



## Distributed optimization with arbitrary local solvers

Chenxin Ma<sup>a</sup>, Jakub Konečný<sup>b</sup>, Martin Jaggi<sup>c</sup>, Virginia Smith<sup>d</sup>, Michael I. Jordan<sup>d</sup>,  
Peter Richtárik<sup>b</sup> and Martin Takáč<sup>a\*</sup>

<sup>a</sup>Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA, USA; <sup>b</sup>School of Mathematics, University of Edinburgh, Old College, Edinburgh, UK; <sup>c</sup>School of Computer And Communication Sciences, EPFL, Lausanne, Switzerland; <sup>d</sup>Division of Computer Science, UC Berkeley, Berkeley, CA, USA

(Received 8 December 2015; accepted 30 December 2016)

With the growth of data and necessity for distributed optimization methods, solvers that work well on a single machine must be re-designed to leverage distributed computation. Recent work in this area has been limited by focusing heavily on developing highly specific methods for the distributed environment. These special-purpose methods are often unable to fully leverage the competitive performance of their well-tuned and customized single machine counterparts. Further, they are unable to easily integrate improvements that continue to be made to single machine methods. To this end, we present a framework for distributed optimization that both allows the flexibility of arbitrary solvers to be used on each (single) machine locally and yet maintains competitive performance against other state-of-the-art special-purpose distributed methods. We give strong primal–dual convergence rate guarantees for our framework that hold for arbitrary local solvers. We demonstrate the impact of local solver selection both theoretically and in an extensive experimental comparison. Finally, we provide thorough implementation details for our framework, highlighting areas for practical performance gains.

**Keywords:** primal-dual algorithm; distributed computing; machine learning; convergence analysis

2010 Mathematics Subject Classification: 68W15; 68W20; 68W10; 68W40

### 1. Motivation

Regression and classification techniques, represented in the general class of regularized loss minimization problems [71], are among the most central tools in modern big data analysis, machine learning, and signal processing. For these tasks, much effort from both industry and academia has gone into the development of highly tuned and customized solvers. However, with the massive growth of available datasets, major roadblocks still persist in the distributed setting, where data no longer fit in the memory of a single computer, and computation must be split across multiple machines in a network [3,7,12,18,22,29,32,34,37,46,52,62,64,67,78].

On typical real-world systems, communicating data between machines is several orders of magnitude slower than reading data from main memory, e.g. when leveraging commodity hardware. Therefore when trying to translate existing highly tuned single machine solvers to the

---

\*Corresponding author. Email: [takac.mt@gmail.com](mailto:takac.mt@gmail.com)

distributed setting, great care must be taken to avoid this significant communication bottleneck [26,74].

While several distributed solvers for the problems of interest have been recently developed, they are often unable to fully leverage the competitive performance of their tuned and customized single machine counterparts, which have already received much more research attention. More importantly, it is unfortunate that distributed solvers cannot automatically benefit from improvements made to the single machine solvers, and therefore are forced to lag behind the most recent developments.

In this paper, we make a step towards resolving these issues by proposing a general communication-efficient distributed framework that can employ arbitrary single machine local solvers and thus directly leverage their benefits and problem-specific improvements. Our framework works in rounds, where in each round the local solvers on each machine find a (possibly weak) solution to a specified subproblem of the same structure as the original master problem. On completion of each round, the partial updates between the machines are efficiently combined by leveraging the primal–dual structure of the global problem [26,35,74]. The framework therefore completely decouples the local solvers from the distributed communication. Through this decoupling, it is possible to balance communication and computation in the distributed setting, by controlling the desired accuracy and thus computational effort spent to determine the solution to each local subproblem. Our framework holds with this abstraction even if the user wishes to use a different local solver on each machine.

## 1.1 Contributions

*Reusability of existing local solvers.* The proposed framework allows for distributed optimization with the use of arbitrary local solvers on each machine. This abstraction makes the resulting framework highly flexible, and means that it can easily leverage the benefits of well-studied, problem-specific single machine solvers. In addition to increased flexibility and ease-of-use, this can result in large performance gains, as single machine solvers for the problems of interest have typically been thoroughly tuned for optimal performance. Moreover, any performance improvements that are made to these local solvers can be automatically translated by the framework into the distributed setting.

*Adaptivity to communication cost.* On real-world compute systems, the cost of communication versus computation typically varies by many orders of magnitude, from high-performance computing environments to very slow disk-based distributed workflow systems such as MapReduce/Hadoop. For optimization algorithms, it is thus essential to accommodate varying amounts of work performed locally per round, while still providing convergence guarantees. Our framework provides exactly such control.

*Strong theoretical guarantees.* In this paper, we extend and improve upon the CoCoA [26] method. Our theoretical convergence rates apply to both smooth and non-smooth losses, and for both CoCoA and CoCoA<sup>\*</sup>, the more general framework presented here. Our new rates exhibit favourable *strong scaling* properties for the class of problems considered, as the number of machines  $K$  increases and the data size is kept fixed. More precisely, while the convergence rate of CoCoA degrades as  $K$  is increased, the stronger theoretical convergence rate here is—in the worst case complexity—*independent* of  $K$ . As only one vector is communicated per round and worker, this favourable scaling might be surprising. Indeed, for existing methods, splitting data among more machines generally increase communication requirements [1,62], which can severely affect overall runtime.

*Primal–dual convergence.* We additionally strengthen the rates by showing stronger primal–dual convergence for both algorithmic frameworks, which are almost tight to their dual-only (or primal-only) counterparts. Primal–dual rates for CoCoA had not previously been analysed in

the general convex case. Our primal–dual rates allow efficient and practical certificates for the optimization quality, e.g. for stopping criteria.

*Experimental results.* Finally, we provide an extensive experimental comparison that highlights the impact of using various arbitrary solvers locally on each machine, with experiments on several real-world, distributed datasets. We compare the performance of CoCoA and CoCoA<sup>+</sup> across these datasets and choices of solvers, in particular illustrating the performance on a 280 GB dataset. Our code is available in an open-source C++ library, at: <https://github.com/optml/CoCoA>.

## 1.2 Outline

The rest of the paper is organized as follows. Section 2 provides context and states the problem of interest, including necessary assumptions and their consequences. In Section 3, we formulate the algorithm in detail and explain how to implement it efficiently in practice. The main theoretical results are presented in Section 4, followed by a discussion of relevant related work in Section 5. Practical experiments demonstrating the strength of the proposed framework are given in Section 6. Finally, we prove the main results in the [appendix](#), in Section A.4.

## 2. Background and problem formulation

To provide context for our framework, we first state traditional complexity measures and convergence rates for single machine algorithms, and then demonstrate that these must be adapted to more accurately represent the performance of an algorithm in the distributed setting.

When running an iterative optimization algorithm  $\mathcal{A}$  on a single machine, its performance is typically measured by the total runtime:

$$\text{TIME}(\mathcal{A}) = \mathcal{I}_{\mathcal{A}}(\epsilon) \times \mathcal{T}_{\mathcal{A}}. \quad (\text{T-A})$$

Here,  $\mathcal{T}_{\mathcal{A}}$  stands for the time it takes to perform a single iteration of algorithm  $\mathcal{A}$ , and  $\mathcal{I}_{\mathcal{A}}(\epsilon)$  is the number of iterations  $\mathcal{A}$  needs to attain an  $\epsilon$ -accurate objective.<sup>1</sup>

On a single machine, most of the state-of-the-art first-order optimization methods can achieve quick convergence in practice in terms of (T-A) by performing a large amount of relatively fast iterations. In the distributed setting, however, time to communicate between two machines can be several orders of magnitude slower than even a single iteration of such an algorithm. As a result, the overall time needed to perform this single iteration can increase significantly.

Distributed timing can therefore be more accurately illustrated using the following practical distributed efficiency model (see also [37]), where

$$\text{TIME}(\mathcal{A}) = \mathcal{I}_{\mathcal{A}}(\epsilon) \times (c + \mathcal{T}_{\mathcal{A}}). \quad (\text{T-B})$$

The extra term  $c$  is the time required to perform one round of communication.<sup>2</sup> As a result, an algorithm that performs well in the setting of (T-A) does not necessarily perform well in the distributed setting (T-B), especially when implemented in a straightforward or naïve way. In particular, if  $c \gg \mathcal{T}_{\mathcal{A}}$ , we could intuitively expect less potential for improvement from fast computation, as most of the time in the method will be spent on communication, not on actual computational effort to solve the problem. In this setting, novel optimization procedures are needed that carefully consider the amount of communication and the distribution of data across multiple machines.

One approach to this challenge is to design novel optimization algorithms from scratch, designed to be efficient in the distributed setting. This approach has one obvious practical drawback: There have been numerous highly efficient solvers developed and fine-tuned to particular problems of interest, as long as the problem fits onto a single machine. These solvers are ideal if run on a single machine, but with the growth of data and necessity of data distribution, they must be re-designed to work in modern data regimes.

Recent work [26,35,65,74–76] has attempted to address this issue by designing algorithms that reduce the communication bottleneck by allowing infrequent communication, while utilizing already existing algorithms as local sub-procedures. The presented work here builds on the promising approach of [26,74] in this direction. See Section 5 for a detailed discussion of the related literature.

The core idea in this line of work is that one can formulate a local subproblem for each individual machine, and run an arbitrary local solver dependent only on local data for a number of iterations—obtaining a partial local update. After each worker returns its partial update, a global update is formed by their aggregation.

The big advantage of this is that companies and practitioners do not have to implement new algorithms that would be suitable for the distributed setting. We provide a way for them to utilize their existing algorithms that work on a single machine, and provide a novel communication protocol on top of this.

In the original work on CoCoA [26], authors provide convergence analysis only for the case when the overall update is formed as an average of the partial updates, and note that in practice it is possible to improve performance by making a longer step in the same direction. The main contribution of this work is a more general convergence analysis of various settings, which enables us to do better than averaging. In one case, we can even sum the partial updates to obtain the overall update, which yields the best result, both in theory and practice. We will see that this can result in significant performance gains, see also [35,65].

In the analysis, we will allow local solvers of arbitrarily weak accuracy, each working on its own subproblem which is defined in a completely data-local way for each machine. The relative accuracy obtained by each local solver will be denoted by  $\Theta \in [0, 1]$ , where  $\Theta = 0$  describes an exact solution of the subproblem, and  $\Theta = 1$  means that the local subproblem objective has not improved at all, for this run of the local solver. This paradigm results in a substantial change in how we analyse efficiency in the distributed setting. The formula practitioners are interested in minimizing thus changes to become:

$$\text{TIME}(\mathcal{A}, \Theta) = \mathcal{I}(\epsilon, \Theta) \times (c + \mathcal{T}_{\mathcal{A}}(\Theta)). \quad (\text{T-C})$$

Here, the function  $\mathcal{T}_{\mathcal{A}}(\Theta)$  represents the time the local algorithm  $\mathcal{A}$  needs to obtain an accuracy of  $\Theta$  on the local subproblem. Note that the number of outer iterations  $\mathcal{I}(\epsilon, \Theta)$  is independent of choice of the inner algorithm  $\mathcal{A}$ , which will also be reflected by our convergence analysis presented in Section 4. Our convergence rates will hold for any local solver  $\mathcal{A}$  achieving local accuracy of  $\Theta$ . For strongly convex problems, the general form will be  $\mathcal{I}(\epsilon, \Theta) = \mathcal{O}(\log(1/\epsilon))/(1 - \Theta)$ . The inverse dependence on  $1 - \Theta$  suggests that there is a limit to how much we can gain by solving local subproblems to high accuracy, i.e. for  $\Theta$  close to 0. There will always be on the order of  $\log(1/\epsilon)$  outer iterations needed. Hence, excessive local accuracy should not be necessary. On the other hand, if  $\Theta \rightarrow 1$ , meaning that the cost and quality of the local solver diminishes, then the number of outer iterations  $\mathcal{I}(\epsilon, \Theta)$  will increase dramatically, which is to be expected.

To illustrate the strength of the paradigm (T-C) compared to (T-B), suppose that we run just a single iteration of gradient descent as the local solver  $\mathcal{A}$ . Within our framework, choosing this local solver would lead to a method which is equivalent to naively distributed gradient descent.<sup>3</sup>

Indeed, running gradient descent for a single iteration would attain a particular value of  $\Theta$ . Note that we typically do not set this explicitly:  $\Theta$  is implicitly chosen by the number of iterations or stopping criterion specified by the user for the local solver. There is no reason to think that the value attained by single iteration of gradient descent would be optimal. For instance, it may be the case that running gradient descent for, say, 200 iterations, instead of just one, would give substantially better results in practice, due to better communication efficiency. Considerations of this form are discussed in detail in Section 6.

In general, one would intuitively expect that the optimal choice would be to have  $\Theta$  such that  $\mathcal{T}_A(\Theta) = \mathcal{O}(1) \times c$ . In practice, however, the best strategy for any given local solver is to estimate the optimal choice by trying several values for the number of local iterations. We discuss the importance of  $\Theta$ , both theoretically and empirically, in Sections 4 and 6.

### 2.1 Problem formulation

Let the training data  $\{\mathbf{x}_i \in \mathbb{R}^d, y_i \in \mathbb{R}\}_{i=1}^n$  be the set of input–output pairs, where  $y_i$  can be real valued or from a discrete set in the case of classification problems. We will assume without loss of generality that  $\forall i: \|\mathbf{x}_i\| \leq 1$ . Many common tasks in machine learning and signal processing can be cast as the following optimization problem:

$$\min_{\mathbf{w} \in \mathbb{R}^d} \left\{ P(\mathbf{w}) := \frac{1}{n} \sum_{i=1}^n \ell_i(\mathbf{x}_i^T \mathbf{w}) + \lambda g(\mathbf{w}) \right\}, \tag{1}$$

where  $\ell_i$  is some convex loss function and  $g$  is a regularizer. Note that  $y_i$  is typically hidden in the formulation of functions  $\ell_i$ . Table 1 lists several common loss functions together with their convex conjugates  $\ell_i^*$  [61].

The dual optimization problem for formulation (1)—as a special case of Fenchel duality—can be written as follows [61,77]:

$$\max_{\boldsymbol{\alpha} \in \mathbb{R}^n} \left\{ D(\boldsymbol{\alpha}) := \frac{1}{n} \left( \sum_{i=1}^n -\ell_i^*(-\alpha_i) \right) - \lambda g^* \left( \frac{1}{\lambda n} X \boldsymbol{\alpha} \right) \right\}, \tag{2}$$

where  $X = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n] \in \mathbb{R}^{d \times n}$ , and  $\ell_i^*$  and  $g^*$  are the convex conjugate functions of  $\ell_i$  and  $g$ , respectively. The convex (Fenchel) conjugate of a function  $\phi: \mathbb{R}^k \rightarrow \mathbb{R}$  is defined as the function  $\phi^*: \mathbb{R}^k \rightarrow \mathbb{R}$ , with  $\phi^*(\mathbf{u}) := \sup_{\mathbf{s} \in \mathbb{R}^k} \{\mathbf{s}^T \mathbf{u} - \phi(\mathbf{s})\}$ .

For simplicity throughout the paper, let us denote

$$f(\boldsymbol{\alpha}) := \lambda g^* \left( \frac{1}{\lambda n} X \boldsymbol{\alpha} \right) \quad \text{and} \quad R(\boldsymbol{\alpha}) := \frac{1}{n} \sum_{i=1}^n \ell_i^*(-\alpha_i), \tag{3}$$

such that  $D(\boldsymbol{\alpha}) \stackrel{(2)+(3)}{=} -f(\boldsymbol{\alpha}) - R(\boldsymbol{\alpha})$ .

Table 1. Examples of commonly used loss functions.

Loss function	$\ell_i(a)$	$\ell_i^*(b)$	Property of $\ell$
Quadratic loss	$\frac{1}{2}(a - y_i)^2$	$\frac{1}{2}b^2 + y_i b$	Smooth
Hinge loss	$\max\{0, y_i - a\}$	$y_i b, b \in [-1, 0]$	Continuous
Squared hinge loss	$(\max\{0, y_i - a\})^2$	$\frac{b^2}{4}, b \in [-\infty, 0]$	Smooth
Logistic loss	$\log(1 + \exp(-y_i a))$	$-\frac{b}{y_i} \log\left(-\frac{b}{y_i}\right) + \left(1 + \frac{b}{y_i}\right) \log\left(1 + \frac{b}{y_i}\right)$	Smooth

It is well known [19,48,61,66] that the first-order optimality conditions give rise to a natural mapping that relates pairs of primal and dual variables. This mapping employs the linear map given by the data  $X$ , and maps any dual variable  $\alpha \in \mathbb{R}^n$  to a primal candidate vector  $\mathbf{w} \in \mathbb{R}^d$  as follows:

$$\mathbf{w}(\alpha) := \nabla g^*(\mathbf{v}(\alpha)) = \nabla g^*\left(\frac{1}{\lambda n} X \alpha\right),$$

where we denote  $\mathbf{v}(\alpha) := (1/\lambda n)X\alpha$ .

For this mapping, under the assumptions that we make in Section 2.2, it holds that if  $\alpha^*$  is an optimal solution of (2), then  $\mathbf{w}(\alpha^*)$  is an optimal solution of (1). In particular, *strong duality* holds between the primal and dual problems. If we define the duality gap function as

$$\text{Gap}(\alpha) := \mathcal{P}(\mathbf{w}(\alpha)) - D(\alpha), \quad (4)$$

then  $\text{Gap}(\alpha^*) = 0$ , which ensures that by solving the dual problem (2) we also solve the original primal problem of interest (1). As we will see later, there are many benefits to leveraging this primal–dual relationship, including the ability to use the duality gap as a certificate of solution quality, and, in the distributed setting, the ability to effectively distribute computation.

*Notation.* We assume that to solve problem (2), we have a network of  $K$  machines at our disposal. The data  $\{\mathbf{x}_i, y_i\}_{i=1}^n$  are residing on the  $K$  machines in a distributed fashion, with every machine holding a subset of the whole dataset. We distribute the dual variables in the same manner, with each dual variable  $\alpha_i$  corresponding to an individual data point  $\mathbf{x}_i$ . The given data distribution is described using a partition  $\mathcal{P}_1, \dots, \mathcal{P}_K$  that corresponds to the indices of the data and dual variables residing on machine  $k$ . Formally,  $\mathcal{P}_k \subseteq \{1, 2, \dots, n\}$  for each  $k$ ;  $\mathcal{P}_k \cap \mathcal{P}_l = \emptyset$  whenever  $k \neq l$ ; and  $\bigcup_{k=1}^K \mathcal{P}_k = \{1, 2, \dots, n\}$ .

Finally, we introduce the following notation dependent on this partitioning. For any  $\mathbf{h} \in \mathbb{R}^n$ , let  $\mathbf{h}_{[k]}$  be the vector in  $\mathbb{R}^n$  defined such that  $(\mathbf{h}_{[k]})_i = h_i$  if  $i \in \mathcal{P}_k$  and 0 otherwise. Note that, in particular,  $\mathbf{h} = \sum_{k=1}^K \mathbf{h}_{[k]}$ . Analogously, we write  $X_{[k]}$  for the matrix consisting only of the columns  $i \in \mathcal{P}_k$ , padded with zeros in all other columns.

## 2.2 Technical assumptions

Here, we first state the properties and assumptions used throughout the paper. We assume that for all  $i \in \{1, \dots, n\}$ , the function  $\ell_i$  in (1) is convex, i.e.  $\forall \lambda \in [0, 1]$  and  $\forall x, y \in \mathbb{R}$  we have  $\ell_i(\lambda x + (1 - \lambda)y) \leq \lambda \ell_i(x) + (1 - \lambda)\ell_i(y)$ .

We also assume that the function  $g$  is 1-strongly convex, i.e. for all  $\mathbf{w}, \mathbf{u} \in \mathbb{R}^d$  it holds that  $g(\mathbf{w} + \mathbf{u}) \geq g(\mathbf{w}) + \langle \nabla g(\mathbf{w}), \mathbf{u} \rangle + \frac{1}{2} \|\mathbf{u}\|^2$ , where  $\nabla g(\mathbf{w})$  is any subgradient<sup>4</sup> of the function  $g$ . Here,  $\|\cdot\|$  denotes the standard Euclidean norm.

Note that we use subgradients in the definition of strong convexity. This is due to the fact that while we will need the function  $g$  to be strongly convex in our analysis, we do not require smoothness. An example used in practice is  $g(\mathbf{w}) = \|\mathbf{w}\|^2 + \lambda' \|\mathbf{w}\|_1$  for some  $\lambda' \in \mathbb{R}$ . Also note that in the problem formulation (1) we have a regularization parameter  $\lambda$ , which controls the strong convexity parameter of the entire second term. Hence, fixing the strong convexity parameter of  $g$  to 1 is not restrictive in this regard. For instance, this setting has been used previously in [15,48,61].

The following assumptions state properties of the functions  $\ell_i$ , which we use only in certain results in the paper. We always explicitly state when we require each assumption.

ASSUMPTION 2.1 (( $1/\gamma$ )-smoothness) Functions  $\ell_i : \mathbb{R} \rightarrow \mathbb{R}$  are  $1/\gamma$ -smooth, if  $\forall i \in \{1, \dots, n\}$  and  $\forall x, h \in \mathbb{R}$  it holds that

$$\ell_i(x+h) \leq \ell_i(x) + h\nabla\ell_i(x) + \frac{1}{2\gamma}h^2, \quad (5)$$

where  $\nabla\ell_i(x)$  denotes the gradient of the function  $\ell_i$ .

ASSUMPTION 2.2 ( $L$ -Lipschitz continuity) Functions  $\ell_i : \mathbb{R} \rightarrow \mathbb{R}$  are  $L$ -Lipschitz continuous, if  $\forall i \in \{1, \dots, n\}$  and  $\forall x, h \in \mathbb{R}$  it holds that

$$|\ell_i(x+h) - \ell_i(x)| \leq L|h|. \quad (6)$$

*Remark 1* As a consequence of having  $1/\gamma$ -smoothness of  $\ell_i$  and 1-strong convexity of  $g$ , we have that the functions  $\ell_i^*(\cdot)$  are  $\gamma$ -strongly convex and  $g^*(\cdot)$  is 1-smooth [57]. These are the properties we will ultimately use as we will be solving the dual problem (2). Note that 1-smoothness of  $g^* : \mathbb{R}^d \rightarrow \mathbb{R}$  means that for all  $\mathbf{x}, \mathbf{h} \in \mathbb{R}^d$ ,

$$g^*(\mathbf{x} + \mathbf{h}) \leq g^*(\mathbf{x}) + \langle \nabla g^*(\mathbf{x}), \mathbf{h} \rangle + \frac{1}{2}\|\mathbf{h}\|^2. \quad (7)$$

The following lemma, which is a consequence of 1-smoothness of  $g^*$  and the definition of  $f$ , will be crucial in deriving a meaningful local subproblem for the proposed distributed framework.

LEMMA 2.3 Let  $f$  be defined in (3). Then for all  $\boldsymbol{\alpha}, \mathbf{h} \in \mathbb{R}^n$  we have

$$f(\boldsymbol{\alpha} + \mathbf{h}) \leq f(\boldsymbol{\alpha}) + \langle \nabla f(\boldsymbol{\alpha}), \mathbf{h} \rangle + \frac{1}{2\lambda n^2} \mathbf{h}^T X^T X \mathbf{h}. \quad (8)$$

*Remark 2* Note that although the above inequality appears as a consequence of the problem structure (2) and of the strong convexity of  $g$ , there are other ways to satisfy it. Hence, our dual analysis holds for all optimization problems of the form  $\max_{\boldsymbol{\alpha}} D(\boldsymbol{\alpha})$ , where  $D(\boldsymbol{\alpha}) = -f(\boldsymbol{\alpha}) - R(\boldsymbol{\alpha})$ , and where  $f$  satisfies inequality (8). However, for the duality gap analysis we naturally do require that the dual problem arises from the primal problem, with  $g$  being strongly convex.

### 3. The framework

In this section we start by giving a general view of the proposed framework, explaining the most important concepts needed to make the framework efficient. In Section 3.1 we discuss the formulation of the local subproblems, and in Section 3.2 we provide specific details and best practices for implementation.

The data distribution plays a crucial role in Algorithm 1, where in each outer iteration indexed by  $t$ , machine  $k$  runs an arbitrary local solver on a problem described only by the data that particular machine owns and other fixed constants or linear functions.

The crucial property is that the optimization algorithm on machine  $k$  changes only coordinates of the dual optimization variable  $\boldsymbol{\alpha}^t$  corresponding to the partition  $\mathcal{P}_k$  to obtain an approximate solution to the local subproblem. We will formally specify this in Assumption 4.1. After each such step, updates from all machines are aggregated to form a new iterate  $\boldsymbol{\alpha}^{t+1}$ . The aggregation parameter  $\nu$  will typically be between  $\nu = 1/K$ , corresponding to averaging, and  $\nu = 1$ , adding.

Here we list the core conceptual properties of Algorithm 1, which are important qualities that allow it to run efficiently.

**Algorithm 1** Improved CoCoA<sup>+</sup> Framework

---

```

1: Input: starting point  $\alpha^0 \in \mathbb{R}^n$ , aggregation parameter  $\nu \in (0, 1]$ , data partition  $\{\mathcal{P}_k\}_{k=1}^K$ 
2: for  $t = 0, 1, 2, \dots$  do
3:   for  $k \in \{1, 2, \dots, K\}$  in parallel over machines do
4:     Let  $\mathbf{h}_{[k]}^t$  be an approximate solution of the local problem (LO), i.e.
           
$$\max_{\mathbf{h}_{[k]} \in \mathbb{R}^n} \mathcal{G}_k(\mathbf{h}_{[k]}; \alpha^t)$$

5:   end for
6:   Set  $\alpha^{t+1} := \alpha^t + \nu \sum_{k=1}^K \mathbf{h}_{[k]}^t$ 
7: end for

```

---

*Locality.* The local subproblem  $\mathcal{G}_k$  (LO) is defined purely based on the data points residing on machine  $k$ , as well as a single shared vector in  $\mathbb{R}^d$  (representing the state of the  $\alpha^t$  variables of the other machines). Each local solver can then run independently and in parallel, i.e. there is no need for communication while solving the local subproblems.

*Local changes.* The optimization algorithm used to solve the local subproblem  $\mathcal{G}_k$  outputs a vector  $\mathbf{h}_{[k]}^t$  with nonzero elements only in coordinates corresponding to variables  $\alpha_{[k]}$  stored locally (i.e.  $i \in \mathcal{P}_k$ ).

*Efficient maintenance.* Given the description of the local problem  $\mathcal{G}_k(\cdot; \alpha^t)$  at time  $t$ , the new local problem  $\mathcal{G}_k(\cdot; \alpha^{t+1})$  at time  $t+1$  can be formed on each machine, requiring only communication of a single vector in  $\mathbb{R}^d$  from each machine  $k$  to the master node, and vice versa, back to each machine  $k$ .

Let us now comment on these properties in more detail. Locality is important for making the method versatile, and is the way we escape the restricted setting described by (T-B) that allows us much greater flexibility in designing the overall optimization scheme. Local changes result from the fact that we distribute coordinates of the dual variables  $\alpha$  in the same manner as the data, and thus only make updates to the coordinates stored locally. As we will see, efficient maintenance of the subproblems can be obtained. For this, a communication-efficient encoding of the current shared state  $\alpha$  is necessary. To this goal, we will in Section 3.2 show that communication of a single  $d$ -dimensional vector is enough to formulate the subproblems (LO) in each round, by carefully exploiting their partly separable structure.

Note that Algorithm 1 is the ‘analysis friendly’ formulation of our algorithm framework, and it is not yet fully illustrative for implementation purposes. In Section 3.2 we will precisely formulate the actual communication scheme, and illustrate how the above properties can be achieved.

Before that, we formulate the precise subproblem  $\mathcal{G}_k$  in the following section.

### 3.1 The local subproblems

We can define a data-local subproblem of the original dual optimization problem (2), which can be solved on machine  $k$  and only requires accessing data which is already available locally, i.e. datapoints with  $i \in \mathcal{P}_k$ . More formally, each machine  $k$  is assigned the following local subproblem, depending only on the previous shared primal vector  $\mathbf{w} \in \mathbb{R}^d$ , and the change in the local dual variables  $\alpha_i$  with  $i \in \mathcal{P}_k$ :

$$\max_{\mathbf{h}_{[k]} \in \mathbb{R}^n} \mathcal{G}_k^{\sigma'}(\mathbf{h}_{[k]}; \alpha). \quad (9)$$

We are now ready to define the local objective  $\mathcal{G}_k^{\sigma'}(\cdot; \boldsymbol{\alpha})$  as follows:

$$\mathcal{G}_k^{\sigma'}(\mathbf{h}_{[k]}; \boldsymbol{\alpha}) := -\frac{1}{K}f(\boldsymbol{\alpha}) - \langle \nabla f(\boldsymbol{\alpha}), \mathbf{h}_{[k]} \rangle - \frac{\lambda\sigma'}{2} \left\| \frac{1}{\lambda n} X_{[k]} \mathbf{h}_{[k]} \right\|^2 - R_k(\boldsymbol{\alpha}_{[k]} + \mathbf{h}_{[k]}), \quad (\text{LO})$$

where  $R_k(\boldsymbol{\alpha}_{[k]}) := (1/n) \sum_{i \in \mathcal{P}_k} \ell_i^*(-\boldsymbol{\alpha}_i)$ . The role of the parameter  $\sigma' \geq 1$  is to measure the ‘difficulty’ of the data partition, in a sense which we will discuss in detail in Section 3.3.

The interpretation of the subproblems defined above is that they will form a quadratic approximation of the smooth part of the true objective  $D$ , which becomes separable over the machines. The approximation keeps the non-smooth  $R$  part intact. The variable  $\mathbf{h}_{[k]}$  expresses the update proposed by machine  $k$ . In this spirit, note also that the approximation coincides with  $D$  at the reference point  $\boldsymbol{\alpha}$ , i.e.  $\sum_{k=1}^K \mathcal{G}_k^{\sigma'}(\mathbf{0}; \boldsymbol{\alpha}) = D(\boldsymbol{\alpha})$ . We will discuss the interpretation and properties of these subproblems in more detail in Section 3.3.

### 3.2 Practical communication-efficient implementation

We now discuss how Algorithm 1 can efficiently be implemented in a distributed environment. Most importantly, we clarify how the ‘local’ subproblems can be formulated and solved while using only local information from the corresponding machines, and we make precise what information needs to be communicated in each round.

Recall that the local subproblem objective  $\mathcal{G}_k^{\sigma'}(\cdot; \boldsymbol{\alpha})$  was defined in (LO). We will now equivalently rewrite this optimization problem, illustrating how it can be expressed using only *local* information. To do so, we use our simplifying notation  $\mathbf{v} = \mathbf{v}(\boldsymbol{\alpha}) := (1/\lambda n) X \boldsymbol{\alpha}$  for a given  $\boldsymbol{\alpha}$ . As we see in the reformulation, it is precisely this vector  $\mathbf{v} \in \mathbb{R}^d$  which contains all the necessary shared information between the machines. Given the vector  $\mathbf{v}$ , the subproblem (LO) can be equivalently written as

$$\begin{aligned} \mathcal{G}_k^{\sigma'}(\mathbf{h}_{[k]}; \mathbf{v}, \boldsymbol{\alpha}_{[k]}) &:= -\frac{\lambda}{K} g^*(\mathbf{v}) - \left\langle \frac{1}{n} X_{[k]}^T \nabla g^*(\mathbf{v}), \mathbf{h}_{[k]} \right\rangle - \frac{\lambda}{2} \sigma' \left\| \frac{1}{\lambda n} X_{[k]} \mathbf{h}_{[k]} \right\|^2 \\ &\quad - R_k(\boldsymbol{\alpha}_{[k]} + \mathbf{h}_{[k]}). \end{aligned} \quad (\text{LO}')$$

Here for the reformulation of the gradient term, we have simply used the chain rule on the objective  $f$  (recall the definition  $f(\boldsymbol{\alpha}) := \lambda g^*(\mathbf{v})$ ), giving

$$\nabla f(\boldsymbol{\alpha})_{[k]} = \frac{1}{n} X_{[k]}^T \nabla g^*(\mathbf{v}).$$

*Practical distributed framework.* In summary, we have seen that each machine can formulate the local subproblem given purely local information (the local data  $X_{[k]}$  as well as the local dual variables  $\boldsymbol{\alpha}_{[k]}$ ). No information about the data or variables  $\boldsymbol{\alpha}$  stored on the other machines is necessary.

The only requirement for the method to work is that between the rounds, the changes in the  $\boldsymbol{\alpha}_{[k]}$  variables on each machine and the resulting global change in  $\mathbf{v}$  are kept consistent, in the sense that  $\mathbf{v}^t = \mathbf{v}(\boldsymbol{\alpha}^t) := (1/\lambda n) X \boldsymbol{\alpha}^t$  must always hold. Note that for the evaluation of  $\nabla g^*(\mathbf{v})$ , the vector  $\mathbf{v}$  is all that is needed. In practice,  $g$  as well as its conjugate  $g^*$  are simple vector-valued regularization functions, the most prominent example being  $g(\mathbf{v}) = g^*(\mathbf{v}) = \frac{1}{2} \|\mathbf{v}\|^2$ .

In the following more detailed formulation of the CoCoA<sup>+</sup> framework shown in Algorithm 2 (an equivalent reformulation of Algorithm 1), the crucial communication pattern of the framework finally becomes more clear: Per round, *only a single vector* (the update on  $\mathbf{v} \in \mathbb{R}^d$ ) needs to be sent over the communication network. The reduce-all operation in line 10 means that each

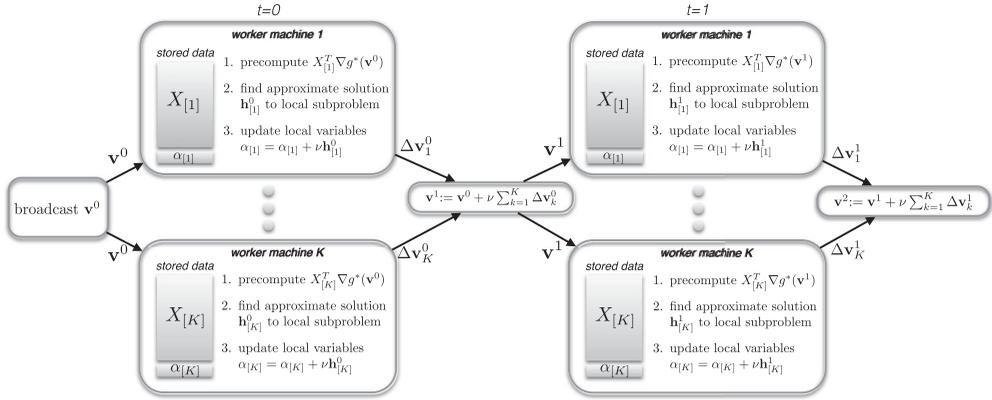


Figure 1. The first two iterations of the improved framework (practical implementation).

machine sends their vector  $\Delta \mathbf{v}_k^t \in \mathbb{R}^d$  to the network, which performs the addition operation of the  $K$  vectors to the old  $\mathbf{v}^t$ . The resulting vector  $\mathbf{v}^{t+1}$  is then communicated back to all machines, so that all have the same copy of  $\mathbf{v}^{t+1}$  before the beginning of the next round.

The framework as shown below in Algorithm 2 clearly maintains the consistency of  $\alpha^t$  and  $\mathbf{v}^t = \mathbf{v}^t(\alpha^t)$  after each round, no matter which local solver is used to approximately solve (LO'). A diagram illustrating the communication and computation involved in the first two full iterations of Algorithm 2 is given in Figure 1.

---

### Algorithm 2 Improved CoCoA<sup>+</sup> Framework, Practical Implementation

---

- 1: **Input:** starting point  $\alpha^0 \in \mathbb{R}^n$ , aggregation parameter  $\nu \in (0, 1]$ , data partition  $\{\mathcal{P}_k\}_{k=1}^K$
  - 2:  $\mathbf{v}^0 := \frac{1}{\lambda n} X \alpha^0 \in \mathbb{R}^d$
  - 3: **for**  $t = 0, 1, 2, \dots$  **do**
  - 4:     **for**  $k \in \{1, 2, \dots, K\}$  **in parallel over machines do**
  - 5:         Precompute  $X_{[k]}^T \nabla g^*(\mathbf{v}^t)$
  - 6:         Let  $\mathbf{h}_{[k]}^t$  be an approximate solution of the local problem (LO'), i.e.
 
$$\max_{\mathbf{h}_{[k]} \in \mathbb{R}^n} \mathcal{G}_k^{\sigma'}(\mathbf{h}_{[k]}; \mathbf{v}^t, \alpha_{[k]}^t) \quad \triangleright \text{computation}$$
  - 7:         Update local variables  $\alpha_{[k]}^{t+1} := \alpha_{[k]}^t + \nu \mathbf{h}_{[k]}^t$
  - 8:         Let  $\Delta \mathbf{v}_k^t := \frac{1}{\lambda n} X_{[k]} \mathbf{h}_{[k]}^t$
  - 9:     **end for**
  - 10:     **reduce all** to compute  $\mathbf{v}^{t+1} := \mathbf{v}^t + \nu \sum_{k=1}^K \Delta \mathbf{v}_k^t \quad \triangleright \text{communication}$
  - 11: **end for**
- 

### 3.3 Compatibility of the subproblems for aggregating updates

In this subsection, we shed more light on the local subproblems on each machine, as defined in (LO) above, and their interpretation. More formally, we show how the aggregation parameter  $\nu$  (controlling the level of adding versus averaging the resulting updates from each machine) and  $\sigma'$  (the subproblem parameter) interplay together, so that in each round they achieve a valid approximation to the global objective function  $D$ .

The role of the subproblem parameter  $\sigma'$  is to measure the difficulty of the given data partition. For the convergence results discussed below to hold,  $\sigma'$  must be chosen not smaller than

$$\sigma' \geq \sigma'_{\min} := \nu \cdot \max_{\mathbf{h} \in \mathbb{R}^n} \{\mathbf{h}^T X^T X \mathbf{h} \mid \mathbf{h}^T \mathbf{G} \mathbf{h} \leq 1\}. \quad (10)$$

Here,  $G$  is the block diagonal submatrix of the data covariance matrix  $X^T X$ , corresponding to the partition  $\{\mathcal{P}_k\}_{k=1}^K$ , i.e.

$$G_{ij} := \begin{cases} \mathbf{x}_i^T \mathbf{x}_j = (X^T X)_{ij} & \text{if } \exists k \text{ such that } i, j \in \mathcal{P}_k, \\ 0 & \text{otherwise.} \end{cases} \quad (11)$$

In this notation, it is easy to see that the crucial quantity defining  $\sigma'_{\min}$  above is written as  $\mathbf{h}^T \mathbf{G} \mathbf{h} = \sum_{k=1}^K \|X_{[k]} \mathbf{h}_{[k]}\|^2$ .

The following lemma shows that if the aggregation and subproblem parameters  $\nu$  and  $\sigma'$  satisfy (10), then the sum of the subproblems  $\sum_k \mathcal{G}_k^{\sigma'}$  will closely approximate the global objective function  $D$ . More precisely, this sum is a block-separable lower bound on  $D$ .

LEMMA 3.1 *Let  $\sigma' \geq 1$  and  $\nu \in [0, 1]$  satisfy (10) (that is  $\sigma' \geq \sigma'_{\min}$ ). Then  $\forall \boldsymbol{\alpha}, \mathbf{h} \in \mathbb{R}^n$ , it holds that*

$$D\left(\boldsymbol{\alpha} + \nu \sum_{k=1}^K \mathbf{h}_{[k]}\right) \geq (1 - \nu)D(\boldsymbol{\alpha}) + \nu \sum_{k=1}^K \mathcal{G}_k^{\sigma'}(\mathbf{h}_{[k]}; \boldsymbol{\alpha}), \quad (12)$$

The following lemma gives a simple choice for the subproblem parameter  $\sigma'$ , which is trivial to calculate for all values of the aggregation parameter  $\nu \in \mathbb{R}$ , and safe in the sense of the desired condition (10) above. Later we will show experimentally (Section 6) that the choice of this safe upper bound for  $\sigma'$  only has a minimal effect on the overall performance of the algorithm.

LEMMA 3.2 *For any aggregation parameter  $\nu \in [0, 1]$ , the choice of the subproblem parameter  $\sigma' := \nu K$  is valid for (10), i.e.  $\nu K \geq \sigma'_{\min}$ .*

## 4. Main results

In this section we state the main theoretical results of this paper. Before doing so, we elaborate on one of the most important aspects of the algorithmic framework: the *quality of approximate local solutions*.

### 4.1 Quality of local solutions

The notion of approximation quality provided by the local solvers is measured according to the following:

ASSUMPTION 4.1 (Quality of local solution) *Let  $\Theta \in [0, 1)$  and  $\boldsymbol{\alpha} \in \mathbb{R}^n$  be fixed, and let  $\mathbf{h}_{[k]}^*$  be the optimal solution of a local subproblem  $\mathcal{G}_k(\cdot; \boldsymbol{\alpha})$ . We assume that the local optimization procedure run on every node  $k \in [K]$  in each iteration  $t$  produces a (possibly random) output  $\mathbf{h}_{[k]}$  satisfying*

$$\mathbb{E}[\mathcal{G}_k(\mathbf{h}_{[k]}^*; \boldsymbol{\alpha}) - \mathcal{G}_k(\mathbf{h}_{[k]}; \boldsymbol{\alpha})] \leq \Theta[\mathcal{G}_k(\mathbf{h}_{[k]}^*; \boldsymbol{\alpha}) - \mathcal{G}_k(\mathbf{0}; \boldsymbol{\alpha})]. \quad (13)$$

The assumption specifies the (relative) accuracy  $\Theta$  obtained on solving the local subproblem  $\mathcal{G}_k$ . Considering the two extreme examples, setting  $\Theta = 0$  would require to find the exact maximum, while  $\Theta = 1$  states that no improvement was achieved at all by the local solver. Intuitively, we would prefer  $\Theta$  to be small, but spending many computational resources to drive  $\Theta$  to 0 can be excessive in practice, since  $\mathcal{G}_k$  is actually not the problem we are interested in solving (2), but is the problem to be solved per communication round. The best choice in practice will therefore be to choose  $\Theta$  such that the local solver runs for a time comparable to the time it takes for a single communication round. This freedom of choice of  $\Theta \in [0, 1]$  is a crucial property of our proposed framework, allowing it to adapt to the full range of communication speeds on real-world systems, ranging from supercomputers on one extreme to very slow communication rounds like MapReduce systems on the other extreme.

In Section 6 we study the impact of different values of this parameter on the overall performance of solving (2).

## 4.2 Complexity bounds

Now we are ready to state the main results. Theorem 4.2 covers the case when  $\forall i$ , the loss function  $\ell_i$  is  $1/\gamma$  smooth, and Theorem 4.3 covers the case when  $\ell_i$  is  $L$ -Lipschitz continuous. For simplicity in the rates, we define the following two quantities:

$$\forall k : \sigma_k := \max_{\alpha_{[k]} \in \mathbb{R}^n} \frac{\|X_{[k]} \alpha_{[k]}\|^2}{\|\alpha_{[k]}\|^2} \quad \text{and} \quad \sigma := \sum_{k=1}^K \sigma_k |\mathcal{P}_k|.$$

**THEOREM 4.2 (Smooth loss functions)** *Assume the loss functions  $\ell_i$  are  $(1/\gamma)$ -smooth  $\forall i \in [n]$ . We define  $\sigma_{\max} = \max_{k \in [K]} \sigma_k$ . Then after  $T$  iterations of Algorithm 2, with*

$$T \geq \frac{1}{\nu(1-\Theta)} \frac{\lambda\gamma n + \sigma_{\max}\sigma'}{\lambda\gamma n} \log \frac{1}{\epsilon_D},$$

it holds that

$$\mathbb{E}[D(\alpha^*) - D(\alpha^T)] \leq \epsilon_D.$$

Furthermore, after  $T$  iterations with

$$T \geq \frac{1}{\nu(1-\Theta)} \frac{\lambda\gamma n + \sigma_{\max}\sigma'}{\lambda\gamma n} \log \left( \frac{1}{\nu(1-\Theta)} \frac{\lambda\gamma n + \sigma_{\max}\sigma'}{\lambda\gamma n} \frac{1}{\epsilon_{\text{Gap}}} \right), \quad (14)$$

we have the expected duality gap

$$\mathbb{E}[\mathcal{P}(\mathbf{w}(\alpha^T)) - D(\alpha^T)] \leq \epsilon_{\text{Gap}}.$$

**THEOREM 4.3 (Lipschitz continuous loss functions)** *Consider Algorithm 2 with Assumption 4.1. Let  $\ell_i(\cdot)$  be  $L$ -Lipschitz continuous, and  $\epsilon_{\text{Gap}} > 0$  be the desired duality gap (and hence an upper*

bound on primal sub-optimality). Then after  $T$  iterations, where

$$\begin{aligned} T &\geq T_0 + \max \left\{ \left\lceil \frac{1}{\nu(1-\Theta)} \right\rceil, \frac{4L^2\sigma\sigma'}{\lambda n^2 \epsilon_{\text{Gap}} \nu(1-\Theta)} \right\}, \\ T_0 &\geq t_0 + \max \left\{ 0, \frac{2}{\nu(1-\Theta)} \left( \frac{8L^2\sigma\sigma'}{\lambda n^2 \epsilon_{\text{Gap}}} - 1 \right) \right\}, \\ t_0 &\geq \max \left\{ 0, \left\lceil \frac{1}{\nu(1-\Theta)} \log \left( \frac{2\lambda n^2 (D(\alpha^*) - D(\alpha^0))}{4L^2\sigma\sigma'} \right) \right\rceil \right\}, \end{aligned} \tag{15}$$

we have that the expected duality gap satisfies

$$\mathbb{E}[\mathcal{P}(\mathbf{w}(\bar{\alpha})) - D(\bar{\alpha})] \leq \epsilon_{\text{Gap}},$$

at the averaged iterate

$$\bar{\alpha} := \frac{1}{T - T_0} \sum_{t=T_0+1}^{T-1} \alpha^t. \tag{16}$$

The most important observation regarding the above result is that we do not impose any assumption on the choice of the local solver, apart from the sufficient decrease condition on the local objective in Assumption 4.1.

Let us now comment on the leading terms of the complexity results. The inverse dependence on  $1 - \Theta$  suggests that it is worth pushing the rate of local accuracy  $\Theta$  down to zero. However, when thinking about overall complexity, we have to bear in mind that achieving high accuracy on the local subproblems might be too expensive. The optimal choice would depend on the time we estimate a round of communication would take. In general, if communication is slow, it would be worth spending more time on solving local subproblems, but not so much if communication is relatively fast. We discussed this trade-off in Section 2.

We achieve a significant speedup by replacing the slow averaging aggregation (as in [26]) by more aggressive adding instead, that is  $\nu = 1$  instead of  $\nu = 1/K$ . Note that the safe subproblem parameter for the averaging case ( $\nu = 1/K$ ) is  $\sigma' := 1$ , while for adding ( $\nu = 1$ ) it is given by  $\sigma' := K$ , both proven in Lemma 3.2. The speedup that results from more aggressive adding is reflected in the convergence rate as shown above, when plugging in the actual parameter values  $\nu$  and  $\sigma'$  for the two cases, as we will illustrate more clearly in the next subsection.

### 4.3 Discussion and interpretations of convergence results

As the above theorems suggest, it is not possible to meaningfully change the aggregation parameter  $\nu$  in isolation. It comes naturally coupled with a particular subproblem.

In this section, we explain a simple way to be able to set the aggregation parameter as  $\nu = 1$ , that is to aggressively add up the updates from each machine. The motivation for this comes from a common practical setting. When solving the SVM dual (Hinge loss:  $\ell_i(a) = \max\{0, y_i - a\}$ ), the optimization problem comes with ‘box constraints’, i.e. for all  $i \in \{1, \dots, n\}$ , we have  $\alpha_i \in [0, 1]$  (see Table 1). The particular values of  $\alpha_i$  being 0 or 1 have a particular interpretation in the context of original problem (1). If we used  $\nu < 1$ , we would never be able to reach the upper boundary of any variable  $\alpha_i$ , when starting the algorithm at  $\mathbf{0}$ . This example illustrates some of the downsides of averaging vs. adding updates, coming from the fact that the step-size from using averaging (by being  $1/K$  times shorter) can result in  $1/K$  times slower convergence.

For the case of aggressive adding, the convergence from Theorem 4.2 becomes:

**COROLLARY 4.4** (Smooth loss functions—adding) *Let the assumptions of Theorem 4.2 be satisfied. If we run Algorithm 1 with  $\nu = 1, \sigma' = K$  for*

$$T \stackrel{(14)}{=} \frac{1}{1 - \Theta} \frac{\lambda\gamma n + \sigma_{\max}K}{\lambda\gamma n} \log \left( \frac{1}{1 - \Theta} \frac{\lambda\gamma n + \sigma_{\max}K}{\lambda\gamma n} \frac{1}{\epsilon_{\text{Gap}}} \right) \quad (17)$$

iterations, we have  $\mathbb{E}[\mathcal{P}(\mathbf{w}(\boldsymbol{\alpha}^T)) - D(\boldsymbol{\alpha}^T)] \leq \epsilon_{\text{Gap}}$ .

On the other hand, if we would just average results (as proposed in [26]), we would obtain following corollary:

**COROLLARY 4.5** (Smooth loss functions—averaging) *Let the assumptions of Theorem 4.2 be satisfied. If we run Algorithm 1 with  $\nu = 1/K, \sigma' = 1$  for*

$$T \stackrel{(14)}{\geq} \frac{1}{1 - \Theta} \frac{K\lambda\gamma n + \sigma_{\max}K}{\lambda\gamma n} \log \left( \frac{1}{1 - \Theta} \frac{K\lambda\gamma n + \sigma_{\max}K}{\lambda\gamma n} \frac{1}{\epsilon_{\text{Gap}}} \right) \quad (18)$$

iterations, we have  $\mathbb{E}[\mathcal{P}(\mathbf{w}(\boldsymbol{\alpha}^T)) - D(\boldsymbol{\alpha}^T)] \leq \epsilon_{\text{Gap}}$ .

Comparing the leading terms in Equations (17) and (18), we see that the leading term for the  $\nu = 1$  choice is  $\mathcal{O}(\lambda\gamma n + \sigma_{\max}K)$ , which is always better than for the  $\nu = 1/K$  case, when the leading term is  $\mathcal{O}(K\lambda\gamma n + \sigma_{\max}K)$ . This strongly suggests that adding in Framework 2 is preferable, especially when  $\lambda\gamma n \gg \sigma_{\max}$ .

An analogous improvement (by a factor on the order of  $K$ ) follows for the case of the sub-linear convergence rate for general Lipschitz loss functions, as shown in Theorem 4.3.

Note that the differences in the convergence rate are bigger for relatively big values of the regularizer  $\lambda$ . When the regularizer is  $\mathcal{O}(1/n)$ , the difference is negligible. This behaviour is also present in practice, as we will illustrate in Section 6.

## 5. Discussion and related work

In this section, we review a number of methods designed to solve optimization problems of the form of interest here, which are typically referred to as regularized empirical risk minimization (ERM) problems in the machine learning literature. This problem class (1), which is formally described in Section 2.1, underlies many prominent methods in supervised machine learning.

*Single machine solvers.* Stochastic Gradient Descent (SGD) is the simplest stochastic method one can use to solve (1), and dates back to the work of Robbins and Monro [55]. We refer the reader to [8,39–41] for a recent theoretical and practical assessment of SGD. Generally speaking, the method is extremely easy to implement, and converges to modest accuracies very quickly, which is often satisfactory in applications in machine learning. On the other hand, the method can sometimes be rather cumbersome because it can be difficult to tune its hyperparameters, and it can be impractical if higher solution accuracy is needed.

The current state of the art for empirical loss minimization with strongly convex regularizers is randomized coordinate ascent on the dual objective—Stochastic Dual Coordinate Ascent (SDCA) [60]. In contrast to primal SGD methods, the SDCA algorithm family is often preferred

as it is free of learning-rate parameters, and has faster (geometric) convergence guarantees. This algorithm and its variants are increasingly used in practice [61,72]. On the other hand, primal-only methods apply to a larger problem class, not only of form (1) that enables formation of dual problem (2) as considered here.

Another class of algorithms gaining attention in recent very few years are ‘variance reduced’ modifications of the original SGD algorithm. They are applied directly to the primal problem (1), but unlike SGD, have the property that the variance of estimates of the gradients tend to zero as they approach the optimal solution. Algorithms such as SAG [58], SAGA [16] and others [17,59] come at the cost of extra memory requirements—they have to store a gradient for each training example. This can be addressed efficiently in the case of generalized linear models, but prohibits its use in more complicated models such as in deep learning. On the other hand, Stochastic Variance Reduced Gradient and its variants [27,28,30,42,73] are often interpreted as ‘memory-free’ methods with variance reduction. However, these methods need to compute the full gradient occasionally to drive the variance reduction, which requires a full pass through the data and is an operation one generally tries to avoid. This and several other practical issues have been recently addressed in [2]. Finally, another class of extensions to SGD are stochastic quasi-Newton methods [6,11]. Despite their clear potential, a lack of theoretical understanding and complicated implementation issues compared to those above may still limit their adoption in the wider community. A stochastic dual Newton ascent (SDNA) method was proposed and analysed in [49]. However, the method needs to be modified substantially before it can be implemented in a distributed environment.

*SGD-based algorithms.* For the empirical loss minimization problems of interest, SGD-based methods are well established. Several distributed variants of SGD have been proposed, many of which build on the idea of a parameter server [18,43,52]. Despite their simplicity and accessibility in terms of implementation, the downside of this approach is that the amount of required communication is equal to the amount of data read locally, since one data point is accessed per machine per round (e.g. mini-batch SGD with a batch size of 1 per worker). These variants are in practice not competitive with the more communication-efficient methods considered in this work, which allow more local updates per communication round.

*One-shot communication schemes.* At the other extreme, there are distributed methods using only a single round of communication, such as [24,36,38,80,81]. These methods require additional assumptions on the partitioning of the data, which are usually not satisfied in practice if the data are distributed “as is”, i.e. if we do not have the opportunity to distribute the data in a specific way beforehand. Furthermore, some cannot guarantee convergence rates beyond what could be achieved if we ignored data residing on all but a single computer, as shown in [64]. Additional relevant lower bounds on the minimum number of communication rounds necessary for a given approximation quality are presented in [1,3].

*Mini-batch methods.* Mini-batch methods (which instead of just one data-example use updates from several examples per iteration) are more flexible and lie within these two communication vs. computation extremes. However, mini-batch versions of both SGD and coordinate descent (CD) [13,14,37,46–48,52–54,61,69,74] suffer from their convergence rate degrading towards the rate of batch gradient descent as the size of the mini-batch is increased. This follows because mini-batch updates are made based on the outdated previous parameter vector  $\mathbf{w}$ , in contrast to methods that allow immediate local updates like CoCoA.

Another disadvantage of mini-batch methods is that the aggregation parameter is harder to tune, as it can lie anywhere in the order of mini-batch size. The optimal choice is often either unknown, or difficult to compute. In the CoCoA setting, the parameter lies in the typically much smaller range given by  $K$ . In this work the aggregation parameter is further simplified and can be simply set to 1, i.e. adding updates, which is achieved by formulating a more conservative local problem as described in Section 3.1.

*Distributed batch solvers.* With traditional batch gradient solvers not being competitive for the problem class (1), improved batch methods have also received much research attention recently, in the single machine case as well as in the distributed setting. In distributed environments, popular methods include the alternating direction method of multipliers (ADMM) [9] as well as quasi-Newton methods such as L-BFGS, which can be attractive because of their relatively low communication requirements. Namely, communication is in the order of a constant number of vectors (the batch gradient information) per full pass through the data.

ADMM also comes with an additional penalty parameter balancing between the equality constraint on the primal variable vector  $\mathbf{w}$  and the original optimization objective [9], which is typically hard to tune in many applications. Nevertheless, the method has been used for distributed SVM training in, e.g. [23]. The known convergence rates for ADMM are weaker than the more problem-tailored methods mentioned we study here, and the choice of the penalty parameter is often unclear in practice.

Standard ADMM and quasi-Newton methods do not allow a gradual trade-off between communication and computation available here. An exception is the approach of Zhang et al. [79], which is similar to our approach in spirit, albeit based on ADMM, in that they allow for the subproblems to be solved inexactly. However, this work focuses on L2-regularized problems and a few selected loss functions, and offers no complexity results.

Interestingly, our proposed CoCoA<sup>+</sup> framework—despite being aimed at cheap stochastic local solvers—does have similarities to block-wise variants of batch proximal methods. In particular, the purpose of our subproblems as defined in (LO) is to form a data-dependent block-separable quadratic approximation to the smooth part of the original (dual) objective (2), while leaving the non-smooth part  $R$  intact (recall that  $R(\boldsymbol{\alpha})$  was defined to collect the  $\ell_i^*$  functions, and is separable over the coordinate blocks). Now if hypothetically each of our regularized quadratic subproblems (LO) were to be minimized exactly, the resulting steps could be interpreted as block-wise proximal Newton-type steps on each coordinate block  $k$  of the dual (2), where the Newton-subproblem is modified to also contain the proximal part  $R$ . This connection only holds for the special case of adding ( $\nu = 1$ ), and would correspond to a carefully adapted step-size in the block-wise Newton case.

One of the main crucial differences of our proposed CoCoA<sup>+</sup> framework compared to all known batch proximal methods (no matter if block-wise or not) is that the latter do require high accuracy subproblem solutions, and do not allow arbitrary solvers of weak accuracy  $\Theta$  such as we do here, see also the next paragraph. Distributed Newton methods have been analysed theoretically only when the subproblems are solved to high precision, see e.g. [64]. This makes the local solvers very expensive and the convergence rates less general than in our framework (which allows weak local solvers). Furthermore, the analysis of [64] requires additional strong assumptions on the data partitioning, such that the local Hessian approximations are consistent between the machines.

*Distributed methods allowing local optimization.* Developing distributed optimization methods that allow for arbitrary weak local optimizers requires carefully devising data-local subproblems to be solved after each communication round.

By making use of the primal–dual structure in the line of work of [31,45,74–76], the CoCoA and CoCoA<sup>+</sup> frameworks proposed here are the first to allow the use of any local solver—of weak local approximation quality—in each round. Furthermore, the approach here also allows more control over the aggregation of updates between machines. The practical variant of the DisDCA algorithm of [74], called DisDCA-p, also allows additive updates but is restricted to coordinate decent (CD) being the local solver, and was initially proposed without convergence guarantees. The work of [75] has provided the first theoretical convergence analysis for an ideal case, when the distributed data parts are all orthogonal to each other, which is an unrealistic setting in practice. DisDCA-p can be recovered as a special case of the CoCoA<sup>+</sup> framework

when using CD as a local solver, if  $|\mathcal{P}_k| = n/K$ , and when using the conservative bound  $\sigma' := K$ ; see also [31,35]. The convergence theory presented here therefore also covers that method, and extends it to arbitrary local solvers.

Since the first version of this work, Accelerated Inexact Dane (AIDE) [50]—a method based on the related set of ideas but applied to the primal problem—was developed. Like CoCoA<sup>+</sup>, AIDE promotes an efficient balance between communication and computation costs in the sense of (T-C).

*Inexact block-CD.* Our framework is related, but not identical, to running an *inexact* version of block coordinate ascent, applied to all blocks in parallel, and to the dual problem. From this perspective, the level of inexactness is controlled by the parameter  $\Theta$  through the use of a (possibly randomized) iterative ‘local’ solver applied to the local subproblems. For previous work on *randomized* block CD, we refer the reader to [68,70].

## 6. Numerical experiments

In this section, we explore numerous aspects of our distributed framework and demonstrate its competitive performance in practice. Section 6.1 first explores the impact of the local solver on overall performance, by comparing examples of various local solvers that can be used in the framework (the improved CoCoA<sup>+</sup> framework as shown in Algorithms 1 and 2) as well as testing the effect of approximate solution quality. The results indicate that the choice of local solver can have a significant impact on overall performance. In Sections 6.2 and 6.3 we further explore framework parameters, looking at the impact of the aggregation parameter  $\nu$  and the subproblem parameter  $\sigma'$ , respectively. Finally, Section 6.5 demonstrates the competitive practical performance of the overall framework on a large 280 GB distributed dataset.

We conduct experiments on three datasets of moderate and large size, namely *rcv1\_test*, *epsilon*, and *splice-site.t*.<sup>5</sup> The details of these datasets are listed in Table 2.

For solving subproblems, we compare numerous local solver methods, as listed in Table 3. We use the Euclidean norm as the regularizer  $g(x) = \|x\|^2$  for all the experiments. All the algorithms are implemented in C++ with MPI, and experiments are run on a cluster of 4 Amazon EC2 m3.xlarge instances. Our open-source code is available online at: <https://github.com/optml/CoCoA>.

Table 2. Datasets used for numerical experiments.

Dataset	$n$	$d$	Size (GB)
rcv1_test	677,399	47,236	1.2
epsilon	400,000	2000	3.1
splice-site.t	4,627,840	11,725,480	273.4

Table 3. Local solvers used in numerical experiments.

CD	Coordinate Descent [51]
APPROX	Accelerated, Parallel and Proximal Coordinate Descent [20,21]
GD	Gradient Descent with Backtracking Line Search [44]
CG	Conjugate Gradient Method [25]
L-BFGS	Quasi-Newton with Limited-Memory BFGS Updating [10]
BB	Barzilai–Borwein Gradient Method [4]
FISTA	Fast Iterative Shrinkage-Thresholding Algorithm [5]

Table 4. Optimal  $H$  for different local solvers and the rcv1\_test dataset.

Local solver	CD	APPROX	GD	CG	L-BFGS	BB	FISTA
$H$	40,000	40,000	20	5	10	15	20

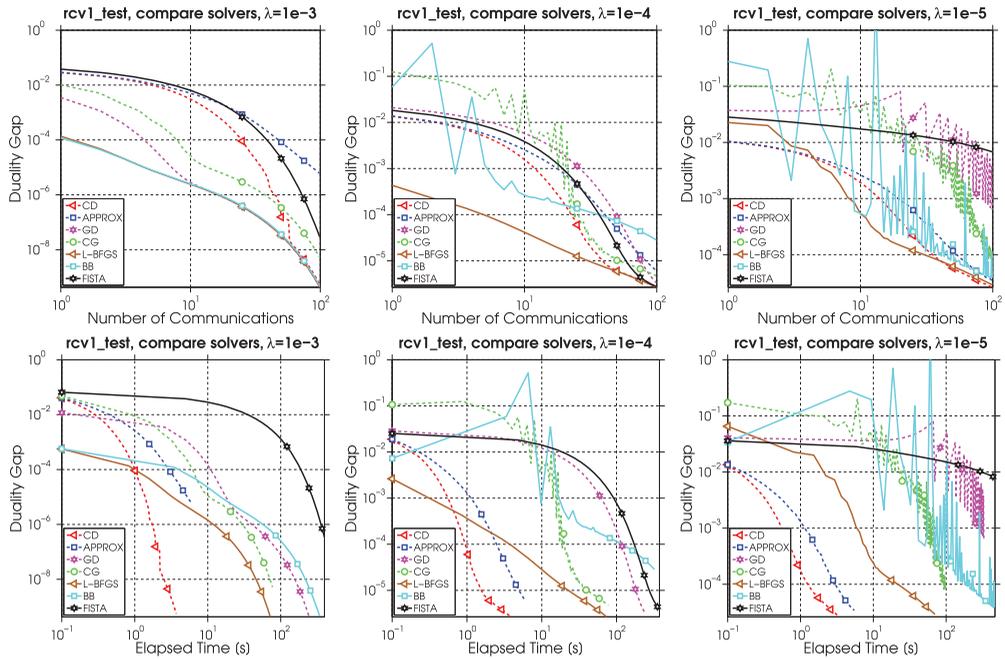


Figure 2. Performance of seven local solvers on the rcv1\_test dataset for three values of the regularization parameter.

### 6.1 Exploration of local solvers within the framework

In this section we compare the performance of our framework for various local solvers and various choices of inner iterations performed by a given local solver, resulting in different local accuracy measures  $\Theta$ . For simplicity, we choose the subproblem parameter  $\sigma' := \nu K$  (see Lemma 3.2) as a simple obtainable and theoretically safe value.

#### 6.1.1 Comparison of different local solvers

Here we compare the performance of the seven local solvers listed in Table 3. We show results for the quadratic loss function  $\ell_i(a) = \frac{1}{2}(a - y_i)^2$  with three different values of the regularization parameter,  $\lambda = 10^{-3}, 10^{-4}$ , and  $10^{-5}$ , and  $g(\cdot)$  being the default Euclidean squared norm regularizer:  $g(\cdot) = \frac{1}{2} \|\cdot\|^2$ . The dataset is rcv1\_test and we ran the CoCoA+ framework for a maximum of  $T = 100$  communication rounds. We set  $\nu = 1$  (adding) and choose  $H$  which gave the best performance in CPU time (see Table 4) for each solver.

From Figure 2, we find that if a high-enough accuracy solution is needed, the coordinate descent (CD) local solver always outperforms the other solvers. However, when a low accuracy solution is sufficient, as is often the case in machine learning applications, and if the regularization parameter is not too small, then L-BFGS performs best. The local subproblems arising with the rcv1\_test dataset are reasonably well conditioned. If more ill-conditioning was present, however, we would expect the APPROX local solver to do better than CD. This is because this

method is an *accelerated* variant of CD. In summary, randomized methods, such as CD and APPROX, and quasi-Newton methods (L-BFGS), perform best on this dataset.

Based on the above observations, it seems reasonable to expect that a method combining the power of both these successful approaches—randomization and second-order information—would perform even better. One might therefore want to look at local solvers based on ideas appearing in [49] or [56].

Note that it is not the goal of this work to decide on what the best local solver is. Our goals are quite the opposite, we provide a framework which allows the incorporation of *any* local solver. This choice might depend on which solvers are readily available to the practitioner/company. It will also depend on the conditioning of the local subproblems, their size, and other similar considerations. Future research will undoubtedly lead to the development of new and better local solvers which can be incorporated within CoCoA<sup>†</sup>.

Finally, note that some of the solvers cannot guarantee strict decrease of the duality gap, and sometimes this fluctuation can be very dramatic.

### 6.1.2 Effect of the quality of local solver solutions on overall performance

Here we discuss how the quality of subproblem solutions affects the overall performance of Algorithm 2. In order to do so, we denote  $H$  as the number of iterations the local solver is run for, within each communication round of the framework. We choose various values for  $H$  for the two local solvers that had the best performance in general, CD [51,60] and L-BFGS [10]. For CD,  $H$  represents the number of local iterations performed on the subproblem. For L-BFGS,  $H$  not only means the number of iterations, but also stands for the size of past information used to approximate the Hessian (i.e. the size of limited memory).

Looking at Figures 3 and 4, we see that for both these local solvers and all values of  $\lambda$ , increasing  $H$  will lead to less iterations of Algorithm 2. Of course, increasing  $H$  comes at the cost of

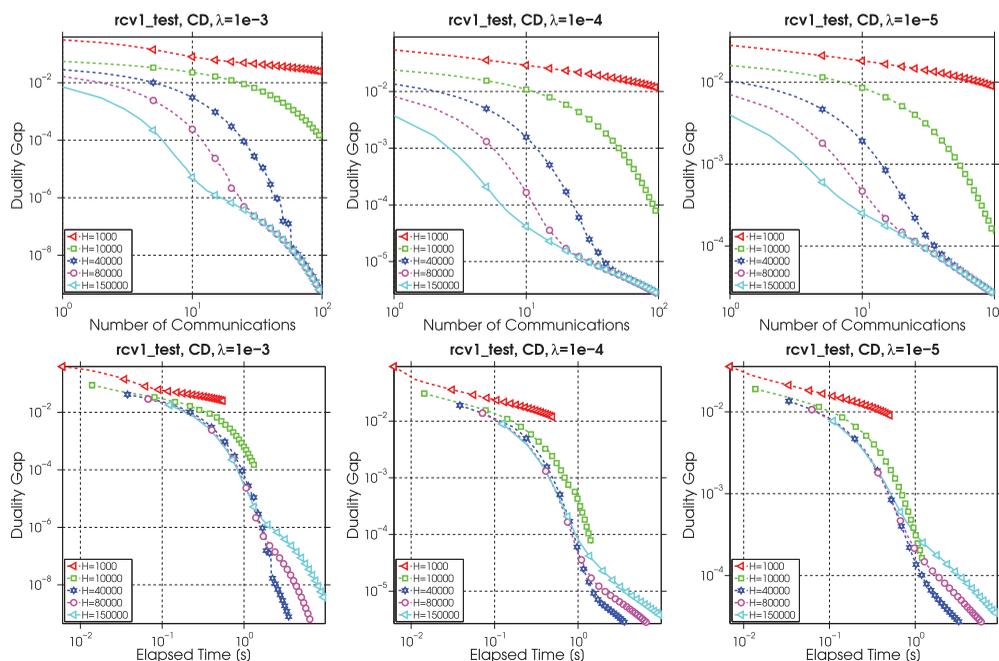


Figure 3. Varying the number of iterations of CD as a local solver.

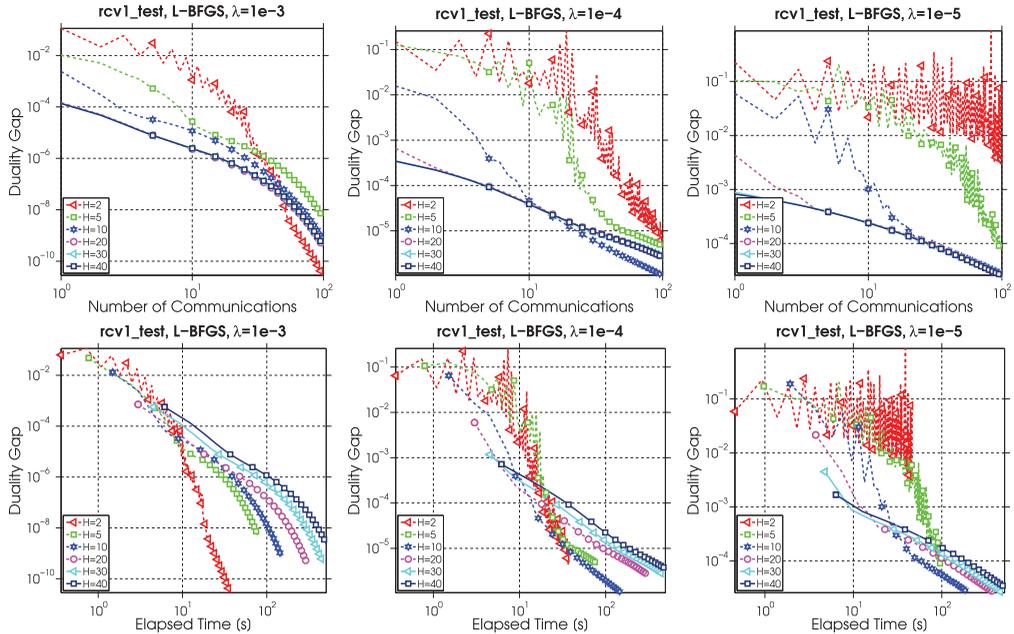


Figure 4. Varying the number of iterations of L-BFGS as a local solver.

the time spent on local solvers increasing. Hence, a larger value of  $H$  is not always the optimal choice with respect to total elapsed time. For example, for the rcv1\_test dataset, when choosing CD to solve the subproblems, choosing  $H$  to be 40,000 uses less time and provides faster convergence. When using L-BFGS,  $H = 10$  seems to be the best choice.

### 6.2 Averaging vs. adding the local updates

In this section, we compare the performance of our algorithm using two different schemes for aggregating partial updates: adding vs. averaging. This corresponds to comparing two extremes for the parameter  $\nu$ , either  $\nu := 1/K$  (averaging partial solutions) or  $\nu := 1$  (adding partial solutions). As discussed in Section 4, adding the local updates ( $\nu = 1$ ) will lead to less iterations than averaging, due to choosing different  $\sigma'$  in the subproblems. We verify this experimentally by considering several of the local solvers listed in Table 3.

We show results for the rcv1\_test dataset, and we apply the quadratic loss function with three different choices for the regularization parameter,  $\lambda = 1e - 03$ ,  $1e - 04$ , and  $1e - 05$ . The experiments in Figures 5–11 indicate that the ‘adding’ strategy will always lead to faster convergence than averaging, even though the difference is minimal when we apply a large number of iterations in the local solver. All the solid plots (adding) outperform the dashed plots (averaging), which indicates the advantage of choosing  $\nu = 1$ . Another note here is that for smaller  $\lambda$ , we will have to spend more iterations to get the same accuracy, because the original objective function (1) is less strongly convex.

### 6.3 The effect of the subproblem parameter $\sigma'$

In this section we consider the effect of the choice of the subproblem parameter on convergence (Figure 12). We plot the duality gap over the number of communications for the rcv1\_test

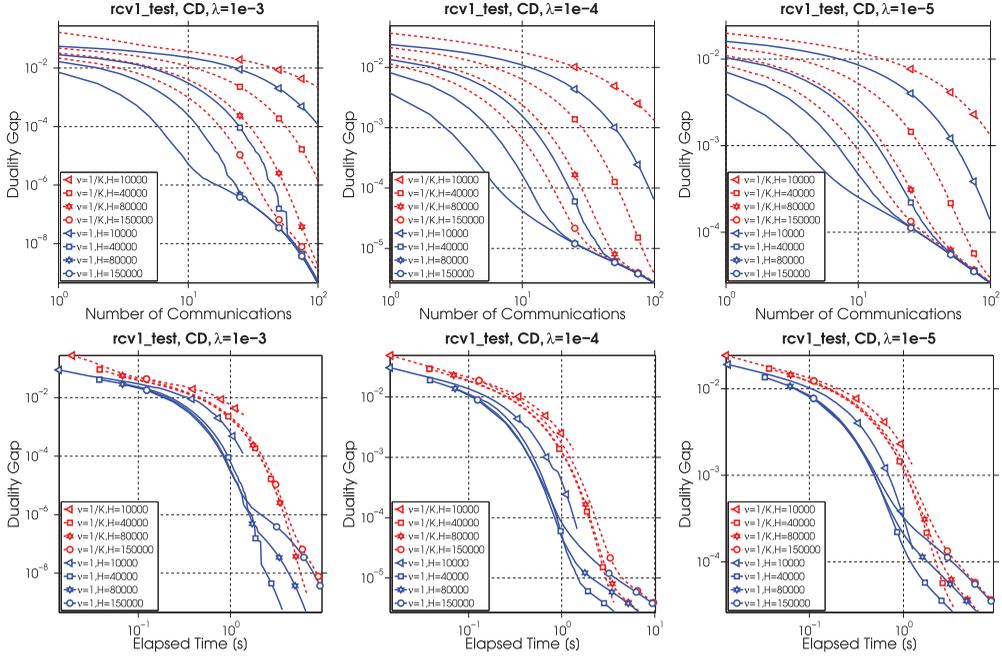


Figure 5. Adding (solid line) vs. averaging (dashed line) for CD as the local solver.

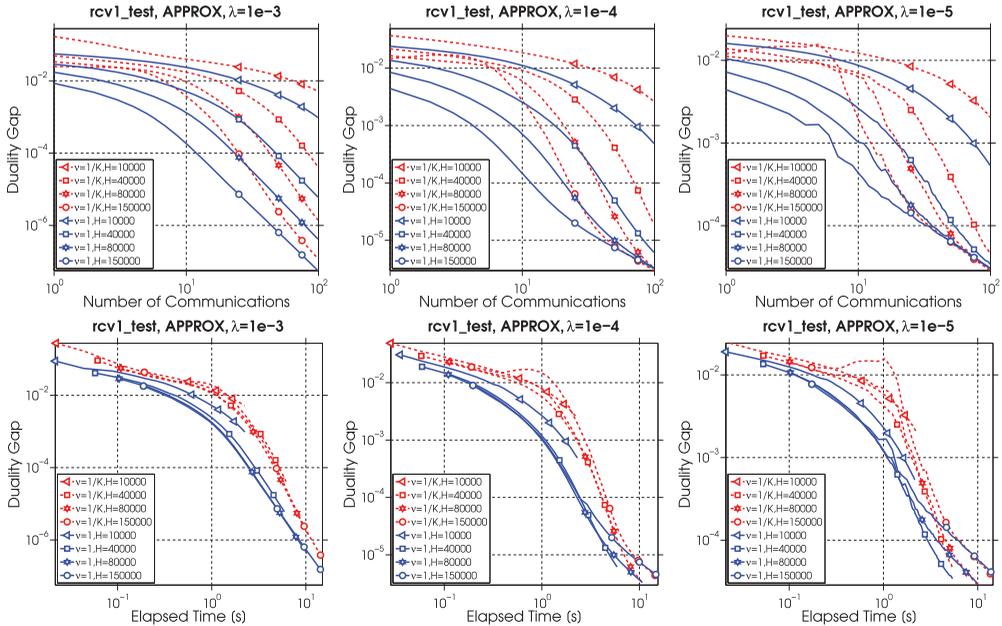


Figure 6. Adding (solid line) vs. averaging (dashed line) for APPROX as the local solver.

and epsilon datasets with quadratic loss, and set  $K = 8$ ,  $\lambda = 10^{-5}$ . For  $\nu = 1$  (adding the local updates), we consider several different values of  $\sigma'$ , ranging from 1 to 8. The value  $\sigma' = 8$  represents the safe upper bound of  $\nu K$ , as given in Lemma 3.2.

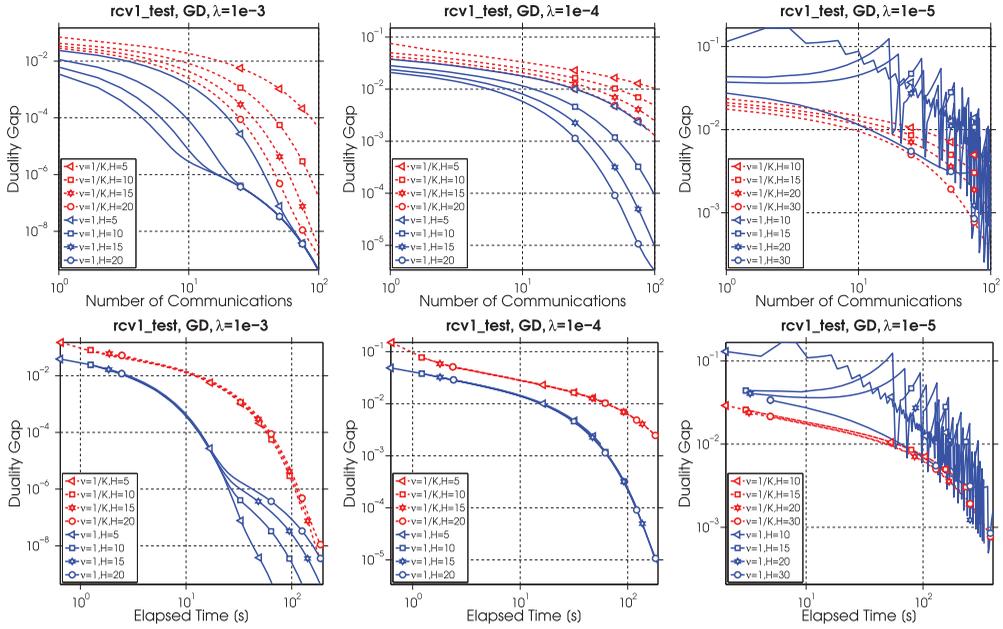


Figure 7. Adding (solid line) vs. averaging (dashed line) for Gradient Descent as the local solver.

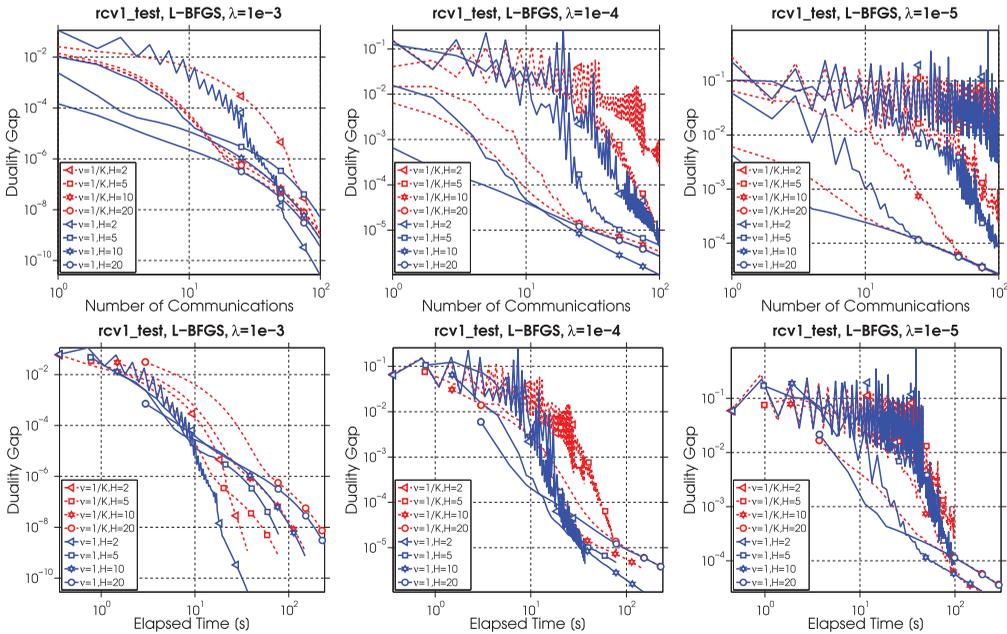


Figure 8. Adding (solid line) vs. averaging (dashed line) for L-BFGS as the local solver.

Decreasing  $\sigma'$  improves performance in terms of communication until a certain point, after which the algorithm diverges. For the rcv1\_test dataset, the optimal convergence occurs around  $\sigma' = 5$ , and diverges fast for  $\sigma' \leq 3$ . For the epsilon dataset,  $\sigma'$  around 6 is the best choice and

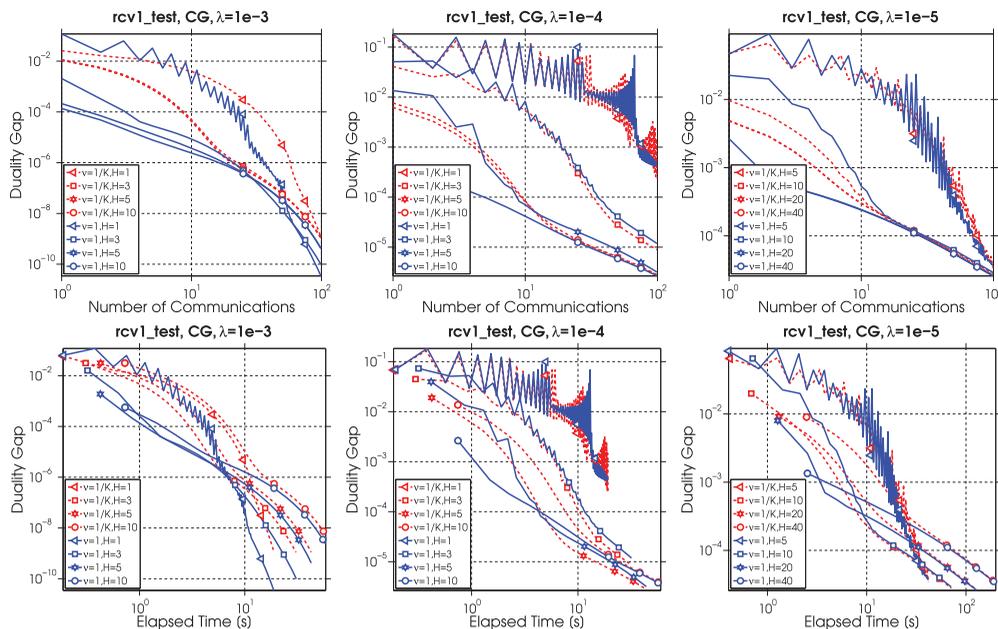


Figure 9. Adding (solid line) vs. averaging (dashed line) for Conjugate Gradient Method as the local solver.

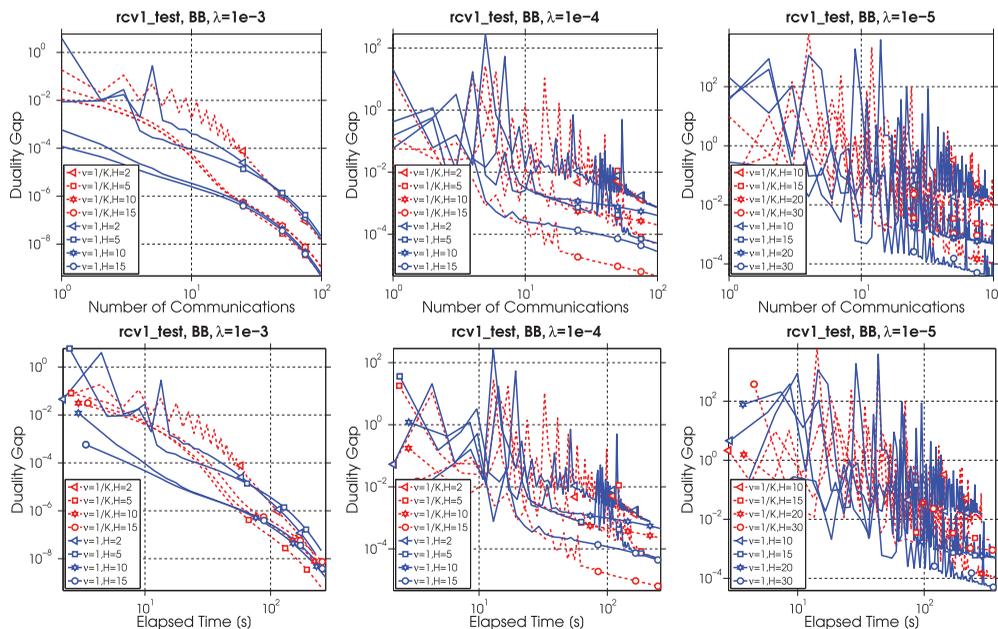


Figure 10. Adding (solid line) vs. averaging (dashed line) for BB as the local solver.

the algorithm will not converge to the optimal solution if  $\sigma' \leq 5$ . However, more importantly, the ‘safe’ upper bound of  $\sigma' := \nu K = 8$  has only slightly worse performance than the practically best (but ‘un-safe’) value of  $\sigma'$ .

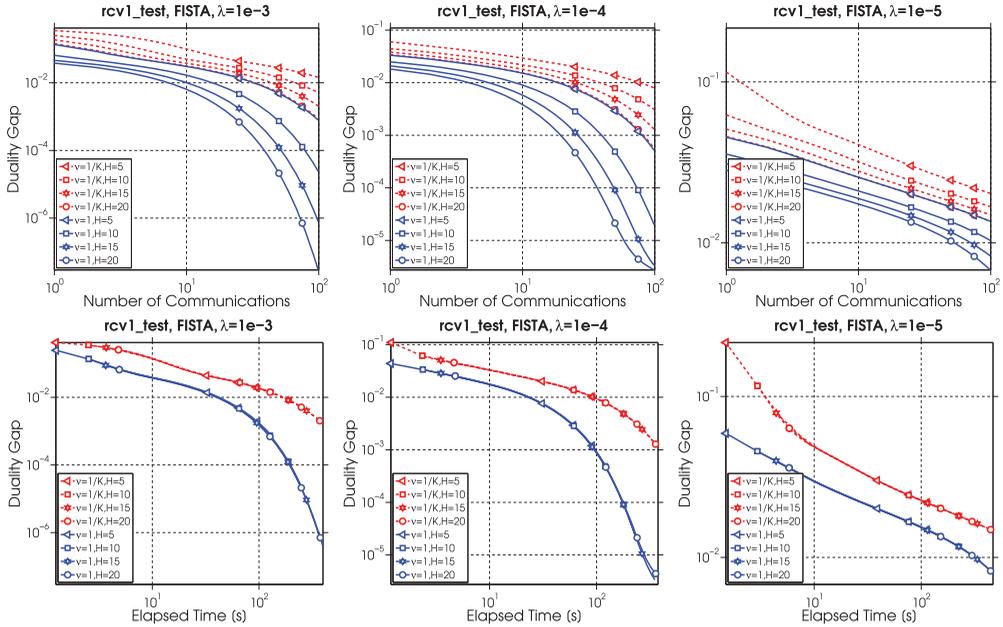


Figure 11. Adding (solid line) vs. averaging (dashed line) for FISTA as the local solver.

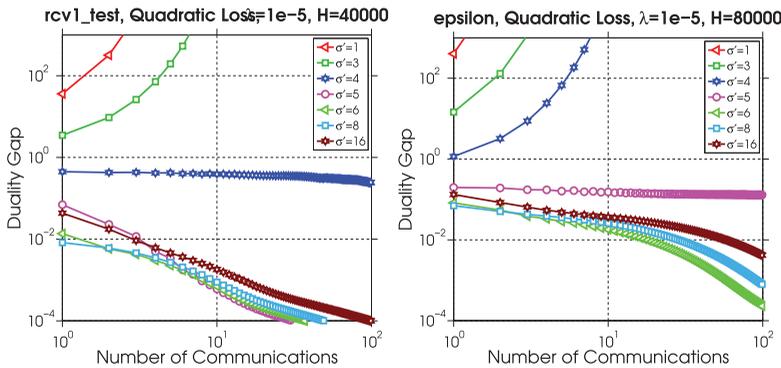


Figure 12. The effect of  $\sigma'$  on convergence for the rcv1\_test and epsilon datasets distributed across eight machines.

### 6.4 Scaling property

Here we demonstrate the ability of our framework to scale with  $K$  (number of machines). We compare the runtime to reach a specific tolerance on duality gap ( $10^{-4}$  and  $10^{-2}$ ) for two choices of  $\nu$ . Looking at Figure 13, we see that when choosing  $\nu = 1$ , the performance improves as the number of machines increases. However, when  $\nu = 1/K$ , the algorithm slows down as  $K$  increases. These observations support our analysis in Section 4.

### 6.5 Performance on a big dataset

As shown in Figure 14, we test the algorithm on the *splice-site.t* dataset, whose size is about 280 GB. We show experiments for three different loss functions  $\ell$ , namely logistic loss, hinge loss and least squares loss (see Table 1). We set  $\lambda = 10^{-6}$  for the squared norm regularizer. The dataset is

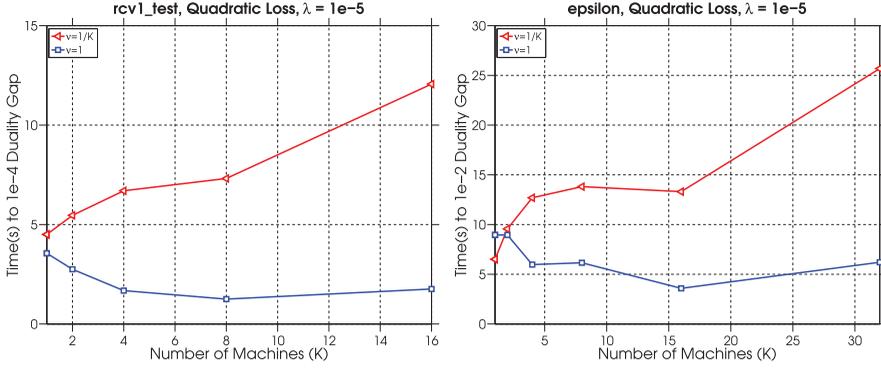


Figure 13. The effect of increasing the number of machines  $K$  on the time (s) to reach a solution with expected duality gap.

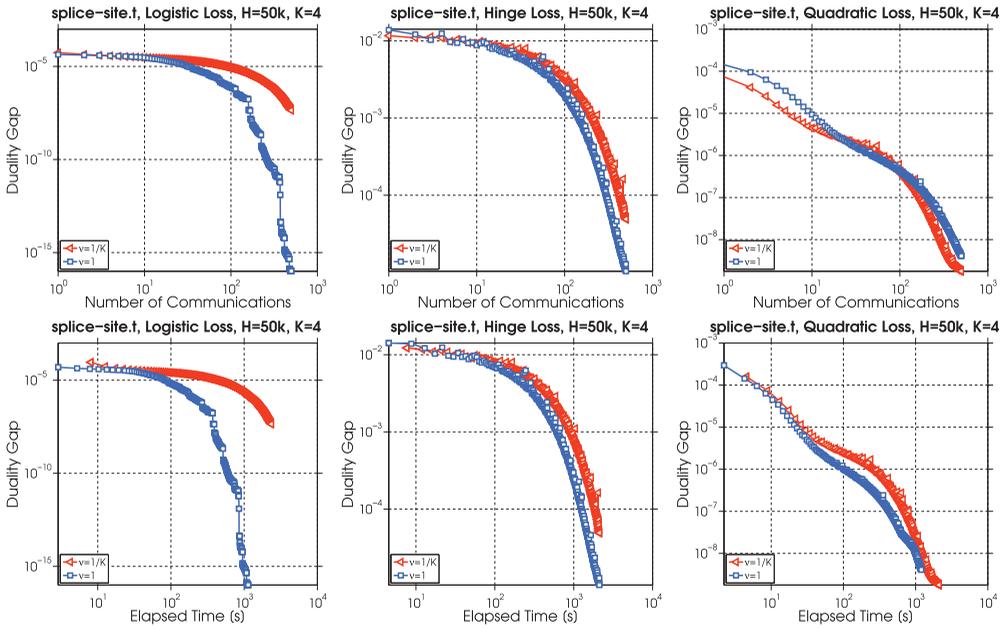


Figure 14. Performance of Algorithm 2 on *splice-site.t* dataset, with three different loss functions.

distributed across  $K = 4$  machines and we use CD as the local solver with  $H = 50,000$ . In all the cases, an optimal solution can be reached in about 20 minutes and again, we observe that setting the aggregation parameter  $\nu := 1$  leads to faster convergence than  $\nu := 1/K$  (averaging).

Also, the number of communication rounds for the three different loss functions are almost the same if we set all the other parameters to be the same. However, the duality gap decreases in a different manner for the three loss functions.

### 6.6 Comparison with other distributed methods

Finally, we compare the CoCoA<sup>+</sup> framework with several competing distributed optimization algorithms. The DiSCO algorithm [78] is a Newton-type method, where in each iteration the updates on iterates are computed inexactly using a Preconditioned Conjugate Gradients (PCG) method. As suggested in [78], in our implementation of DISCO we apply the Stochastic Average

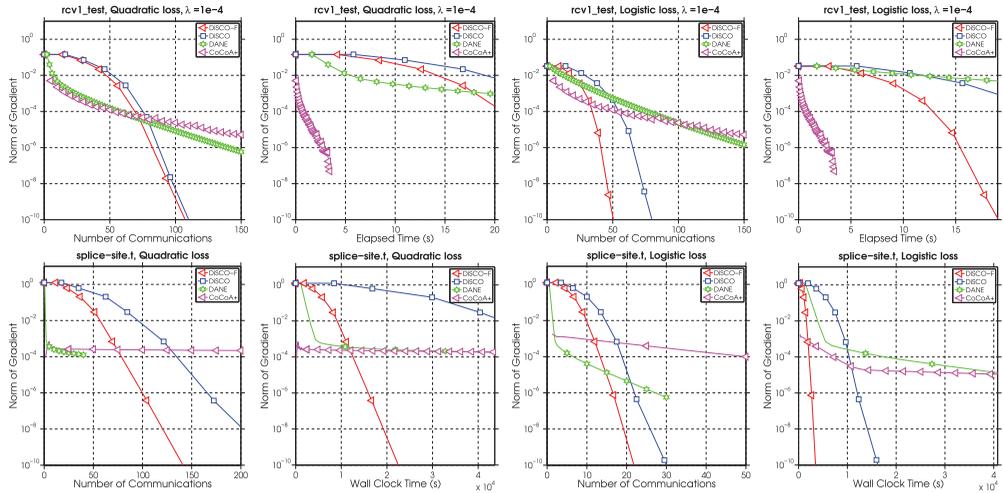


Figure 15. Performance of several distributed frameworks on solving (1) with different losses on two datasets.

Gradient (SAG) method [58] as the solver to get the initial solutions for each local machine and solve the linear system during PCG. DiSCO-F [33], improves on the computational efficiency of original DiSCO, by partitioning the data across features rather than examples. The DANE algorithm [63] is another distributed Newton-type method, where in each iteration there are two rounds of communication. Also, a subproblem is to be solved in each iteration to obtain updates. For each of these algorithms, we tune the hyperparameters manually to optimize performance.

The experiments are conducted on a compute cluster with  $K = 4$  machines. We run all algorithms using two datasets. Since not all methods are primal-based in nature, it is difficult to continue using duality gap as a measure of optimality. Instead, the norm of the gradient of the primal objective function (1) is used to compare the relative quality of the solutions obtained.

As shown in Figure 15, in terms of the number of communications, CoCoA<sup>+</sup> usually converges more rapidly than competing methods during the early iterations, but tends to get slower later on in the iterative process. This illustrates that the Newton-type methods can accelerate in the vicinity of the optimal solution, as expected. However, CoCoA<sup>+</sup> can still beat other methods in running time. The main reason for this is the fact that the subproblems in our framework can be solved more efficiently, compared with DiSCO and DANE.

## 7. Conclusion

We present CoCoA<sup>+</sup>, a novel framework that enables fast and communication-efficient *additive aggregation* in distributed primal–dual optimization. We analyse the theoretical complexity of CoCoA<sup>+</sup>, giving strong primal–dual convergence rates with outer iterations scaling independent of the number of machines. We extended the basic theory to allow for non-smooth loss functions, arbitrary strongly convex regularizers, and primal–dual convergence results. Our experimental results show significant speedups in terms of runtime over previous methods, including the original CoCoA framework as well as other state-of-the-art methods.

## Acknowledgments

This paper presents a substantial extension of an earlier conference article which appeared in ICML 2015—Proceedings of the 32th International Conference on Machine Learning, 2015 [35]. A conference article [65] exploring non-strongly convex regularizers is currently under review.

## Disclosure statement

No potential conflict of interest was reported by the authors.

## Notes

1. While for many algorithms the cost of a single iteration will vary throughout the iterative process, this simple model will suffice for our purpose to highlight the key issues associated with extending algorithms to a distributed framework.
2. For simplicity, we assume here a fixed network architecture, and compare only to classes of algorithms that communicate a single vector in each iteration, rendering  $c$  to be a constant, assuming we have a fixed number of machines. Most first-order algorithms would fall into this class.
3. Note that this is not obvious at this point. They are identical, subject to choice of local subproblems as specified in Section 3.1.
4. A subgradient of a convex function  $\phi$  in a point  $\mathbf{x}' \in \mathbb{R}^d$  is defined as any  $\xi \in \mathbb{R}^d$  satisfying for all  $\mathbf{x} \in \mathbb{R}^d$ ,  $\phi(\mathbf{x}) \geq \phi(\mathbf{x}') + \langle \xi, \mathbf{x} - \mathbf{x}' \rangle$ .
5. The datasets are available at <http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>.
6. Note that the case of weakly convex  $\ell_i^*(\cdot)$  is explicitly allowed here as well, as the Lemma holds for the case  $\gamma = 0$ .

## References

- [1] Y. Arjevani and O. Shamir, *Communication Complexity of Distributed Convex Learning and Optimization*, NIPS – Advances in Neural Information Processing Systems 28, Montreal, 2015, pp. 1747–1755.
- [2] R. Babanezhad, M.O. Ahmed, A. Virani, M. Schmidt, J. Konečný, and S. Sallinen, *Stop Wasting My Gradients: Practical SVRG*, NIPS – Advances in Neural Information Processing Systems 28, Montreal, 2015.
- [3] M.F. Balcan, A. Blum, S. Fine, and Y. Mansour, *Distributed Learning, Communication Complexity and Privacy*, COLT, April, 2012, pp. 26.1–26.22.
- [4] J. Barzilai and J.M. Borwein, *Two-point step size gradient methods*, IMA J. Numer. Analysis. 8 (1988), pp. 141–148.
- [5] A. Beck and M. Teboulle, *A fast iterative shrinkage-thresholding algorithm for linear inverse problems*, SIAM J. Imaging Sci. 2 (2009), pp. 183–202.
- [6] A. Bordes, L. Bottou, and P. Gallinari, *SGD-QN: Careful quasi-Newton stochastic gradient descent*, J. Mach. Learning Res. 10 (2009), pp. 1737–1754.
- [7] L. Bottou, *Large-Scale Machine Learning with Stochastic Gradient Descent*, Proceedings of COMPSTAT'2010, Paris, Springer, 2010, pp. 177–186.
- [8] L. Bottou, *Stochastic gradient descent tricks*, in *Neural Networks: Tricks of the Trade*, G. Montavon, G. Orr, and K.-R. Müller, eds., Springer, New York, 2012, pp. 421–436.
- [9] S. Boyd, N. Parikh, E.C.B. Peleato, and J. Eckstein, *Distributed optimization and statistical learning via the alternating direction method of multipliers*, Found. Trends Mach. Learning. 3 (2010), pp. 1–122.
- [10] R.H. Byrd, P. Lu, J. Nocedal, and C. Zhu, *A limited memory algorithm for bound constrained optimization*, SIAM J. Sci. Comput. 16 (1995), pp. 1190–1208.
- [11] R.H. Byrd, S. Hansen, J. Nocedal, and Y. Singer, *A Stochastic Quasi-Newton Method for Large-Scale Optimization*, preprint (2014). Available at arXiv:1401.7020.
- [12] W. Chen, Z. Wang, and J. Zhou, *Large-scale L-BFGS using MapReduce*, Advances in Neural Information Processing Systems 27, Montreal, 2014, pp. 1332–1340.
- [13] D. Csiba and P. Richtárik, *Primal method for ERM with flexible mini-batching schemes and non-convex losses* (2015). Available at arXiv:1506.02227, pp. 1–13.
- [14] D. Csiba and P. Richtárik, *Importance sampling for minibatches* (2016). Available at arXiv:1602.02283, pp. 1–19.
- [15] D. Csiba, Z. Qu, and P. Richtárik, *Stochastic Dual Coordinate Ascent with Adaptive Probabilities*, ICML – Proceedings of The 32nd International Conference on Machine Learning, Lille, 2015, pp. 674–683.
- [16] A. Defazio, F. Bach, and S. Lacoste-Julien, *SAGA: A Fast Incremental Gradient Method with Support for Non-Strongly Convex Composite Objectives*, Advances in Neural Information Processing Systems 27, Montreal, 2014, pp. 1646–1654.
- [17] A. Defazio, J. Domke, and T. Caetano, *Finito: A Faster, Permutable Incremental Gradient Method for Big Data Problems*. ICML – Proceedings of The 31st International Conference on Machine Learning, Beijing, 2014, pp. 1125–1133.
- [18] J. Duchi, M.I. Jordan, and B. McMahan, *Estimation, Optimization, and Parallelism When Data is Sparse*, NIPS – Advances in Neural Information Processing Systems 26, Lake Tahoe, 2013, pp. 2832–2840.
- [19] C. Dünnér, S. Forte, M. Takáč, and M. Jaggi, *Primal–dual rates and certificates*, ICML 2016 – Proceedings of the 33rd International Conference on Machine Learning, New York, 2016.
- [20] O. Fercoq and P. Richtárik, *Accelerated, parallel and proximal coordinate descent*, SIAM J. Optim. 25 (2015), pp. 1997–2023.
- [21] O. Fercoq and P. Richtárik, *Optimization in high dimensions via accelerated, parallel, and proximal coordinate descent*, SIAM Rev. 58 (2016), pp. 739–771.

- [22] O. Fercoq, Z. Qu, P. Richtárik, and M. Takáč, *Fast Distributed Coordinate Descent for Non-strongly Convex Losses*, IEEE Workshop on Machine Learning for Signal Processing, Reims, 2014.
- [23] P.A. Forero, A. Cano, and G.B. Giannakis, *Consensus-based distributed support vector machines*, J. Mach. Learning Res. 11 (2010), pp. 1663–1707.
- [24] C. Heinze, B. McWilliams, and N. Meinshausen, *DUAL-LOCO: Distributing Statistical Estimation Using Random Projections*, AISTATS – Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, Cadiz, 2016, pp. 875–883.
- [25] M.R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Stand. (1952).
- [26] M. Jaggi, V. Smith, M. Takáč, J. Terhorst, S. Krishnan, T. Hofmann, and M.I. Jordan, *Communication-Efficient Distributed Dual Coordinate Ascent*, Advances in Neural Information Processing Systems 27, Montreal, 2014, pp. 3068–3076.
- [27] R. Johnson and T. Zhang, *Accelerating Stochastic Gradient Descent Using Predictive Variance Reduction*, Advances in Neural Information Processing Systems 26, Lake Tahoe, 2013, pp. 315–323.
- [28] J. Konečný and P. Richtárik, *Semi-stochastic gradient descent methods*, preprint (2013). Available at arXiv:1312.1666.
- [29] J. Konečný, B. McMahan, and D. Ramage, *Federated optimization: Distributed optimization beyond the datacenter*, preprint (2015). Available at arXiv:1511.03575.
- [30] J. Konečný, J. Liu, P. Richtárik, and M. Takáč, *Mini-batch semi-stochastic gradient descent in the proximal setting*, IEEE J. Selected Topics Signal Process. 10 (2016), pp. 242–255.
- [31] C.P. Lee and D. Roth, *Distributed Box-Constrained Quadratic Optimization for Dual Linear SVM*, ICML – Proceedings of the 32th International Conference on Machine Learning, Lille, 2015, pp. 987–996.
- [32] C. Ma and M. Takáč, *Partitioning data on features or samples in communication-efficient distributed optimization?*, preprint (2015). Available at arXiv:1510.06688.
- [33] C. Ma and M. Takáč, *Distributed inexact damped newton method: Data partitioning and load-balancing*, ArXiv e-prints (2016).
- [34] C. Ma, R. Tappenden, and M. Takáč, *Linear convergence of the randomized feasible descent method under the weak strong convexity assumption*, preprint (2015). Available at arXiv:1506.02530.
- [35] C. Ma, V. Smith, M. Jaggi, M.I. Jordan, P. Richtárik, and M. Takáč, *Adding vs. Averaging in Distributed Primal–Dual Optimization*, ICML – Proceedings of the 32nd International Conference on Machine Learning, Lille, 2015, pp. 1973–1982.
- [36] G. Mann, R. McDonald, M. Mohri, N. Silberman, and D.D. Walker, *Efficient Large-Scale Distributed Training of Conditional Maximum Entropy Models*, Advances in Neural Information Processing Systems 22, Vancouver, 2009, pp. 1231–1239.
- [37] J. Mareček, P. Richtárik, and M. Takáč, *Distributed block coordinate descent for minimizing partially separable functions*, in *Numerical Analysis and Optimization 2014*, Springer Proceedings in Mathematics and Statistics, Vol. 134, M. Al-Baali, L. Grandinetti, and A. Purnama, eds., Springer, Muscat, 2015, pp. 261–288.
- [38] B. McWilliams, C. Heinze, N. Meinshausen, and G. Krummenacher, *LOCO: Distributing ridge regression with random projections*, preprint (2014). Available at arXiv:1406.3469.
- [39] E. Moulines and F.R. Bach, *Non-asymptotic Analysis of Stochastic Approximation Algorithms for Machine Learning*, Advances in Neural Information Processing Systems 24, Granada, 2011, pp. 451–459.
- [40] D. Needell, R. Ward, and N. Srebro, *Stochastic Gradient Descent, Weighted Sampling, and the Randomized Kaczmarz Algorithm*, Advances in Neural Information Processing Systems 27, Montreal, 2014, pp. 1017–1025.
- [41] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro, *Robust stochastic approximation approach to stochastic programming*, SIAM J. Optim. 19 (2009), pp. 1574–1609.
- [42] A. Nitanda, *Stochastic Proximal Gradient Descent with Acceleration Techniques*, Advances in Neural Information Processing Systems 27, Montreal, 2014, pp. 1574–1582.
- [43] F. Niu, B. Recht, C. Ré, and S.J. Wright, *Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent*, NIPS – Advances in Neural Information Processing Systems 24, Granada, 2011.
- [44] J. Nocedal and S. Wright, *Numerical Optimization*, 2nd ed., Springer Series in Operations Research and Financial Engineering, Springer, New York, 2006.
- [45] D. Pechyony, L. Shen, and R. Jones, *Solving Large Scale Linear SVM with Distributed Block Minimization*, ACM International Conference on Information and Knowledge Management, Glasgow, 2011.
- [46] Z. Qu and P. Richtárik, *Coordinate descent with arbitrary sampling I: Algorithms and complexity*, Optim. Methods Softw. 31 (2016), pp. 829–857.
- [47] Z. Qu and P. Richtárik, *Coordinate descent with arbitrary sampling II: Expected separable overapproximation*, Optim. Methods Softw. 31 (2016), pp. 858–884.
- [48] Z. Qu, P. Richtárik, and T. Zhang, *Quartz: Randomized Dual Coordinate Ascent with Arbitrary Sampling*, Advances in Neural Information Processing Systems 28, Montreal, 2015, pp. 865–873.
- [49] Z. Qu, P. Richtárik, and O. Fercoq, *SDNA: Stochastic Dual Newton Ascent for Empirical Risk Minimization*, ICML – Proceedings of the 33rd International Conference on Machine Learning, Vol. 48, New York, 2016.
- [50] S.J. Reddi, J. Konečný, P. Richtárik, B. Póczos, and A. Smola, *Aide: Fast and communication efficient distributed optimization*, preprint (2016). Available at arXiv:1608.06879.
- [51] P. Richtárik and M. Takáč, *Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function*, Math. Program. 144 (2014), pp. 1–38.

- [52] P. Richtárik and M. Takáč, *Distributed coordinate descent method for learning with big data*, J. Mach. Learn. Res. 17 (2016), pp. 1–25.
- [53] P. Richtárik and M. Takáč, *On optimal probabilities in stochastic coordinate descent methods*, Optim. Lett. 10 (2016), pp. 1233–1243.
- [54] P. Richtárik and M. Takáč, *Parallel coordinate descent methods for big data optimization*, Math. Program. 156 (2016), pp. 433–484.
- [55] H. Robbins and S. Monro, *A stochastic approximation method*, Ann. Math. Stat. 22 (1951), pp. 400–407.
- [56] D.G. Robert, M. Gower, and P. Richtárik, *Stochastic Block BFGS: Squeezing More Curvature Out of Data*, ICML 2016 – Proceedings of the 33rd International Conference on Machine Learning, New York, 2016, pp. 1869–1878.
- [57] R.T. Rockafellar, *Convex Analysis*, Princeton University Press, Princeton, 2015.
- [58] M. Schmidt, N.L. Roux, and F. Bach, *Minimizing finite sums with the stochastic average gradient*, preprint (2013). Available at arXiv:1309.2388.
- [59] S. Shalev-Shwartz, *SDCA without duality*, preprint (2015). Available at arXiv:1502.06177.
- [60] S. Shalev-Shwartz and T. Zhang, *Stochastic dual coordinate ascent methods for regularized loss minimization*, J. Mach. Learning Res. 14 (2013), pp. 567–599.
- [61] S. Shalev-Shwartz and T. Zhang, *Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization*, Math. Program. (2014), pp. 1–41.
- [62] O. Shamir and N. Srebro, *Distributed Stochastic Optimization and Learning*, Allerton, Urbana-Champaign, 2014.
- [63] O. Shamir, N. Srebro, and T. Zhang, *Communication efficient distributed optimization using an approximate newton-type method*, preprint (2013). Available at arXiv.org.
- [64] O. Shamir, N. Srebro, and T. Zhang, *Communication Efficient Distributed Optimization Using an Approximate Newton-type Method*, ICML – Proceedings of the 31st International Conference on Machine Learning, Beijing, 2014.
- [65] V. Smith, S. Forte, M.I. Jordan, and M. Jaggi,  *$L_1$ -Regularized Distributed Optimization: A Communication-Efficient Primal–Dual Framework*, preprint (2015). Available at arXiv.org.
- [66] M. Takáč, A. Bijral, P. Richtárik, and N. Srebro, *Mini-batch Primal and Dual Methods for SVM*, ICML – Proceedings of the 30th International Conference on Machine Learning, Atlanta, 2013.
- [67] M. Takáč, P. Richtárik, and N. Srebro, *Distributed mini-batch SDCA*, preprint (2015). Available at arXiv:1507.08322.
- [68] R. Tappenden, P. Richtárik, and B. Büke, *Separable approximations and decomposition methods for the augmented Lagrangian*, Optim. Methods Softw. 30 (2013), pp. 643–668.
- [69] R. Tappenden, M. Takáč, and P. Richtárik, *On the Complexity of Parallel Coordinate Descent*, preprint (2015). Available at arXiv math.OC.
- [70] R. Tappenden, P. Richtárik, and J. Gondzio, *Inexact coordinate descent: Complexity and preconditioning*, J. Optim. Theory Appl. 170 (2016), pp. 144–176.
- [71] V.N. Vapnik, *Statistical Learning Theory*, Vol. 1, Wiley, New York, 1998.
- [72] S.J. Wright, *Coordinate descent algorithms*, Math. Program. 151 (2015), pp. 3–34.
- [73] L. Xiao and T. Zhang, *A proximal stochastic gradient method with progressive variance reduction*, SIAM J. Optim. 24 (2014), pp. 2057–2075.
- [74] T. Yang, *Trading Computation for Communication: Distributed Stochastic Dual Coordinate Ascent*, Advances in Neural Information Processing Systems 26, Lake Tahoe, 2013.
- [75] T. Yang, S. Zhu, R. Jin, and Y. Lin, *Analysis of distributed stochastic dual coordinate ascent*, preprint (2013). Available at arXiv:1312.1031.
- [76] H.-F. Yu, C.-J. Hsieh, K.-W. Chang, and C.-J. Lin, *Large linear classification when data cannot fit in memory*, ACM Trans. Knowl. Discov. Data 5 (2012), pp. 1–23.
- [77] G.-X. Yuan, C.-H. Ho, and C.-J. Lin, *Recent advances of large-scale linear classification*, Proc. IEEE 100 (2012), pp. 2584–2603.
- [78] Y. Zhang and X. Lin, *DiSCO: Distributed Optimization for Self-concordant Empirical Loss*, ICML – Proceedings of the 32nd International Conference on Machine Learning, Lille, 2015, pp. 362–370.
- [79] C. Zhang, H. Lee, and K.G. Shin, *Efficient Distributed linear classification algorithms via the alternating direction method of multipliers*, AISTATS – Proceedings of the 15th International Conference on Artificial Intelligence and Statistics, La Palma, 2012.
- [80] Y. Zhang, J.C. Duchi, and M.J. Wainwright, *Communication-efficient algorithms for statistical optimization*, J. Mach. Learn. Res. 14 (2013), pp. 3321–3363.
- [81] M.A. Zinkevich, M. Weimer, A.J. Smola, and L. Li, *Parallelized Stochastic Gradient Descent*, Advances in Neural Information Processing Systems 23, Vancouver, 2010, pp. 1–37.

## Appendix. Proofs

### Proof of Lemma 2.3

Since  $g$  is 1-strongly convex,  $g^*$  is 1-smooth, and thus we can use (7) as follows:

$$\begin{aligned} f(\boldsymbol{\alpha} + \mathbf{h}) &= \lambda g^* \left( \frac{1}{\lambda n} X \boldsymbol{\alpha} + \frac{1}{\lambda n} X \mathbf{h} \right) \\ &\stackrel{(7)}{\leq} \lambda \left( g^* \left( \frac{1}{\lambda n} X \boldsymbol{\alpha} \right) + \left\langle \nabla g^* \left( \frac{1}{\lambda n} X \boldsymbol{\alpha} \right), \frac{1}{\lambda n} X \mathbf{h} \right\rangle + \frac{1}{2} \left\| \frac{1}{\lambda n} X \mathbf{h} \right\|^2 \right) \\ &= f(\boldsymbol{\alpha}) + \langle \nabla f(\boldsymbol{\alpha}), \mathbf{h} \rangle + \frac{1}{2\lambda n^2} \mathbf{h}^T X^T X \mathbf{h}. \end{aligned}$$

### Proof of Lemma 3.1

Indeed,

$$\begin{aligned} D \left( \boldsymbol{\alpha} + \nu \sum_{k=1}^K \mathbf{h}_{[k]} \right) &= D(\boldsymbol{\alpha} + \nu \mathbf{h}) \\ &\stackrel{(2)}{=} \frac{1}{n} \sum_{i=1}^n -\ell_i^*(-\alpha_i - \nu h_i) - \lambda g^* \left( \frac{1}{\lambda n} X(\boldsymbol{\alpha} + \nu \mathbf{h}) \right) \\ &\stackrel{(3)}{=} \frac{1}{n} \sum_{i=1}^n -\ell_i^*(-\alpha_i - \nu h_i) - f(\boldsymbol{\alpha} + \nu \mathbf{h}) \\ &\stackrel{(8)}{\geq} \frac{1-\nu}{n} \sum_{i=1}^n -\ell_i^*(-\alpha_i) + \nu \frac{1}{n} \sum_{i=1}^n -\ell_i^*(-\alpha_i - h_i) \\ &\quad - f(\boldsymbol{\alpha}) - \nu \langle \nabla f(\boldsymbol{\alpha}), \mathbf{h} \rangle - \nu^2 \frac{1}{2\lambda n^2} \mathbf{h}^T X^T X \mathbf{h} \\ &\stackrel{(2),(10)}{\geq} (1-\nu)D(\boldsymbol{\alpha}) - \nu \sum_{k=1}^K R_k(\boldsymbol{\alpha}_{[k]} + \mathbf{h}_{[k]}) \\ &\quad - \nu \frac{1}{K} \sum_{k=1}^K f(\boldsymbol{\alpha}) - \nu \sum_{k=1}^K \langle \nabla f(\boldsymbol{\alpha}), \mathbf{h}_{[k]} \rangle - \nu \sigma' \frac{1}{2\lambda n^2} \mathbf{h}^T \mathbf{G} \mathbf{h} \\ &\stackrel{(L0)}{=} (1-\nu)D(\boldsymbol{\alpha}) + \nu \frac{1}{K} \mathcal{G}_k^{\sigma'}(\mathbf{h}_{[k]}; \boldsymbol{\alpha}), \end{aligned}$$

where the first inequality follows from Jensen's inequality and the last equality follows from the block diagonal definition of  $G$  given in (11), i.e.

$$\mathbf{h}^T \mathbf{G} \mathbf{h} = \sum_{k=1}^K \mathbf{h}_{[k]} X_{[k]}^T X_{[k]} \mathbf{h}_{[k]}. \quad (\text{A1})$$

### Proof of Lemma 3.2

Considering  $\mathbf{h} \in \mathbb{R}^n$  with zeros in all coordinates except those that belong to the  $k$ th block  $\mathcal{P}_k$ , we have  $\mathbf{h}^T X^T X \mathbf{h} = \mathbf{h}^T \mathbf{G} \mathbf{h}$ , and thus  $\sigma' \geq \nu$ . Let  $\mathbf{h}_{[k,l]}$  denote  $\mathbf{h}_{[k]} - \mathbf{h}_{[l]}$ . Since  $X^T X$  is a positive semi-definite matrix, for  $k, l \in \{1, \dots, K\}$ ,  $k \neq l$  we have

$$0 \leq \mathbf{h}_{[k,l]}^T X^T X \mathbf{h}_{[k,l]} = \mathbf{h}_{[k]}^T X^T X \mathbf{h}_{[k]} + \mathbf{h}_{[l]}^T X^T X \mathbf{h}_{[l]} - 2\mathbf{h}_{[k]}^T X^T X \mathbf{h}_{[l]}. \quad (\text{A2})$$

By taking any  $\mathbf{h} \in \mathbb{R}^n$  for which  $\mathbf{h}^T \mathbf{G} \mathbf{h} \leq 1$ , in view of (10), we obtain

$$\begin{aligned} \mathbf{h}^T X^T X \mathbf{h} &= \sum_{k=1}^K \sum_{l=1}^K \mathbf{h}_{[k]}^T X^T X \mathbf{h}_{[l]} = \sum_{k=1}^K \mathbf{h}_{[k]}^T X^T X \mathbf{h}_{[k]} + \sum_{k \neq l} \mathbf{h}_{[k]}^T X^T X \mathbf{h}_{[l]} \\ &\stackrel{(A2)}{\leq} \sum_{k=1}^K \mathbf{h}_{[k]}^T X^T X \mathbf{h}_{[k]} + \sum_{k \neq l} \frac{1}{2} [\mathbf{h}_{[k]}^T X^T X \mathbf{h}_{[k]} + \mathbf{h}_{[l]}^T X^T X \mathbf{h}_{[l]}] \\ &= K \sum_{k=1}^K \mathbf{h}_{[k]}^T X^T X \mathbf{h}_{[k]} = K \mathbf{h}^T \mathbf{G} \mathbf{h} \leq K. \end{aligned}$$

Therefore, we can conclude that  $\nu \mathbf{h}^T X^T X \mathbf{h} \leq \nu K$  for all  $\mathbf{h}$  included in the definition (10) of  $\sigma'_{\min}$ , proving the claim.

### Proofs of Theorems 4.2 and 4.3

Before we state the proofs of the main theorems, we will write and prove a few crucial lemmas.

LEMMA A.1 *Let  $\ell_i^*$  be strongly<sup>6</sup> convex with convexity parameter  $\gamma \geq 0$  with respect to the norm  $\|\cdot\|$ ,  $\forall i \in [n]$ . Then for all iterations  $t$  of Algorithm 1 under Assumption 4.1, and any  $s \in [0, 1]$ , it holds that*

$$\mathbb{E}[D(\boldsymbol{\alpha}^{t+1}) - D(\boldsymbol{\alpha}^t)] \geq \nu(1 - \Theta) \left( s \text{Gap}(\boldsymbol{\alpha}^t) - \frac{\sigma'^2 s^2}{2\lambda n^2} R^t \right), \quad (A3)$$

where

$$R^t := -\frac{\lambda \gamma n(1-s)}{\sigma' s} \|\mathbf{u}^t - \boldsymbol{\alpha}^t\|^2 + \sum_{k=1}^K \|X(\mathbf{u}^t - \boldsymbol{\alpha}^t)_{[k]}\|^2, \quad (A4)$$

for  $\mathbf{u}^t \in \mathbb{R}^n$  with

$$-u_i^t \in \partial \ell_i(\mathbf{w}(\boldsymbol{\alpha}^t)^T \mathbf{x}_i). \quad (A5)$$

*Proof* For the sake of notation, we will write  $\boldsymbol{\alpha}$  instead of  $\boldsymbol{\alpha}^t$ ,  $\mathbf{w}$  instead of  $\mathbf{w}(\boldsymbol{\alpha}^t)$  and  $\mathbf{u}$  instead of  $\mathbf{u}^t$ .

Now, let us estimate the expected change of the dual objective. Using the definition of the dual update  $\boldsymbol{\alpha}^{t+1} := \boldsymbol{\alpha}^t + \nu \sum_k \mathbf{h}_{[k]}$  resulting in Algorithm 2, we have

$$\begin{aligned} \mathbb{E}[D(\boldsymbol{\alpha}^t) - D(\boldsymbol{\alpha}^{t+1})] &= \mathbb{E} \left[ D(\boldsymbol{\alpha}) - D(\boldsymbol{\alpha} + \nu \sum_{k=1}^K \mathbf{h}_{[k]}) \right] \\ &\stackrel{(12)}{\leq} \mathbb{E} \left[ D(\boldsymbol{\alpha}) - (1 - \nu)D(\boldsymbol{\alpha}) - \nu \sum_{k=1}^K \mathcal{G}_k^{\sigma'}(\mathbf{h}_{[k]}^t; \boldsymbol{\alpha}) \right] \\ &= \nu \mathbb{E} \left[ D(\boldsymbol{\alpha}) - \sum_{k=1}^K \mathcal{G}_k^{\sigma'}(\mathbf{h}_{[k]}^t; \boldsymbol{\alpha}) \right] \\ &= \nu \mathbb{E} \left[ D(\boldsymbol{\alpha}) - \sum_{k=1}^K \mathcal{G}_k^{\sigma'}(\mathbf{h}_{[k]}^*; \boldsymbol{\alpha}) + \sum_{k=1}^K \mathcal{G}_k^{\sigma'}(\mathbf{h}_{[k]}^*; \boldsymbol{\alpha}) - \sum_{k=1}^K \mathcal{G}_k^{\sigma'}(\mathbf{h}_{[k]}^t; \boldsymbol{\alpha}) \right] \\ &\stackrel{(13)}{\leq} \nu \left( D(\boldsymbol{\alpha}) - \sum_{k=1}^K \mathcal{G}_k^{\sigma'}(\mathbf{h}_{[k]}^*; \boldsymbol{\alpha}) + \Theta \left( \sum_{k=1}^K \mathcal{G}_k^{\sigma'}(\mathbf{h}_{[k]}^*; \boldsymbol{\alpha}) - \underbrace{\sum_{k=1}^K \mathcal{G}_k^{\sigma'}(\mathbf{0}; \boldsymbol{\alpha})}_{D(\boldsymbol{\alpha})} \right) \right) \\ &= \nu(1 - \Theta) \left( \underbrace{D(\boldsymbol{\alpha}) - \sum_{k=1}^K \mathcal{G}_k^{\sigma'}(\mathbf{h}_{[k]}^*; \boldsymbol{\alpha})}_C \right). \quad (A6) \end{aligned}$$

Now, let us upper bound the  $C$  term (we will denote by  $\mathbf{h}^* = \sum_{k=1}^K \mathbf{h}_{[k]}^*$ ):

$$\begin{aligned}
C &\stackrel{(2),(\text{LO})}{=} \frac{1}{n} \sum_{i=1}^n (\ell_i^*(-\alpha_i - h_i^*) - \ell_i^*(-\alpha_i)) + \langle \nabla f(\boldsymbol{\alpha}), \mathbf{h} \rangle + \sum_{k=1}^K \frac{\lambda}{2} \sigma' \left\| \frac{1}{\lambda n} X \mathbf{h}_{[k]}^* \right\|^2 \\
&\leq \frac{1}{n} \sum_{i=1}^n (\ell_i^*(-\alpha_i - s(u_i - \alpha_i)) - \ell_i^*(-\alpha_i)) + \langle \nabla f(\boldsymbol{\alpha}), s(\mathbf{u} - \boldsymbol{\alpha}) \rangle + \sum_{k=1}^K \frac{\lambda}{2} \sigma' \left\| \frac{1}{\lambda n} X s(\mathbf{u} - \boldsymbol{\alpha})_{[k]} \right\|^2 \\
&\stackrel{\text{Strong conv.}}{\leq} \frac{1}{n} \sum_{i=1}^n \left( s \ell_i^*(-u_i) + (1-s) \ell_i^*(-\alpha_i) - \frac{\gamma}{2} (1-s) s (u_i - \alpha_i)^2 - \ell_i^*(-\alpha_i) \right) \\
&\quad + \langle \nabla f(\boldsymbol{\alpha}), s(\mathbf{u} - \boldsymbol{\alpha}) \rangle + \sum_{k=1}^K \frac{\lambda}{2} \sigma' \left\| \frac{1}{\lambda n} X s(\mathbf{u} - \boldsymbol{\alpha})_{[k]} \right\|^2 \\
&= \frac{1}{n} \sum_{i=1}^n \left( s \ell_i^*(-u_i) - s \ell_i^*(-\alpha_i) - \frac{\gamma}{2} (1-s) s (u_i - \alpha_i)^2 \right) + \langle \nabla f(\boldsymbol{\alpha}), s(\mathbf{u} - \boldsymbol{\alpha}) \rangle + \sum_{k=1}^K \frac{\lambda}{2} \sigma' \left\| \frac{1}{\lambda n} X s(\mathbf{u} - \boldsymbol{\alpha})_{[k]} \right\|^2.
\end{aligned}$$

The convex conjugate maximal property implies that

$$\ell_i^*(-u_i) \stackrel{(\text{A5})}{=} -u_i \mathbf{w}(\boldsymbol{\alpha})^T \mathbf{x}_i - \ell_i(\mathbf{w}(\boldsymbol{\alpha})^T \mathbf{x}_i). \quad (\text{A7})$$

Moreover, from the definition of the primal and dual optimization problems (1), (2), we can write the duality gap as

$$\begin{aligned}
\text{Gap}(\boldsymbol{\alpha}) &:= \mathcal{P}(\mathbf{w}(\boldsymbol{\alpha})) - D(\boldsymbol{\alpha}) \\
&\stackrel{(1),(2)}{=} \frac{1}{n} \sum_{i=1}^n (\ell_i(\mathbf{x}_i^T \mathbf{w}(\boldsymbol{\alpha})) + \ell_i^*(-\alpha_i)) + \lambda g(\mathbf{w}(\boldsymbol{\alpha})) + \lambda g^*(\mathbf{v}(\boldsymbol{\alpha})) \\
&= \frac{1}{n} \sum_{i=1}^n (\ell_i(\mathbf{x}_i^T \mathbf{w}(\boldsymbol{\alpha})) + \ell_i^*(-\alpha_i)) + \lambda g(\nabla g^*(\mathbf{v}(\boldsymbol{\alpha}))) + \lambda g^*(\mathbf{v}(\boldsymbol{\alpha})) \\
&= \frac{1}{n} \sum_{i=1}^n (\ell_i(\mathbf{x}_i^T \mathbf{w}(\boldsymbol{\alpha})) + \ell_i^*(-\alpha_i)) + \lambda \mathbf{v}(\boldsymbol{\alpha})^T \mathbf{w}(\boldsymbol{\alpha}) \\
&= \frac{1}{n} \sum_{i=1}^n (\ell_i(\mathbf{x}_i^T \mathbf{w}(\boldsymbol{\alpha})) + \ell_i^*(-\alpha_i) + \mathbf{w}(\boldsymbol{\alpha})^T \mathbf{x}_i \alpha_i). \quad (\text{A8})
\end{aligned}$$

Hence,

$$\begin{aligned}
C &\stackrel{(\text{A7})}{\leq} \frac{1}{n} \sum_{i=1}^n \left( -s u_i \mathbf{w}(\boldsymbol{\alpha})^T \mathbf{x}_i - s \ell_i(\mathbf{w}(\boldsymbol{\alpha})^T \mathbf{x}_i) - s \ell_i^*(-\alpha_i) \underbrace{-s \mathbf{w}(\boldsymbol{\alpha})^T \mathbf{x}_i \alpha_i + s \mathbf{w}(\boldsymbol{\alpha})^T \mathbf{x}_i \alpha_i}_0 \right) \\
&\quad + \frac{1}{n} \sum_{i=1}^n \left( -\frac{\gamma}{2} (1-s) s (u_i - \alpha_i)^2 \right) + \langle \nabla f(\boldsymbol{\alpha}), s(\mathbf{u} - \boldsymbol{\alpha}) \rangle + \sum_{k=1}^K \frac{\lambda}{2} \sigma' \left\| \frac{1}{\lambda n} X s(\mathbf{u} - \boldsymbol{\alpha})_{[k]} \right\|^2 \\
&= \frac{1}{n} \sum_{i=1}^n (-s \ell_i(\mathbf{w}(\boldsymbol{\alpha})^T \mathbf{x}_i) - s \ell_i^*(-\alpha_i) - s \mathbf{w}(\boldsymbol{\alpha})^T \mathbf{x}_i \alpha_i) \\
&\quad + \frac{1}{n} \sum_{i=1}^n \left( s \mathbf{w}(\boldsymbol{\alpha})^T \mathbf{x}_i (\alpha_i - u_i) - \frac{\gamma}{2} (1-s) s (u_i - \alpha_i)^2 \right) \\
&\quad + \frac{1}{n} \mathbf{w}(\boldsymbol{\alpha})^T X s(\mathbf{u} - \boldsymbol{\alpha}) + \sum_{k=1}^K \frac{\lambda}{2} \sigma' \left\| \frac{1}{\lambda n} X s(\mathbf{u} - \boldsymbol{\alpha})_{[k]} \right\|^2 \\
&\stackrel{(\text{A8})}{=} -s \text{Gap}(\boldsymbol{\alpha}) - \frac{\gamma}{2} (1-s) s \frac{1}{n} \sum_{i=1}^n \|\mathbf{u} - \boldsymbol{\alpha}\|^2 + \frac{\sigma' s^2}{2 \lambda n^2} \sum_{k=1}^K \|X(\mathbf{u} - \boldsymbol{\alpha})_{[k]}\|^2. \quad (\text{A9})
\end{aligned}$$

Now, the claimed improvement bound (A3) follows by plugging (A9) into (A6). ■

LEMMA A.2 If  $\ell_i$  are  $L$ -Lipschitz continuous for all  $i \in [n]$ , then

$$\forall t : R^t \leq 4L^2 \underbrace{\sum_{k=1}^K \sigma_k |\mathcal{P}_k|}_{=:\sigma}, \quad (\text{A10})$$

where

$$\sigma_k := \max_{\alpha_{[k]} \in \mathbb{R}^n} \frac{\|X_{[k]} \alpha_{[k]}\|^2}{\|\alpha_{[k]}\|^2}. \quad (\text{A11})$$

*Proof* For general convex functions, the strong convexity parameter is  $\gamma = 0$ , and hence the definition of  $R^t$  becomes

$$R^t \stackrel{(\text{A4})}{=} \sum_{k=1}^K \|X(\mathbf{u}^t - \alpha^t)_{[k]}\|^2 \stackrel{(\text{A11})}{\leq} \sum_{k=1}^K \sigma_k \|\mathbf{u}^t - \alpha^t\|_{[k]}^2 \stackrel{\text{Lemma 3 in [60]}}{\leq} \sum_{k=1}^K \sigma_k |\mathcal{P}_k| 4L^2. \quad \blacksquare$$

#### A.4.1 Proof of Theorem 4.3

At first let us estimate expected change of dual feasibility. By using the main Lemma A.1, we have

$$\begin{aligned} \mathbb{E}[D(\alpha^*) - D(\alpha^{t+1})] &= \mathbb{E}[D(\alpha^*) - D(\alpha^{t+1}) + D(\alpha^t) - D(\alpha^t)] \\ &\stackrel{(\text{A3})}{=} D(\alpha^*) - D(\alpha^t) - \nu(1 - \Theta)s \text{Gap}(\alpha^t) + \nu(1 - \Theta) \frac{\sigma'}{2\lambda} \left(\frac{s}{n}\right)^2 R^t \\ &\stackrel{(4)}{=} D(\alpha^*) - D(\alpha^t) - \nu(1 - \Theta)s(\mathcal{P}(\mathbf{w}(\alpha^t)) - D(\alpha^t)) + \nu(1 - \Theta) \frac{\sigma'}{2\lambda} \left(\frac{s}{n}\right)^2 R^t \\ &\leq D(\alpha^*) - D(\alpha^t) - \nu(1 - \Theta)s(D(\alpha^*) - D(\alpha^t)) + \nu(1 - \Theta) \frac{\sigma'}{2\lambda} \left(\frac{s}{n}\right)^2 R^t \\ &\stackrel{(\text{A10})}{\leq} (1 - \nu(1 - \Theta)s)(D(\alpha^*) - D(\alpha^t)) + \nu(1 - \Theta) \frac{\sigma'}{2\lambda} \left(\frac{s}{n}\right)^2 4L^2\sigma. \end{aligned} \quad (\text{A12})$$

Using (A12) recursively, we have

$$\begin{aligned} \mathbb{E}[D(\alpha^*) - D(\alpha^t)] &= (1 - \nu(1 - \Theta)s)^t (D(\alpha^*) - D(\alpha^0)) \\ &\quad + \nu(1 - \Theta) \frac{\sigma'}{2\lambda} \left(\frac{s}{n}\right)^2 4L^2\sigma \sum_{j=0}^{t-1} (1 - \nu(1 - \Theta)s)^j \\ &= (1 - \nu(1 - \Theta)s)^t (D(\alpha^*) - D(\alpha^0)) + \nu(1 - \Theta) \frac{\sigma'}{2\lambda} \left(\frac{s}{n}\right)^2 4L^2\sigma \frac{1 - (1 - \nu(1 - \Theta)s)^t}{\nu(1 - \Theta)s} \\ &\leq (1 - \nu(1 - \Theta)s)^t (D(\alpha^*) - D(\alpha^0)) + s \frac{4L^2\sigma\sigma'}{2\lambda n^2}. \end{aligned} \quad (\text{A13})$$

The choice of  $s = 1$  and  $t = t_0 := \max\{0, \lceil (1/\nu(1 - \Theta)) \log(2\lambda n^2 (D(\alpha^*) - D(\alpha^0)) / (4L^2\sigma\sigma')) \rceil\}$  will lead to

$$\begin{aligned} \mathbb{E}[D(\alpha^*) - D(\alpha^t)] &\leq (1 - \nu(1 - \Theta))^{t_0} (D(\alpha^*) - D(\alpha^0)) + \frac{4L^2\sigma\sigma'}{2\lambda n^2} \\ &\leq \frac{4L^2\sigma\sigma'}{2\lambda n^2} + \frac{4L^2\sigma\sigma'}{2\lambda n^2} = \frac{4L^2\sigma\sigma'}{\lambda n^2}. \end{aligned} \quad (\text{A14})$$

Now, we are going to show that

$$\forall t \geq t_0 : \mathbb{E}[D(\alpha^*) - D(\alpha^t)] \leq \frac{4L^2\sigma\sigma'}{\lambda n^2 (1 + \frac{1}{2}\nu(1 - \Theta)(t - t_0))}. \quad (\text{A15})$$

Clearly, (A14) implies that (A15) holds for  $t = t_0$ . Now imagine that it holds for any  $t \geq t_0$  then we show that it also has to hold for  $t + 1$ . Indeed, using

$$s = \frac{1}{1 + \frac{1}{2}\nu(1 - \Theta)(t - t_0)} \in [0, 1] \quad (\text{A16})$$

we obtain

$$\begin{aligned}
 \mathbb{E}[D(\boldsymbol{\alpha}^*) - D(\boldsymbol{\alpha}^{t+1})] &\stackrel{(A12)}{\leq} (1 - \nu(1 - \Theta)s)(D(\boldsymbol{\alpha}^*) - D(\boldsymbol{\alpha}^t)) + \nu(1 - \Theta) \frac{\sigma'}{2\lambda} \left(\frac{s}{n}\right)^2 4L^2\sigma \\
 &\stackrel{(A15)}{\leq} (1 - \nu(1 - \Theta)s) \frac{4L^2\sigma\sigma'}{\lambda n^2(1 + \frac{1}{2}\nu(1 - \Theta)(t - t_0))} + \nu(1 - \Theta) \frac{\sigma'}{2\lambda} \left(\frac{s}{n}\right)^2 4L^2\sigma \\
 &\stackrel{(A16)}{=} \frac{4L^2\sigma\sigma'}{\lambda n^2} \left( \frac{1 + \frac{1}{2}\nu(1 - \Theta)(t - t_0) - \nu(1 - \Theta) + \nu(1 - \Theta)\frac{1}{2}}{(1 + \frac{1}{2}\nu(1 - \Theta)(t - t_0))^2} \right) \\
 &= \frac{4L^2\sigma\sigma'}{\lambda n^2} \underbrace{\left( \frac{1 + \frac{1}{2}\nu(1 - \Theta)(t - t_0) - \frac{1}{2}\nu(1 - \Theta)}{(1 + \frac{1}{2}\nu(1 - \Theta)(t - t_0))^2} \right)}_D.
 \end{aligned}$$

Now, we will upper bound  $D$  as follows:

$$\begin{aligned}
 D &= \frac{1}{1 + \frac{1}{2}\nu(1 - \Theta)(t + 1 - t_0)} \underbrace{\frac{(1 + \frac{1}{2}\nu(1 - \Theta)(t + 1 - t_0))(1 + \frac{1}{2}\nu(1 - \Theta)(t - 1 - t_0))}{(1 + \frac{1}{2}\nu(1 - \Theta)(t - t_0))^2}}_{\leq 1} \\
 &\leq \frac{1}{1 + \frac{1}{2}\nu(1 - \Theta)(t + 1 - t_0)},
 \end{aligned}$$

where in the last inequality we have used the fact that geometric mean is less or equal to arithmetic mean.

If  $\bar{\boldsymbol{\alpha}}$  is defined as (16), then we obtain that

$$\begin{aligned}
 \mathbb{E}[\text{Gap}(\bar{\boldsymbol{\alpha}})] &= \mathbb{E} \left[ \text{Gap} \left( \sum_{t=T_0}^{T-1} \frac{1}{T - T_0} \boldsymbol{\alpha}^t \right) \right] \leq \frac{1}{T - T_0} \mathbb{E} \left[ \sum_{t=T_0}^{T-1} \text{Gap}(\boldsymbol{\alpha}^t) \right] \\
 &\stackrel{(A3),(A10)}{\leq} \frac{1}{T - T_0} \mathbb{E} \left[ \sum_{t=T_0}^{T-1} \left( \frac{1}{\nu(1 - \Theta)s} (D(\boldsymbol{\alpha}^{t+1}) - D(\boldsymbol{\alpha}^t)) + \frac{4L^2\sigma\sigma's}{2\lambda n^2} \right) \right] \\
 &= \frac{1}{\nu(1 - \Theta)s} \frac{1}{T - T_0} \mathbb{E}[D(\boldsymbol{\alpha}^T) - D(\boldsymbol{\alpha}^{T_0})] + \frac{4L^2\sigma\sigma's}{2\lambda n^2} \\
 &\leq \frac{1}{\nu(1 - \Theta)s} \frac{1}{T - T_0} \mathbb{E}[D(\boldsymbol{\alpha}^*) - D(\boldsymbol{\alpha}^{T_0})] + \frac{4L^2\sigma\sigma's}{2\lambda n^2}. \tag{A17}
 \end{aligned}$$

Now, if  $T \geq \lceil 1/\nu(1 - \Theta) \rceil + T_0$  such that  $T_0 \geq t_0$  we obtain

$$\begin{aligned}
 \mathbb{E}[\text{Gap}(\bar{\boldsymbol{\alpha}})] &\stackrel{(A17),(A15)}{\leq} \frac{1}{\nu(1 - \Theta)s} \frac{1}{T - T_0} \left( \frac{4L^2\sigma\sigma'}{\lambda n^2(1 + \frac{1}{2}\nu(1 - \Theta)(T_0 - t_0))} \right) + \frac{4L^2\sigma\sigma's}{2\lambda n^2} \\
 &= \frac{4L^2\sigma\sigma'}{\lambda n^2} \left( \frac{1}{\nu(1 - \Theta)s} \frac{1}{T - T_0} \frac{1}{1 + \frac{1}{2}\nu(1 - \Theta)(T_0 - t_0)} + \frac{s}{2} \right). \tag{A18}
 \end{aligned}$$

Choosing

$$s = \frac{1}{(T - T_0)\nu(1 - \Theta)} \in [0, 1] \tag{A19}$$

gives us

$$\mathbb{E}[\text{Gap}(\bar{\boldsymbol{\alpha}})] \stackrel{(A18),(A19)}{\leq} \frac{4L^2\sigma\sigma'}{\lambda n^2} \left( \frac{1}{1 + \frac{1}{2}\nu(1 - \Theta)(T_0 - t_0)} + \frac{1}{(T - T_0)\nu(1 - \Theta)} \frac{1}{2} \right). \tag{A20}$$

To have the right-hand side of (A20) smaller than  $\epsilon_{\text{Gap}}$ , it is sufficient to choose  $T_0$  and  $T$  such that

$$\frac{4L^2\sigma\sigma'}{\lambda n^2} \left( \frac{1}{1 + \frac{1}{2}\nu(1 - \Theta)(T_0 - t_0)} \right) \leq \frac{1}{2}\epsilon_{\text{Gap}}, \tag{A21}$$

$$\frac{4L^2\sigma\sigma'}{\lambda n^2} \left( \frac{1}{(T - T_0)\nu(1 - \Theta)} \frac{1}{2} \right) \leq \frac{1}{2}\epsilon_{\text{Gap}}. \tag{A22}$$

Hence, if

$$t_0 + \frac{2}{v(1-\Theta)} \left( \frac{8L^2\sigma\sigma'}{\lambda n^2 \epsilon_{\text{Gap}}} - 1 \right) \leq T_0,$$

$$T_0 + \frac{4L^2\sigma\sigma'}{\lambda n^2 \epsilon_{\text{Gap}} v(1-\Theta)} \leq T,$$

then (A21) and (A22) are satisfied.

#### A.4.2 Proof of Theorem 4.2

If the function  $\ell_i(\cdot)$  is  $(1/\gamma)$ -smooth then  $\ell_i^*(\cdot)$  is  $\gamma$ -strongly convex with respect to the  $\|\cdot\|$  norm. From (A4), we have

$$\begin{aligned} R^t &\stackrel{(A4)}{=} -\frac{\lambda\gamma n(1-s)}{\sigma's} \|\mathbf{u}^t - \boldsymbol{\alpha}^t\|^2 + \sum_{k=1}^K \|X(\mathbf{u}^t - \boldsymbol{\alpha}^t)_{[k]}\|^2 \\ &\stackrel{(A11)}{\leq} -\frac{\lambda\gamma n(1-s)}{\sigma's} \|\mathbf{u}^t - \boldsymbol{\alpha}^t\|^2 + \sum_{k=1}^K \sigma_k \|(\mathbf{u}^t - \boldsymbol{\alpha}^t)_{[k]}\|^2 \\ &\leq -\frac{\lambda\gamma n(1-s)}{\sigma's} \|\mathbf{u}^t - \boldsymbol{\alpha}^t\|^2 + \sigma_{\max} \sum_{k=1}^K \|(\mathbf{u}^t - \boldsymbol{\alpha}^t)_{[k]}\|^2 \\ &= \left( -\frac{\lambda\gamma n(1-s)}{\sigma's} + \sigma_{\max} \right) \|\mathbf{u}^t - \boldsymbol{\alpha}^t\|^2. \end{aligned} \tag{A23}$$

If we plug

$$s = \frac{\lambda\gamma n}{\lambda\gamma n + \sigma_{\max}\sigma'} \in [0, 1] \tag{A24}$$

into (A23), we obtain that  $\forall t : R^t \leq 0$ . Putting the same  $s$  into (A3) will give us

$$\begin{aligned} \mathbb{E}[D(\boldsymbol{\alpha}^{t+1}) - D(\boldsymbol{\alpha}^t)] &\stackrel{(A3),(A24)}{\geq} v(1-\Theta) \frac{\lambda\gamma n}{\lambda\gamma n + \sigma_{\max}\sigma'} \text{Gap}(\boldsymbol{\alpha}^t) \\ &\geq v(1-\Theta) \frac{\lambda\gamma n}{\lambda\gamma n + \sigma_{\max}\sigma'} D(\boldsymbol{\alpha}^*) - D(\boldsymbol{\alpha}^t). \end{aligned} \tag{A25}$$

Using the fact that  $\mathbb{E}[D(\boldsymbol{\alpha}^{t+1}) - D(\boldsymbol{\alpha}^t)] = \mathbb{E}[D(\boldsymbol{\alpha}^{t+1}) - D(\boldsymbol{\alpha}^*)] + D(\boldsymbol{\alpha}^*) - D(\boldsymbol{\alpha}^t)$ , we have

$$\mathbb{E}[D(\boldsymbol{\alpha}^{t+1}) - D(\boldsymbol{\alpha}^*)] + D(\boldsymbol{\alpha}^*) - D(\boldsymbol{\alpha}^t) \stackrel{(A25)}{\geq} v(1-\Theta) \frac{\lambda\gamma n}{\lambda\gamma n + \sigma_{\max}\sigma'} D(\boldsymbol{\alpha}^*) - D(\boldsymbol{\alpha}^t)$$

which is equivalent to

$$\mathbb{E}[D(\boldsymbol{\alpha}^*) - D(\boldsymbol{\alpha}^{t+1})] \leq \left( 1 - v(1-\Theta) \frac{\lambda\gamma n}{\lambda\gamma n + \sigma_{\max}\sigma'} \right) D(\boldsymbol{\alpha}^*) - D(\boldsymbol{\alpha}^t). \tag{A26}$$

Therefore if we denote by  $\epsilon_D^t = D(\boldsymbol{\alpha}^*) - D(\boldsymbol{\alpha}^t)$  we have that

$$\begin{aligned} \mathbb{E}[\epsilon_D^t] &\stackrel{(A26)}{\leq} \left( 1 - v(1-\Theta) \frac{\lambda\gamma n}{\lambda\gamma n + \sigma_{\max}\sigma'} \right)^t \epsilon_D^0 \leq \left( 1 - v(1-\Theta) \frac{\lambda\gamma n}{\lambda\gamma n + \sigma_{\max}\sigma'} \right)^t \\ &\leq \exp \left( -tv(1-\Theta) \frac{\lambda\gamma n}{\lambda\gamma n + \sigma_{\max}\sigma'} \right). \end{aligned}$$

The right-hand side will be smaller than some  $\epsilon_D$  if

$$t \geq \frac{1}{v(1-\Theta)} \frac{\lambda\gamma n + \sigma_{\max}\sigma'}{\lambda\gamma n} \log \frac{1}{\epsilon_D}.$$

Moreover, to bound the duality gap, we have

$$v(1-\Theta) \frac{\lambda\gamma n}{\lambda\gamma n + \sigma_{\max}\sigma'} \text{Gap}(\boldsymbol{\alpha}^t) \stackrel{(A25)}{\leq} \mathbb{E}[D(\boldsymbol{\alpha}^{t+1}) - D(\boldsymbol{\alpha}^t)] \leq \mathbb{E}[D(\boldsymbol{\alpha}^*) - D(\boldsymbol{\alpha}^t)].$$

Therefore,  $\text{Gap}(\boldsymbol{\alpha}^t) \leq (1/\nu(1-\Theta))(\lambda\gamma n + \sigma_{\max}\sigma')/\lambda\gamma n \epsilon_D^t$ . Hence if  $\epsilon_D \leq \nu(1-\Theta)(\lambda\gamma n/(\lambda\gamma n + \sigma_{\max}\sigma'))\epsilon_{\text{Gap}}$  then  $\text{Gap}(\boldsymbol{\alpha}^t) \leq \epsilon_{\text{Gap}}$ . Therefore after

$$t \geq \frac{1}{\nu(1-\Theta)} \frac{\lambda\gamma n + \sigma_{\max}\sigma'}{\lambda\gamma n} \log \left( \frac{1}{\nu(1-\Theta)} \frac{\lambda\gamma n + \sigma_{\max}\sigma'}{\lambda\gamma n} \frac{1}{\epsilon_{\text{Gap}}} \right)$$

iterations, we have obtained a duality gap less than  $\epsilon_{\text{Gap}}$ .