Using big steps in coordinate descent primal-dual algorithms

Pascal Bianchi, Olivier Fercoq

Using Big Steps in Coordinate Descent Primal-Dual Algorithms

Pascal Bianchi and Olivier Fercoq

***Abstract*— The Vũ-Condat algorithm is a standard method for finding a saddle point of a Lagrangian involving a differentiable function. Recent works have tried to adapt the idea of random coordinate descent to this algorithm, with the aim to efficiently solve some regularized or distributed optimization problems. A drawback of these approaches is that the admissible step sizes can be small, leading to slow convergence. In this paper, we introduce a coordinate descent primal-dual algorithm which is provably convergent for a wider range of step size values than previous methods. In particular, the condition on the step-sizes depends on the *coordinate-wise* Lipschitz constant of the differentiable function's gradient. We discuss the application of our method to distributed optimization and large scale support vector machine problems.**

I. Introduction

In applications such as machine learning or multiagent systems, one is often faced with the issue of solving large scale optimization problems containing both differentiable and non-differentiable terms. In this context, random coordinate descent (CD) algorithms are amongst the most popular approaches. They consist in updating only a subset of the components of the estimate at each iteration, the other coordinates being maintained to their past value. Not only it is often computationally easier to evaluate a single coordinate of the gradient vector rather than the whole vector, but the conditions under which the CD version of the algorithm is provably convergent are generally weaker than in the case of their deterministic counterpart. For instance, in the CD version of the gradient descent [1] (or its proximal variant [2]), the *step size* used in the algorithm when updating a given coordinate i can be chosen to be inversely proportional to the *coordinate-wise* Lipschitz constant of the differentiable function along its ith coordinate, rather than the global Lipschitz constant (as would be the case in a standard gradient descent). Hence, the introduction of coordinate descent allows to use *longer step sizes* which results in a more attractive performance.

There is a rich literature on CD version of *primal* methods. Richtárik and Takáč [2] apply CD to the minimization of a sum of two convex functions $f + g$. The algorithm of [2] is analyzed under the additional assumption that function g is *separable* in the sense that for each $x \in \mathcal{X}$, $g(x) = \sum_{i=1}^{n} g_i(x^{(i)})$ for some functions $g_i : \mathcal{X}_i \to]-\infty, +\infty]$, where $x^{(i)}$ stands for the ith coordinate of x.

Comparatively, less work has been done regarding the application of CD to primal dual methods *i.e.*, when one seeks to find a saddle point of the Lagrangian. In the case where the optimization problem contains a separable and a strongly convex function, Zhang and Xiao [3] introduce a stochastic CD primal-dual algorithm and analyze its convergence rate. In 2013, Iutzeler et al. [4] proved that random coordinate descent can be successfully applied to fixed point iterations of firmly non-expansive (FNE) operators. It is known that the ADMM can be written as a fixed point algorithm of a FNE operator, which led the authors of [4] to propose a coordinate descent version of ADMM with application to distributed optimization. The key idea behind the convergence proof of [4] is to establish the so-called stochastic Fejér monotonicity of the sequence of iterates as noted by [5]. In a more general setting than [4], Combettes *et al.* in [5] and Bianchi *et al.* [6] extend the proof to the so-called α-averaged operators, which include FNE operators as a special case. This generalization allows to apply the coordinate descent principle to a broader class of primal-dual algorithms which is no longer restricted to the ADMM or the Douglas Rachford algorithm. For instance, Forward-Backward splitting is considered in [5] and a particular case of the Vũ-Condat algorithm is considered in [6].

However, in the approach of [4], [5], [6] the convergence conditions are identical to the ones of the brute method, the one without coordinate descent. These conditions involve the global Lipschitz constant of the gradient the differentiable term instead than its coordinate-wise Lipschitz constants. In practice, it means that the application of coordinate descent to primal-dual algorithm as suggested by [5] and [6] is restricted to the use of potentially small step sizes. One of the major benefits of coordinate descent is lost.

In this paper, we provide a CD primal-dual algorithm with a broader range of admissible step sizes. Our numerical experiments show that remarkable performance gains can be obtained when using larger step sizes. We review applications to asynchronous distributed optimization and machine learning.

The paper is organized as follows. In Section II, we formulate the problem. In Section III we review some applications to distributed optimization and to support vector machines (SVM). The main algorithm is introduced in SectionIV. The algorithm is instantiated in the case of distributed optimization in Section V. Numerical experiments are provided in Section VI in the case of SVM.

II. Optimization Framework

We consider the optimization problem

$$\inf_{x \in \mathcal{X}} f(x) + g(x) + h(Mx) \tag{1}$$

LTCI, CNRS, Télécom ParisTech, Université Paris-Saclay, 75013, Paris, France. E-mails: forename.name@telecom-paristech.fr.

This work has been supported by the Orange/Telecom ParisTech think tank Phi-TAB and the ANR project ODISSEE

where $\mathcal{X}$ is a Euclidean space, $M : \mathcal{X} \to \mathcal{Y}$ is a linear operator onto a second Euclidean space $\mathcal{Y}$; functions $f : \mathcal{X} \to \mathbb{R}$, $g : \mathcal{X} \to (-\infty, +\infty]$ and $h : \mathcal{Y} \to]-\infty, +\infty]$ are assumed proper, closed and convex; the function f is moreover assumed differentiable. We assume that $\mathcal{X}$ and $\mathcal{Y}$ are product spaces of the form $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$ and $\mathcal{Y} = \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_p$ for some integers n, p. For any $x \in \mathcal{X}$, we use the notation $x = (x^{(1)}, \ldots, x^{(n)})$ to represent the (block of) coordinates of x (similarly for $y = (y^{(1)}, \ldots, y^{(p)})$ in $\mathcal{Y}$). Under the standard qualification condition $0 \in \mathrm{ri}(M\mathrm{dom}g - \mathrm{dom}h)$ (where dom and ri stand for domain and relative interior, respectively), a point $x \in \mathcal{X}$ is a minimizer of (1) if and only if there exists $y \in \mathcal{Y}$ such that (x, y) is a saddle point of the Lagrangian function

$$L(x, y) = f(x) + g(x) + \langle y, Mx \rangle - h^\star(y)$$

where $\langle \, . \, , . \, \rangle$ is the inner product and $h^\star : y \mapsto \sup_{z \in \mathcal{Y}} \langle y, z \rangle - h(z)$ is the Fenchel-Legendre transform of h.

Vũ [7] and Condat [8] separately proposed a primal-dual algorithm allowing to handle f explicitly, and requiring one evaluation of the gradient of f at each iteration. The algorithm can be also be casted into a more general formalism [9], [10].

III. Applications

A. Distributed Optimization

Consider a set of $n > 1$ computing agents that cooperate to solve the minimization problem

$$\inf_{u \in \mathbb{R}^d} \sum_{i=1}^{n} (f_i(u) + g_i(u)) \tag{2}$$

where $d \geq 1$ is an integer and where f_i, g_i are two private cost functions available at Agent i. Typically, f_i represents a differentiable data fitting term corresponding to the data locally accessible at node i, while g_i represents a regularization term. Here, the purpose is to design a distributed iterative algorithm where at a each iteration, each active agent updates a local estimate in the parameter space $\mathcal{X}$ based on the sole knowledge of this agent's private cost functions and on an information it received from its neighbors through some communication network. Eventually, the local estimates will converge to a common value (or consensus) which is a minimizer (assumed to exist) of the aggregate cost function of Problem (2). In practice, it is assumed that the agents are able to exchange information over a certain communication graph $G = (V, E)$ where $V = \{1, \ldots, n\}$ represents the set of agents and where E is a set of undirected edges such that $\{i, j\} \in E$ if and only if the agents i and j are likely to communicate. As noted by [6], the problem (2) is an instance of the general problem (1) by setting $\mathcal{X} = (\mathbb{R}^d)^{\otimes N}$ and for every $x \in \mathcal{X}$, $f(x) = \sum_{i=1}^n f_i(x^{(i)})$, $g(x) = \sum_{i=1}^n g_i(x^{(i)})$ and finally $h(Mx)$ is chosen as an indicator function equal to zero if $x^{(1)} = \cdots = x^{(n)}$ and to $+\infty$ otherwise. We briefly recall the specific choice h and M leading to a distributed algorithm (see [6] for more details).

The idea is to ensure consensus separately over all the edges of the graph. This way, the constraints are localized at the edge level, but are equivalent to global consensus as soon as G is connected. Mathematically, for any $e = \{i, j\} \in E$, let M_e be the linear operator from $\mathcal{X}$ to $\mathbb{R}^d \times \mathbb{R}^d$ such that $M_e x = (x^{(i)}, x^{(j)})$ for any $x \in \mathcal{X}$ (here we assume some implicit ordering of the nodes). Define M as the linear operator generated by stacking vertically the $(M_e)_{e \in E}$, that is $M : \mathcal{X} \to \mathcal{Y}$ where $\mathcal{Y} = (\mathbb{R}^d \times \mathbb{R}^d)^{|E|}$. For every $y \in \mathcal{Y}$, decompose y as $(y_e)_{e \in E}$ where each $y_e \in \mathbb{R}^d \times \mathbb{R}^d$ is a pair of vectors of $\mathbb{R}^d$. Denote by $h(y)$ the function equal to zero if every pair y_e has its two components equal, and to zero otherwise. As made clear in [6], the quantity $h(Mx)$ is equal to zero if x has all its components equal and to $+\infty$ otherwise, as soon as G is connected. Hence, the problem (2) can be casted into the form (1).

By applying the Vũ-Condat algorithm on that problem, [6] obtains a distributed proximal gradient algorithm, which they call DADMM+. Applying on the top of that the idea of coordinate descent, they may choose to update at each iteration only the subset of coordinates which are physically handled by a sole agent, chosen at random. This approach has the advantage of making the algorithm *asynchronous*: at each time n, an agent picked at random wakes up, updates its own variables and sends the result of its computation to its neighbors in the graph.

The drawback of the algorithm lies in its convergence conditions. The conditions on the step size are related to the global Lipschitz constant of the gradient of $f(x) = \sum_{i=1}^n f_i(x^{(i)})$ and the minimum node-degree in the graph. First, these constants are by definition unknown (recall that each function f_i is only known by Agent i). Second, the step size are limited by the Agent having the *worst* local Lipschitz constant, and by the *smallest* node degree. In this paper, we prove that these convergence conditions are too conservative, and establish that each Agent might select its local step size as a function of its own local Lipschitz constant and its own degree.

B. Support Vector Machines

We consider a set of n observations gathered into a data matrix $A \in \mathbb{R}^{m \times n}$ and labels $b \in \mathbb{R}^n$ and we intend to solve the following Support Vector Machine (SVM) problem:

$$\min_{w \in \mathbb{R}^m, w_0 \in \mathbb{R}} \sum_{i=1}^{n} C_i \max\left(0, 1 - b_i((A^\top w)_i + w_0)\right) + \frac{\lambda}{2} \|w\|_2^2$$

where $C_1, \ldots, C_n$ are positive weights typically allowing to cope with potentially unbalanced classes. As is common practice for this problem, we solve instead the Dual Support Vector Machine problem:

$$\max_{x \in \mathbb{R}^n} -\frac{1}{2\lambda} \|AD(b)x\|_2^2 + e^T x - \sum_{i=1}^{n} I_{[0, C_i]}(x_i) - I_{b^\perp}(x)$$

where $D(b)$ is the diagonal matrix containing the labels, e is the vector whose components are all equal to one, I_S is the indicator function of a set S (equal to zero on that set, to $+\infty$ outside) and $b^\perp$ is the orthogonal space to the linear span of the vector b. Some authors proposed to fix the

bias w_0 to 0 in order to make the problem easier to solve [11]. Here, we intend to solve the initial problem with a non-zero bias. Setting $f(x) = \frac{1}{2}\|AD(b)x\|_2^2 - e^T x$, $g(x) = \sum_{i=1}^n I_{[0,C_i]}(x_i)$, $h(y) = I_{b^\perp}(y)$ and $M = I$, the problem falls again into the formulation (1).

Other application examples are provided in [12].

IV. Main Algorithm

A. Notation

We note $M = (M_{j,i} : i \in \{1,\dots,n\}, j \in \{1,\dots,p\})$ where $M_{j,i} : \mathcal{X}_i \to \mathcal{Y}_j$ are the block components of M. For each $j \in \{1,\dots,p\}$, we introduce the set

$$I(j) := \Big\{ i \in \{1,\dots,n\} \;:\; M_{j,i} \neq 0 \Big\}.$$

Otherwise stated, the jth component of vector Mx only depends on x through the coordinates $x^{(i)}$ such that $i \in I(j)$. We denote by

$$m_j := \mathrm{card}(I(j))$$

the number of such coordinates. Without loss of generality, we assume that $m_j \neq 0$ for all j. For all $i \in \{1,\dots,n\}$, we define

$$J(i) := \Big\{ j \in \{1,\dots,p\} \;:\; M_{j,i} \neq 0 \Big\}.$$

Note that for every pair (i,j), the statements $i \in I(j)$ and $j \in J(i)$ are equivalent.

Recall that $\mathcal{Y} = \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_p$. For every $j \in \{1,\dots,p\}$, we use the notation $\boldsymbol{\mathcal{Y}}_j := \mathcal{Y}_j^{I(j)}$. An arbitrary element $\boldsymbol{u}$ in $\boldsymbol{\mathcal{Y}}_j$ will be represented by $\boldsymbol{u} = (\boldsymbol{u}(i) \;:\; i \in I(j))$. We define $\boldsymbol{\mathcal{Y}} := \boldsymbol{\mathcal{Y}}_1 \times \cdots \times \boldsymbol{\mathcal{Y}}_p$. An arbitrary element $\boldsymbol{y}$ in $\boldsymbol{\mathcal{Y}}$ will be represented as $\boldsymbol{y} = (\boldsymbol{y}^{(1)}, \dots, \boldsymbol{y}^{(p)})$. This notation is recalled in Table I below.

TABLE I

Standing notation.

Space	Element
$\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$	$x = (x^{(i)} : i \in \{1,\dots,n\})$
$\mathcal{Y} = \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_p$	$y = (y^{(j)} : j \in \{1,\dots,p\})$
$\boldsymbol{\mathcal{Y}}_j = \mathcal{Y}_j^{I(j)}$	$\boldsymbol{u} = (\boldsymbol{u}(i) : i \in I(j))$
$\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{Y}}_1 \times \cdots \times \boldsymbol{\mathcal{Y}}_p$	$\boldsymbol{y} = (\boldsymbol{y}^{(j)} : j \in \{1,\dots,p\})$ where $\boldsymbol{y}^{(j)} = (\boldsymbol{y}^{(j)}(i) : i \in I(j))\ \forall j$

If ℓ is an integer, $\gamma = (\gamma_1,\dots,\gamma_\ell)$ is a collection of positive real numbers and $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_\ell$ is a product of Euclidean spaces, we introduce the weighted norm $\|\,.\,\|_\gamma$ on $\mathcal{A}$ given by $\|u\|_\gamma^2 = \sum_{i=1}^\ell \gamma_i \|u^{(i)}\|_{\mathcal{A}_i}^2$ for every $u = (u^{(1)},\dots,u^{(\ell)})$ where $\|\,.\,\|_{\mathcal{A}_i}$ stand for the norm on $\mathcal{A}_i$. If $F : \mathcal{A} \to]-\infty,+\infty]$ denotes a convex proper lower-semicontinuous function, we introduce the proximity operator $\mathrm{prox}_{\gamma,F} : \mathcal{A} \to \mathcal{A}$ defined for any $u \in \mathcal{A}$ by

$$\mathrm{prox}_{\gamma,F}(u) := \arg\min_{w \in \mathcal{A}} \big[F(w) + \frac{1}{2}\|w-u\|_{\gamma^{-1}}^2 \big]$$

where we use the notation $\gamma^{-1} = (\gamma_1^{-1},\dots,\gamma_\ell^{-1})$. We denote by $\mathrm{prox}_{\gamma,F}^{(i)} : \mathcal{A} \to \mathcal{A}_i$ the ith coordinate mapping of $\mathrm{prox}_{\gamma,F}$ that is, $\mathrm{prox}_{\gamma,F}(u) = (\mathrm{prox}_{\gamma,F}^{(1)}(u),\dots,\mathrm{prox}_{\gamma,F}^{(\ell)}(u))$ for any $u \in \mathcal{A}$. The notation $D_{\mathcal{A}}(\gamma)$ (or simply $D(\gamma)$ when no ambiguity occurs) stands for the diagonal operator on $\mathcal{A} \to \mathcal{A}$ given by $D_{\mathcal{A}}(\gamma)(u) = (\gamma_1 u^{(1)},\dots,\gamma_\ell u^{(\ell)})$ for every $u = (u^{(1)},\dots,u^{(\ell)})$.

Finally, the adjoint of a linear operator B is denoted $B^\star$. The spectral radius of a square matrix A is denoted by $\rho(A)$.

B. Algorithm

Consider Problem (1). Let $\sigma = (\sigma_1,\dots,\sigma_p)$ and $\tau = (\tau_1,\dots,\tau_n)$ be two tuples of positive real numbers. Consider an independent and identically distributed sequence $(i_k : k \in \mathbb{N}^*)$ with uniform distribution on $\{1,\dots,n\}$. The proposed primal-dual CD algorithm consists in updating four sequences $x_k \in \mathcal{X}$, $w_k \in \mathcal{X}$, $z_k \in \mathcal{Y}$ and $\boldsymbol{y}_k \in \boldsymbol{\mathcal{Y}}$. It is provided in Algorithm 1 below.

Algorithm 1 Coordinate-descent primal-dual algorithm

Initialization: Choose $x_0 \in \mathcal{X}$, $\boldsymbol{y}_0 \in \boldsymbol{\mathcal{Y}}$.
For all $i \in \{1,\dots,n\}$, set $w_0^{(i)} = \sum_{j \in J(i)} M_{j,i}^\star \boldsymbol{y}_0^{(j)}(i)$.
For all $j \in \{1,\dots,p\}$, set $z_0^{(j)} = \frac{1}{m_j}\sum_{i \in I(j)} \boldsymbol{y}_0^{(j)}(i)$.
Iteration k: Define:

$$\overline{y}_{k+1} = \mathrm{prox}_{\sigma,h^\star}\left(z_k + D(\sigma)Mx_k\right)$$
$$\overline{x}_{k+1} = \mathrm{prox}_{\tau,g}\Big(x_k - D(\tau)\left(\nabla f(x_k) + 2M^\star \overline{y}_{k+1} - w_k\right)\Big).$$

For $i = i_{k+1}$ and for each $j \in J(i_{k+1})$, update:

$$x_{k+1}^{(i)} = \overline{x}_{k+1}^{(i)}$$
$$\boldsymbol{y}_{k+1}^{(j)}(i) = \overline{y}_{k+1}^{(j)}$$
$$w_{k+1}^{(i)} = w_k^{(i)} + \sum_{j \in J(i)} M_{j,i}^\star \left(\boldsymbol{y}_{k+1}^{(j)}(i) - \boldsymbol{y}_k^{(j)}(i)\right)$$
$$z_{k+1}^{(j)} = z_k^{(j)} + \frac{1}{m_j}\left(\boldsymbol{y}_{k+1}^{(j)}(i) - \boldsymbol{y}_k^{(j)}(i)\right).$$

Otherwise, set $x_{k+1}^{(i)} = x_k^{(i)}$, $w_{k+1}^{(i)} = w_k^{(i)}$, $z_{k+1}^{(j)} = z_k^{(j)}$ and $\boldsymbol{y}_{k+1}^{(j)}(i) = \boldsymbol{y}_k^{(j)}(i)$.

Remark. In Algorithm 1, it is worth noting that quantities $(\overline{x}_{k+1}, \overline{y}_{k+1})$ do not need to be explicitly calculated. At iteration k, only the coordinates

$$\overline{x}_{k+1}^{(i_{k+1})} \text{ and } \overline{y}_{k+1}^{(j)}, \quad \forall j \in J(i_{k+1})$$

are needed to perform the update. When g is separable, it can be easily checked that other coordinates do not need to be computed. From a computational point of view, it is often the case that the evaluation of the above coordinates is less demanding than the computation of the whole vectors $\overline{x}_{k+1}$, $\overline{y}_{k+1}$. Practical examples are provided in Section VI.

For every $i \in \{1,\dots,n\}$, we denote by $U_i : \mathcal{X}_i \to \mathcal{X}$ the linear operator such that all coordinates of $U_i(u)$ are zero except the ith coordinate which coincides with u: $U_i(u) = (0,\cdots,0,u,0,\cdots,0)$. Our convergence result holds under the following assumptions.

Assumption 4.1: a) The functions f, g, h are closed proper and convex.

b) The function f is differentiable on $\mathcal{X}$.

c) For every $i \in \{1,\dots,n\}$, there exists $\beta_i \geq 0$ such that for any $x \in \mathcal{X}$, any $u \in \mathcal{X}_i$,

$$f(x+U_i u) \leq f(x) + \langle \nabla f(x), U_i u\rangle + \frac{\beta_i}{2}\|u\|^2_{\mathcal{X}_i}\,.$$

d) The random sequence $(i_k)_{k\in\mathbb{N}^*}$ is independent with uniform distribution on $\{1,\dots,n\}$.

e) For every $i \in \{1,\dots,n\}$,

$$\tau_i < \frac{1}{\beta_i + \rho\left(\sum_{j\in J(i)} m_j \sigma_j M^\star_{j,i} M_{j,i}\right)}\,.$$

We denote by $\mathcal{S}$ the set of saddle points of the Lagrangian function L. Otherwise stated, a couple $(x_*, y_*) \in \mathcal{X}\times\mathcal{Y}$ lies in $\mathcal{S}$ if and only if it satisfies the following inclusions

$$0 \in \nabla f(x_*) + \partial g(x_*) + M^\star y_* \tag{3}$$
$$0 \in -Mx_* + \partial h^\star(y_*)\,. \tag{4}$$

We shall also refer to elements of $\mathcal{S}$ as primal-dual solutions.

Theorem 4.2: Let Assumption 4.1 hold true and suppose that $\mathcal{S} \neq \emptyset$. Let $(x_k, \boldsymbol{y}_k)$ be a sequence generated by Algorithm 1. Almost surely, there exists $(x_*, y_*) \in \mathcal{S}$ s.t.

$$\lim_{k\to\infty} x_k = x_*$$
$$\lim_{k\to\infty} \boldsymbol{y}_k^{(j)}(i) = y_*^{(j)} \qquad (\forall j \in \{1,\dots,p\},\ \forall i \in I(j))\,.$$

The proof of the following Theorem is found in [12]. It is worth noting that, under the stated assumption on the step-size, the stochastic Fejér monotonicity of the sequence of iterates, which is the key idea in [4], [5], [6], does not hold (a counter-example is provided in [12]). Our proof is different and relies on the introduction of an adequate Lyapunov function.

Remark. A quite similar algorithm was proposed by [13], unfortunately we have not been able to understand the convergence proof. Moreover, the algorithm of [13] uses small step sizes which yields potentially slow convergence phenomena as discussed in [12].

C. Special Cases

1) The Case $m_1 = \cdots = m_p = 1$: We consider the special case $m_1 = \cdots = m_p = 1$. Otherwise stated, the linear operator M has a single nonzero component $M_{j,i}$ per row $j \in \{1,\dots,p\}$.

For each $j \in \{1,\dots,p\}$, the vector $\boldsymbol{y}_k^{(j)}$ is reduced to a single value $\boldsymbol{y}_k^{(j)}(i) \in \mathcal{Y}_j$ where i is the unique index such that $M_{j,i} \neq 0$. We simply denote this value by $y_k^{(j)}$. Algorithm 1 simplifies to Algorithm 2 below.

2) The Case $h = 0$: Instanciating Algorithm 1 in the special case $h = 0$, it boils down to the following CD forward-backward algorithm:

$$x_{k+1}^{(i)} = \begin{cases} \mathrm{prox}^{(i)}_{\tau,g}\left(x_k - D(\tau)\nabla f(x_k)\right), & \text{if } i = i_{k+1},\\ x_k^{(i)}, & \text{otherwise.}\end{cases} \tag{5}$$

Algorithm 2 Coordinate-descent primal-dual algorithm - Case $m_1 = \cdots = m_p = 1$.

Initialization: Choose $x_0 \in \mathcal{X}$, $y_0 \in \mathcal{Y}$.

Iteration k: Define:

$$\overline{y}_{k+1} = \mathrm{prox}_{\sigma,h^\star}\left(y_k + D(\sigma)Mx_k\right)$$
$$\overline{x}_{k+1} = \mathrm{prox}_{\tau,g}\left(x_k - D(\tau)\big(\nabla f(x_k) + M^\star(2\overline{y}_{k+1} - y_k)\big)\right).$$

For $i = i_{k+1}$ and for each $j \in J(i_{k+1})$, update:

$$x_{k+1}^{(i)} = \overline{x}_{k+1}^{(i)}$$
$$y_{k+1}^{(j)} = \overline{y}_{k+1}^{(j)}\,.$$

Otherwise, set $x_{k+1}^{(i)} = x_k^{(i)}$,$y_{k+1}^{(j)} = y_k^{(j)}$.

As a consequence, Algorithm 1 allows to recover the CD proximal gradient algorithm of [2] with the notable difference that we do *not* assume the separability of g. On the other hand, Assumption 4.1(e) becomes $\tau_i < 1/\beta_i$ whereas in the separable case, [2] assumes $\tau_i = 1/\beta_i$.

V. Application to Distributed Optimization

We apply our algorithm to the special instance described in Section III-A. Here our approach is identical to the one of [6] (the difference with [6] lies in the convergence condition). We will therefore skip the technical details behind the instanciation of the random coordinate descent and directly provide the algorithm, we refer the interested reader to [12].

The Distributed Asynchronous Primal Dual Algorithm (DAPD) method is described in Algorithm 3. The notation $\mathcal{N}_i$ stands for the neighborhood of a node i in the graph and d_i is the degree of node i.

Algorithm 3 DAPD.

Initialization: Each node $i \in \{1,\dots,n\}$ chooses $x_0^{(i)}$ and $(\lambda_0^{(i,j)} : j \in \mathcal{N}_i)$

Iteration k: A node $i = i_{k+1}$ wakes up uniformly at random

$$\lambda_{k+1}^{(i,j)} = \frac{\lambda_k^{(i,j)} - \lambda_k^{(j,i)}}{2} + \frac{x_k^{(i)} - x_k^{(j)}}{2\rho}$$
$$x_{k+1}^{(i)} = \mathrm{prox}_{\tau_i g_i/d_i}\left(x_k^{(i)} - \frac{\tau_i}{d_i}\nabla f_i(x_k^{(i)}) + \tau_i \Delta_k^{(i)}\right)$$

where $\Delta_k^{(i)} = d_i^{-1}\sum_{j\in\mathcal{N}_i}(\lambda_k^{(j,i)} + \rho^{-1}(x_k^{(j)} - x_k^{(i)}))$.
Node i communicates $x_{k+1}^{(i)}$ and $\lambda_{k+1}^{(i,j)}$ to each neighbor j
All nodes $\ell \neq i$ do not modify their iterates $x_{k+1}^{(\ell)} = x_k^{(\ell)}$ and $\lambda_{k+1}^{(\ell,j)} = \lambda_k^{(\ell,j)}$ for all $j \in \mathcal{N}_\ell$.

Theorem 5.1: Assume that G is connected and that (2) has a minimizer. Assume that the sequence of active nodes $(i_k)_{k\in\mathbb{N}^*}$ is iid and uniformly distributed. Assume that for every $i = 1,\dots,n$, ∇f_i is L_i-Lipschitz continuous and

$$\tau_i{}^{-1} - \rho^{-1} > \frac{L_i}{d_i} \tag{6}$$

where d_i is the degree of node i. Then, almost surely and for any initial value, there exists a minimizer $x^\star$ of (2) such that for every $i = 1, \dots, n$, the sequence $x_k^{(i)}$ generated by the DAPD converges almost surely to a $x^\star$.

We now compare this result with [6]. In [6], the same step $\tau_1 = \cdots = \tau_n = \tau$ was used for all Agents and the convergence condition was $\tau^{-1} - \rho^{-1} > \frac{\max_i L_i}{2 \min_i d_i}$. Apart from the factor 2, the condition (6) on the step size is generally milder and allows for larger step sizes. In addition, the conditions only involve parameters that are local.

VI. Numerical Experiment

We used one processor of a computer with Intel Xeon CPUs at 2.80GHz. In the experiment[1], we consider the SVM problem discussed in Section III-B. We consider the the RCV1 dataset where A is a sparse $m \times n$ matrix with m = 20,242, n = 47,236 and 0.157 % of nonzero entries and we take $C_i = \frac{1}{n}$ for all i and $\lambda = \frac{1}{4n}$. For this dataset, $\|A\|^2 \approx 40 \max_i \|Ae_i\|^2$, which means that using small step sizes would lead to a roughly 40 times slower algorithm. This situation is not uncommon and is one of the reasons why coordinate descent methods are attractive. We compare our method with

- SCDA [11]: note that SDCA simply forgets $I_{b^\perp}(x)$ in order to be able to apply the classical coordinate descent method a thus will not converge to an optimal solution.
- RCD [14]: at each iteration, the algorithm selects two coordinates randomly and performs a coordinate descent step according to these two variables. Updating two variables at a times allows us to satisfy the linear constraint at each iteration.

We can see on Figure 1 the decrease of the SVM duality gap for each algorithm. SDCA is very efficient in the beginning and converges quickly. However, as the method does not take into account the intercept, it does not converge to the optimal solution and stagnates after a few passes on the data. Algorithm 1 allows step sizes nearly as long as SDCA's and taking into account the coupling constraint represents only marginal additional work. Hence, the objective value decreases nearly as fast for SDCA in the beginning without sacrificing the intercept, leading to a smaller objective value in the end. The RCD method of [14] does work but is not competitive in terms of speed of convergence. We also tried the C implementation of LIBSVM but it needed 175s to solve the (medium-size) RCV1 problem.

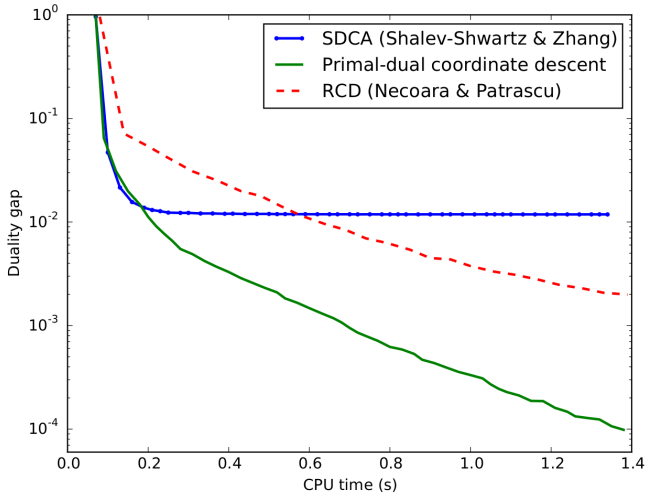

Fig. 1. Comparison of dual algorithms for the resolution of linear SVM on the RCV1 dataset. We report the value of the duality gap after a post-processing to recover feasible primal and dual variables. Primal variables are recovered as suggested in [11] and the intercept is recovered by exact minimization of the primal objective given the other primal variables. When dual iterates are not feasible, we project them onto the dual feasible set before computing the dual objective. We stopped each algorithm after 100 passes through the data.

References

[1] Yurii Nesterov, "Efficiency of coordinate descent methods on huge-scale optimization problems," *SIAM Journal on Optimization*, vol. 22, no. 2, pp. 341–362, 2012.

[2] Peter Richtárik and Martin Takáč, "Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function," *Mathematical Programming*, vol. 144, no. 1-2, pp. 1–38, 2014.

[3] Yuchen Zhang and Lin Xiao, "Stochastic primal-dual coordinate method for regularized empirical risk minimization," *arXiv preprint arXiv:1409.3257*, 2014.

[4] Franck Iutzeler, Pascal Bianchi, Philippe Ciblat, and Walid Hachem, "Asynchronous distributed optimization using a randomized alternating direction method of multipliers," in *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*. IEEE, 2013, pp. 3671–3676.

[5] Patrick L Combettes and Jean-Christophe Pesquet, "Stochastic quasi-Fejér block-coordinate fixed point iterations with random sweeping," *SIAM Journal on Optimization*, vol. 25, no. 2, pp. 1221–1248, 2015.

[6] Pascal Bianchi, Walid Hachem, and Franck Iutzeler, "A stochastic coordinate descent primal-dual algorithm and applications to large-scale composite optimization," *arXiv preprint arXiv:1407.0898*, 2014.

[7] Bang Công Vũ, "A splitting algorithm for dual monotone inclusions involving cocoercive operators," *Advances in Computational Mathematics*, vol. 38, no. 3, pp. 667–681, 2013.

[8] Laurent Condat, "A primal–dual splitting method for convex optimization involving Lipschitzian, proximable and linear composite terms," *Journal of Optimization Theory and Applications*, vol. 158, no. 2, pp. 460–479, 2013.

[9] Damek Davis and Wotao Yin, "A three-operator splitting scheme and its optimization applications," *arXiv preprint arXiv:1504.01032*, 2015.

[10] Puya Latafat and Panagiotis Patrinos, "Asymmetric forward-backward-adjoint splitting for solving monotone inclusions involving three operators," *arXiv preprint arXiv:1602.08729*, 2016.

[11] Shai Shalev-Shwartz and Tong Zhang, "Stochastic dual coordinate ascent methods for regularized loss minimization," *Journal of Machine Learning Research*, vol. 14, pp. 567–599, 2013.

[12] Olivier Fercoq and Pascal Bianchi, "A coordinate descent primal-dual algorithm with large step size and possibly non separable functions," *arXiv preprint arXiv:1508.04625*, 2015.

[13] Jean-Christophe Pesquet and Audrey Repetti, "A class of randomized primal-dual algorithms for distributed optimization," *Journal of Nonlinear Convex Analysis*, vol. 16, no. 12, 2015.

[14] Ion Necoara and Andrei Patrascu, "A random coordinate descent algorithm for optimization problems with composite objective function and linear coupled constraints," Tech. Rep., Politehnica University of Bucharest, 2012.

[1]Code available on `https://github.com/ofercoq/lightning`