A Stochastic Coordinate Descent Primal-Dual Algorithm and Applications to Large-Scale Composite Optimization

P. Bianchi, W. Hachem and F. Iutzeler

Abstract—Based on the idea of randomized coordinate descent of $\alpha$-averaged operators, we provide a randomized primal-dual algorithm. The algorithm builds upon a variant of a recent (deterministic) algorithm proposed by Vu and Condat. Next, we address two applications of our method. (i) In the case of stochastic approximation methods, the algorithm can be used to split a composite objective function into blocks, each of these blocks being processed sequentially by the computer. (ii) In the case of distributed optimization, we consider a set of $N$ agents having private composite objective functions and seeking to find a consensus on the minimum of the aggregate objective. In that case, our method yields a distributed iterative algorithm where each agent use both local computations and message passing in an asynchronous manner. Numerical results demonstrate the attractive performance of the method in the framework of large scale machine learning applications.

Index Terms—Distributed Optimization, Large-scale Learning, Coordinate Descent, Consensus algorithms, Primal-Dual Algorithm.

I. INTRODUCTION

A. Context

A broad class of signal processing and machine learning problems can be formally stated as minimization problems of the form

$$\text{Minimize } \sum_{i=1}^{T} \ell(x; Z_i) + r(x) \text{ w.r.t. } x \in \mathcal{X}$$  \hspace{1cm} (1)

where $\mathcal{X}$ is a Euclidean space, $(Z_1, \ldots, Z_T)$ is a collection of data examples, $\ell$ is a loss function such that $\ell(x; Z_i)$ represents the inadequacy between the $i$th example and a probabilistic model represented by a vector $x$ to be estimated. Finally, $r$ stands for a regularization term which prevents the occurrence of overly complex solutions. A typical example of loss function is $\ell(x, Z_i) = (Y_i - \langle X_i, x \rangle)^2$ where $Z_i = (X_i, Y_i) \in \mathcal{X} \times \mathbb{R}$ is composed of a response $Y_i \in \mathbb{R}$ to be explained as a linear function of feature vector $X_i \in \mathcal{X}$. A typical regularization term is $r(x) = \|x\|_1$ i.e. the $\ell_1$-norm of a vector $x$ [1] (see [2] for other examples). There is wide variety of optimization algorithms for solving (1). Celebrated examples include the standard (sub)gradient descent or the forward-backward algorithm also referred to as proximal gradient algorithm [3]. Such algorithms iteratively generate a sequence $x^k (k = 0, 1, 2, \ldots)$ which converges to a minimizer of (1) under some hypotheses. As the number of examples $T$ becomes large, the aforementioned methods face some limitations. First, as the update of $x^k$ involves the complete data set, the computational cost per iteration is likely to become prohibitive. Second, large data set are often distributed by nature and may thus be impossible to process as a whole. Such issues can be handled using a divide and conquer approach where the data set is split into $N$ chunks or batches, each of those being processed separately. The following frameworks will be investigated in this paper.

Incremental algorithms. We are considering iterative algorithms for which the estimate $x^k$ is updated at time $k$ using a single batch chosen at random amongst the $N$ batches. Incremental algorithms fall within the broader class of stochastic approximation methods. The archetypal example is the stochastic gradient algorithm [4]. Other recent methods includes the stochastic averaged gradient (SAG) algorithm proposed in [5], the MISO algorithm [6], the stochastic dual coordinate ascent (SDCA) [7], etc.

Distributed optimization. An alternative way to separately process the batches is to associate each of them to a distinct processing unit, also referred to as agent. In this context, it is assumed that agents are able to process local data and exchange or merge their outputs by message passing through an interconnection network [8]. Thus, each iteration of the algorithm both involves local computations and communications between agents. Distributed algorithms can either be synchronous or asynchronous. In the synchronous case, a scheduler makes sure that all the agents completed their local computations before their outputs can be merged. Otherwise stated, the algorithm proceeds with the next iteration provided that all agents returned their result. In the asynchronous case, we assume that each agent finishes a local computation at some random time instant, passes its output to some agents in its neighborhood, then turns idle (or performs some other independent task) or proceeds with another local computation with no need to wait for the other agents to terminate their own computations.

Asynchronous distributed optimization is a promising framework in order to scale up machine learning problems involving massive data sets (we refer to [9] or the more recent
survey [10]). Early works on distributed optimization include [11], [12] where a network of processors seeks to optimize some objective function known by all agents (possibly up to some additive noise). More recently, numerous works extended this kind of algorithm to more involved multi-agent scenarios, see [13]–[23].

B. Contributions

Let \( \mathcal{X} \) and \( \mathcal{Y} \) be two Euclidean spaces and let \( M : \mathcal{X} \to \mathcal{Y} \) be a linear operator. Given two real convex functions \( f \) and \( g \) on \( \mathcal{X} \) and a real convex function \( h \) on \( \mathcal{Y} \), we consider the minimization problem

\[
\inf_{x \in \mathcal{X}} f(x) + g(x) + h(Mx)
\]

(2)

where \( f \) is differentiable and its gradient \( \nabla f \) is Lipschitz-continuous. Our contributions are as follows.

1) V̂u and Condat have separately proposed an algorithm to solve (2) in [24] and [25] respectively. Elaborating on this algorithm, we provide an iterative algorithm for solving (2) which we refer to as ADMM+ (Alternating Direction Method of Multipliers plus) because it includes the well known ADMM [26], [27] as a special case. The algorithm also has some similarities with a recent variation on ADMM by [28].

2) As a second main theoretical ingredient, we investigate the use of randomized coordinate descent on Krasnosel’skii-Mann iterations. In their simplest form, Krasnosel’skii-Mann iterations can be interpreted as fixed point iterations of so called \( \alpha \)-averaged operators, which have a contraction-like property. Interestingly, ADMM+ as well as many other algorithms (gradient descent, proximal gradient algorithm, ADMM, etc.) are special instances of Krasnosel’skii-Mann iterations [3]. The idea behind stochastic coordinate descent is to update only a random subset of coordinates at each iteration. This leads to a perturbed version of the initial Krasnosel’skii-Mann iterations which can nevertheless be shown to preserve the sought convergence properties of the initial unperturbed version.

Stochastic coordinate descent has been mainly studied in the literature in the special case of proximal gradient algorithms [29]–[31]. The idea of stochastic coordinate descent on a more general class of averaged operator has been introduced in [32] (see also the recent preprint [33] in the same line of thought). Note that the approach of [32] was limited to unrelaxed firmly non expansive (or 1/2-averaged) operators, well-suited for studying ADMM which was the algorithm of interest in [32].

3) Putting together both ingredients above, we apply our findings to large-scale optimization problems arising in signal processing and machine learning contexts. We show that the general idea of stochastic coordinate descent provides a unified framework allowing to derive stochastic algorithms of different kinds. More precisely, we derive two application examples. First, we introduce a new incremental algorithm by applying stochastic coordinate descent on the top of ADMM+. The algorithm is referred to as Stochastic Minibatch Primal Dual algorithm (SMPD). Second, we propose a new asynchronous distributed optimization algorithm that we refer to as Distributed Asynchronous Primal Dual algorithm (DAPD). The algorithm can be used to efficiently solve an optimization problem over a network of communicating agents. Here, we recall that the asynchronous refers to the fact that any node can wake up at random at any moment. Upon waking up, this node processes its data, sends a limited amount of information to some of its neighbors, then turns idle. We also remark that the algorithm proposed in [32] was not fully decentralized nor fully asynchronous. Indeed, when a node wakes up, this node was required in that paper to act as a “local scheduler” by activating one or more of its neighbors.

The paper is organized as follows. Section II is devoted to the introduction of ADMM+ algorithm and its relation with the Primal-Dual algorithms of V̂u [24] and Condat [25], we also show how ADMM+ includes both the standard ADMM and the Forward-Backward algorithm (also referred to as proximal gradient algorithm) as special cases. In Section III, we provide our main result on the convergence of Krasnosel’skii-Mann iterations with randomized coordinate descent. This enables us to derive, in Section IV, an incremental stochastic algorithm from the ADMM+. Section V addresses the problem of asynchronous distributed optimization. Finally, Section VI provides numerical results in the context of large-scale logistic regression.

II. A PRIMAL DUAL ALGORITHM

A. Problem statement

We consider Problem (2). Denoting by \( \Gamma_0(\mathcal{X}) \) the set of proper lower semi-continuous convex functions on \( \mathcal{X} \to (-\infty, \infty) \) and by \( \| \cdot \| \) the norm on \( \mathcal{X} \), we make the following assumptions:

**Assumption 1** The following facts hold true:

(i) \( f \) is a convex differentiable function on \( \mathcal{X} \),
(ii) \( g \in \Gamma_0(\mathcal{X}) \) and \( h \in \Gamma_0(\mathcal{Y}) \).

We consider the case where \( M \) is injective (in particular, it is implicit that \( \dim(\mathcal{X}) \leq \dim(\mathcal{Y}) \)). In the latter case, we denote by \( S = \text{Im}(M) \) the image of \( M \) and by \( M^{-1} \) the inverse of \( M \) on \( S \to \mathcal{X} \). We emphasize the fact that the inclusion \( S \subset \mathcal{Y} \) might be strict. We denote by \( \nabla \) the gradient operator.
Assumption 2 The following facts hold true:
(i) \( M \) is injective,
(ii) \( \nabla(f \circ M^{-1}) \) is \( L \)-Lipschitz continuous on \( S \).

We denote by \( \text{dom} q \) the domain of a function \( q \) and by \( \text{ri} S \) the relative interior of a set \( S \) in a Euclidean space.

Assumption 3 The infimum of Problem (2) is attained. Moreover, the following qualification condition holds
\[
0 \in \text{ri}(\text{dom} h - M \text{dom} g).
\]

The dual problem corresponding to the primal problem (2) is written
\[
\min_{\lambda \in \mathcal{Y}} (f + g)^*(-M^*\lambda) + h^*(\lambda)
\]
where \( q^* \) denotes the Legendre-Fenchel transform of a function \( q \) and where \( M^* \) is the adjoint of \( M \). With the assumptions (1) and (3), the classical Fenchel-Rockafellar duality theory [34], [35] shows that
\[
\min_{x \in \mathcal{X}} f(x) + g(x) + h(Mx) = -\inf_{\lambda \in \mathcal{Y}} (f + g)^*(-M^*\lambda) + h^*(\lambda),
\]
and the infimum at the right hand member is attained. Furthermore, denoting by \( \partial g \) the subdifferential of a function \( g \in \Gamma_2(\mathcal{X}) \), any point \((\bar{x}, \bar{\lambda}) \in \mathcal{X} \times \mathcal{Y} \) at which the above equality holds satisfies
\[
\begin{cases}
0 \in \nabla f(\bar{x}) + \partial g(\bar{x}) + M^*\bar{\lambda} \\
0 \in -M\bar{x} + \partial h^*(\bar{\lambda})
\end{cases}
\]
and conversely. Such a point is called a primal-dual point.

B. A Primal-Dual Algorithm

We denote by \( \langle \cdot, \cdot \rangle \) the inner product on \( \mathcal{X} \). We keep the same notation \( \| \cdot \| \) to represent the norm on both \( \mathcal{X} \) and \( \mathcal{Y} \). For some parameters \( \rho, \tau > 0 \), we consider the following algorithm which we shall refer to as ADMM+.

\[
\begin{align}
\zeta^{k+1} &= \arg\min_{w \in \mathcal{Y}} \left[ h(w) + \frac{\|w - (Mx^k + \rho \lambda^k)\|^2}{2\rho} \right] \\
\lambda^{k+1} &= \lambda^k + \rho^{-1} (Mx^k - \zeta^{k+1}) \\
u^{k+1} &= (1 - \tau \rho^{-1})Mx^k + \tau \rho^{-1} \zeta^{k+1} \\
x^{k+1} &= \arg\min_{w \in \mathcal{X}} \left[ g(w) + \langle \nabla f(x^k), w \rangle + \frac{\|Mw - u^{k+1} + \tau \lambda^{k+1}\|^2}{2\tau} \right]
\end{align}
\]

Theorem 1 Let Assumptions (1) and (3) hold true. Assume that \( \tau^{-1} - \rho^{-1} > L/2 \). For any initial value \((x^0, \lambda^0) \in \mathcal{X} \times \mathcal{Y} \), the sequence \((x^k, \lambda^k) \) defined by ADMM+ converges to a primal-dual point \((x^*, \lambda^*) \) of (3) as \( k \to \infty \).

Remark 1 In the special case when \( f = 0 \) (that is \( L = 0 \)), the condition \( \tau^{-1} - \rho^{-1} > L/2 \) can be further weakened to \( \tau^{-1} - \rho^{-1} \geq 0 \). It is therefore possible to set \( \tau = \rho \) and thus have a single instrumental parameter to tune in the algorithm. Note also that in \( f = 0 \) the algorithm is provably convergent with no need to require the injectivity of \( M \).

Before providing the proof of Theorem 1, let us introduce the following notation. For any function \( g \in \Gamma_0(\mathcal{X}) \) we denote by \( \text{prox}_g \) its proximity operator defined by
\[
\text{prox}_g(x) = \arg\min_{w \in \mathcal{X}} \left[ g(w) + \frac{1}{2}\|w - x\|^2 \right].
\]

The ADMM+ is an instance of the primal dual algorithm recently proposed by Vû [24] and Condat [25], see also [36]:

Theorem 2 ([24], [25]) Given a Euclidean space \( \mathcal{E} \), consider the minimization problem \( \inf_{y \in \mathcal{E}} f(y) + g(y) + h(y) \) where \( g, h \in \Gamma_0(\mathcal{E}) \) and where \( f \) is convex and differentiable on \( \mathcal{E} \) with an \( L \)-Lipschitz continuous gradient. Assume that the infimum is attained and that \( 0 \in \text{ri}(\text{dom} h - \text{dom} g) \).

\[
\lambda^{k+1} = \text{prox}_{\rho^{-1}h}(\lambda^k + \rho^{-1}g^k) \\
y^{k+1} = \text{prox}_{\rho g}(y^k - \tau \nabla f(y^k) - \tau(2 \lambda^{k+1} - \lambda^k))
\]

Then for any initial value \((y^0, \lambda^0) \in \mathcal{E} \times \mathcal{E} \), the sequence \((y^k, \lambda^k) \) converges to a primal-dual point \((y^*, \lambda^*) \), i.e., a solution of the equation
\[
\inf_{y \in \mathcal{E}} f(y) + g(y) + h(y) = -\inf_{\lambda \in \mathcal{E}} (f + g)^*(-\lambda) + h^*(\lambda).
\]

Elaborating on Theorem 2 we are now ready to establish the Theorem 1.

C. Proof of Theorem 1

By setting \( \mathcal{E} = S \) and by assuming that \( \mathcal{E} \) is equipped with the same inner product as \( \mathcal{Y} \), one can notice that the functions \( f = f \circ M^{-1}, g = g \circ M^{-1} \) and \( h \) satisfy the conditions of Theorem 2. Moreover, since \( (f + g)^* = (f + g)^* \circ M^* \), one can also notice that \((x^*, \lambda^*) \) is a primal-dual point associated with Eq. (3) and only if \((Mx^*, \lambda^*) \) is a primal-dual point associated with Eq. (7).

To recover ADMM+ from the iterations (6a)-(6c), the starting point is Moreau’s identity [35, Th. 14.3] which reads
\[
\text{prox}_{\rho^{-1}h}(x) + \rho^{-1} \text{prox}_{\rho h}(\rho x) = x.
\]

Setting \( x^k = M^{-1} y^k \) and
\[
\begin{align}
\zeta^{k+1} &= \arg\min_{w \in \mathcal{Y}} \left[ h(w) + \frac{\|w - (Mx^k + \rho \lambda^k)\|^2}{2\rho} \right] \\
\lambda^{k+1} &= \lambda^k + \rho^{-1} (Mx^k - \zeta^{k+1}) \\
u^{k+1} &= (1 - \tau \rho^{-1})Mx^k + \tau \rho^{-1} \zeta^{k+1}
\end{align}
\]
Equation (6a) can be rewritten thanks to Moreau’s identity \( \lambda^{k+1} = \lambda^k + \rho^{-1}(Mx^k - \zeta^{k+1}) \).

Now, Equation (6b) can be rewritten as
\[
y^{k+1} = \arg\min_{w \in \mathcal{E}} (\bar{g}(w) + \langle \nabla f(y^k), w \rangle + \frac{\|w - y^k + \tau(2 \lambda^{k+1} - \lambda^k)\|^2}{2\tau})
\]
Upon noting that \( \bar{g}(Mx) = g(x) \) and \( \langle \nabla f(y^k), Mx \rangle = \langle (M^{-1})^* \nabla f(M^{-1}Mx), Mx \rangle = \langle \nabla f(x^k), x \rangle \), the above equation becomes
\[
x^{k+1} = \arg\min_{w \in \mathcal{X}} (\nabla f(x^k), w) + \frac{\|Mw - u^{k+1} + \tau \lambda^{k+1}\|^2}{2\tau}
\]
where

\[ u^{k+1} = Mx^k + \tau (\lambda^k - \lambda^{k+1}) \]
\[ = (1 - \rho^{-1}\tau)Mx^k + \tau \rho^{-1}z^{k+1} \cdot \]

The iterates \((z^{k+1}, \lambda^{k+1}, u^{k+1}, x^{k+1})\) are those of the ADMM+.

D. The case \(f \equiv 0\) and the link with ADMM

In the special case \(f \equiv 0\) and \(\tau = \rho\), sequence \((u^{k})_{k \in \mathbb{N}}\) coincides with \((z^{k})_{k \in \mathbb{N}}\). Then, ADMM+ boils down to the standard ADMM whose iterations are given by:

\[ z^{k+1} = \text{argmin}_{w \in \mathcal{Y}} \left[ h(w) + \frac{1}{2\rho} \| w - Mx^k - \rho \lambda^k \| \right] \]
\[ \lambda^{k+1} = \lambda^k + \rho^{-1}(Mx^k - z^{k+1}) \]
\[ x^{k+1} = \text{argmin}_{w \in \mathcal{X}} \left[ g(w) + \frac{1}{2\rho} \| Mw - z^{k+1} + \rho \lambda^{k+1} \| \right] \cdot \]

E. The case \(h \equiv 0\) and the link with the Forward-Backward algorithm

In the special case \(h \equiv 0\) and \(M = I\), it can be easily verified that \(\lambda^k\) is null for all \(k \geq 1\) and \(u^k = x^k\). Then, ADMM+ boils down to the standard Forward-Backward algorithm whose iterations are given by:

\[ x^{k+1} = \text{argmin}_{w \in \mathcal{X}} \left[ g(w) + \frac{1}{2\tau} \| w - (x^k - \tau \nabla f(x^k)) \| \right] \]
\[ = \text{prox}_{\tau g}(x^k - \tau \nabla f(x^k)) \cdot \]

One can remark that \(\rho\) has disappeared thus it can be set as large as wanted so the condition on stepsize \(\tau\) from Theorem 1 boils down to \(\tau < 2/L\). Applications of this algorithm with particular functions appear in well known learning methods such as ISTA [37].

F. Comparison to the original Vţu-Condat algorithm

We emphasize the fact that ADMM+ is a variation on the Vţu-Condat algorithm. The original Vţu-Condat algorithm is in general sufficient and, in many contexts, has even better properties than ADMM+ from an implementation point of view. Indeed, whereas the Vţu-Condat algorithm handles the operator \(M\) explicitly, the step 4d in ADMM+ can be delicate to implement in certain applications, i.e., when \(M\) has no convenient structure allowing to easily compute the \(\text{argmin}\) (the same remark holds of course for ADMM which is a special case of ADMM+).

This potential drawback is however not an issue in many other scenarios where the structure of \(M\) is such that step 4d is affordable. In Sections 4V and 4V we shall provide such scenarios where ADMM+ is especially relevant. In particular, ADMM+ is not only easy to implement but it is also provably convergent under weaker assumptions on the step sizes, as compared to the original Vţu-Condat algorithm.

Also, the injectivity assumption on \(M\) could be seen as restrictive at first glance. First, the latter assumption is in fact not needed when \(f = 0\) as noted above. Second, it is trivially satisfied in the application scenarios which motivate this paper (see the next sections).

III. COORDINATE DESCENT

A. Averaged operators and the primal-dual algorithm

Let \(\mathcal{H}\) be a Euclidean space\(^1\). For \(0 < \alpha \leq 1\), a mapping \(T : \mathcal{H} \rightarrow \mathcal{H}\) is \(\alpha\)-averaged if the following inequality holds for any \(x, y \in \mathcal{H}\):

\[ \|Tx - Ty\|^2 \leq \|x - y\|^2 - \frac{1 - \alpha}{\alpha} \| (1 - T)x - (1 - T)y \|^2 \cdot \]

A 1-averaged operator is said non-expansive. A \(\frac{1}{2}\)-averaged operator is said firmly non-expansive. The following Lemma can be found in [35, Proposition 5.15, pp.80].

**Lemma 1 (Krasnosel’skii-Mann iterations)** Assume that \(T : \mathcal{H} \rightarrow \mathcal{H}\) is \(\alpha\)-averaged and that the set \(\text{fix}(T)\) of fixed points of \(T\) is non-empty. Consider a sequence \((\eta_k)_{k \in \mathbb{N}}\) such that \(0 \leq \eta_k \leq 1/\alpha\) and \(\sum_k \eta_k (1/\alpha - \eta_k) = \infty\). For any \(x^0 \in \mathcal{H}\), the sequence \((x^k)_{k \in \mathbb{N}}\) recursively defined on \(\mathcal{H}\) by \(x^{k+1} = x^k + \eta_k (Tx^k - x^k)\) converges to some point in \(\text{fix}(T)\).

On the product space \(\mathcal{Y} \times \mathcal{Y}\), consider the operator

\[ V = \begin{pmatrix} \tau^{-1} I_Y & \rho I_Y \\ I_Y & \rho I_Y \end{pmatrix} \cdot \]

When \(\tau^{-1} - \rho^{-1} > 0\), one can easily check that \(V\) is positive definite. In this case, we endow \(\mathcal{Y} \times \mathcal{Y}\) with an inner product \(\langle \cdot, \cdot \rangle\) defined as \(\langle \zeta, \varphi \rangle_V = \langle V\zeta, V\varphi \rangle\) where \(\langle \cdot, \cdot \rangle\) stands for the natural inner product on \(\mathcal{Y} \times \mathcal{Y}\). We denote by \(\mathcal{H}_V\) the corresponding Euclidean space.

In association with Lemma 1 the following lemma is at the heart of the proof of Theorem 2.

**Lemma 2 ([24], [25])** Let Assumptions 2 hold true. Assume that \(\tau^{-1} - \rho^{-1} > L/2\). Let \((\lambda^{k+1}, y^{k+1}) = T(\lambda^k, y^k)\) where \(T\) is the transformation described by Equations 6a–6b. Then \(T\) is an \(\alpha\)-averaged operator on \(\mathcal{H}_V\) with \(\alpha = (2 - \alpha_1)^{-1}\) and \(\alpha_1 = (L/2)(\tau^{-1} - \rho^{-1})^{-1}\).

Note that \(\tau^{-1} - \rho^{-1} > L/2\) implies that \(1 > \alpha_1 \geq 0\) and thus that \(\alpha\) verifies \(1/2 \leq \alpha < 1\) which matches the definition of \(\alpha\)-averaged operators.

B. Randomized Krasnosel’skii Mann Iterations

Consider the space \(\mathcal{H} = \mathcal{H}_1 \times \cdots \times \mathcal{H}_J\) for some \(J \in \mathbb{N}^* = \mathbb{N} - \{0\}\) where for any \(j\), \(\mathcal{H}_j\) is a Euclidean space. Assume that \(\mathcal{H}\) is equipped with the scalar product \(\langle x, y \rangle = \sum_{j=1}^J \langle x_j, y_j \rangle_{\mathcal{H}_j}\) where \(\langle \cdot, \cdot \rangle_{\mathcal{H}_j}\) is the scalar product in \(\mathcal{H}_j\). For \(j \in \{1, \ldots, J\}\), we denote by \(T_j : \mathcal{H} \rightarrow \mathcal{H}\) the components of the output of operator \(T : \mathcal{H} \rightarrow \mathcal{H}\) corresponding to \(\mathcal{H}_j\), we thus have \(T_x = (T_1 x, \ldots, T_J x)\). We denote by \(2^J\) the power set of \(J = \{1, \ldots, J\}\). For any \(\kappa \in 2^J\), we define the operator \(T^{(\kappa)} : \mathcal{H} \rightarrow \mathcal{H}\) by \(T^{(\kappa)} x = T_j x\) if \(j \in \kappa\) and \(T^{(\kappa)} x = x_j\) otherwise. On some probability space \((\Omega, \mathcal{F}, \mathbb{P})\), we introduce a random i.i.d. sequence \((\xi^k)_{k \in \mathbb{N}}\) such that \(\xi^k : \Omega \rightarrow 2^J\) i.e. \(\xi^k(\omega)\) is a subset of \(J\). We assume that the following holds:

\[ \forall j \in J, \exists \kappa \in 2^J, j \in \kappa \text{ and } \mathbb{P}(\xi_1 = \kappa) > 0 \cdot (8) \]
Let $T$ be an $\alpha$-averaged operator, instead of considering the iterates $x^{k+1} = x^k + \eta_k(T x^k - x^k)$, we are now interested in a stochastic coordinate descent version of this algorithm that consists in iterates of the type $x^{k+1} = x^k + \eta_k(T^{(\epsilon_k^1)} x^k - x^k)$.

**Theorem 3** Let $T : \mathcal{H} \to \mathcal{H}$ be $\alpha$-averaged and $\text{fix}(T) \neq \emptyset$. Assume that for all $k$, the sequence $(\eta_k)_{k \in \mathbb{N}}$ satisfies

$$0 < \lim \inf_{k} \eta_k \leq \lim \sup_{k} \eta_k < \frac{1}{\alpha}.$$

Let $(\xi k)_{k \in \mathbb{N}}$ be a random i.i.d. sequence on $2^\mathcal{J}$ such that Condition (\ref{eq:condition}) holds. Then, for any deterministic initial value $x_0$, the iterated sequence

$$x^{k+1} = x^k + \eta_k(T^{(\epsilon_k^1)} x^k - x^k) \quad (9)$$

converges almost surely to a random variable supported by $\text{fix}(T)$.

**Proof:** The main idea behind the proof can be found in [32]. Define the operator $U = (1 - \eta_k) I + \eta_k T$ (we omit the index $k$ in $U$ to simplify the notation); similarly, define $U(\kappa) = (1 - \eta_k) I + \eta_k T(\kappa)$. Remark that the operator $U$ is $(\alpha \eta_k)$-averaged.

The iteration (9) reads $x^{k+1} = U(\xi^{k+1}) x^k$. Set $p_k = \mathbb{P}(\xi_1 = \kappa)$ for any $\kappa \in 2^\mathcal{J}$. Denote by $||x||^2 = \langle x, x \rangle$ the squared norm in $\mathcal{H}$. Define a new inner product $\langle x \cdot y \rangle = \sum_{\kappa \in \mathcal{J}}^J q_j(x_j, y_j)$ on $\mathcal{H}$ where $q_j = \sum_{\kappa \in 2^\mathcal{J}} \eta_k \mathbb{P}(\xi_k = \kappa)$ and let $||.||^2 = x \cdot x$ be its associated squared norm. Consider any $x^* \in \text{fix}(T)$. Conditionally to the sigma-field $\mathcal{F}^k = \sigma(\xi_1, \ldots, \xi_k)$ we have

$$\mathbb{E} ||x^{k+1} - x^*||^2 | \mathcal{F}^k = \sum_{\kappa \in 2^\mathcal{J}} p_k \left( U(\kappa) x^k - x^* \right)^2 = \sum_{\kappa \in 2^\mathcal{J}} p_k \sum_{j \in \kappa} q_j \left( \|U x^k - x^*\|^2 + \sum_{\kappa \in 2^\mathcal{J}} p_{j \notin \kappa} q_j x_j^k - x_j^* \right)^2$$

$$= \|x^k - x^*\|^2 + \sum_{\kappa \in 2^\mathcal{J}} p_k \sum_{j \in \kappa} q_j \left( \|U x^k - x^*\|^2 - \|x_j^k - x_j^*\|^2 \right)$$

$$= \|x^k - x^*\|^2 + \sum_{j=1}^{J} \left( \|U x^k - x^*\|^2 - \|x_j^k - x_j^*\|^2 \right)$$

$$= \|x^k - x^*\|^2 + \|U x^k - x^*\|^2 - \|x^k - x^*\|^2$$

Using that $U$ is $(\alpha \eta_k)$-averaged and that $x^*$ is a fixed point of $U$, the term enclosed in the parentheses is no larger than $\frac{1 - \alpha \eta_k}{\alpha \eta_k} \| (1 - U)x^k \|^2$. As $1 - U = \eta_k (1-T)$, we obtain:

$$\mathbb{E} ||x^{k+1} - x^*||^2 | \mathcal{F}^k \leq ||x^k - x^*||^2 - \eta_k (1 - \alpha \eta_k) \| (1 - T)x^k \|^2 \quad (10)$$

which shows that $\|x^k - x^*\|^2$ is a nonnegative supermartingale with respect to the filtration $(\mathcal{F}_k)$. As such, it converges with probability one towards a random variable that is finite almost everywhere.

Given a countable dense subset $H$ of $\text{fix}(T)$, there is a probability one set on which $\|x^k - x\| \to X_\epsilon \in [0, \infty)$ for all $x \in H$. Let $x^* \in \text{fix}(T)$, let $\epsilon > 0$, and choose $x \in H$ such that $\|x^* - x\| \leq \epsilon$. With probability one, we have

$$\|x^k - x^*\| \leq \|x^k - x\| + \|x - x^*\| \leq X_\epsilon + 2\epsilon$$

for $k$ large enough. Similarly, $\|x^k - x^*\| \geq X_\epsilon - 2\epsilon$ for $k$ large enough. We therefore obtain:

**C1:** There is a probability one set on which $\|x^k - x^*\|$ converges for every $x^* \in \text{fix}(T)$.

Getting back to (10), taking the expectations on both sides of this inequality and iterating over $k$, we obtain

$$\sum_{k=0}^{\infty} \eta_k(1 - \alpha \eta_k) \mathbb{E} \| (1 - T)x^k \|^2 \leq \|x^0 - x^*\|^2$$

Using the assumption on $(\eta_k)_{k \in \mathbb{N}}$, it is straightforward to see that $\sum_{k=0}^{\infty} \eta_k(1 - \alpha \eta_k) = +\infty$ and thus that $\sum_{k=0}^{\infty} \mathbb{E} \| (1 - T)x^k \|^2$ is finite. By Markov’s inequality and Borel Cantelli’s lemma, we therefore obtain:

**C2:** $(1 - T)x^k \to 0$ almost surely.

We now consider an elementary event in the probability one set where C1 and C2 hold. On this event, since $\|x^k - x^*\|$ converges for $x^* \in \text{fix}(T)$, the sequence $(x^k)_{k \in \mathbb{N}}$ is bounded. Since $T$ is $\alpha$-averaged, it is continuous, and C2 shows that all the accumulation points of $(x^k)_{k \in \mathbb{N}}$ are in $\text{fix}(T)$. It remains to show that these accumulation points reduce to one point. Assume that $x_1^*$ is an accumulation point. By C1, $\|x^k - x_1^*\|$ converges. Therefore, $\lim \|x^k - x_1^*\| = \lim \inf \|x^k - x_1^*\|$, which shows that $x_1^*$ is unique.

**Remark 2** At the time of writing the paper, the work [33] was brought to our knowledge. A result similar to Theorem 3 is presented in the framework of Hilbert spaces, random summable errors (dealt with by relying on the notion of quasi-Féjer monotonicity) and multiple blocks.

By Lemma 2, ADMM+ iterates are generated by the action of an $\alpha$-averaged operator. Theorem 3 shows then that a stochastic coordinate descent version of ADMM+ converges towards a primal-dual point. This result will be exploited in two directions: first, we describe a stochastic minibatch algorithm, where a large dataset is randomly split into smaller chunks. Second, we develop an asynchronous version of ADMM+ in the context where it is distributed on a graph.

IV. **APPLICATION TO INCREMENTAL OPTIMIZATION**

**A. Problem Setting**

Given an integer $N > 1$, consider the problem of minimizing a sum of composite functions

$$\inf_{x \in \mathcal{X}} \sum_{n=1}^{N} (f_n(x) + g_n(x)) \quad (11)$$

where we make the following assumption:

**Assumption 4** For each $n = 1, \ldots, N$,

(i) $f_n$ is a convex differentiable function on $\mathcal{X}$, and its gradient $\nabla f_n$ is $L$-Lipschitz continuous on $\mathcal{X}$ for some $L \geq 0$.

(ii) $g_n \in \mathcal{G}_0(\mathcal{X})$.

(iii) The infimum of Problem (11) is attained.

(iv) $\cap_{n=1}^{N} \text{ri dom } g_n \neq \emptyset$.

This problem arises for instance in large-scale learning applications where the learning set is too large to be handled
as a single block. Stochastic minibatch algorithms consist in splitting the data set into \( N \) chunks and to process each chunk in some order, one at a time. The quantity \( f_n(x) + g_n(x) \) measures the inadequacy between the model (represented by parameter \( x \)) and the \( n \)-th chunk of data. Typically, \( f_n \) stands for a data fitting term whereas \( g_n \) is a regularization term which penalizes the occurrence of erratic solutions. As an example, the case where \( f_n \) is quadratic and \( g_n \) is the \( \ell_1 \)-norm reduces to the popular LASSO problem \cite{R租j}. 

B. Instantiating ADMM+

We derive our stochastic minibatch algorithm as an instance of ADMM+ coupled with a randomized coordinate descent. To that end let us first rephrase Problem \ref{eq:main} as  

\[
\inf_{x \in \mathcal{X}^N} \sum_{n=1}^N (f_n(x_n) + g_n(x_n)) + \iota_C(x)
\]  

(12)

where the notation \( x_n \) represents the \( n \)-th component of any \( x \in \mathcal{X}^N \), \( \iota_A \) is the indicator function of a set \( A \) (null on \( A \) and infinite outside this set), and \( \mathcal{C} \) is the space of vectors \( x \in \mathcal{X}^N \) such that \( x_1 = \cdots = x_N \). On the space \( \mathcal{X}^N \), we set \( f(x) = \sum_n f_n(x_n) \), \( g(x) = \sum_n g_n(x_n) \), \( h(x) = \iota_C(x) \) and \( M = I_{\mathcal{X}^N} \) the identity matrix. Problem \ref{eq:main2} can be rewritten as  

\[
\inf_{x \in \mathcal{X}^N} f(x) + g(x) + h(Mx).
\]  

(13)

We define the natural scalar product on \( \mathcal{X}^N \) as \( \langle x, y \rangle = \sum_{n=1}^N \langle x_n, y_n \rangle \). Applying ADMM+ to Problem \ref{eq:main2} leads to the following procedure:  

\[
\begin{align*}
    z^{k+1} &= \text{proj}_C(x^k + \rho \lambda^k) \\
    \lambda^{k+1} &= \lambda^k + \rho \left( x^k - z^{k+1} \right) \\
    u^{k+1} &= (1 - \tau \rho^{-1}) x^k + \tau \rho^{-1} z^{k+1} \\
    x^{k+1} &= \arg \min_{w \in \mathcal{X}} \left\{ g_n(w) + \langle \nabla f_n(x^k), w \rangle \right\} \\
    &\quad + \frac{\|w - u^{k+1} + \tau \lambda^{k+1}\|^2}{2\tau}
\end{align*}
\]

where \( \text{proj}_C \) is the orthogonal projection onto \( \mathcal{C} \) (observe that the \( \text{proj} \) of an indicator function on a closed convex set coincides with the orthogonal projection on that set). Note that for any \( x \in \mathcal{X}^N \), \( \text{proj}_C(x) \) is equal to \((\bar{x}, \cdots, \bar{x})\) where \( \bar{x} \) is the average of vector \( x \) \( i.e., \bar{x} = N^{-1} \sum_n x_n \). Consequently, the components of \( z^{k+1} \) are equal and coincide with \( \bar{x}^k \) and \( \bar{\lambda}^k \) where \( \bar{x}^k \) and \( \bar{\lambda}^k \) are the averages of \( x^k \) and \( \lambda^k \) respectively. By inspecting the \( \lambda^{k+1} \)-update equation above, we notice that the latter equality simplifies even further by noting that \( \lambda^{k+1} = 0 \) or, equivalently, \( \bar{\lambda}^k = 0 \) for all \( k \geq 1 \) if the algorithm is started with \( \lambda^0 = 0 \). Finally, for any \( n \) and \( k \geq 1 \), the above iterations reduce to  

\[
\begin{align*}
    \bar{x}^k &= \frac{1}{N} \sum_{n=1}^N x_n^k \\
    \lambda^{k+1} &= \lambda^k + \rho \left( x^k - \bar{x}^k \right) \\
    u^{k+1} &= (1 - \tau \rho^{-1}) x^k + \tau \rho^{-1} \bar{x}^k \\
    x^{k+1} &= \text{prox}_{\tau g_n} \left[ \nabla f_n(x^k) - \tau \lambda^{k+1} \right]
\end{align*}
\]

where we recall the notation \( \text{prox} \) defined in \cite{R租j}. These iterations can be written more compactly as

**Minibatch ADMM+**

Initialization: \((x^0, \lambda^0)\) s.t. \( \sum_n \lambda^0_n = 0 \).

Do

- \( \bar{x}^k = \frac{1}{N} \sum_{n=1}^N x_n^k \),
- For batches \( n = 1, \ldots, N \), do \( \lambda^{k+1} = \lambda^k + \rho \left( x^k - \bar{x}^k \right) \) \( x^{k+1} = \text{prox}_{\tau g_n} \left[ (1 - 2\tau \rho^{-1}) x^k - \tau \nabla f_n(x^k) + 2\tau \rho^{-1} \bar{x}^k - \tau \lambda^k \right] \).

- Increment \( k \).

The following result is a straightforward consequence of Theorem \ref{thm:main}.

**Theorem 4** Let Assumption \ref{assum:main} hold true and assume that \( \tau^2 - \rho^{-1} > L/2 \). Then for any initial point \((x^0, \lambda^0)\) such that \( \lambda^0 = 0 \), there exists a primal-dual point \((x^*, \lambda^*)\) of Problem \ref{eq:main} such that \((\bar{x}^k, \bar{\lambda}^k)\) converges to \((x^*, \lambda^*)\).

At each step \( k \), the iterations given above involve the whole set of functions \( f_n, g_n (n = 1, \ldots, N) \). Our aim is now to propose an algorithm which involves a single couple of functions \( (f_n, g_n) \) per iteration.

C. A Stochastic Minibatch Primal Dual algorithm

We are now in position to state the main algorithm of this section. The proposed **Stochastic Minibatch Primal Dual algorithm (SMPD)** is obtained upon applying the randomized coordinate descent on the minibatch ADMM+:

**SMPD Algorithm:**

Initialization: \((x^0, \lambda^0)\).

Do

- Define \( \bar{x}^k \) and \( \bar{\lambda}^k \) as  

\[
\bar{x}^k = \frac{1}{N} \sum_{n=1}^N x_n^k, \quad \bar{\lambda}^k = \frac{1}{N} \sum_{n=1}^N \lambda_n^k.
\]

- Pick up the value of \( \xi^{k+1} \).

- For Batch \( n = \xi^{k+1} \), set  

\[
\begin{align*}
    \lambda^{k+1} &= \lambda^k - \bar{\lambda}^k + \frac{x_n^k - \bar{x}^k}{\rho} \\
    x^{k+1} &= \text{prox}_{\tau g_n} \left[ (1 - 2\tau \rho^{-1}) x_n^k - \tau \nabla f_n(x_n^k) \\
    &\quad - \tau \lambda^{k+1} + 2\tau \rho^{-1} \bar{x}^k \right].
\end{align*}
\]

- For all batches \( n \neq \xi^{k+1} \), set \( \lambda^{k+1} = \lambda_n^k, x^{k+1} = x_n^k \).

- Increment \( k \).

**Assumption 5** The random sequence \((\xi^k)_{k \in \mathbb{N}^+}\) is i.i.d. and satisfies \( \mathbb{P}(|\xi^1| = n) > 0 \) for all \( n = 1, \ldots, N \).

**Theorem 5** Let Assumptions \ref{assum:main} and \ref{assum:main2} hold true. Assume that \( \tau^2 - \rho^{-1} > L/2 \). For any initial point \((x^0, \lambda^0)\), the sequence \( \bar{x}^k \) generated by the SMPD algorithm converges almost surely
to a random variable supported by the set of minimizers of Problem \([13]\).

This theorem is proven in Appendix \([A]\).

V. DISTRIBUTED OPTIMIZATION

Consider a set of \(N > 1\) computing agents that cooperate to solve the minimization problem (11). Here, \(f_n, g_n\) are two private functions available at Agent \(n\). Our purpose is to design a random distributed (or decentralized) iterative algorithm where, at each iteration, each active agent updates a local estimate in the parameter space \(\mathcal{X}\) based on the sole knowledge of its private functions and on information it received from its neighbors through some communication network. Eventually, the local estimates will converge to a common consensus value which is a minimizer of the aggregate function of problem (11) if any.

Instances of this problem appear in learning applications where massive training data sets are distributed over a network and processed by distinct machines \([38]\), \([39]\), in resource allocation problems for communication networks \([17]\), or in statistical estimation problems by sensor networks \([16]\), \([40]\).

A. Network Model and Problem Formulation

We represent the network as an undirected graph \(G = (V, E)\) where \(V = \{1, \ldots, N\}\) is the set of agents/nodes and \(E\) is the set of edges. Representing an edge by a set \(\{n, m\}\) with \(n, m \in V\), we write \(m \sim n\) whenever \(\{n, m\} \in E\). Practically, \(n \sim m\) means that agents \(n\) and \(m\) can communicate with each other.

Assumption 6 \(G\) is connected, and \(n \neq m\) for all \(\{n, m\} \in E\).

Let us introduce some notation. For any \(x \in \mathcal{X}^{|V|}\), we denote by \(x_n\) the components of \(x\), i.e., \(x = (x_n)_{n \in V}\). We introduce the functions \(f\) and \(g\) on \(\mathcal{X}^{|V|} \to (-\infty, +\infty]\) as \(f(x) = \sum_{n \in V} f_n(x_n)\) and \(g(x) = \sum_{n \in V} g_n(x_n)\). Clearly, Problem (11) is equivalent to the minimization of \(f(x) + g(x)\) under the constraint that all components of \(x\) are equal.

As done in Section IV, we can rephrase the optimization problem in the form of (2) and apply ADMM+ where the last term \(h(Mx)\) shall coincide with the indicator function of \(C\). However, simply setting \(h = \iota_C\) and \(M\) as the identity would not lead to a distributed algorithm. Loosely speaking, we must define \(h\) and \(M\) in such a way that it encodes the communication graph. Our goal will be to ensure global consensus through local consensus over every edge of the graph.

For any \(\epsilon \in E\), say \(\epsilon = \{n, m\}\), we define the linear operator \(M_\epsilon : \mathcal{X}^{|V|} \to \mathcal{X}^2\) as \(M_\epsilon(x) = (x_n, x_m)\) assuming \(n < m\) to avoid any ambiguity on the definition of \(M\).

We construct the linear operator \(M : \mathcal{X}^{|V|} \to \mathcal{Y} \triangleq \mathcal{X}^2|E|\) as \(Mx = (M_\epsilon(x))_{\epsilon \in E}\) where we assume some (irrelevant) ordering on the edges. Any vector \(y \in \mathcal{Y}\) will be written as \(y = (y_\epsilon)_{\epsilon \in E}\) where, writing \(\epsilon = \{n, m\} \in E\), the component \(y_\epsilon\) will be represented by the couple \(y_\epsilon = (y_n(n), y_m(m))\) with \(n < m\). Note that this notation is abusive since it tends to indicate that \(y_\epsilon\) has more than two components. However, it will turn out to be convenient in the sequel. We also introduce the subspace of \(\mathcal{X}^2\) defined as \(C_2 = \{(x, x) : x \in \mathcal{X}\}\). Finally, we define \(h : \mathcal{Y} \to (-\infty, +\infty]\) as

\[
h(y) = \sum_{\epsilon \in E} \iota_{C_2}(y_\epsilon). \tag{17}
\]

We consider the following problem:

\[
\inf_{x \in \mathcal{X}^{|V|}} f(x) + g(x) + h(Mx). \tag{18}
\]

Lemma 3 Let Assumption 6 hold true. The minimizers of (18) are the tuples \((x^*, \ldots, x^*)\) where \(x^*\) is any minimizer of (11).

Proof: Assume that Problem (18) has a minimizer \(\bar{x} = (x_1, \ldots, x_{|V|})\). Then

\[
\bar{h}(M \bar{x}) = \sum_{\epsilon \in \{n, m\} \in E} \iota_{C^2}(x_n, x_m) = 0.
\]

Since the graph \(G\) is connected, this equation is satisfied if and only if \(\bar{x} = (x^*, \ldots, x^*)\) for some \(x^* \in \mathcal{X}\). The result follows.

B. Instantiating ADMM+

We now apply ADMM+ to solve the problem (18). Since the newly defined function \(h\) is separable with respect to the \((y_\epsilon)_{\epsilon \in E}\), we get

\[
\text{prox}_{\rho h}(y) = (\text{prox}_{\rho \iota_{C_2}}(y_\epsilon))_{\epsilon \in E} = (\bar{y}_n, \bar{y}_m)_{\epsilon \in E}
\]

where \(\bar{y}_n = (y_n(n) + y_m(m))/2\) if \(\epsilon = \{n, m\}\). With this in mind, the update equation (4a) of ADMM+ is written as \(z^{k+1} = ((z^{k+1}_n, z^{k+1}_m))_{\epsilon \in E}\) where \(z^{k+1}_\epsilon = (x^{k}_n + x^{k}_m)/2 + \rho(\lambda^{k}_n(n) + \lambda^{k}_m(m))/2\) for any \(\epsilon = \{n, m\} \in E\). Plugging this inequality into Eq. (4b), it can be seen that \(\lambda^{k}_n(n) = -\lambda^{k}_m(m)\). Therefore, \(\lambda^{k+1}_\epsilon = (x^{k}_n + x^{k}_m)/2\) for any \(k \geq 1\). Moreover, \(\lambda^{k+1}_\epsilon(n) = \lambda^{k}_\epsilon(n) + (x^{k}_n - x^{k}_m)/(2\rho)\).

Let us now instantiate Equations (4c) and (4d). Observe that the \(n\)-th component of the vector \(M^*M\bar{x}\) coincides with \(d_n x_n\) where \(d_n\) is the degree (i.e., the number of neighbors) of node \(n\). From Eq. (4d), the \(n\)-th component of \(x^{k+1}\) is written

\[
x^{k+1}_n = \text{prox}_{\tau d_n}(d_n \frac{(M^*(u^{k+1} - \tau \lambda^{k+1})) - \tau \nabla f_n(x^{k}_n)}{n})
\]

where for any \(y \in \mathcal{Y}\),

\[
(M^*y)_n = \sum_{m:\{m, n\} \in E} y_{m}(m)(n)
\]

is the \(n\)-th component of \(M^*y \in \mathcal{X}^{|V|}\). Plugging Eq. (4c) together with the expressions of \(\bar{z}_{\{n, m\}}^{k+1}\) and \(\lambda_{\{n, m\}}^{k+1}(n)\) in the argument of \(\text{prox}_{\tau d_n}/d_n\), we get after a small calculation

\[
x^{k+1}_n = \text{prox}_{\tau d_n}(d_n \frac{(1 - \tau \rho^{-1})x^{k}_n - \tau \nabla f_n(x^{k}_n)}{d_n} + \frac{\tau}{d_n} \sum_{m:\{m, n\} \in E} (\rho^{-1} x^{k}_m - \lambda_{\{n, m\}}^{k}(n))).
\]
The algorithm is finally described by the following procedure:
Prior to the clock tick $k + 1$, the node $n$ has in its memory the variables $x_n^k$, $\{\lambda_{(n,m)}^k(m)\}_{m \sim n}$, and $\{x_m^k\}_{m \sim n}$.

**Distributed ADMM+**

Initialization: $(x^0, \lambda^0)$ s.t. $\lambda^0_{(n,m)}(n) = -\lambda^0_{(n,m)}(m)$ for all $m \sim n$.

Do

- For all $n \in V$, Agent $n$ performs the following operations:
  \[
  \lambda_{(n,m)}^{k+1}(n) = \lambda_{(n,m)}^k(n) + \frac{x_n^k - x_m^k}{2\rho}
  \]  
  \[
  x_m^{k+1} = \text{prox}_{\tau g_n/d_n} \left( (1 - \tau \rho^{-1})x_m^k - \tau \frac{\tau}{d_n} \nabla f_n(x_m^k) + \frac{\tau}{d_n} \sum_{m':(n,m) \in E} (\rho^{-1}x_m^k - \lambda_{(n,m)}^k(n)) \right)
  \]

- For all $n \in V$, Agent $n$ sends the parameter $x_m^{k+1}$ to its neighbors,
- Increment $k$.

The proof of the following result is provided in Appendix [B].

**Theorem 6** Let Assumptions 2 and 6 hold true. Assume that

\[
\tau^{-1} - \rho^{-1} > \frac{L}{2d_{\text{min}}}
\]

where $d_{\text{min}}$ is the minimum of the nodes' degrees in the graph $G$. For any initial value $(x^0, \lambda^0)$, let $(x^k)_{k \in \mathbb{N}}$ be the sequence produced by the Distributed ADMM+. Then there exists a minimizer $x^*$ of Problem (11) such that for all $n \in V$, $(x_n^k)_{k \in \mathbb{N}}$ converges to $x^*$.

**C. A Distributed Asynchronous Primal Dual Algorithm**

In the distributed synchronous case, at each clock tick, a central scheduler activates all the nodes of the network simultaneously and monitors the communications that take place between these nodes once they have finished their prox($\cdot$) and gradient operations. The meaning we give to “distributed asynchronous algorithm” is that there is no central scheduler and that any node can wake up randomly at any moment independently of the other nodes. This mode of operation brings clear advantages in terms of complexity and flexibility.

The proposed **Distributed Asynchronous Primal Dual** algorithm (DAPD) is obtained by applying the randomized coordinate descent on the above algorithm. As opposed to the latter, the resulting algorithm has the following attractive property: at each iteration, a single agent, or possibly a subset of agents chosen at random, are activated. More formally, let $(\xi^k)_{k \in \mathbb{N}}$ be a sequence of i.i.d. random variables valued in $2^V$. The value taken by $\xi^k$ represents the agents that will be activated and perform a prox on their $x$ variable at moment $k$. The asynchronous algorithm goes as follows:

**DAPD Algorithm:**

Initialization: $(x^0, \lambda^0)$.

Do

- Select a random set of agents $\xi^{k+1} = \mathcal{A}$.
- For all $n \in \mathcal{A}$, Agent $n$ performs the following operations:
  - For all $m \sim n$, do
    \[
    \lambda_{(n,m)}^{k+1}(n) = \lambda_{(n,m)}^k(n) + \frac{x_n^k - x_m^k}{2\rho} \]
    \[\lambda_{(n,m)}^{k+1}(m) = \lambda_{(n,m)}^k(m) + \frac{x_m^k - x_n^k}{2\rho} \]
    \[x_m^{k+1} = \text{prox}_{\tau g_n/d_n} \left( (1 - \tau \rho^{-1})x_m^k - \tau \frac{\tau}{d_n} \nabla f_n(x_m^k) + \frac{\tau}{d_n} \sum_{m':(n,m) \in E} (\rho^{-1}x_m^k + \lambda_{(n,m)}^{k+1}(m)) \right) \]
  - For all $m \sim n$, send $\{x_n^{k+1}, \lambda_{(n,m)}^{k+1}(n)\}$ to Neighbor $m$.
  - For all agents $n \notin \mathcal{A}$, $x_n^{k+1} = x_n^k$, and $\lambda_{(n,m)}^{k+1}(n) = \lambda_{(n,m)}^k(n)$ for all $m \sim n$.
  - Increment $k$.

**Assumption 7** The collections of sets $\{\mathcal{A}_1, \mathcal{A}_2, \ldots\}$ such that $\mathbb{P}[\xi^t = \mathcal{A}_1]$ is positive satisfies $\bigcup \mathcal{A}_1 = V$.

In other words, any agent is selected with a positive probability. The following theorem is proven in Appendix [C].

**Theorem 7** Let Assumptions 2, 6 and 7 hold true. Assume that condition (20) holds true. Let $(x_n^{k+1})_{n \in V}$ be the output of the DAPD algorithm. For any initial value $(x^0, \lambda^0)$, the sequences $x_1^k, \ldots, x_V^k$ converge almost surely as $k \to \infty$ to a random variable $x^*$ supported by the set of minimizers of Problem (11).

Before turning to the numerical illustrations, we note that the very recent paper [41] also deals with asynchronous primal-dual distributed algorithms by relying on the idea of random coordinate descent.

**VI. NUMERICAL ILLUSTRATIONS**

**Problem.** We address the problem of the regularized logistic regression. Denoting by $m$ the number of observations and by $p$ the number of features, the optimization problem is written

\[
\min_{x \in \mathbb{R}^p} \frac{1}{m} \sum_{t=1}^m \log \left( 1 + e^{-y_t a_t^T x} \right) + \mu r(x)
\]

where the $(y_t)_{t=1}^m$ are in $\{-1, +1\}$, the $(a_t)_{t=1}^m$ are in $\mathbb{R}^p$, $\mu > 0$ is a scalar, and $r(\cdot)$ a regularization function (typically a 1 or 2-norm). Let $(B_n)_{n=1}^N$ denote a partition of $\{1, \ldots, m\}$. The optimization problem is then written

\[
\min_{x \in \mathbb{R}^p} \sum_{n=1}^N \left( \sum_{t \in B_n} \frac{1}{m} \log \left( 1 + e^{-y_t a_t^T x} \right) \right) + \mu r(x)
\]

or

\[
\min_{x \in \mathbb{R}^{Np}} \sum_{n=1}^N \left( \sum_{t \in B_n} \frac{1}{m} \log \left( 1 + e^{-y_t a_t^T x} \right) + \frac{\mu}{N} r(x_n) \right)
\]

(23)
by considering variables in \( \mathbb{R}^{NP} \) of which each block is related to one batch. Obviously Problems (21), (22) and (24) are equivalent and Problem (24) is in the form of (12).

**Datasets.** In the whole section, we will perform our simulations on four classical datasets:

<table>
<thead>
<tr>
<th>name</th>
<th>m</th>
<th>p</th>
<th>density</th>
</tr>
</thead>
<tbody>
<tr>
<td>covtype</td>
<td>581012</td>
<td>54</td>
<td>dense</td>
</tr>
<tr>
<td>alpha</td>
<td>500000</td>
<td>500</td>
<td>sparse</td>
</tr>
<tr>
<td>realsim</td>
<td>72309</td>
<td>20958</td>
<td>sparse</td>
</tr>
<tr>
<td>rcvl</td>
<td>20242</td>
<td>47236</td>
<td>sparse</td>
</tr>
</tbody>
</table>

The datasets covtype, realsim, and rcvl are taken from the LIBSVM website[^1] and alpha was from the Pascal 2008 Large Scale Learning challenge[^2]. We preprocessed the dense datasets so that each feature has zero mean and unit variance.

**Models.** We consider two different situations:

**A. Distributed Optimization.** At each iteration, one agent chosen randomly processes its private batch of data and communicates with its neighbors. In this setup, we evaluate the DAPD algorithm described in Section V.

**B. Minibatch.** At each iteration, a random batch of data is processed. In this setup, we evaluate the SMPD algorithm described in Section IV.

### A. Distributed Optimization

**Setup.** We consider the case where the dataset is scattered over a network. Indeed, massive data sets are often distributed on different physical machines communicating together by means of an interconnection network [8, Chap. 2.5] and many algorithms have been implemented for independent threads or processes running on distant cores, closer to the data (see e.g. [20], [9] for MapReduce implementation of ADMM, [22] for Spark implementation). To do so, we introduce a graph \( G = (V, E) \) representing the connections between the agents.

The goal for the agents is now to perform \( \ell_2 \)-regularized logistic regression in a decentralized manner (we chose \( \ell_2 \) regularization here for comparison purposes and to illustrate the flexibility of our algorithm). This problem is written:

\[
\min_{x \in \mathbb{R}^N} \sum_{n=1}^{N} \left( \sum_{\ell \in E_n} \frac{1}{m} \log \left( 1 + e^{-y_n a_{\ell}^T x_n} \right) + \frac{\mu}{2N} \| x_n \|_2^2 \right) + \sum_{\epsilon \in E} c_{\epsilon} (y_{\epsilon}).
\]

the third function is the same function as in Section V Problem (18). The regularization parameter \( \mu \) was also set to \( 10^{-4} \).

**Algorithms comparison** We compare here some distributed optimization algorithms for which one or more agents wake up, process their data, and send information to their neighbors: no coordinator/fusion center is present to collect or manage data. The considered algorithms are:

- DGD: the synchronous distributed algorithm. Here, each agent performs a gradient descent then exchanges with its neighbors according to the Metropolis rule.
- ABG: the asynchronous broadcast gradient [42]. In this setup, one agent wakes up and sends its information to its neighbors. Any of these neighbors replaces its current value with the mean of this value and the received value then performs a gradient descent.
- PWG: the pairwise gossip gradient. In this setup, one agent wakes up, and selects one neighbor. Then each of the two agents performs a gradient descent, then exchanges and replaces its value by taking the mean between the former and the received value.
- DAPD: our DAPD algorithm.

For DGD, ABG, and PWG, the stepsizes have been taken decreasing as \( \gamma_0 / t^{0.75} \). The other parameters (including \( \gamma_0 \)) were chosen automatically in sets of the form \( \text{parameter}_{\text{theory}} \times 10^6 \).

Whereas DAPD can allow for multiple agents to wake at each iteration, we considered only the single active agent case as it does not change much the practical implementation. It is thus underperforming compared to a multiple awaking agents scenario. The stepsizes of DAPD have been chosen automatically in sets of the form \( \text{parameter}_{\text{theory}} \times 10^6 \) for \( \tau \) and \( \rho = 2 \tau \) for fairness in terms of number of step sizes explored.

The (total) functional cost was evaluated with the value at agent 1 (the agents are indistinguishable from a network point of view) and plotted versus the number of local gradients used.

In Figure 1, we plot the \( \ell_2 \)-regularized logistic cost at some agent versus the number of local gradients used. We solved this problem for each dataset on a \( 10 \times 10 \) 2D toroidal grid (100 agents) by assigning the same number of observations per agent. We observe that the DAPD is significantly quicker than the other stochastic gradient methods. Finally, we also remark that the quantity of information exchanged per iteration for DAPD is roughly a vector of length shorter than \( 2NP \) (8p with our graph) which means that the number of transmissions is in general quite small compared to the size of the whole dataset (roughly \( TP \)).

In Figure 2, we plot the same quantities for the rcvl dataset but now the same number of observations are dispatched over i) a \( 5 \times 5 \) toroidal grid (25 agents) and ii) a 50-nodes complete network.

### B. Minibatch

**Setup.** Consider Problem (21) with \( \ell_1 \) regularization, the goal of minibatch processing is to find a minimum of this problem by using one randomly selected batch per iteration. The regularization parameter \( \mu \) was set to \( 10^{-4} \).

**Algorithms comparison** We compare here optimization algorithm capable of handling composite functions with a non-smooth part:

- ISTA: the (deterministic) Forward-Backward algorithm applied to Eq. (21). One can remark that as mentioned in Section IV when only two functions are considered,
where chosen automatically in a set of observations used divided by the total number of observations, of effective passes over the data, computed as the number of the tuning process.

For all stochastic algorithms, the algorithm parameters where chosen automatically in a set of 10 possible values of the form parameter_{theory} × 2^k and taking the parameter leading to the lowest functional cost. For the SMPD, we chose τ automatically as above and took ρ = 2τ out of fairness for the tuning process.

We plot hereunder the functional cost versus the number of effective passes over the data, computed as the number of observations used divided by the total number of observations, in order to compare full-batch and mini-batch algorithms more fairly.

In Figure 3 we plot the ℓ1-regularized logistic cost versus the number of passes over the data, each dataset being split into 5 batches. The point reached by LibLinear is represented by the black star * (except for the dataset alpha for which it produced a memory error). We remark that the SMPD offers comparable performance to MISO when the number of features is moderate. When the number of features becomes large, the SMPD performs quite poorly at the beginning but matches FISTA and eventually MISO after a few tens of passes; this may be due to the fact that x̂ is not sparse at first as the sparsification is made at each batch and thus may differ before the dual variables make them match. Overall, the stochastic methods derived from ADMM+ (SMPD, and even MISO which can be seen as a coordinate descent of the Forward-Backward) can have interesting performances in many different situations.

In Figure 4 we plot the same quantities for the covtype dataset but with different batch sizes.

For the deterministic algorithms (ISTA, FISTA), the Lipschitz constant of the first function was taken equal to the upper-bound 0.25 max_{i=1,...,m} \{||a_i||_2^2\}.

For all stochastic algorithms, the algorithm parameters where chosen automatically in a set of 10 possible values of the form parameter_{theory} × 2^k and taking the parameter leading to the lowest functional cost. For the SMPD, we chose τ automatically as above and took ρ = 2τ out of fairness for the tuning process.

We plot hereunder the functional cost versus the number of effective passes over the data, computed as the number of observations used divided by the total number of observations,
VII. CONCLUSIONS AND PERSPECTIVES

This paper introduced a general framework for stochastic coordinate descent. The framework was used on a new algorithm called ADMM+ which has roots in a recent work by Vü and Condat. As a byproduct, we obtained a stochastic approximation algorithm which can be used to handle distinct data blocks sequentially. We also obtained an asynchronous distributed algorithm which enables the processing of distinct blocks on different machines. Future works include an analysis of the convergence rate of our algorithms along with efficient stepizes strategies.

APPENDIX

A. Proof of Theorem 5

Let us define \((\hat{f}, \hat{g}, h, M) = (f, g, h, 1_X\times)\) where the functions \(f, g, h\) and \(M\) are the ones defined in Section IV-B. Then the iterates \((\lambda_{n+1}^k, x_{n+1}^k)_{n=1}^{N}\) described by Equations (14) coincide with the iterates \((\lambda_{n+1}^{k+1}, y_{n+1}^{k+1})\) described by Equations (6). If we write these equations more compactly as \((\lambda_{n+1}^{k+1}, x_{n+1}^{k+1}) = T(\lambda_n^k, x_n^k)\) where the operator \(T\) acts in the space \(\mathcal{H} = X^N \times X^N\), then Lemma 2 shows that \(T\) is \(\alpha\)-averaged. Defining the selection operator \(S_n\) on \(\mathcal{H}\) as \(S_n(\lambda, x) = (\lambda_n, x_n)\), we obtain that \(\mathcal{H} = S_1(\mathcal{H}) \times \cdots \times S_N(\mathcal{H})\) up to an element reordering. To be compatible with the notation of Section IV-B, assume that \(J = N\) and that the random sequence \(\xi_\rho\) driving the SMPD algorithm is set valued in \(\{1, \ldots, N\} \subset 2^J\). In order to establish Theorem 5, we need to show that the iterates \((\lambda_{n+1}^{k+1}, x_{n+1}^{k+1})\) provided by the SMPD algorithm are those who satisfy the equation \((\lambda_{n+1}^{k+1}, x_{n+1}^{k+1}) = T(\lambda_n^{k+1}, x_n^{k+1})\), as described in Theorem 5. Then by the direct application of Theorem 3.

Let us start with the \(\lambda\)-update equation. Since \(h = \zeta_\rho\), its Legendre-Fenchel transform is \(h^* = \iota_{C^\perp}\) where \(C^\perp\) is the orthogonal complement of \(C\) in \(X^N\). Consequently, if we write \((\eta^{k+1}, \xi^{k+1}) = T(\lambda^{k+1}, x^{k+1})\), then by Eq. (6a),

\[
\eta^{k+1} = \lambda^{k+1} - \lambda^k + \frac{x^k - x^{k+1}}{\rho} \quad n = 1, \ldots, N
\]

Notice that in general, \(\lambda^k \neq 0\) because in the SMPD algorithm, only one component is updated at a time. If \(\{n\} = \xi^{k+1}\), then \(\lambda_{n+1}^{k+1} = \eta_{n+1}^{k+1}\) which is Eq. (15). All other components of \(\lambda^k\) are carried over to \(\lambda_{n+1}^{k+1}\).

By Equation (6b) we also get

\[
\eta^{k+1} = \text{prox}_{\tau g_n} \left[ x_n^k - \tau \nabla f_n(x_n^k) - \tau (2\lambda^{k+1}_n - \lambda^k) \right] 
\]

If \(\{n\} = \xi^{k+1}\), then \(x_{n+1}^{k+1} = \eta_{n+1}^{k+1}\) can easily be shown to be given by Eq. (16).

B. Proof of Theorem 6

The proof simply consists in checking that the assumptions of Theorem 1 are satisfied. To that end, we compute the Lipschitz constant \(L\) of \(\nabla (f \circ M^{-1})\) as a function of \(\hat{L}\). Recall that \(S\) is the image of \(M\). For any \(y \in S\), note that

\[
\nabla (f \circ M^{-1})(y) = M(M^*M)^{-1} \nabla f(M^{-1}y) 
\]

Using the definition of \(M\), the operator \(M^*M\) is diagonal. More precisely, for any \(x \in \mathbb{R}^{\mid V\mid}\), say \(x = (x_n)_{n \in V}\), the \(n\)th component of \((M^*M)x\) coincides with \(d_n x_n\) where \(d_n = \text{card}\{m \in V : n \sim m\}\) is the degree of node \(n\) in the graph \(G\). Thus, \(\|M(M^*M)^{-1}x\|^2 = \sum_{n \in V} d_n^{-1} \|x_n\|^2\). As a consequence of the latter equality and (26), for any \((y, y') \in S^2\), say \(y = Mx\) and \(y' = Mx'\), one has

\[
\|\nabla (f \circ M^{-1})(y) - \nabla (f \circ M^{-1})(y')\|^2 = \sum_{n} d_n^{-1} \|\nabla f_n(x_n) - \nabla f_n(x'_n)\|^2 
\]

Under the stated hypotheses, we can write for all \(n\),

\[
\|\nabla f_n(x_n) - \nabla f_n(x'_n)\|^2 \leq \hat{L}^2 \|x_n - x'_n\|^2 
\]

Thus,

\[
\|\nabla (f \circ M^{-1})(y) - \nabla (f \circ M^{-1})(y')\|^2 \leq \hat{L}^2 (d_{min})^2 \|x - x'\|^2 
\]

C. Proof of Theorem 4

Let \((\hat{f}, \hat{g}, h) = (f \circ M^{-1}, M^*M^{-1}, h)\) where \(f, g, h\) and \(M\) are those of Problem (19). For these functions, write Equations (6) as \((\lambda_{n+1}^{k+1}, y_{n+1}^{k+1}) = T(\lambda_n^{k+1}, y_n^{k+1})\). By Lemma 2, the operator \(T\) is an \(\alpha\)-averaged operator acting on the space \(\mathcal{S} \times \mathcal{S}\), where \(\mathcal{S}\) is the image of \(\hat{X}^{\mid V\mid}\) by \(M\). For any \(n \in V\), let \(S_n\) be the selection operator on \(\mathcal{H}\) defined as \(S_n(\lambda, x) = \lambda(n) \in E_n \subset \mathbb{R}, \xi_n \in x_n\). Then it is easy to see that up to an element reordering, \(\mathcal{H} = S_1(\mathcal{H}) \times \cdots \times S_N(\mathcal{H})\). Identifying the set \(J\) introduced before the statement of Theorem 4 with \(V\), the operator \(T(\xi^{k+1})\) is defined as follows: if \(n \notin \xi^{k+1}\), then \(S_n(T(\xi^{k+1})(\lambda, x)) = S_n(\lambda, x)\) while if \(n \in \xi^{k+1}\), then \(S_n(T(\xi^{k+1})(\lambda, x)) = S_n(\lambda, x)\). We know by Theorem 3 that the sequence \((\lambda^{k+1}, x^{k+1}) = T(\xi^{k+1})(\lambda^k, x^k)\) converges almost surely to a primal-dual point of Problem (T). This implies by Lemma 4 that the sequence \(x^k\) converges almost surely to \((x^*, \ldots, x^*)\) where \(x^*\) is a minimizer of Problem (11).

We therefore need to prove that the operator \(T(\xi^{k+1})\) is translated into the DAPD algorithm. The definition (17) of \(h\) shows that

\[
h^*(\phi) = \sum_{e \in E} \iota_{C^\perp}(\phi_e) 
\]

where \(C^\perp = \{ (x, -x) : x \in \mathcal{X} \}\). Therefore, writing

\[
(\eta^{k+1}, \xi^{k+1} = M\xi^{k+1}) = T(\lambda^k, y^k = Mx^k), 
\]

Equation (6a) shows that

\[
\eta^{k+1} = \text{prox}_{C^\perp}(\lambda^k + \rho^{-1}y^k) 
\]

Notice that contrary to the case of the synchronous algorithm (19), there is no reason here for which \(\text{prox}_{C^\perp}(\lambda^k) = 0\).
Getting back to (\(\lambda^{k+1}, Mx^{k+1}\)) = \(T(\xi^{k+1}) (\lambda^{k}, y^{k}) = Mx^{k}\)), we therefore obtain that for all \(n \in \xi^{k+1}\) and all \(m \sim n\),
\[
\lambda_{\{n,m\}}^{k+1}(n) = \frac{\lambda^{k}_{n,m}(n) - \lambda^{k}_{n,m}(m)}{2} + \frac{y_{n,m}(n) - y_{n,m}(m)}{2\rho} = \frac{\lambda^{k}_{n,m}(n) - \lambda^{k}_{n,m}(m)}{2} + \frac{x^{k}_{n} - x^{k}_{m}}{2\rho}.
\]

Recall now that Eq. (6b) can be rewritten as
\[
q^{k+1} = \arg\min_{w \in S} g(w) + \langle \nabla f(y^{k}), w \rangle + \frac{1}{\tau} \|w - y^{k} + \tau (2\lambda^{k+1} - \lambda^{k})\|^{2}/2\tau
\]

Upon noting that \(g(Mx) = g(x)\) and \(\langle \nabla f(y^{k}), Mx \rangle = (\langle M^{-1} \nabla f(M^{-1} Mx^{k}), Mx \rangle = \langle \nabla f(x^{k}), x \rangle\), the above equation becomes
\[
v^{k+1} = \arg\min_{w \in \mathcal{X}} g(w) + \langle \nabla f(x^{k}), w \rangle + \frac{1}{\tau} \|M(w - x^{k}) + \tau (2\lambda^{k+1} - \lambda^{k})\|^{2}/2\tau.
\]

Recall that \((M^{*}Mx)_{n} = d_{n}x_{n}\). Hence, for all \(n \in \xi^{k+1}\), we get after some computations
\[
x^{k+1}_{n} = \text{prox}_{\tau_{g_{n,d}}}(x^{k}_{n} - \frac{\tau}{d_{n}} \nabla f_{n}(x^{k}_{n}) - \frac{\tau}{d_{n}} (M^{*}(2\lambda^{k+1} - \lambda^{k}))_{n}).
\]

Using the identity \((M^{*}y)_{n} = \sum_{m;\{n,m\} \in E} y_{n,m}(n)\), one can check that this equation coincides with the \(x\)-update equation in the DAPD algorithm.

**REFERENCES**


