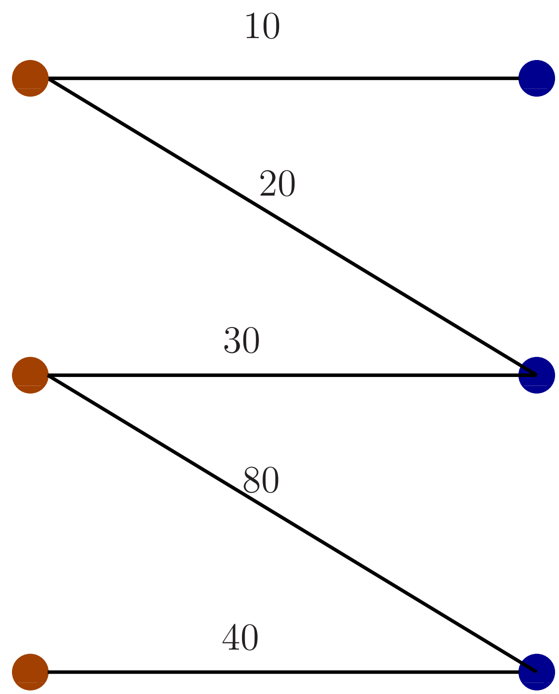
Scheduling Algorithms

- Tassiulas and Ephremides (1992) proposed
  - *Max-Weight* scheduling algorithm, that is
    - each time among all *allowable* schedules
    - choose the one with maximum *weight*
    - where, weight is induced by (function of) *queue-size*

- They showed it to be *throughput* optimal
  - In a very general network setup

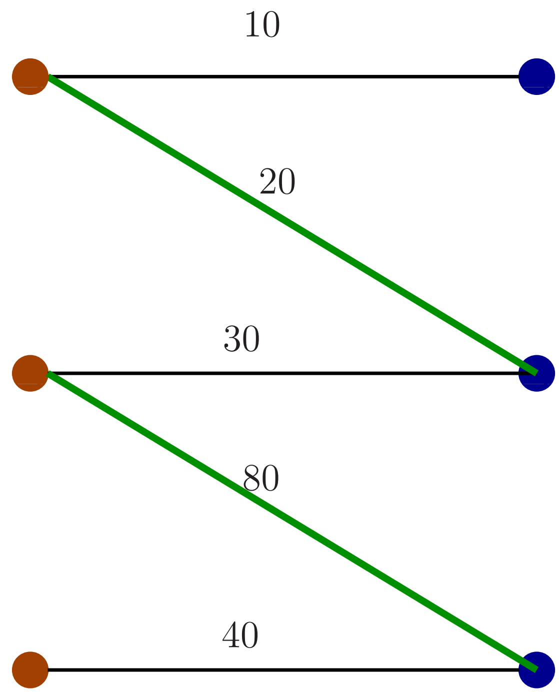
- Next, lets looks at some examples of this algorithm

An Example of Scheduling



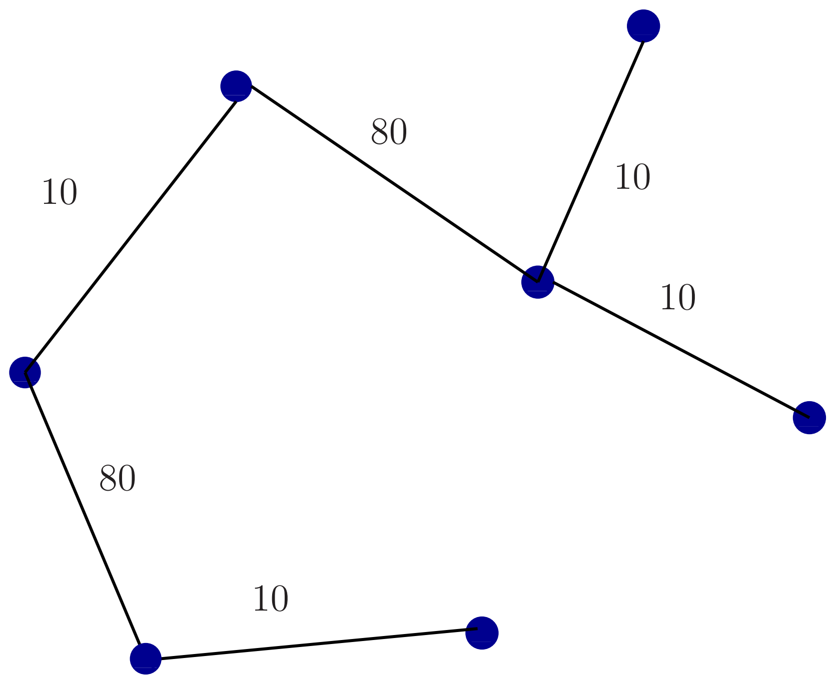
Switch Bipartite Graph

An Example of Scheduling



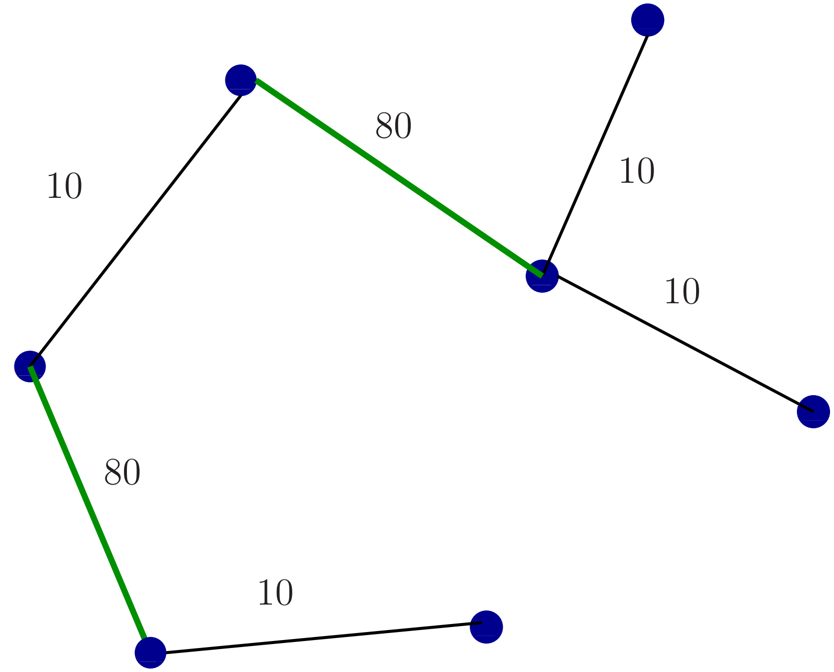
Maximum Weight Matching

An Example of Scheduling



Wireless Interference Graph

An Example of Scheduling



Maximum Weight Matching

Scheduling Algorithms

- The Max-Weight algorithm has good performance
  - ◦ However, finding such schedules can become difficult
  - ◦ For example, finding Max Weight Matching in a graph
    - – requires $O(n^3)$ operations (for a centralized algorithm)
  - ◦ It becomes worse when constraints correspond to *independent set*
    - – it is NP-hard and hard to approximate

→ Is it possible to have any throughput optimal simple solution?

- ◦ Yes, gossip comes to *rescue*

Scheduling Algorithm via Gossip

- Algorithm SCH-GOSSIP
  - Let $S(\tau)$ be schedule at time τ.
  - Select $S(\tau+1)$ as follows:
    - generate schedule $R(\tau+1)$ by RANDOM
    - select $S(\tau+1) = \text{MIX}(S(\tau), R(\tau))$

  (Similar to Tassiulas (1998) and Giaccone-Prabhakar-S (2002))

- For SCH-GOSSIP, we obtain the following (Modiano-S-Zussman (2006))
- **Theorem 1.** Let there exists finite $\delta, \delta_1, \gamma > 0$ such that for any τ,

  P1. $\Pr\left(R(\tau+1)\text{equal to Max Wt Schedule}\right) > \delta$,

  P2. $\Pr\left(W(S(\tau+1)) \geq (1-\gamma)\max\{W(S(\tau)), W(R(\tau+1))\}\right) > \delta_1$.

  Then, the algorithm is $1-\gamma-2\sqrt{\frac{\delta_1}{\delta}}$ approximation of throughput optimal algorithm.

Scheduling Algorithm via Gossip

- Theorem implies that, to obtain simple distributed high-throughput algorithm
  - We need a distributed *sampler*, and
  - A distributed *comparison* algorithm
- Distributed random sampler
  - Can be obtained using simple, local random schemes
    - for matchings, k-factors, independent set,...
- Distributed comparison algorithm
  - Comparing weights is the same as comparing averages

  $\rightarrow$ Gossip algorithm for averaging can be used
- There are many other examples where averaging is useful *subroutine*
  - Computing Top-k eigenvalue, distributed LP, (some) asynchronous optimization, etc

A Bit of History

- Distributed algorithm based on linear dynamics for
  - Averaging or Agreement problem
  - Was first studied by
    - Tsitsiklis (1984), and
    - Tsitsiklis-Bertsekas-Athans (1986)

- Similar algorithm has been re-discovered in many other contexts
  - Load balancing by Rabani-Sinclair-Wanka (1998)

- A good re-cap of the history
  - In recent paper by Blondel-Hendrickx-Olshevsky-Tsitsiklis (2005)

Rest of the Talk

- Averaging
  - Setup, Problem and Quantity of interest
- Algorithm I
  - Randomized terative averaging
- Algorithm II
  - Based on property of exponential variables
- Algorithm III
  - Based on non-reversible Random Walks
- Summary

Averaging

Setup

- Some notation
  - Consider a network of n nodes
  - $G = (\{1, \dots, n\}, E)$ is the corresponding network graph
    - edge $(i, j) \in E$ iff nodes i and j are connected
  - Node i has some real value x_i
    - let vector $\bar{x} = [x_i]$
- We wish to design algorithm for the following task: at each node,
  - compute $x_{ave} = \frac{\sum_i x_i}{n}$
- Quantity of interest: computation time
  - How long does it take to obtain good approximation of x_{ave} at all nodes

Time Model

- Synchronous: slotted time (discrete)
  - In matching model multiple node pairs exchange messages
    - these pairs form a matching
  - In push (pull) model a node contacts exactly one other node and sends message
    - however, a node may be contacted by multiple nodes
- Asynchronous: continuous time
  - Each node has its independent exponential clock of rate 1
    - a node performs an operation only when its clock ticks
  - No two clocks tick at the same time
    - only one pair is performing an operation at an instance
- We will consider asynchronous time model
  - However, *all* results extend to synchronous time models as well
  - Of course, it requires a lot more work (and we've done it)

Algorithm I

Stephen Boyd Arpita Ghosh Balaji Prabhakar

Asynchronous Averaging Algorithm

- Time step of algorithm is any of the n clock tick
  - Equivalently, tick of an exponential clock of rate n
- Algorithm
  - Initially, at time $t = 0$, for $i = 1, \ldots, n$,
    - node i sets its estimate, $x_i(0) = x_i$
  - At time t_k (k^{th} clock-tick)
    - one of the n nodes becomes active at random, say i
    - node i contacts one of its neighbor, say j, with prob. P_{ij}
    - nodes i and j set their new estimates as follows:

$$x_i(t_k) = x_j(t_k) = \frac{1}{2}\left(x_i(t_{k-1}) + x_j(t_{k-1})\right).$$

- Question: how long does it take to compute average given graph G and matrix P?

Quantity of Interest

- **Definition 1.** ϵ-Averaging time, $T_{ave}(\epsilon, P)$, as

$$T_{ave}(\epsilon, P) = \sup_{x(0)} \inf \left\{ t \ : \ \Pr\left(\frac{\|x(t) - x_{ave}\mathbf{1}\|}{\|x(0)\|} > \epsilon \right) < \epsilon \right\}.$$

for gossip algorithm based on P

- Some notation
  - Let $E(P) = \{(i, j) \in E \ : \ P_{ij} > 0$ or $P_{ji} > 0\}$
  - Let $G(P) = (V, E(P))$
  - It is required that $G(P)$ is connected for $T_{ave}(\epsilon, P) < \infty$

Main Result

- **Theorem 2.** For any $0 < \epsilon < 1$,

$$\frac{\log \epsilon^{-1}}{2n \log \lambda_{\max}^{-1}(W)} \leq T_{ave}(\epsilon, P) \leq \frac{3 \log \epsilon^{-1}}{n \log \lambda_{\max}^{-1}(W)},$$

  - where $\lambda_{\max}(W)$ is the second largest eigenvalue of

$$W = I - \frac{1}{2n} D + \frac{1}{2n} \left(P + P^T \right),$$

  with $D = diag(D_1, \ldots, D_n)$ such that

$$D_i = \sum_{k=1}^{n} (P_{ik} + P_{ki}).$$

- If $P = P^T$, then $W = I - \frac{1}{n}(I - P)$

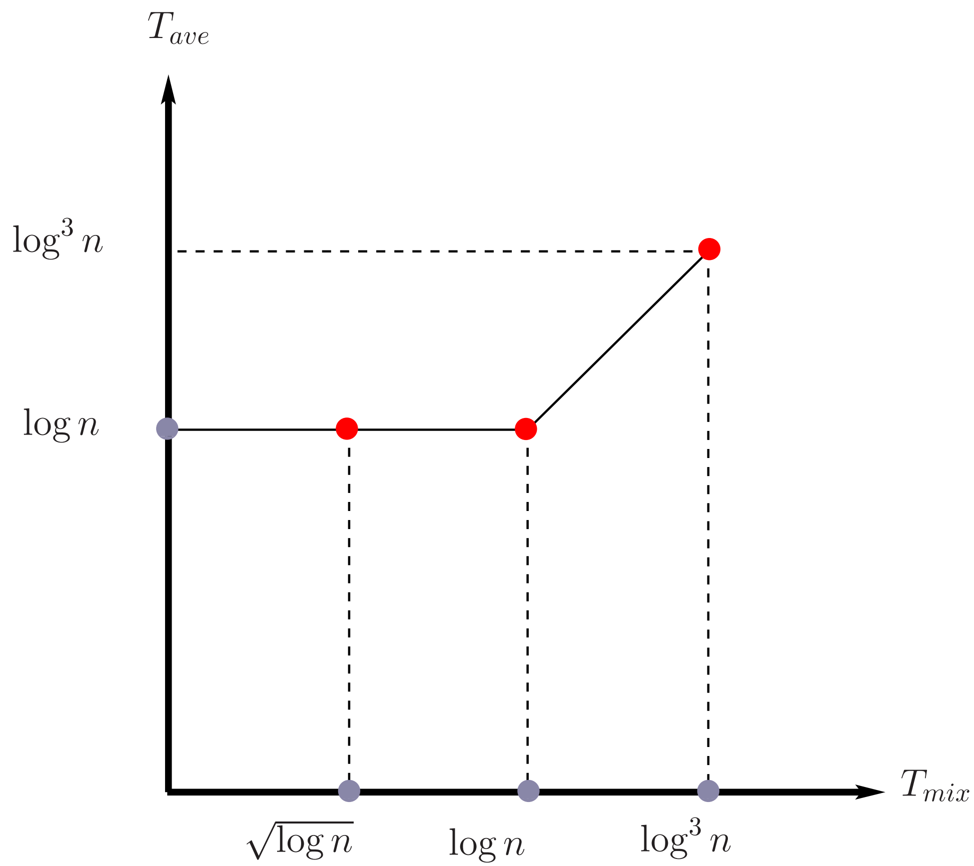
  $\rightarrow \lambda_{\max}(W) = 1 - \frac{1}{n}(1 - \lambda_2(P))$,

  $\rightarrow T_{ave}(\epsilon, P) \sim \frac{\log \epsilon^{-1}}{1 - \lambda_2(P)}$

Pictorial view of Theorem 2

- Relation between averaging and mixing time ($\epsilon = n^{-2}$)

$$T_{ave} = \Theta\left(T_{mix} + \log n\right)$$



Some Implications

- Let's consider the complete graph

- Lower Bound

  ◦ For any P, $\lambda_2(P) \in [-1, 1]$

  ◦ Thus, $(1 - \lambda_2(P))^{-1} \geq 0.5$

  $\rightarrow T_{ave}(\epsilon, P) \geq 0.5 \log \epsilon^{-1}$

  ◦ This recovers result of Karp et. al. (2000)

- An upper bound: consider $P = [1/n]$

  ◦ $\lambda_2(P) = 0$

  $\rightarrow$ That is, $T_{ave}(\epsilon, P) \leq 3n \log \epsilon^{-1}$

  ◦ In particular, $T_{ave}(1/n^2, P) = O(\log n)$

  ◦ This recovers results of Kempe et al. (2003)

Fastest Averaging Algorithm

- Given graph G, the fastest averaging algorithm corresponds to
  - Communication matrix P such that
    - corresponding W has minimum $\lambda_{\max}(W)$
  - For any P, $W = W^T$
- Fastest averaging can be posed as the following optimization problem:

$$\min \lambda_{\max}(W), \text{ given}$$
$$W = \frac{1}{n}\sum_{ij} P_{ij} W_{ij}$$
$$P_{ij} \geq 0,\ P_{ij} = 0 \text{ if } (i,j) \notin E$$
$$\sum_j P_{ij} = 1,\ \forall i.$$
$$\text{where, } W_{ij} = I - \frac{(e_i - e_j)(e_i - e_j)^T}{2}.$$

- This is known to be a Semi-Definite Program (SDP)
  - Shown recently by Boyd, Diaconis and Xiao (2003)
  - Can be solved in polynomial time

Distributed Optimization

- The SDP corresponding to fast averaging algorithm
  - Can be solved by a centralized procedure
  - We need a distributed gossip algorithm to do so

- We obtain a distributed gossip subgradient method based on
  - Problem structure,
  - Gossip for averaging, and
  - Recent results on approx. subgradient by Kiwiel (2004)

An Example

- Next, we evaluate performance of algorithm
  - ◦ Wireless network
    - – how long does it take to average?

- We'll see
  - ◦ Geometric random graph, $G(n,r)$, as a model for wireless network
  - ◦ Compute $\lambda_{\max}$ for $G(n,r)$-conformant P

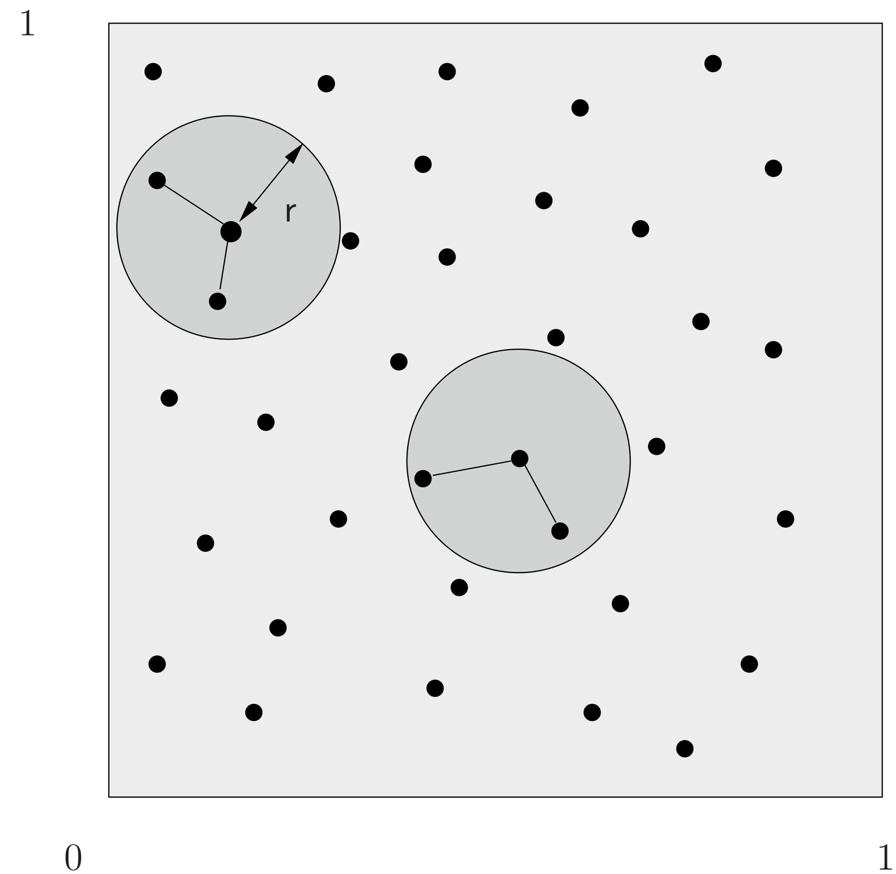
  $\rightarrow$ Evaluate of averaging time on $G(n,r)$

Wireless Network

- Wireless networks are formed by nodes placed in ad hoc manner in some geographic area
  - Two nodes within transmission range of each other can communicate

- Gupta and Kumar (2000) introduced Geometric random graph as a model for such ad hoc wireless networks
  - n nodes are thrown uniformly at random into a unit disc
  - Two nodes within distance r of each other are connected
    - r represents the transmission radius
  - Denoted as $G(n, r)$

An Example of $G(n, r)$



Averaging in $G(n, r)$

- To facilitate communication in network, the property of connectivity is desirable
    - for $r = 0$ the graph is disconnected
    - for $r = 1$ the graph is completely connected

  → What is the smallest value of r such that graph is connected?

  → How does probability of connectivity change with r?

- $G(n, r)$ exhibits threshold property in connectivity with critical $r_c = \Theta\left(\sqrt{\log n/n}\right)$
    - When $r < (1 - \epsilon)r_c$, $G(n, r)$ is disconnected w.p. tending to 1
    - When $r > (1 + \epsilon)r_c$, $G(n, r)$ is connected w.p. tending to 1

Averaging in $G(n, r)$

- We are interested in computing eigenvalues of $G(n, r)$-conformant P
  - In particular, what are the eigenvalues of P for natural gossip algorithm?
  - And, what is the smallest second eigenvalue?

- We obtain the following characterization of eigenvalues
- **Theorem 3.** For $G(n, r)$ with $r = \omega\left(\sqrt{\log n/n}\right)$, the second eigenvalue corresponding to natural gossip algorithm as well as the fastest gossip algorithm scales as $1 - \Theta(r^2)$.

$\longrightarrow$ For natural and fastest averaging algorithms,

$$T_{ave}(\epsilon, P) = \Theta(r^{-2} \log \epsilon^{-1})$$

Summary: Algorithm I

- Randomize gossip algorithm based on P
  - $T_{ave}(P) \sim \Theta\left(T_{mix}(P) + \log n\right)$

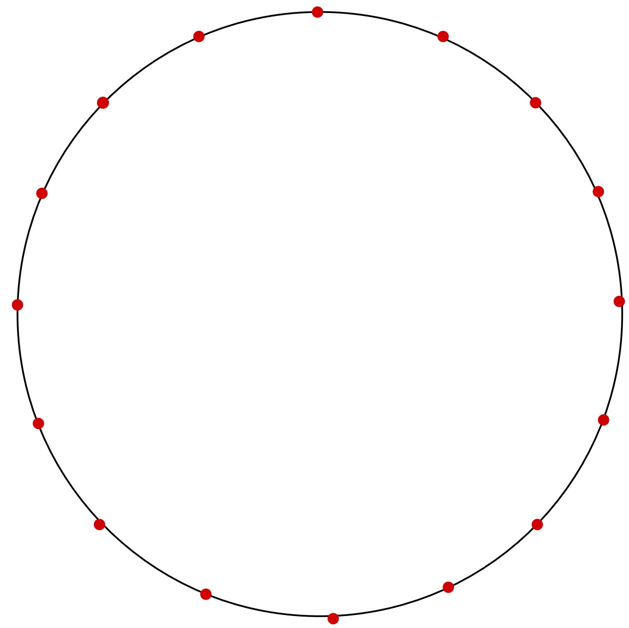
  $\rightarrow$ pair-wise constraint impose penalty of additive $\log n$

- Fastest gossip algorithm correspond to fastest mixing Markov chain
  - This can be posed as an SDP
  - It can be optimized via gossip algorithm

- Question: does this class of algorithms provide best performance?
  - Lets consider a ring graph

Ring Graph

- We'll consider performance of algorithm on ring graph
  - Each node the n nodes is connected to two of its neighbors



  - For any reversible (symmetric) P, $1 - \lambda_{max} = O(1/n^2)$

$\rightarrow T_{ave}(\epsilon) = \Omega(n^2 \log \epsilon^{-1})$ (achieved by symmetric RW)

  - Contrast this with centralized algorithm requiring $\Theta(n)$ time !

Ring Graph

- For a large class of *ring-type* graphs
  - For any reversible random walk, $1 - \lambda_{max} \sim 1/\text{diameter}^2$

  $\rightarrow T_{ave} \sim \log \epsilon^{-1} \times \text{diameter}^2$

- **Definition 2.** For a proper subset $S \subset V$, the *conductance* of a symmetric stochastic matrix P denoted as $\Phi(P)$ is defined as

$$\Phi_P = \min_{S \subset V : |S| \leq n/2} \frac{\sum_{i \in S, j \in S^c} (P_{ij} + P_{ji})}{|S|}.$$

- More generally,
  - $\Phi(P)^2/2 \leq (1 - \lambda_{\max}(P)) \leq 2\Phi(P)$

  $\rightarrow \log \epsilon^{-1}/\Phi(P) \leq T_{ave} \leq \log \epsilon^{-1}/\Phi(P)^2$
  - Usually, for reversible walks (i.e. P), its closer to the upper bound (like ring graph)

- Next, we consider a heuristic lower bound on T_{ave}

Heuristic Lower Bound

- We present a heuristic argument for lower bound
  - On exact averaging time, T_{ave}

- The lower bound suggests $T_{ave} = \Omega(1/\Phi(P))$
  - Where matrix $P = [P_{ij}]$ corresponds to the capacity induced on edges by the algorithm

- Consider the minimizing cut in definition of conductance $\Phi(P)$
  - Denoted by (S, S^c) with $|S| \leq n/2$
  - That minimizes $\frac{\sum_{i \in S, j \notin S^c} P_{ij}}{|S|}$
  - Across this cut, data crosses at rate $O(n\Phi(P))$

Heuristic Lower Bound

- For *exact* averaging
  - All nodes need to exchange information with all other nodes
  - That is, $\Theta(n)$ amount of information exchange need to happen through each cut
- Thus, the minimal time required for *exact* averaging

  $\rightarrow$ Is at least $\Omega(1/\Phi(P))$

  $\rightarrow$ $T_{ave} = \Omega(1/\Phi(P))$
- For example, for ring graph
  - $\Phi(P) = O(1/n)$ for any algorithm

  $\rightarrow$ That is, $T_{ave} = \Omega(n)$
- This naturally raises the following question
  - Is it possible to have T_{ave} scaling linearly in $1/\Phi(P)$?

Algorithm II

Damon Mosk Aoyama

Information Spreading

- We'll first consider a related task of spreading information
- As before, consider network of n nodes
  - ◦ $G = (\{1, \ldots, n\}, E)$ is the corresponding network graph
    - – edge $(i, j) \in E$ iff nodes i and j are connected
  - ◦ Node i has some information $\mathcal{I}_i$
- We wish to design algorithm for the following task:
  - ◦ spread information of each node to all the n nodes
  - ◦ same as computing minimum of values at all nodes
- Quantity of interest: computation time
  - ◦ How long does it take to spread information to all nodes

Asynchronous Averaging Algorithm

- Time step of algorithm is any of the n clock ticks
  - Equivalently, tick of an exponential clock of rate n
- Algorithm
  - Initially, at time $t = 0$, for $i = 1, \ldots, n$,
    - set of information node i, $S_i(0) = \{\mathcal{I}_i\}$
  - At time t_k (k^{th} clock-tick),
    - one of the n node becomes active at random, say i
    - node i contacts one of its neighbor, say j, with prob. P_{ij}
    - nodes i and j exchange all of each others information:

$$S_i(t_k) = S_j(t_k) = S_i(t_{k-1}) \cup S_j(t_{k-1}).$$

- Question: how long does it take to spread all information to all nodes given graph G and matrix P?

Quantity of Interest

- **Definition 3.** ϵ-Spreading time, of a communication matrix P, denoted by $T_{\mathsf{spr}}(\epsilon, P)$, is

$$T_{\mathsf{spr}}(\epsilon, P) = \sup_{i \in V} \inf\{t : \Pr(|S_i(t)| < n) \leq \epsilon\}.$$

- Some notation
    - Let $E(P) = \{(i, j) \in E \ : \ P_{ij} > 0$ or $P_{ji} > 0\}$
    - Let $G(P) = (V, E(P))$
    - It is required that $G(P)$ is connected for $T_{\mathsf{spr}}(\epsilon, P) < \infty$

Performance of Inf. Spr. Algorithm

- **Definition 4.** For a proper subset $S \subset V$, the *conductance* of a symmetric stochastic matrix P denoted as $\Phi(P)$ is defined as

$$\Phi_P = \min_{S \subset V : |S| \le n/2} \frac{\sum_{i \in S, j \in S^c} (P_{ij} + P_{ji})}{|S|}.$$

- **Theorem 4.** For any $\epsilon \in (0,1)$, the ϵ-spreading time, $T_{\mathsf{spr}}(\epsilon, P)$ is bounded as follows:

$$T_{\mathsf{spr}}(\epsilon, P) = O\left(\frac{\log n + \log \epsilon^{-1}}{\Phi(P)}\right).$$

- Next, we consider algorithm for averaging

Exponential Random Variable

- A random variable X has exponential distribution with rate λ

$$\Pr(X > t) = \exp(-\lambda t), \quad t \in \mathbb{R}_+$$

- Consider the following well-known property of exponential distribution
  - Let $X_1, \ldots, X_n$ be independent exponential variables
    - with parameters $\lambda_1, \ldots, \lambda_n$
  - Let $X^* = \min_{i=1}^n X_i$, then
    - X^* is exponential variable with rate $\sum_{i=1}^n \lambda_i$
- This naturally suggests an algorithm for computing average

Algorithm II

- The algorithm is described as follows
  - Initially, for $i = 1, \ldots, n$, node i has the value $x_i \geq 1$.
  - Each node i generates r independent random numbers $W_1^i, \ldots, W_r^i$, where the distribution of each W_ℓ^i is exponential with rate x_i (i.e., with mean $1/x_i$).
  - Each node i computes, for $\ell = 1, \ldots, r$, an estimate $\hat{W}_\ell^i$ of the minimum $\mathbf{W}_\ell = \min_{i=1}^n W_\ell^i$.
    - it can be computed using the information spreading algorithm
  - Each node i computes $\hat{x}_i = \frac{r}{\sum_{\ell=1}^r \hat{W}_\ell^i}$ as its estimate of $\sum_{i=1}^n x_i$.

Performance of Algorithm II

- **Theorem 5.** Given an information spreading algorithm with ϵ spreading time $\mathcal{T}_\epsilon$, the ϵ-averaging time of Algorithm II, $T_{ave}(\epsilon)$ is bounded above as

$$T_{ave}(\epsilon) = O\left(\epsilon^{-2} \log \epsilon^{-1} \mathcal{T}_{\epsilon/2}\right).$$

- Specifically, when Algorithm II uses the info. spr. algo. described earlier

$$T_{ave}(\epsilon) = O\left(\frac{\epsilon^{-2} \log \epsilon^{-1} (\log n + \log \epsilon^{-1})}{\Phi(P)}\right).$$

- The above utilizes good Large deviation properties of exponential distribution

Comparison: Algorithm I v/s Algorithm II

- Recall that for any random walk on G with transition matrix P

$$\frac{\Phi^2(P)}{2} \leq 1 - \lambda_{max}(P) \leq 2\Phi(P).$$

- From this, we find that
  - For Algorithm I

$$T_{ave}(\epsilon) \sim \frac{\log \epsilon^{-1}}{\Phi(P)^2}$$

  - For Algorithm II

$$T_{ave}(\epsilon) \sim \frac{\epsilon^{-2} \log^2(n + \epsilon^{-1})}{\Phi(P)}$$

- Is it possible to obtain best of both Algorithms
  - Scaling of $O(\log \epsilon^{-1}/\Phi(P))$?

Algorithm III

- We (Jung-S (2006)) find an algorithm
  - Based on non-reversible (non-symmetric) random walk on graph
  - Specifically, given any stochastic matrix P
  - Our gossip algorithm performs as follows:

$$T_{ave}(\epsilon) = O\left(\frac{\Delta \log \epsilon^{-1}}{\Phi(P)}\right)$$

where Δ is the maximum vertex degree of network graph

$\rightarrow$ The performance scales linearly in $1/\Phi(P)$, and

- Scales linearly in $\log \epsilon^{-1}$
- But has additional Δ factor

Summary

- Randomized Gossip Algorithm
  - An attractive algorithmic solution for sensor and peer-to-peer networks
  - Averaging: a very useful *subroutine* to distributed computation
- Algorithms I
  - Strongly related to Mixing time
- Algorithms II
  - Performance related to conductance

  $\longrightarrow$ Better scaling in number of nodes compared to Algorithm I
  - But, poor scaling with respect to ϵ
- Algorithm III:
  - Can we find good non-rev. RW in distributed fashion?