

# Computing Separable Functions via Gossip<sup>[1]</sup>


Presentation by Daniel Jones and Kishalay Banerjee


# Separable Functions

Suppose  $f$  is a function of a vector  $\vec{x}$ . We say that  $f$  is additively separable if there exists functions  $f_i$  such that  $f(\vec{x}) = \sum f_i(x_i)$ .

Here, our Separable Function is defined as -

vector of values


$$f(\vec{x}, S) = \sum_{i \in S} f_i(x_i)$$

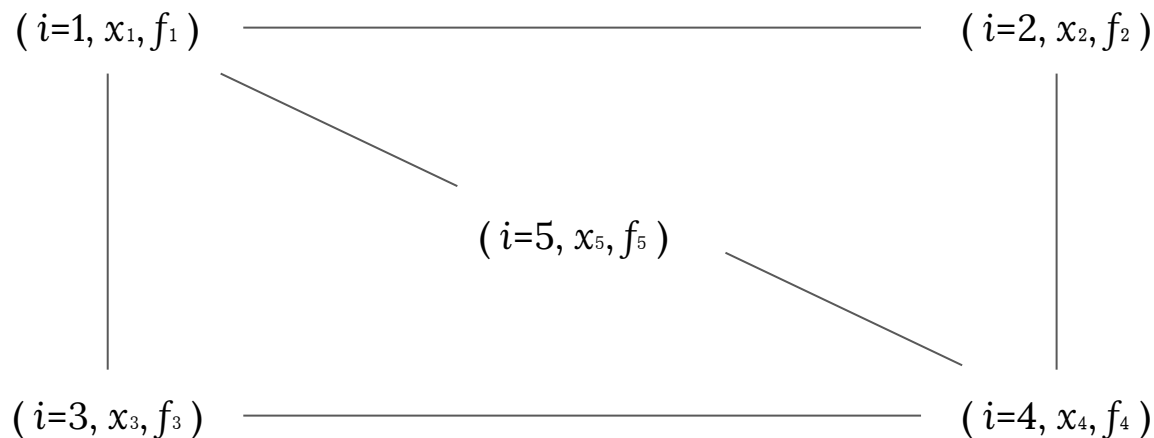


set S of node  
identifiers  $i = \{ 1, \dots, n \}$

# Separable Functions on a Network

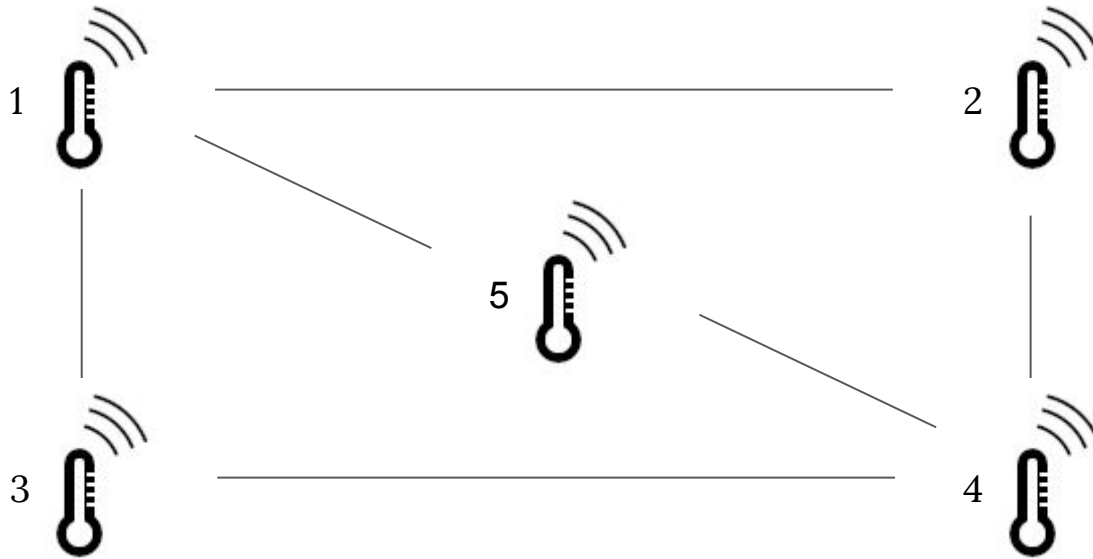
An *undirected, connected* graph  $G = (V, E)$  where each node  $i$  has a value  $x_i$  and function  $f_i$ .

Can we compute  $f(\vec{x}, V)$  ?



# Wireless Sensor Networks

Limitations: an ad-hoc network with unreliable wireless connections and no master nodes.



# Questions

1. Does an algorithm exist that can compute separable functions on a network in a totally distributed manner?
2. Can we find an upper-bound on such an algorithms execution time?

# Questions

1. Does an algorithm exist that can compute separable functions on a network in a totally distributed manner?

This paper describes a randomized, distributed algorithm which cannot compute exact values, instead it computes an estimate on each node, within an accuracy of  $(1 \pm \epsilon)$  multiplicative factor, with probability of at least  $(1 - \delta)$ .

2. Can we find an upper-bound on such an algorithms execution time?

This paper defines the computation time in terms of  $\epsilon$  and  $\delta$ , the  $(\epsilon, \delta)$ -computing time, this is then reduced to the time it takes for information to spread through the network, called the  $\delta$ -information-spreading time.

The  $\delta$ -information-spreading time is used to derive an upper bound on the algorithms execution time. Running the algorithm for the  $(\epsilon, \delta)$ -computing time upper-bound gives a probabilistic guarantee of the networks estimate.

# Time Model

Since this is a distributed computation, a **time model** determines the instances when nodes communicate with each other. Two such time models are considered in this paper -

1. Synchronous Time Model - In this model, time is slotted commonly across all nodes in the network. In any time slot, a particular node may contact one of its neighbouring nodes according to a random choice that is independent of other nodes' choices.
2. Asynchronous Time Model - In this model, each node has a clock that marks the jumps of a Poisson(1) process. When a particular node becomes active, it contacts one of its neighbours at random. In this system, we consider time to be discrete.

The paper considers the running time of algorithms in absolute time, which is the number of time slots in the synchronous model (or equivalently, the number of time jumps divided by  $n$ , in the asynchronous model).

# Avoiding Double Counting

In our model, *nodes do not have permanent identifiers*. If we were to naively pass values through the network, each node may receive the same message more than once. Without identifiers, there is no way to avoid counting repeated values twice (*double counting*).

Instead of using all the values (thus including repetitions), if we use minimum of the values at each node, the problem of double counting is eliminated.

But how do we transform the problem of computing a summation of values, into computing their minimum?



# A Special Property of $\text{Exp}(\lambda)$

The algorithm makes use of a key property of the Exponential distribution, which is -

If  $W_1, W_2, \dots, W_n$  are  $n$  independent random variables such that  $W_i \sim \text{Exp}(\lambda_i)$ ,  
Then,  $\min(W_i) \sim \text{Exp}(\sum \lambda_i)$

We proved this in the first problem sheet!

# Empirical Means and Expectations

- The empirical mean of a random variable is  $\frac{\sum_i \text{observation}_i}{\text{number of observations}}$
- We also know that  $\text{Exp}(a)$  has an expectation of  $\frac{1}{a}$
- Equate the two to get an estimate of  $a$ :  $a = \frac{\text{number of observations}}{\sum_i \text{observation}_i}$
- Therefore, for an exponential distribution with rate  $\sum \lambda_i$ :

$$\sum_i \lambda_i = \frac{\text{number of observations}}{\sum_i \text{observation}_i}$$

- Since  $\min(W_i) \sim \text{Exp}(\sum \lambda_i)$  we can draw the observations we need from  $\min(W_i)$ .

# COMP Algorithm

The COMP algorithm describes the steps by which the nodes compute estimates of  $f(\vec{x}) = \sum f_i(x_i)$ .

1. Each node starts with its value,  $x_i$ , and function,  $f_i$ . These are used to compute a value  $y_i$  s.t.  $y_i = f_i(x_i)$  with  $y_i \geq 1$ .
2. Each node  $i$  generates  $r$  independent random numbers distributed as  $\text{Exp}(y_i)$ . Each of which is an observation taken by a single node.
3. Each node  $i$  computes an estimates the minimum of each the  $n$  sets of  $r$  observations. This is done using an information spreading algorithm, SPREAD.
4. Each node  $i$  computes an estimate of  $\sum f_i(x_i)$  given by

$$\sum_i f_i(x_i) = \frac{r}{\sum_{l=1}^r \min_{i=1}^n (\text{observation}_l^i)}$$

# Computation Time

Given an algorithm  $C$ , and two positive quantities  $\epsilon$  and  $\delta$ , the minimum time the algorithm must run, such that the probabilities of the estimates of the function at each node are within the  $(1 \pm \epsilon)$  neighbourhood with at least a probability of  $(1 - \delta)$ , is termed as the  $(\epsilon, \delta)$  computing time of  $C$ , and is defined as follows -

$$T_C^{\text{cmp}}(\epsilon, \delta) = \sup_{f \in \mathcal{F}} \sup_{\vec{x} \in \mathbf{R}^n} \inf \left\{ \tau : \forall t \geq \tau, \Pr \left( \cup_{i=1}^n A_i^\epsilon(t) \right) \leq \delta \right\}$$

The algorithm described in the paper does not compute exact values of the separable function at each node in the network. Rather, an estimate is computed at each node, and then, the probability of that estimate lying within the  $(1 \pm \epsilon)$  neighbourhood of the true value is analysed. If  $\hat{y}_i(t)$  is the estimate of the function at node  $i$  and time  $t$ , the event that  $\hat{y}_i(t)$  does not lie in the  $(1 \pm \epsilon)$  neighbourhood is denoted as follows -

$$A_i^\epsilon(t) = \{ \hat{y}_i(t) \notin [(1 - \epsilon)f(\vec{x}, V), (1 + \epsilon)f(\vec{x}, V)] \}$$

# Upper Bound on Computation Time

Using the COMP algorithm, the paper has reduced the main problem into one of information spreading.

They then obtain an upper-bound on the  $\delta$ -information-spreading time of this gossip algorithm in terms of the conductance of a stochastic matrix  $P$  (a matrix representing the probability of communication between two nodes).

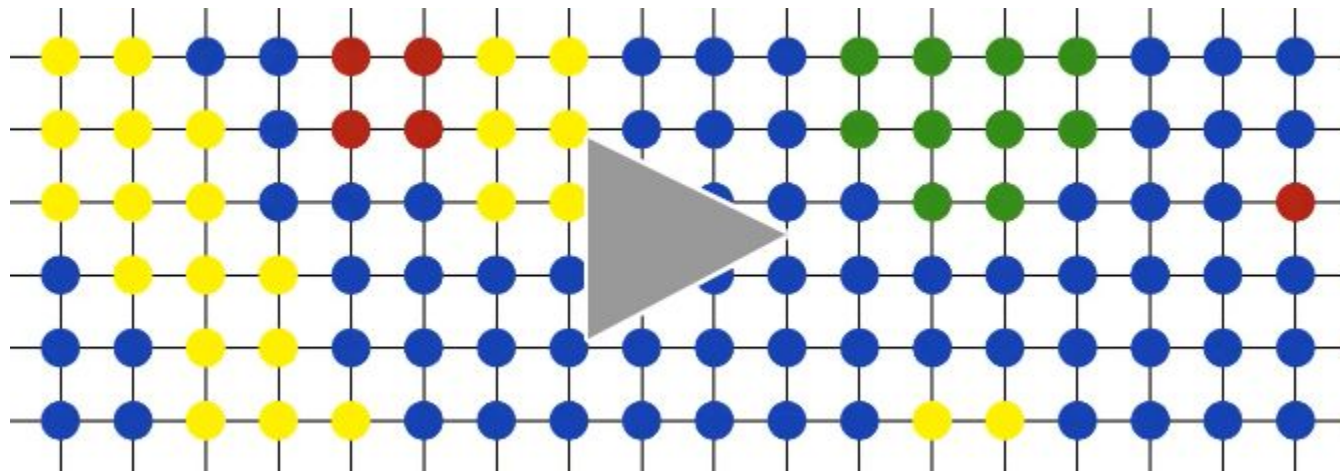
$$T_{\mathcal{P}}^{spr}(\delta) = O\left(\frac{\log n + \log \delta^{-1}}{\Phi(P)}\right)$$

Giving an upper-bound on the  $(\epsilon, \delta)$ -computing time:

$$T_{\mathcal{A}}^{cmp}(\epsilon, \delta) = O\left(\epsilon^{-2}(1 + \log \delta^{-1})T_{\mathcal{D}}^{spr}(\delta/2)\right)$$

# Averaging Colours on a Grid

For a grid network with colour values on each node, use the **COMP** and **SPREAD** algorithms to mix the colour of each node in the network (code can be found on [GitHub](#)).



# References

1. Mosk-Aoyama, Damon, and Devavrat Shah. "Computing separable functions via gossip." Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing. ACM, 2006. ([PDF](#))
2. Shah, Devavrat. "Gossip algorithms." Foundations and Trends® in Networking 3.1 (2009): 1-125. ([PDF](#))
3. S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Gossip algorithms: Design, analysis and applications. In Proceedings of IEEE INFOCOM 2005, pages 1653–1664, 2005.

Thank you!