INTRODUCTION TO DISTRIBUTED OPTIMIZATION AND GOSSIP ALGORITHMS

MATHIEU EVEN, SUPERVISED BY LAURENT MASSOULIÉ, APRIL 2020 TO JULY 2020

Abstract. In order to make predictions, Machine Learning requires heavy computations that depend on a set of data. In simple problems where no large amount of data is required, a *centralized* approach can be used. However, modern applications involve learning over exponentially large sets of data, that cannot be stored on a unique central unique. This leads to considering new optimization approaches, that are distributed. In this introduction to an area of research, we explain how simple optimization procedures (gradient descent under smoothness and strong convexity assumption) are adapted in a distributed setting.

Contents

1. Introduction 2
2. Problem Formulation 2
2.1. Notations and Different Assumptions 2
2.2. Centralized Scheme 3
2.3. Decentralized Scheme 3
3. Preliminaries on Optimization Methods 3
3.1. Full Gradient Descent 3
3.2. Coordinate Gradient Descent 4
3.3. Accelerating Gradient Descents 4
4. Synchronous Gossip Algorithms 4
4.1. Simple Synchronous Gossip Algorithm 5
4.2. Generalization to strongly convex and smooth functions f_i 5
4.3. Existing "Fast" Algorithms 5
5. Asynchronous Gossip Algorithms 6
5.1. Generalization to the Distributed Optimization Problem 6
5.2. Accelerating Asynchronous Gossip 7
6. Works and Future Perspectives 8
6.1. A Finer Modelling of Asynchrony in the Decentralized Scheme 8
6.2. Acceleration in the Decentralized Scheme 9
6.3. Ideas for Future Works and Conclusion 11
References 11

1. Introduction

In machine learning, a broad cast of problems require to find the minimizer of a certain function in order to compute an estimator. Often, this function takes the form:

$$F(x) = \sum_{i=1}^{n} f_i(x), \tag{1.1}$$

where $x \in \mathbb{R}^d$ is the variable to compute and each f_i depends on a subset of the data. Take for instance the classical logistic regression: given observed variables $X = (X_1, ..., X_N) \in \mathbb{R}^{N\times d}$ and their respective observed outputs $Y = (Y_1, ..., Y_n) \in \{0, 1\}^{N\times d'}$, one desires to find for the next entry X_{N+1}, in function of its value, the associated output. A classical procedure is to assume a linear relation between X_i and Y_i noised by a Gaussian centered variable: $Y_i = \theta^T X_i + \varepsilon_i$. In order to find θ, the statistician must then solve the following optimization problem:

$$\theta^* \in \arg\min_{\theta\in\mathbb{R}^{d'\times d}} \sum_{i=1}^{N} l_2(X_i, Y_i, \theta), \tag{1.2}$$

where $l_2(X_i, Y_i, \theta) = ||Y_i - \theta^T X_i||^2$ (squared euclidian distance). Equation 1.2 hence has the general form of Equation 1.1, each f_i depending on a set of data X_i, Y_i. If the data are centralized *i.e.* all the f_i's are accessible on a local central unit, many procedures exist in order to solve this problem [Bubeck, 2014], involving gradient descent flavoured algorithms.

Assuming centralized data is nowadays unrealistic for complicated machine learning problems. Indeed, as the amount of data at stake is increasing way faster than computer power and memories, one may find itself in cases where data are stacked in different clouds, computers, or hard-drive. The statistician can then copy these data on a central unit. However, that is not always the case. This can happen for different reasons:

- 1) Local memory is too small;
- 2) Local computational power is not enough;
- 3) Privacy issues: data are not allowed to be moved from their location.

In cases 1) and 3), the optimization procedure must take place without copying data on the central unit. In case 2), data can be copied on a central unit, but external computational power is needed. From these considerations, two main structures exist for optimization algorithms in our distributed framework: *centralized* algorithms, and *decentralized* ones.

Centralized algorithms take a *master-slave* structure: a central unit (*master*) has access to the parameter θ to compute, and workers that all have access to a small set of data and to some computational power. At each iteration of the algorithm, the central unit sends the parameter θ to a worker, who then computes a local gradient, sends it back to the central unit who finally update the parameter. Decentralized algorithms do not assume the existence of a central unit. Data are stacked on the dones of a communication network, and must communicate with their neighbors in order to reach consensus on the desired parameter θ. In this introduction to a area of research, we will only consider the decentralized framework.

This introduction to a area of research will proceed in the following way. Section 2 will formalize notations and define both centralized and decentralized structures.

2. Problem Formulation

We formalize the problem in this Section.

2.1. Notations and Different Assumptions.

We want to solve the optimization problem:

$$\min_{x\in\mathbb{R}^d} \sum_{i=1}^{n} f_i(x), \tag{2.1}$$

for some functions f_i defined on $\mathbb{R}^d$ and some integer n. Several different assumptions can be made on the *local* functions f_i. We denote $f(x) = \sum_i f_i(x)$ We will mainly talk about the following one.

Assumption 1. *Each f_i for $i \in \mathbb{N}$ is assumed L_i-smooth and σ_i-strongly convex,* i.e. $\forall x, y \in \mathbb{R}^d$*:*

$$f_i(x) \leq f_i(y) + \langle \nabla f_i(y), x - y \rangle + \frac{L_i}{2}||x - y||^2,$$
$$f_i(x) \geq f_i(y) + \langle \nabla f_i(y), x - y \rangle + \frac{\sigma_i}{2}||x - y||^2. \tag{2.2}$$

We denote $L_{\max} = \max_i L_i$ and $\sigma_{\min} = \min_i \sigma_i$ the global complexity numbers.

2.2. **Centralized Scheme.** In the centralized framework, a central node stacks the parameter x to compute. The functions $f_i, i \in [n]$ are stacked on n different workers, who can compute derivatives $\nabla f_i(y), y \in \mathbb{R}^d$, Fenchel conjugates f_i^*, etc. An iteration of an optimization algorithm in this scheme, has three steps. First, the central unit selects some $i \in [n]$ (or a subset of indices), possibly at random, and sends the central parameter x_t to i (resp. to the selected workers). On unit i (resp. selected workers), the local computer, knowing x_t, computes some gradient $\nabla f_i(x_t)$ (or another quantity), and send it to the central unit. Finally, the central unit may perform an update on x_t thanks to the received gradient.

2.3. **Decentralized Scheme.** The communication network is represented by an undirected graph $G = (V, E)$ on the set of nodes $V = [n]$, and is assumed to be connected. Two nodes are said to be neighbors in the graph, and we write $i \sim j$, if $(ij) \in E$. Each node $i \in V$ has access to local function f_i. Let us denote $F(x) = \sum_{i \in [n]} f_i(x_i)$ for $x \in \mathbb{R}^{n \times d}$ the *augmented problem* where $x_i \in \mathbb{R}^d$ is stacked at node i. Computing gradients and communicating them between two neighboring nodes $i \sim j$ is assumed to take time $\tau_{ij} > 0$. This constant takes into account both the communication and computation times. The problem can then be formulated as follows:

$$\min_{x \in \mathbb{R}^{n \times d} : x_1 = ... = x_n} F(x), \tag{2.3}$$

where $x_1 = ... = x_n$ enforces consensus on all the nodes. We add the following structural constraints:

- *Local computations:* node i can compute first-order characteristics, such as ∇f_i or f_i^*;
- *Local communications:* node i can send information to neighboring nodes $j \sim i$.

These operations may be performed asynchronously and in parallel, and each node possesses a local version $x_i \in \mathbb{R}^d$ of the global parameter.

3. Preliminaries on Optimization Methods

Some classical methods in order to solve optimization problems such as 2.1 exist. This section present two main ideas, that will be re-used after. We consider the following optimization problem:

$$\min_{x \in \mathbb{R}^p} J(x), \tag{3.1}$$

for some functional J defined on $\mathbb{R}^p$. J is assumed σ-strongly convex and $L-$smooth.

3.1. **Full Gradient Descent.** The most basic method in order to solve 3.1 is *full gradient descent*. Starting from a point $x_0 \in \mathbb{R}^p$, one constructs iterates $x_t, t \in \mathbb{N}$ that will converge to x^* the minimizer of J. From x_t, using $J(x) - J(x_t) \leq \langle \nabla J(x_t), x - x_t \rangle + \frac{L}{2}||x - x_t||^2$, x_{t+1} will minimize the following local problem:

$$\min_{x \in \mathbb{R}^p} \langle \nabla J(x_t), x - x_t \rangle + \frac{L}{2}||x - x_t||^2, \tag{3.2}$$

giving:

$$x_{t+1} = x_t - \frac{1}{L}\nabla J(x_t). \tag{3.3}$$

This is the *full gradient descent* algorithm. It gives the following bound:

Proposition 1 (Full Gradient Method)**.** *For $t \geq 0$, starting from x_0, one has:*

$$J(x_t) - J(x^*) \leq (J(x_0) - J(x^*))(1 - \frac{\sigma}{L})^t. \tag{3.4}$$

The real positive number $\frac{\sigma}{L}$ is called the complexity parameter *of the problem: it controls how well one can perform optimization procedures on J.*

3.2. **Coordinate Gradient Descent.** Often, computing a full gradient ∇J can be costly if p is too large. An alternative to considering the local optimization problem 3.2 consists in taking this minimization problem not on $x \in \mathbb{R}^p$, but for $x \in E_{I_t}$, where I_t is a set of indices included in $[p]$ and E_{I_t} is the sub-euclidean space where only coordinates $i \in I$ are taken. That leads to the following update, where L_{I_t} is the smoothness parameter in the subspace considered ($L_{I_t} \leq L$):

$$x_{t+1} = x_t - \frac{1}{L_{I_t}} \nabla_{I_t} J(x_t). \tag{3.5}$$

This *Coordinate Gradient Method* is widely used in practice. Indeed, it enables *1)* to take smaller and local smoothness parameters; *2)* to compute cheaper local gradients; *3)* parallelization is easier. The speed of the algorithm is, as before, controlled by the complexity parameter of the problem. The optimizer usually chooses he subset of indices in a random and *i.i.d.* way. However, alternatives exist, such as the *Marcov Chain Gradient Descent* [Sun et al., 2018], where indices are chosen following an underlying Marcov Chain process.

3.3. **Accelerating Gradient Descents.** The bottleneck of these methods are the complexity parameter $\frac{\sigma}{L}$: if it is too small, the algorithm will converge slowly. Indeed, if L is too big in front of σ, the gradient step, weighted by $\frac{1}{L}$, is then too small. Taking a bigger step would make the sequence (x_t) explode. One cannot do better than this, intuitively. However, Nesterov found a way to make bigger steps [Nesterov, 1983], introducing *accelerated gradient methods*. The intuition is the following: instead of considering only one sequence (x_t), take a second one (v_t), called *momentum*. Before performing a gradient step, compute a contraction $z_t = (1-\theta)x_t + \theta v_t$, where θ is *small*. Perform a classical gradient step on z_t:

$$x_{t+1} = z_t - \frac{1}{L} \nabla J(z_t), \tag{3.6}$$

and a bigger step on v_t, using the same gradient:

$$v_{t+1} = v_t - \eta \nabla J(z_t). \tag{3.7}$$

with $1/L \ll \eta$. v_t should intuitively explode. However, the contraction step if well parametered counterbalances η taken too big, leading to a better speed of convergence thanks to bigger steps:

$$J(x_t) - J(x^*) \leq (J(x_0) - J(x^*))(1 - \sqrt{\frac{\sigma}{L}})^t. \tag{3.8}$$

This acceleration can be adapted to coordinate gradient descent [Nesterov and Stich, 2017], giving similar results.

4. Synchronous Gossip Algorithms

Taking local functions $f_i(x) = \frac{1}{2}||x - c_i||^2$ for some constant vectors $c_i \in \mathbb{R}^d$ leads to the *Global Averaging Problem*, consisting in finding the mean $\bar{c} = \frac{1}{n}\sum_i c_i$ of local values in a distributed fation, as $(\bar{c}, ..., \bar{c})$ is the solution to the augmented problem defined at equation 2.3. Algorithms solving this averaging problem are *Gossip Algorithms*. Two types of Gossip algorithms can be extracted from the literature: synchronous ones, where all nodes communicate with each others simultaneously [Scaman et al., 2017, Dimakis et al., 2010, Berthier et al., 2018], and asynchronous ones [Boyd et al., 2006, Nedic and Ozdaglar, 2009, Hendrikx et al., 2018], where at a defined time $t \geq 0$, only a pair of adjacent nodes can communicate. Intuitively, in synchronous frameworks, the communication speed is slower than in asynchronous ones, as all nodes are slowed by the slowest node. Hence our interest in this paper in asynchronous algorithms. The rates of convergence of Gossip Algorithms are captured by the smallest non negative eigenvalue γ of the Laplacian Matrix of graph G, weighted by some weights ν_{ij}.

Definition 1 (Graph Laplacian)**.** *Let $(\nu_{ij})_{(ij)\in E}$ a set of non negative real numbers. The Laplacian of the graph G weighted by the ν_{ij}'s is the matrix with entries $-\nu_{ij}$ for $(ij) \in E$, $\sum_{j\sim i} \nu_{ij}$ for (ii) and 0 otherwise. In this paper, the notation ν_{ij} will stand for the weights of the Laplacian.*

See [B Mohar and Oellermann, 1991] for a broader study on graph laplacians. In the synchronous setting, all nodes are allowed to share a common clock, which enables them to perform operations synchronously. Formally, a gossip matrix is defined as follows:

Definition 2 (Gossip Matrix)**.** *A gossip matrix is a matrix $W \in \mathbb{R}^{n\times n}$ such that:*

- $\forall (i,j) \in [n]^2$, $W_{i,j} > 0 \implies i \sim j$ *or* $i = j$ *(supported by G),*
- $\forall i \in [n], \sum_{j\sim i} W_{i,j} = 1$ *(stochastic),*
- $\forall (i,j) \in [n]^2, W_{i,j} = W_{j,i}$ *(symmetric).*

4.1. **Simple Synchronous Gossip Algorithm.** Iteratively, at times $t = 0, 1, 2, ...$, if $x(t) = (x_i(t))_i \in \mathbb{R}^{n\times d}$ describes the information stacked locally at each node ($x_i(t)$ being the vector at node i), the operation $x(t+1) = Wx(t)$ is made. It is to be noted that, thanks to the sparsity of the gossip matrix, this operation is local: for all node i,

$$x_i(t+1) = \sum_{j\sim i} W_{ij} x_j(t), \tag{4.1}$$

where $i \sim j$ if they are neighbors or if $i = j$. The convergence bound will be stated below. Intuitively, at each iteration, each node i sends a proportion of its mass to each one of its neighbour, the condition $\sum_{j\sim i} W_{ij} = 1$ being the mass conservation.

Proposition 2 (Simple Synchronous Gossip)**.** *Let γ_W be the eigengap of the laplacian of G weighted by $\nu_{ij} = 1 - W_{ij}$ at each edge. Then, for all $k = 0, 1, 2...$:*

$$||x(k) - \bar{c}|| \leq (1 - \gamma_W)^k ||c - \bar{c}||. \tag{4.2}$$

Starting from this simplest gossip algorithm, works aim at both improving the rate of convergence and generalizing this to the original problem with more general functions f_i.

4.2. **Generalization to strongly convex and smooth functions f_i.** These gossip matrices enable [Scaman et al., 2017] to make a dual formulation of the augmented problem 2.3, in order to treat the consensus constraint. More precisely, given W a gossip matrix, and $\sqrt{W}$ its symmetric and non-negative square root (W is symmetric and non-negative), the augmented problem 2.3 is equivalent to:

$$\min_{\lambda \in \mathbb{R}^{n\times d}} F^*(\lambda\sqrt{W}), \tag{4.3}$$

where $F^*(x) = \max_{y\in\mathbb{R}^{n\times d}} \langle y, x\rangle - F(y)$ is the Fenchel conjugate of F, well defined thanks to strong convexity. The minimization problem 4.3 can easily be solved by a gradient descent:

$$\lambda_{t+1} = \lambda_t - \eta \nabla F^*(\lambda_t\sqrt{W})\sqrt{W}, \tag{4.4}$$

for η small enough. The change of variable $y_t = \lambda_t\sqrt{W}$ leads to:

$$y_{t+1} = y_t - \eta \nabla F^*(y_t) W. \tag{4.5}$$

The sparsity of the gossip matrix hence leads to local updates:

$$y_{t+1,i} = y_{t,i} - \eta \sum_{j\sim i} W_{ij} \nabla f_j^*(y_{t,j}), \tag{4.6}$$

making y_t computable in a decentralized way. Then, the link with primal variable x_t is made through the formula $x_{t,i} = \nabla f_i^*(y_t)$.

4.3. **Existing "Fast" Algorithms.** [Scaman et al., 2017] use the dual gradient descent described above and a *Nesterov Acceleration* (see [Bubeck, 2014] for details) on the gradient descent in order to obtain an optimal algorithm (*SSDA*) for this synchronous problem. Their major work lies in proving the optimality of the method. If the error to the consensus at iteration t is ε_t, then they get:

$$\varepsilon_t \leq (1 - \sqrt{\gamma_W \frac{\sigma_{\min}}{L_{\max}}})^t \tag{4.7}$$

instead of $(1-\gamma_W \frac{\sigma_{\min}}{L_{\max}})^t$ for non accelerated methods. However, in practice, their speed-up is only visible on asymptotic times. [Berthier et al., 2018] managed to find a really efficient alternative.

Indeed, synchronous algorithms being a deterministic process, they write gossip iterations as a polynomial of the gossip matrix W:

$$X_t = P_t(W),\ \text{degree}(P_t) \leq t. \tag{4.8}$$

Simple gossip gives $P_t = X^t$ (t iterations). *Chebyshev Acceleration* (*SSDA* applied to the averaging problem) leads to P_t being the t^{th} *Chebyshev polynomial*. A polynomial of degree t corresponds to t communications. That being stated, the gossip problem is then formulated as an optimization problem over polynomials of degree bounded by t. This optimization problem is then solved using orthogonal polynomials with respect to the spectral measure of the gossip matrix W. This enables them to use the whole spectrum of the gossip matrix. From this point, algebraic miracles such as order-2 recurrence formulas for these polynomials, and an approximation of the spectrum of W using the d-dimension of the graph enables to find fast iterations, giving for optimal polynomial P_t the t^{th} Jacobi polynomial.

5. Asynchronous Gossip Algorithms

Above considerations assume the existence of a common clock, shared by all nodes in the network, enabling them to make operations synchronously. Asynchronous algorithms do not make that assumption. The first asynchronous setting is the *P.p.p. model* (stands for Poisson point procces), introduced by [Boyd et al., 2006] for the global averaging problem. In this so-called asynchronous model, time is indexed in a continuous way, by $t \in \mathbb{R}^+$. For every edge $e = (ij) \in E$, let $\mathcal{P}_{ij}$ be a Poisson point process (*P.p.p.*) of constant intensity $p_{ij} > 0$ that we will call "clocks", all independent from each other. Updates will be ruled by these processes: at every clock ticking of $\mathcal{P}_{ij}$, nodes i and j update the value they stack by the mean $\frac{x_i+x_j}{2}$. If we write $\mathcal{P} = \cup_{ij\in E}\mathcal{P}_{ij}$, $\mathcal{P}$ is a *P.p.p.* of intensity $I := \sum_{ij\in E} p_{ij}$. See [Klenke, 2014] for a study of these point processes. For this algorithm, we have the following bound to control the second moment of the error.

Proposition 3 (Asynchronous Gossip)**.** *Let γ_p be the smallest non null eigenvalue of the Laplacian of the graph, weighted by $\nu_{ij} = p_{ij}$. For $t \in \mathbb{R}^+$, we have, where $c = x_0$ and $x_t \in \mathbb{R}^{n\times d}$ is the vector stacked on the nodes at time t:*

$$\mathbb{E}[||x_t - \bar{c}||^2] \leq \exp(-t\sigma_p)||c - \bar{c}||^2. \tag{5.1}$$

Proof. When ij activated at time t, multiply $x(t)$ by $W_{ij} = I_n - \frac{{}^t(e_i-e_j)(e_i-e_j)}{2}$. By observing that $W_{ij}^2 = W_{ij}$ and that $\sum_{ij} p_{ij}W_{ij} = II_n - L$, where L is the laplacian of the graph weighted by the p_{ij}, we get that, with R_t^2 the squared error to the consensus at time t:

$$\begin{aligned}
\mathbb{E}^{\mathcal{F}_t}[R_{t+dt}^2] =& (1 - Idt)\mathbb{E}^{\mathcal{F}_t}[R_{t+dt}^2|\text{no activations in } [t, t+dt]] \\
&+ dt\sum_{ij} p_{ij}\mathbb{E}^{\mathcal{F}_t}[R_{t+dt}^2|ij \text{ activated in } [t, t+dt]] + o(dt) \\
&= R_t^2 - dt(x(t) - \bar{c})^T \sum_{ij} W_{ij}(x(t) - \bar{c}) \\
&\leq R_t^2 - dt\sigma_p R_t^2.
\end{aligned}$$

Then, taking the mean, dividing by $dt \to 0$ and integrating concudes the proof. □

5.1. Generalization to the Distributed Optimization Problem.

From this asynchronous scheme, different works [Pu et al., 2020, Hendrikx et al., 2018] have been done to generalize it to strongly convex and smooth objective functions f_i. As presented synchronous algorithms used a dual formulation, we are going to present a similar method here. Note that it is not the only existing one, but it appears to us to be the most convenient one. Again, a standard way to deal with the constraint $x_i = ... = x_n$, is to use a dual formulation, by introducing a dual variable λ indexed by the edges.We first introduce a matrix $A \in \mathbb{R}^{n\times E}$ such that $\text{Ker}(\text{A}^{\text{T}}) = \text{Vect}(\mathbb{I})$ where $\mathbb{I}$ is the constant vector $(1, ..., 1)^\top$ of dimension n. The constraint $x_i = ... = x_n$ can then be written

$A^T x = 0$. The dual problem reads as follows:

$$\min_{x\in\mathbb{R}^{n\times d}, A^T x=0} \sum_{i=1}^n f_i(x)$$
$$= \min_{x\in\mathbb{R}^{n\times d}} \max_{\lambda\in\mathbb{R}^E} \sum_{i=1}^n f_i(x) - \langle A^T x, \lambda\rangle.$$

Let the Fenchel conjugate of F be noted $F^*(x) = \max_{y\in R^n} \langle x, y\rangle - F(y)$, well defined thanks to strong convexity. Let $F_A^*(\lambda) = F^*(A\lambda)$ for $\lambda \in \mathbb{R}^{E\times d}$. The dual problem reads:

$$\min_{x\in\mathbb{R}^{n\times d}, x_1=\ldots=x_n} F(x) = \max_{\lambda\in R^{E\times d}} -F_A^*(\lambda) \tag{5.2}$$

$F_A^*(\lambda) = \sum_{i=1}^n f_i^*((A\lambda)_i)*$ is thus to be minimized over the dual variable $\lambda \in \mathbb{R}^{E\times d}$. The goal being to minimize F under consensus and local update constraints, a parallel needs to be made between minimization method on the dual problem and on the primal one. As $F_A^*(\lambda) = \max_{x\in\mathbb{R}^{n\times d}} -F(x) + \langle A\lambda, x\rangle$, writing the gradient at the optimum of this maximization problem gives, for any λ, a primal variable x uniquely associated, through the formula $\nabla F(x) = A\lambda$. In the gossip problem, this gives $x = A\lambda + c$, bridging the gap between primal and dual formulations. A gradient step on the dual variable λ alongside coordinate (ij) is: $\nabla_{ij} F_A^*(\lambda) = (Ae_{ij})^T \nabla F^*(A\lambda)$, where $\nabla F^*(A\lambda) = (\nabla f_i^*((A\lambda)_i))$. The quantities $\nabla f_i^*((A\lambda)_i)$ are locally computable at each node, hence the following asumption on the matrix A:

$$\forall (ij) \in E, Ae_{ij} = \mu_{ij}(e_i - e_j) \tag{5.3}$$

for some non-null constants μ_{ij}. We define $\mu_{ij} = -\mu_{ji}$ for this writing to be consistent. This gives $\nabla_{ij} F_A^*(\lambda) = \mu_{ij}(e_i - e_j)^T \nabla F^*(A\lambda) = \mu_{ij}(\nabla f_i^*((A\lambda)_i) - \nabla f_j^*((A\lambda)_j))$, a local quantity when edge ij is activated. This matrix A is a square root of the laplacian of the graph weighted by the μ_{ij}^2's. More generally, the gradient step on dual variables, alonside coordinate ij, will be, when ij activated at iteration k (corresponding to time t_k):

$$\lambda_{t_{k+1}} = \lambda_{t_k} - \frac{1}{(\sigma_i^{-1} + \sigma_j^{-1})\mu_{ij}^2} U_{ij} \nabla_{ij} F_A^*(\lambda_{t_k}). \tag{5.4}$$

Denoting $y_k = A\lambda_{t_k}$, we get the following (local) formula, when ij activated:

$$y_{k+1,i} = y_{k,i} - \frac{(\nabla f_i^*(y_{k,i}) - \nabla f_j^*(y_{k,j}))}{(\sigma_i^{-1} + \sigma_j^{-1})}, \tag{5.5}$$

$$y_{k+1,j} = y_{k,j} + \frac{(\nabla f_i^*(y_{k,i}) - \nabla f_j^*(y_{k,j}))}{(\sigma_i^{-1} + \sigma_j^{-1})}. \tag{5.6}$$

In the Gossip Averaging problem, a coordinate gradient step on the dual variable is thus exactly a local averaging on the primal problem. These updates enable us to compute the $y_{k,i}$ locally ($y_{k,i}$ computed at node i). Finally, if (y_k) has a limit y^* when $k \to \infty$ that minimizes F^*, as $F^*(y) = \sup_{x\in\mathbb{R}^{n\times d}}(-F(x) + \langle x, y\rangle)$, and as $F^{**} = F$, the desired solution to the primal problem is $x = \nabla F^*(y)$. The returned solution at node i after our gossip procedure will be $\nabla_i f_i^*(y_i)$, where y_i is the computed local variable through our gossip procedure.

Proposition 4. *For the coordinate gradient descent described in Equation 5.4, for $t \in \mathbb{R}^+$, one has the follwing bound that control both dual and primal variables:*

$$\mathbb{E}[F_A^*(\lambda_t) - F_A^*(\lambda^*)] \leq [F_A^*(\lambda_0) - F_A^*(\lambda^*)] \exp(\gamma_p \times \frac{\sigma_{\min}}{L_{\max}}). \tag{5.7}$$

5.2. **Accelerating Asynchronous Gossip.** Accelerating Asynchronous Gossip in the same way that it is done for Synchronous Gossip is however more difficult. Indeed, as highlighted above through our dual formulations, where synchronous gossip used *full gradient descents*, asynchronous gossip can be formulated as a *coordinate gradient descent algorithm*. The difficulty lies in the stochasticity of this method: product of random matrices often get a variance too big in front of the desired speed. Accelerating asynchronous gossip is thus easier to study in the Optimization framework introduced in last subsection. Applying an accelerated coordinate gradient descent

[Nesterov and Stich, 2017] should give an accelerated rate of convergence. That has been done by [Hendrikx et al., 2018]. However, the obtained algorithm is not *fully asynchronous* in the following sense: Accelerated Coordinate Gradient Descent Method on the dual problem require at every iteration a contraction of the variables, and a coordinate gradient step. Translating this on the primal variable leads to a local update for the coordinate gradient step (an asynchronous operation), and to a contraction on all nodes. That last operation is not asynchronous; all nodes must perform the same contraction at the same time. That requires knowledge that is not available in th *P.p.p. model*. Still, the rate of convergence obtained is accelerated.

6. Works and Future Perspectives

In this section, personal works done between Spring 2020 and August 2020 related to the previous sections are presented.

6.1. A Finer Modelling of Asynchrony in the Decentralized Scheme.

Previous works on asynchronous Gossip suffer from not being fully asynchronous in real applications. The historical *P.p.p. model* presented in Section 5 is the most studied model. However, although qualified as so, this model cannot be programmed in a fully distributed and asynchronous structure: it assumes that communications and computations are made instantly. Two solutions appear to us in order to solve this problem in the modelling: either when a node i receives information from a neighbor j at a time $t \geq 0$, assume that this information is delayed, or forbid communications with a *busy* (*i.e.* communicating or computing) edge to avoid delayed information. These two modellings of asynchrony are respectively inspired by existing works done in an asynchronous but centralized frameworks (*perturbed iterates*, [Leblond et al., 2016, Niu et al., 2011]), by works related to telecommunication modelling (*Loss-Networks*, [Kelly, 1991]). In the *perturbed iterate* modelling, cited works in the centralized setting consider a central unit, that delegate computations to workers. Asynchrony lies in the fact that these workers do not wait for the central unit to perform updates on the model: they send whenever they can computed gradients. In order to update the parameter on the central unit, the steps available are thus perturbed (*delayed*) gradients [Mania et al., 2015]. However, our work is done in the second modelling: nodes behave as in the *P.p.p. model*, but are made *busy* and hence non-available for other nodes for a time $\tau_{ij} > 0$ after their activation. The asynchrony here lies in the fact that, by making communicating and computing nodes *busy*, gradients received are not out-of-date. Moreover, only *local* clocks are needed. Our focus on asynchrony is motivated by their empirical execution speed: we aim at building a framework for their analysis in order to show their efficiency over synchronous ones. Such a construction enables us to extract the quantities of interest, giving us a better understanding of the communication network and the quantities at stake, while being programmable in a fully asynchronous and distributed way. The communication models we considered and analyzed are presented below.

Loss-Network Model *LNM*: In this model, edges are activated following the same procedure as in the *P.p.p. model*, with processes of intensity p_{ij}. However, when an edge ij is activated, it will become "busy" for an interval of time of length τ_{ij}. A busy node cannot be activated, leading to an underlying graph that follows a Markov-Chain process. This aims at taking into account asynchrony in a finer way. Indeed, this model describes the following behaviour of the nodes: each node has an exponential clock of intensity $\frac{1}{2}\sum_{j\sim i} p_{ij}$, and at each ringing, it selects a neighbor j with probability $p_{ij}/\sum_{k\sim i} p_{ik}$. If this neighbor j is not busy, i and j can communicate, becoming busy for a time-lapse of length τ_{ij}. However, if node j or node i is busy, the communication does not happen. Hence, we have no asynchrony problem here, and all the quantities that are being communicated are available at each activation. We can think of this procedure as classical gossip on an underlying random graph, that follows a Markov-Chain process. However, for this model to be a true representation of reality, every node needs to know wether its neighbors are busy or not, a realistic assumption when delays are caused by low bandwidth.

Refined Loss-Network Model *RLNM(ε)*: Take the above procedure from the node point of view, and when $i \in V$ is called and wants to communicate with $j \sim i$, it becomes busy for a time of length $\varepsilon\tau_{ij}$ in order to check if j is busy or not. The Loss-Network model above is the scenario where $\varepsilon << 1$, realistic when the time it takes to send a simple request is negligible in front of sending a whole gradient.

The main goal of considering these alternative modellings is to be programmable in a fully asynchronous and distributed way, while having theoretical guarantees that match existing synchronous methods. We obtained such results, summarized in the following theorem.

Theorem 1 (Discrete-time rate of convergence in the Loss-Network model)**.** *The Lyapunov function L_t studied is defined as follows, for discretized time $t \in \mathbb{N}$:*

$$\forall t \in \mathbb{N}, L_t = \frac{1}{T}\sum_{s=t}^{t+T-1} F_A^*(\lambda_s) - F_A^*(\lambda^*). \tag{6.1}$$

We have:

$$\liminf_{t\to\infty, t\in\mathbb{N}} \frac{1}{t}\log(\mathbb{E}[L_t]) \leq -\rho, \tag{6.2}$$

for ρ the smallest non-null eigenvalue of the laplacian of the graph, with ν_{ij} taken as:

$$\alpha \times \frac{\sigma_{\min}}{L_{\max}} \times \frac{\tilde{\tau}_{ij}^{-1}\min_{(kl)\sim(ij)}\frac{\tilde{\tau}_{ij}}{\tilde{\tau}_{kl}}}{Id_{\max}(\log(|E|)+\log(I\tilde{\tau}_{\max}))^2}, \tag{6.3}$$

where $\alpha = \frac{32e^2}{\log(1-(1-e^{-1})e^{-1})^2}$ is a universal constant.

Quantitatively, having $\tilde{\tau_{ij}}$ and the local maximum over adjacent edges mean that a node is slowed down by its neighbors. Moreover, it means that in this model, the effective waiting time is $\tilde{\tau}_{ij}$, bigger than τ_{ij} due to blocking edges. When passing in continuous time, the discrete-time rate of convergence defined in Equation (6.3) is multiplied by a factor of order I. In the synchronous setting, the weights of the Laplacian of the graph are taken as τ_{max}^{-1}. In *RLNM*, we have local weights $\tilde{\tau}_{ij}$ up to constant factors. As a consequence, when some nodes slow down the whole process, *RLNM* has an improved communication rate as expected, a result that becomes even truer when the variance in terms of communication delays in the graph becomes higher. In a graph where all edges have the same delay τ_{ij} (a case conducive to synchronous algorithms), *RLNM* only loses a factor $\frac{1}{d\log(n)}$, expected not too big due to the logarithm.

6.2. **Acceleration in the Decentralized Scheme.** As highlighted in Section 5.2, no general accelerated Gossip algorithm exists in a synchronous setting. We propose an asynchronous version of the algorithm proposed by [Hendrikx et al., 2018] where contractions are made continuously, leading to a fully asynchronous algorithm, the first one in the *P.p.p.* model. We call our algorithm *CACDM* (Continuously Accelerated Coordinate Gradient Method). It is defined and analyzed in the dual variable λ, and requires a momentum variable v (still a dual variable). This will lead to two primal variables stacked at each node. Let us first define our algorithm on dual variables. Time $t \in \mathbb{R}^+$ is here indexed continuously. *CACDM* involves two operations: continuous contractions, and local updates when a local *P.p.p.* ticks. The intuition behind *CACDM* is that the local continuous contractions enable the momemtum variable (v_t) to make bigger gradient steps, by difusing them continuously on variable λ_t.

i) Continuous Contractions: For all times $t \in \mathbb{R}^+$, make the infinitesimal contraction

$$\begin{pmatrix}\lambda_{t+dt}\\ v_{t+dt}\end{pmatrix} = \begin{pmatrix}1-dtI\theta & dtI\theta\\ dtI\theta & 1-dtI\theta\end{pmatrix}\begin{pmatrix}\lambda_t\\ v_t\end{pmatrix}, \tag{6.4}$$

between times t and $t+dt$, on the dual variables. Between times $s < t$, if there is no activation, it consists in performing the contraction:

$$\begin{pmatrix}\lambda_t\\ v_t\end{pmatrix} = \exp\left((t-s)I\begin{pmatrix}-\theta & \theta\\ \theta & -\theta\end{pmatrix}\right)\begin{pmatrix}\lambda_s\\ v_s\end{pmatrix}, \tag{6.5}$$

where we have:

$$\exp\left(tI\begin{pmatrix}-\theta & \theta\\ \theta & -\theta\end{pmatrix}\right) = \begin{pmatrix}\frac{1+e^{-2I\theta t}}{2} & \frac{1-e^{-2I\theta t}}{2}\\ \frac{1-e^{-2I\theta t}}{2} & \frac{1+e^{-2I\theta t}}{2}\end{pmatrix}.$$

ii) Local Updates: When edge (ij) is activated at time $t \geq 0$, define the coordinate gradient step:

$$\eta_{ij,t} = -\begin{pmatrix}\frac{1}{2\mu_{ij}^2}U_{ij}\nabla_{ij}F_A^*(\lambda_t)\\ \frac{\theta}{\sigma_A p_{ij}}U_{ij}\nabla_{ij}F_A^*(\lambda_t)\end{pmatrix} \tag{6.6}$$

where σ_A is the strong convexity parameter of F_A^*, $U_{ij} = e_{ij}e_{ij}^T$, and perform the gradient step:

$$\begin{pmatrix}\lambda_t\\ v_t\end{pmatrix} \overset{t}{\leftarrow} \begin{pmatrix}\lambda_t\\ v_t\end{pmatrix} + \eta_{ij,t} \tag{6.7}$$

on the dual variables λ_t and v_t.
More formally, the stochastic process defined above is the following, where $Y_t = (\lambda_t, v_t)^T$, and N_{ij} are independent *P.p.p.* of intensities p_{ij}:

$$dY_t = I\begin{pmatrix}-\theta & \theta\\ \theta & -\theta\end{pmatrix}Y_t dt + \sum_{(ij)\in E} dN_{ij}(t)\eta_{ij,t}. \tag{6.8}$$

This procedure is an asynchronous one: the length $t-s$ between two activations of an edge that appears in the exponential contraction (6.5) is a local variable, and only needs a local clock to be computed. Denote:

$$L_t = ||v_t - \lambda^*||^{*2} + \frac{2\theta^2 S^2}{\sigma_A^2}(F_A^*(\lambda_t) - F_A^*(\lambda^*)) \tag{6.9}$$

the Lyapunov function we will study, with λ^* an optimizer of F_A^* and $||.||^*$ the euclidian norm on the orthogonal of Ker(A).

Theorem 2. *For the* CACDM *algorithm, if* $\theta = \sqrt{\frac{\sigma_A}{IS^2}}$ *for* S *verifying the inequality* $S^2 \geq \sup_{(ij)\in E}\frac{\mu_{ij}^2 e_{ij}^T A^* A e_{ij}(\sigma_i^{-1}+\sigma_j^{-1})}{2p_{ij}^2}$, *we have for all* $t \in \mathbb{R}^+$*:*

$$\mathbb{E}[L_t] \leq L_0 e^{-I\theta t}. \tag{6.10}$$

And thus, on the primal variables that we denote X_t*:*

$$||X_t - \bar{C}||^2 \leq \frac{L_{\max}}{\sigma_A}L_0 e^{-I\theta t} \tag{6.11}$$

Remarks on the bound: σ_A is the strong convexity parameter of F_A^*. It can be lower-bounded by $\frac{\gamma_{\text{asynch}}/I}{L_{\max}}$, where γ_{asynch} is the smallest eigenvalue of the laplacian of the graph weighted by p_{ij} (non accelerated *P.p.p.* rate of convergence), it is divided by I so that the entries p_{ij} sum to 1. Under a bounded-degree assumption, S^{-2} is of order $nd_{\max}/\sigma_{\min}$. Finally, we get the following rate of convergence, giving us both an acceleration in terms of optimization complexity, and in terms of communication:

$$I \times \sqrt{\frac{\sigma_{\min}}{L_{\max}} \times \frac{\gamma_{\text{asynch}}}{Id_{\max}}}. \tag{6.12}$$

Taking $I = 1$ (re-normalizing time) and the simple averaging problem, leads to an improved rate n^{-2} on the line graph instead of n^{-3}. For the 2D-Grid, we get $n^{-3/2}$ instead of n^{-2}. These rates are the same as [Dimakis et al., 2008, Hendrikx et al., 2018, Loizou and Richtárik, 2018]. However, our algorithm does not require to know the number of activations performed on the whole network, and only requires local clocks. Moreover, it works for any graph and for the more general problem of distributed optimization of smooth and strongly convex functions.

Operations in the Primal Variables: node i stacks two primal variables x_i and y_i respectively primal surrogates of λ and v. CACDM leads to the following operations on x, y. Continuously shrink x and y at all times and at all nodes:

$$\begin{pmatrix} x_{i,t+dt} \\ y_{i,t+dt} \end{pmatrix} = \begin{pmatrix} 1 - dtI\theta & dtI\theta \\ dtI\theta & 1 - dtI\theta \end{pmatrix} \begin{pmatrix} x_{i,t} \\ y_{i,t} \end{pmatrix}, \tag{6.13}$$

and when ij activated, perform on x_i, x_i' (and reciprocally on x_j, x_j') the following gradient steps:

$$x_{i,t} \overset{t}{\leftarrow} x_{i,t} - \frac{\nabla f_i^*(x_t(i)) - \nabla f_j^*(x_t(j))}{(\sigma_i^{-1} + \sigma_j^{-1})}, \tag{6.14}$$

$$y_{i,t} \overset{t}{\leftarrow} y_{i,t} - \frac{\theta}{\sigma_A}(\nabla f_i^*(x_t(i)) - \nabla f_j^*(x_t(j)). \tag{6.15}$$

6.3. **Ideas for Future Works and Conclusion.** We presented in this introduction to an area of research different methods, their pros and cons, and personal works related to a simple optimization problem (1.1). Directly related to Section 6.1, many questions arise. The first one being the eventual optimality of our rate of convergence, and its dependency in all the variables of the problem. Then, that question tackled, what could possibly be the best graph architecture, knowing the delays τ_{ij} between all nodes? Such a construction would be a very interesting work to do. More generally, from this introduction to distributed Optimization, we could broaden the type of optimization problems tackled distributedly. One could think of performing distributed Bandit problems or Reinforcement Learning over a whole network of communication, Distributed Principal Component Analysis, etc. Finally, when studying learning problems, we in fact have more structure than the one assumed in this introduction: he local functions f_i are generally linked to one another. The simplest model would assume them to be drawn from *i.i.d.* data. How to best use such a structure is not really understood at the moment. Some works have been done ine the centralized scheme, but none in a decentralized one.

References

[B Mohar and Oellermann, 1991] B Mohar, Y Alavi, G. C. and Oellermann, O. (1991). The laplacian spectrum of graphs. *Graph theory, combinatorics, and applications*.

[Berthier et al., 2018] Berthier, R., Bach, F., and Gaillard, P. (2018). Accelerated gossip in networks of given dimension using jacobi polynomial iterations.

[Boyd et al., 2006] Boyd, S., Ghosh, A., Prabhakar, B., and Shah, D. (2006). Randomized gossip algorithms.

[Bubeck, 2014] Bubeck, S. (2014). Convex optimization: Algorithms and complexity.

[Dimakis et al., 2008] Dimakis, A. D. G., Sarwate, A. D., and Wainwright, M. J. (2008). Geographic gossip: Efficient averaging for sensor networks. *IEEE Transactions on Signal Processing*, 56(3):1205–1216.

[Dimakis et al., 2010] Dimakis, A. G., Kar, S., Moura, J. M. F., Rabbat, M. G., and Scaglione, A. (2010). Gossip algorithms for distributed signal processing. *Proceedings of the IEEE*, 98(11):1847–1864.

[Hendrikx et al., 2018] Hendrikx, H., Bach, F., and Massoulié, L. (2018). Accelerated decentralized optimization with local updates for smooth and strongly convex objectives.

[Kelly, 1991] Kelly, F. P. (1991). Loss networks. *The Annals of Applied Probability*, 1(3):319–378.

[Klenke, 2014] Klenke, A. (2014). *The Poisson Point Process*, pages 543–561. Springer London, London.

[Leblond et al., 2016] Leblond, R., Pedregosa, F., and Lacoste-Julien, S. (2016). Asaga: Asynchronous parallel saga.

[Loizou and Richtárik, 2018] Loizou, N. and Richtárik, P. (2018). Accelerated gossip via stochastic heavy ball method.

[Mania et al., 2015] Mania, H., Pan, X., Papailiopoulos, D., Recht, B., Ramchandran, K., and Jordan, M. I. (2015). Perturbed iterate analysis for asynchronous stochastic optimization.

[Nedic and Ozdaglar, 2009] Nedic, A. and Ozdaglar, A. (2009). Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 54(1):48–61.

[Nesterov and Stich, 2017] Nesterov, Y. and Stich, S. U. (2017). Efficiency of the accelerated coordinate descent method on structured optimization problems. *SIAM Journal on Optimization*, 27(1):110–123.

[Nesterov, 1983] Nesterov, Y. E. (1983). A method for solving the convex programming problem with convergence rate o($1/k^2$).*Dokl.Akad.NaukSSSR*, 269 : 543 − −547.

[Niu et al., 2011] Niu, F., Recht, B., Re, C., and Wright, S. J. (2011). Hogwild!: A lock-free approach to parallelizing stochastic gradient descent.

[Pu et al., 2020] Pu, S., Shi, W., Xu, J., and Nedic, A. (2020). Push-pull gradient methods for distributed optimization in networks. *IEEE Transactions on Automatic Control*, pages 1–1.

[Scaman et al., 2017] Scaman, K., Bach, F., Bubeck, S., Lee, Y. T., and Massoulié, L. (2017). Optimal algorithms for smooth and strongly convex distributed optimization in networks.

[Sun et al., 2018] Sun, T., Sun, Y., and Yin, W. (2018). On markov chain gradient descent.