CoCoA: A General Framework for Communication-Efficient Distributed Optimization

Virginia Smith VSMITH@BERKELEY.EDU
Division of Computer Science
University of California
Berkeley, CA 94720, USA

Simone Forte SIMONE.FORTE@GESS.ETHZ.CH
Department of Computer Science
ETH Zürich
8006 Zürich, Switzerland

Chenxin Ma CHM514@LEHIGH.EDU
Martin Takáč TAKAC.MT@GMAIL.COM
Industrial and Systems Engineering Department
Lehigh University
Bethlehem, PA 18015, USA

Michael I. Jordan JORDAN@CS.BERKELEY.EDU
Division of Computer Science and Department of Statistics
University of California
Berkeley, CA 94720, USA

Martin Jaggi MARTIN.JAGGI@EPFL.CH
School of Computer and Communication Sciences
EPFL
1015 Lausanne, Switzerland

Abstract

The scale of modern datasets necessitates the development of efficient distributed optimization methods for machine learning. We present a general-purpose framework for the distributed environment, CoCoA, that has an efficient communication scheme and is applicable to a wide variety of problems in machine learning and signal processing. We extend the framework to cover general non-strongly convex regularizers, including L1-regularized problems like lasso, sparse logistic regression, and elastic net regularization, and show how earlier work can be derived as a special case. We provide convergence guarantees for the class of convex regularized loss minimization objectives, leveraging a novel approach in handling non-strongly convex regularizers and non-smooth loss functions. The resulting framework has markedly improved performance over state-of-the-art methods, as we illustrate with an extensive set of experiments on real distributed datasets.

Keywords: Convex optimization, distributed systems, large-scale machine learning, parallel and distributed algorithms

1. Introduction

Distributed computing architectures have come to the fore in modern machine learning, in response to the challenges arising from a wide range of large-scale learning applications. Distributed architectures offer the promise of scalability by increasing both computational and storage capacities. A critical challenge in realizing this promise of scalability is to develop efficient methods for communicating and coordinating information between distributed machines, taking into account the specific needs of machine-learning algorithms. On most distributed systems, the communication of data between machines is vastly more expensive than reading data from main memory and performing local computation. Moreover, the optimal trade-off between communication and computation can vary widely depending on the dataset being processed, the system being used, and the objective being optimized. It is therefore essential for distributed methods to accommodate flexible communication-computation profiles while still providing convergence guarantees.

Although numerous distributed optimization methods have been proposed, the mini-batch optimization approach has emerged as one of the most popular paradigms for tackling this communication-computation tradeoff (see, e.g., Takáč et al., 2013; Shalev-Shwartz and Zhang, 2013b; Qu et al., 2015; Richtárik and Takáč, 2016). Mini-batch methods are often developed by generalizing classical stochastic methods to process multiple data points at a time, which helps to alleviate the communication bottleneck by enabling more distributed computation per round of communication. However, while the need to reduce communication would suggest large mini-batch sizes, the theoretical convergence rates of these methods degrade with increased mini-batch size, reverting to the rates of classical (batch) gradient methods. Empirical results corroborate these theoretical rates, and in practice, mini-batch methods have limited flexibility to adapt to the communication-computation tradeoffs that would maximally leverage parallel execution. Moreover, because mini-batch methods are typically derived from a specific single-machine solver, these methods and their associated analyses are often tailored to specific problem instances and can suffer both theoretically and practically when applied outside of their restricted problem setting.

In this work, we propose a framework, CoCoA[1], that addresses these two fundamental limitations. First, we allow arbitrary local solvers to be used on each machine in parallel. This allows our framework to directly incorporate state-of-the-art, application-specific single-machine solvers in the distributed setting. Second, we share information between machines in our framework with a highly flexible communication scheme. This allows the amount of communication to be easily tailored to the problem and system at hand, in particular allowing for the case of significantly reduced communication in the distributed environment.

A key step in providing these features in our framework is to first define meaningful subproblems for each machine to solve in parallel, and to then combine updates from the subproblems in an efficient manner. Our method and convergence results rely on noting that, depending on the distribution of the data (e.g., by feature or by training point), and whether we solve the problem in the primal or the dual, certain machine learning objectives can be

1. CoCoA-v1 (Jaggi et al., 2014) and CoCoA$^+$ (Ma et al., 2015a,b) are predecessors of this work. We continue to use the name CoCoA for the more general framework proposed here, and show how earlier work can be derived as a special case (Section 4). Portions of this newer work appear in SF's Master's Thesis (Forte, 2015) and Smith et al. (2015).

more easily decomposed into subproblems in the distributed environment. In particular, we categorize common machine learning objectives into several cases, and use duality to help decompose these objectives. Using primal-dual information in this manner not only allows for highly efficient methods (achieving, e.g., up to 50x speedups compared to state-of-the-art distributed methods), but also allows for strong primal-dual convergence guarantees and practical benefits such as computation of the duality gap for use as an accuracy certificate and stopping criterion.

1.1 Contributions

General framework. We develop a communication-efficient primal-dual framework that is applicable to a broad class of convex optimization problems. Notably, in contrast to earlier work of Yang (2013); Jaggi et al. (2014); Ma et al. (2015a); and Ma et al. (2015b), our generalized, cohesive framework: (1) specifically incorporates difficult cases of L_1 regularization and other non-strongly convex regularizers; (2) allows for the flexibility of distributing the data by either feature or training point; and (3) can be run on either a primal or dual formulation, which we show to have significant theoretical and practical implications.

Flexible communication and local solvers. Two key advantages of the proposed framework are its communication efficiency and ability to employ off-the-shelf single-machine solvers internally. On real-world systems, the cost of communication versus computation can vary widely, and it is thus advantageous to permit a flexible amount of communication depending on the setting at hand. Our framework provides exactly such control. Moreover, we allow arbitrary solvers to be used on each machine, which permits the reuse of existing code and the benefits from multi-core or other optimizations therein.

Primal-dual rates. We derive convergence rates for our framework, leveraging a novel approach in the analysis of primal-dual rates for non-strongly convex regularizers. The proposed technique is a significant improvement over simple smoothing techniques used in, e.g., Nesterov (2005); Shalev-Shwartz and Zhang (2014); and Zhang and Lin (2015) that enforce strong convexity by adding a small L_2 term to the objective. Our results include primal-dual rates and certificates for the general class of linear regularized loss minimization, and we show how earlier work can be derived as a special case of our more general approach.

Experimental comparison. The proposed framework yields order-of-magnitude speedups (as much as 50$\times$ faster) compared to state-of-the-art methods for large-scale machine learning. We demonstrate these performance gains with an extensive experimental comparison on real-world distributed datasets. We additionally explore properties of the framework itself, including the effect of running the framework in the primal or the dual. All algorithms for comparison are implemented in Apache Spark and run on Amazon EC2 clusters. Our code is open-source and publicly available at: `github.com/gingsmith/proxcocoa`.

2. Background and Setup

In this paper we develop a general framework for minimizing problems of the following form:

$$\ell(\mathbf{u}) + r(\mathbf{u})\,, \tag{I}$$

for convex functions ℓ and r. Frequently the first term ℓ is an empirical loss over the data, taking the form $\sum_i \ell_i(\mathbf{u})$, and the second term r is a regularizer, e.g., $r(\mathbf{u}) = \lambda\|\mathbf{u}\|_p$. This formulation includes many popular methods in machine learning and signal processing, such as support vector machines, linear and logistic regression, lasso and sparse logistic regression, and many others.

2.1 Definitions

The following standard definitions will be used throughout the paper.

Definition 1 (L-Lipschitz Continuity)**.** *A function $h : \mathbb{R}^m \to \mathbb{R}$ is* L-Lipschitz continuous *if $\forall \mathbf{u}, \mathbf{v} \in \mathbb{R}^m$, we have*

$$|h(\mathbf{u}) - h(\mathbf{v})| \leq L\|\mathbf{u} - \mathbf{v}\|\,. \tag{1}$$

Definition 2 (L-Bounded Support)**.** *A function $h : \mathbb{R}^m \to \mathbb{R} \cup \{+\infty\}$ has* L-bounded support *if its effective domain is bounded by L, i.e.,*

$$h(\mathbf{u}) < +\infty \;\Rightarrow\; \|\mathbf{u}\| \leq L\,. \tag{2}$$

Definition 3 ($(1/\mu)$-Smoothness)**.** *A function $h : \mathbb{R}^m \to \mathbb{R}$ is* $(1/\mu)$-smooth *if it is differentiable and its derivative is $(1/\mu)$-Lipschitz continuous, or equivalently*

$$h(\mathbf{u}) \leq h(\mathbf{v}) + \langle \nabla h(\mathbf{v}), \mathbf{u} - \mathbf{v}\rangle + \frac{1}{2\mu}\|\mathbf{u} - \mathbf{v}\|^2 \qquad \forall \mathbf{u}, \mathbf{w} \in \mathbb{R}^m\,. \tag{3}$$

Definition 4 (μ-Strong Convexity)**.** *A function $h : \mathbb{R}^m \to \mathbb{R}$ is* μ-strongly convex *for $\mu \geq 0$ if*

$$h(\mathbf{u}) \geq h(\mathbf{v}) + \langle s, \mathbf{u} - \mathbf{v}\rangle + \frac{\mu}{2}\|\mathbf{u} - \mathbf{v}\|^2 \qquad \forall \mathbf{u}, \mathbf{v} \in \mathbb{R}^m\,, \tag{4}$$

for any $s \in \partial h(\mathbf{v})$, where $\partial h(\mathbf{v})$ denotes the subdifferential of h at $\mathbf{v}$.

2.2 Primal-Dual Setting

Numerous methods have been proposed to solve (I), and these methods generally fall into two categories: *primal methods*, which run directly on the primal objective, and *dual methods*, which instead run on the dual formulation of the primal objective. In developing our framework, we present an abstraction that allows for either a primal or a dual variant of our framework to be run. In particular, to solve the input problem (I), we consider mapping the problem to one of the following two general problems:

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^n} \quad \left[\quad \mathcal{O}_A(\boldsymbol{\alpha}) := \quad f(A\boldsymbol{\alpha}) \;+\; g(\boldsymbol{\alpha}) \qquad \right] \tag{A}$$

$$\min_{\mathbf{w} \in \mathbb{R}^d} \quad \left[\quad \mathcal{O}_B(\mathbf{w}) := \quad f^*(\mathbf{w}) \;+\; g^*(-A^\top \mathbf{w}) \qquad \right] \tag{B}$$

Here $\boldsymbol{\alpha} \in \mathbb{R}^n$ and $\mathbf{w} \in \mathbb{R}^d$ are parameter vectors, $A := [\mathbf{x}_1; \ldots; \mathbf{x}_n] \in \mathbb{R}^{d \times n}$ is a data matrix with column vectors $\mathbf{x}_i \in \mathbb{R}^d$, $i \in \{1, \ldots, n\}$, and the functions f^* and g_i^* are the *convex conjugates* of f and g_i, respectively.

The dual relationship in problems (A) and (B) is known as Fenchel-Rockafellar duality (Borwein and Zhu, 2005, Theorem 4.4.2). We provide a self-contained derivation of the duality in Appendix B. Note that while dual problems are typically presented as a pair of (min, max) problems, we have equivalently reformulated (A) and (B) to both be minimization problems in accordance with their roles in our framework.

Given $\boldsymbol{\alpha} \in \mathbb{R}^n$ in the context of (A), a corresponding vector $\mathbf{w} \in \mathbb{R}^d$ for problem (B) is obtained by:

$$\mathbf{w} = \mathbf{w}(\boldsymbol{\alpha}) := \nabla f(A\boldsymbol{\alpha})\,. \tag{5}$$

This mapping arises from first-order optimality conditions on the f-part of the objective. The duality gap, given by:

$$G(\boldsymbol{\alpha}) := \mathcal{O}_A(\boldsymbol{\alpha}) - [-\mathcal{O}_B(\mathbf{w}(\boldsymbol{\alpha}))] \tag{6}$$

is always non-negative, and under strong duality, the gap will reach zero only for an optimal pair $(\boldsymbol{\alpha}^\star, \mathbf{w}^\star)$. The duality gap at any point provides a practically computable upper bound on the unknown primal as well as dual optimization error (suboptimality), since

$$\mathcal{O}_A(\boldsymbol{\alpha}) \geq \mathcal{O}_A(\boldsymbol{\alpha}^\star) \geq -\mathcal{O}_B(\mathbf{w}^\star) \geq -\mathcal{O}_B(\mathbf{w}(\boldsymbol{\alpha}))\ .$$

In developing the proposed framework, noting the duality between (A) and (B) has many benefits, including the ability to compute the duality gap, which acts as a certificate of the approximation quality. It is also useful as an analysis tool, helping us to present a cohesive framework and relate this work to the prior work of Yang (2013); Jaggi et al. (2014); and Ma et al. (2015b,a). As a word of caution, note that we avoid prescribing the name "primal" or "dual" directly to either of the problems (A) or (B), as we demonstrate below that their role as primal or dual can change depending on the application problem of interest.

2.3 Assumptions and Problem Cases

Our main assumptions on problem (A) are that f is $(1/\tau)$-smooth, and the function g is separable, i.e., $g(\boldsymbol{\alpha}) = \sum_i g_i(\alpha_i)$, with each g_i having L-bounded support. Given the duality between the problems (A) and (B), this can be equivalently stated as assuming that in problem (B), f^* is τ-strongly convex, and the function $g^*(-A^\top \mathbf{w}) = \sum_i g_i^*(-\mathbf{x}_i^\top \mathbf{w})$ is separable with each g_i^* being L-Lipschitz.

For clarity, in Table 1 we relate our assumptions on objectives (A) and (B) to the general input problem (I). Suppose, as in equation (I), we would like to find a minimizer of the general objective $\ell(\mathbf{u}) + r(\mathbf{u})$. Depending on the smoothness of the function ℓ and the strong convexity of the function r, we will be able to map the input function (I) to one (or both) of the objectives (A) and (B) based on our assumptions.

In particular, we outline three separate cases: Case I, in which the function ℓ is smooth and the function r is strongly convex; case II, in which ℓ is smooth, and r is non-strongly convex and separable; and case III, in which ℓ is non-smooth and separable, and r is strongly convex. The union of these cases will capture most commonly-used applications of linear regularized loss minimization problems. In Section 3, we will see that different variants of our framework may be realized depending on which of these three cases we consider when solving the input problem (I).

Table 1: Criteria for Objectives (A) and (B).

	Smooth ℓ	Non-smooth, separable ℓ
Strongly convex r	Case I: Obj (A) or (B)	Case III: Obj (B)
Non-strongly convex, separable r	Case II: Obj (A)	–

2.4 Running Examples

To illustrate the three cases in Table 1, we consider several examples below. These applications will serve as running examples throughout the paper, and we will revisit them in our experiments (Section 6). Further applications and details are provided in Section 5.

1. *Elastic Net Regression (Case I: map to either (A) or (B)).* We can map elastic-net regularized least squares regression,

$$\min_{\mathbf{u}\in\mathbb{R}^p} \ \tfrac{1}{2}\|A\mathbf{u}-\mathbf{b}\|_2^2 + \eta\lambda\|\mathbf{u}\|_1 + (1-\eta)\frac{\lambda}{2}\|\mathbf{u}\|_2^2\,, \tag{7}$$

   to either objective (A) or (B). To map to objective (A), we let: $f(A\boldsymbol{\alpha}) = \frac{1}{2}\|A\boldsymbol{\alpha}-\mathbf{b}\|_2^2$ and $g(\boldsymbol{\alpha}) = \sum_i g_i(\alpha_i) = \sum_i \eta\lambda|\alpha_i| + (1-\eta)\frac{\lambda}{2}\alpha_i^2$, setting n to be the number of features and d the number of training points. To map to (B), we let: $g(-A^\top\mathbf{w}) = \sum_i g_i^*(-\mathbf{x}_i^\top\mathbf{w}) = \sum_i \frac{1}{2}(\mathbf{x}_i^\top\mathbf{w}-\mathbf{b}_i)^2$ and $f^*(\mathbf{w}) = \eta\lambda\|\mathbf{w}\|_1 + (1-\eta)\frac{\lambda}{2}\|\mathbf{w}\|_2^2$, setting d to be the number of features and n the number of training points. We discuss in Section 3 how the choice of mapping elastic net regression to either (A) or to (B) will result in one of two variants of our framework, and can have implications on the distribution scheme and overall performance of the method.

2. *Lasso (Case II: map to (A)).* We can represent L_1-regularized least squares regression by mapping the model:

$$\min_{\mathbf{u}\in\mathbb{R}^p} \ \tfrac{1}{2}\|A\mathbf{u}-\mathbf{b}\|_2^2 + \lambda\|\mathbf{u}\|_1 \tag{8}$$

   to objective (A), letting $f(A\boldsymbol{\alpha}) = \frac{1}{2}\|A\boldsymbol{\alpha}-\mathbf{b}\|_2^2$ and $g(\boldsymbol{\alpha}) = \sum_i g_i(\alpha_i) = \sum_i \lambda|\alpha_i|$. In this mapping, n represents the number of features, and d the number of training points. Note that we cannot map the lasso objective to (B) directly, as f^* must be τ-strongly convex and the L_1-norm is non-strongly convex.

3. *Support Vector Machine (Case III: map to (B)).* We can represent a hinge loss support vector machine (SVM) by mapping the model:

$$\min_{\mathbf{u}\in\mathbb{R}^p} \ \frac{1}{m}\sum_{i=1}^{m} \max\left\{0, 1-y_i(\mathbf{x}_i^\top\mathbf{u})\right\} + \tfrac{\lambda}{2}\|\mathbf{u}\|_2^2\,, \tag{9}$$

   to objective (B), letting $g^*(-A^\top\mathbf{w}) = \sum_i g_i^*(-\mathbf{x}_i^\top\mathbf{w}) = \sum_i \frac{1}{n}\max\{0, 1-y_i\mathbf{x}_i^\top\mathbf{w}\}$ and $f^*(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2$. In this mapping, d represents the number of features, and n the number of training points. Note that we cannot map the hinge loss SVM primal to objective (A) directly, as f must be $(1/\tau)$-smooth and the hinge loss is non-smooth.

2.5 Data Partitioning

To view our setup in the distributed environment, we suppose that the dataset A is distributed over K machines according to a partition $\{\mathcal{P}_k\}_{k=1}^K$ of the *columns* of $A \in \mathbb{R}^{d\times n}$. We denote the size of the partition on machine k by $n_k = |\mathcal{P}_k|$. For machine $k \in \{1,\dots,K\}$ and weight vector $\boldsymbol{\alpha} \in \mathbb{R}^n$, we define $\boldsymbol{\alpha}_{[k]} \in \mathbb{R}^n$ as the n-vector with elements $(\boldsymbol{\alpha}_{[k]})_i := \alpha_i$ if $i \in \mathcal{P}_k$ and $(\boldsymbol{\alpha}_{[k]})_i := 0$ otherwise. Analogously, we write $A_{[k]}$ for the corresponding group of columns of A, and zeros elsewhere (note that columns can correspond to either training examples or features, depending on the application). We discuss these distribution schemes in greater detail in Section 3.

3. CoCoA

We first describe the proposed framework, CoCoA, at a high level, and then discuss two approaches for using the framework in practice: CoCoA in the primal, where we consider (A) to be the primal objective and run the framework on this problem directly, and CoCoA in the dual, where we instead consider (B) to be the primal objective, and then run the framework on the dual (A). Note that in both approaches, the aim will be to compute a minimizer of the problem (A) in a distributed fashion; the main difference will be whether we view (A) as the primal objective or as the dual objective.

3.1 The Generalized Framework

The goal of our framework is to find a global minimizer of the objective (A), while distributing computation based on the partitioning of the dataset A across machines (Section 2.5). As a first step, note that distributing the update to the function g in objective (A) is straightforward, as we have required that this term is separable according to the partitioning of our data, i.e., $g(\boldsymbol{\alpha}) = \sum_{i=1}^n g_i(\alpha_i)$. However, the same does not hold for the term $f(A\boldsymbol{\alpha})$. To minimize this part of the objective in a distributed fashion, we propose minimizing a quadratic approximation of the function, which allows the minimization to separate across machines. We make this approximation precise in the following subsection.

Data-local quadratic subproblems. In the general CoCoA framework (Algorithm 1), we distribute computation by defining a data-local subproblem of the optimization problem (A) for each machine. This simpler problem can be solved on machine k and only requires accessing data which is already available locally, i.e., the columns $A_{[k]}$. More formally, each machine k is assigned the following local subproblem, which depends only on the previous shared vector $\mathbf{v} := A\boldsymbol{\alpha} \in \mathbb{R}^d$, and the local data $A_{[k]}$:

$$\min_{\Delta\boldsymbol{\alpha}_{[k]}\in\mathbb{R}^n} \ \mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]}; \mathbf{v}, \boldsymbol{\alpha}_{[k]}) \,, \tag{10}$$

where

$$\mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]}; \mathbf{v}, \boldsymbol{\alpha}_{[k]}) := \frac{1}{K} f(\mathbf{v}) + \mathbf{w}^\top A_{[k]}\Delta\boldsymbol{\alpha}_{[k]} + \frac{\sigma'}{2\tau}\left\|A_{[k]}\Delta\boldsymbol{\alpha}_{[k]}\right\|^2 + \sum_{i\in\mathcal{P}_k} g_i(\boldsymbol{\alpha}_i + \Delta\boldsymbol{\alpha}_{[k]_i}),$$

and $\mathbf{w} := \nabla f(\mathbf{v})$. Here we let $\Delta\boldsymbol{\alpha}_{[k]}$ denote the change of local variables α_i for indices $i \in \mathcal{P}_k$, and we set $(\Delta\boldsymbol{\alpha}_{[k]})_i := 0$ for all $i \notin \mathcal{P}_k$. It is important to note that the subproblem (10)

Algorithm 1 Generalized CoCoA Distributed Framework

Input: Data matrix A distributed column-wise according to partition $\{\mathcal{P}_k\}_{k=1}^K$, aggregation parameter $\gamma \in (0,1]$, and parameter σ' for the local subproblems $\mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]}; \mathbf{v}, \boldsymbol{\alpha}_{[k]})$. Starting point $\boldsymbol{\alpha}^{(0)} := \mathbf{0} \in \mathbb{R}^n$, $\mathbf{v}^{(0)} := \mathbf{0} \in \mathbb{R}^d$.
for $t = 0, 1, 2, \ldots$ **do**
  for $k \in \{1, 2, \ldots, K\}$ **in parallel over computers do**
    call local solver, returning a Θ-approximate solution $\Delta\boldsymbol{\alpha}_{[k]}$ of the local subproblem (10)
    update local variables $\boldsymbol{\alpha}_{[k]}^{(t+1)} := \boldsymbol{\alpha}_{[k]}^{(t)} + \gamma\, \Delta\boldsymbol{\alpha}_{[k]}$
    return updates to shared state $\Delta\mathbf{v}_k := A_{[k]}\Delta\boldsymbol{\alpha}_{[k]}$
  end for
  reduce $\mathbf{v}^{(t+1)} := \mathbf{v}^{(t)} + \gamma \sum_{k=1}^K \Delta\mathbf{v}_k$
end for

is simple in the sense that it is always a quadratic objective (apart from the g_i term). The subproblem does not depend on the function f itself, but only its linearization at the fixed shared vector $\mathbf{v}$. This property additionally simplifies the task of the local solver, especially for cases of complex functions f.

Framework parameters γ and σ'. There are two parameters that must be set in our framework: γ, the aggregation parameter, which controls how the updates from each machine are combined, and σ', the subproblem parameter, which is a data-dependent term measuring the difficulty of the data partitioning $\{\mathcal{P}_k\}_{k=1}^K$. These terms play a crucial role in the convergence of the method, as we demonstrate in Section 4. In practice, we provide a simple and robust way to set these parameters: For a given aggregation parameter $\gamma \in (0,1]$, the subproblem parameter σ' will be set as $\sigma' := \gamma K$, but can also be improved in a data-dependent way as we discuss below. In general, as we show in Section 4, setting $\gamma := 1$ and $\sigma' := K$ will guarantee convergence while delivering our fastest convergence rates.

Definition 5 (Data-dependent aggregation parameter)**.** *In Algorithm 1, the aggregation parameter γ controls the level of adding ($\gamma := 1$) versus averaging ($\gamma := \frac{1}{K}$) of the partial solutions from all machines. For our convergence results (Section 4) to hold, the subproblem parameter σ' must be chosen not smaller than*

$$\sigma' \geq \sigma'_{min} := \gamma \max_{\boldsymbol{\alpha} \in \mathbb{R}^n} \frac{\|A\boldsymbol{\alpha}\|^2}{\sum_{k=1}^K \|A_{[k]}\boldsymbol{\alpha}_{[k]}\|^2}\,. \tag{11}$$

The simple choice of $\sigma' := \gamma K$ is valid for (11), i.e.,

$$\gamma K \geq \sigma'_{min}\,.$$

In some cases, it will be possible to give a better (data-dependent) choice for σ', closer to the actual bound given in σ'_{min}.

Subproblem Interpretation. Here we provide further intuition behind the data-local subproblems (10). The local objective functions $\mathcal{G}_k^{\sigma'}$ are defined to closely approximate the global objective in (A) as the "local" variable $\Delta\boldsymbol{\alpha}_{[k]}$ varies, which we will see in the analysis (Appendix D, Lemma 1). In fact, if the subproblem were solved exactly, this could be interpreted as a data-dependent, block-separable proximal step, applied to the f part of the objective (A) as follows:

$$\sum_{k=1}^{K} \mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]}; \mathbf{v}, \boldsymbol{\alpha}_{[k]}) \;=\; R + f(\mathbf{v}) + \nabla f(\mathbf{v})^\top A\Delta\boldsymbol{\alpha} + \frac{\sigma'}{2\tau}\Delta\boldsymbol{\alpha}^\top \begin{bmatrix} A_{[1]}^\top A_{[1]} & & 0 \\ & \ddots & \\ 0 & & A_{[K]}^\top A_{[K]} \end{bmatrix} \Delta\boldsymbol{\alpha}\,,$$

where $R = \sum_{i\in[n]} g_i(-\boldsymbol{\alpha}_i - \Delta\boldsymbol{\alpha}_i)$.

However, note that in contrast to traditional proximal methods, our algorithm does *not* assume that this subproblem is solved to high accuracy, as we instead allow the use of local solvers of any approximation quality Θ.

Reusability of existing single-machine solvers. Our local subproblems (10) have the appealing property of being very similar in structure to the global problem (A), with the main difference being that they are defined on a smaller (local) subset of the data, and are simpler because they are not dependent on the shape of f. For a user of CoCoA, this presents a major advantage in that existing single machine-solvers can be directly re-used in our distributed framework (Algorithm 1) by employing them on the subproblems $\mathcal{G}_k^{\sigma'}$.

Therefore, problem-specific tuned solvers which have already been developed, along with associated speed improvements (such as multi-core implementations), can be easily leveraged in the distributed setting. We quantify the dependence on local solver performance with the following assumption and remark, and relate this performance to our global convergence rates in Section 4.

Assumption 1 (Θ-approximate solution)**.** *We assume that there exists $\Theta \in [0, 1)$ such that $\forall k \in [K]$, the local solver at any outer iteration t produces a (possibly) randomized approximate solution $\Delta\boldsymbol{\alpha}_{[k]}$, which satisfies*

$$\mathbb{E}\big[\mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]}; \mathbf{v}, \boldsymbol{\alpha}_{[k]}) - \mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]}^\star; \mathbf{v}, \boldsymbol{\alpha}_{[k]})\big] \leq \Theta\left(\mathcal{G}_k^{\sigma'}(\mathbf{0}; \mathbf{v}, \boldsymbol{\alpha}_{[k]}) - \mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]}^\star; \mathbf{v}, \boldsymbol{\alpha}_{[k]})\right), \tag{12}$$

where

$$\Delta\boldsymbol{\alpha}_{[k]}^\star \in \arg\min_{\Delta\boldsymbol{\alpha}\in\mathbb{R}^n} \; \mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]}; \mathbf{v}, \boldsymbol{\alpha}_{[k]}), \;\; \forall k \in [K]\,. \tag{13}$$

Remark 1. *In practice, the time spent solving the local subproblems in parallel should be chosen comparable to the required time of a communication round, for best overall efficiency on a given system. We study this trade-off both in theory (Section 4) and experiments (Section 6).*

Remark 2. *Note that the accuracy parameter Θ does not have to be chosen a priori: Our convergence results (Section 4) are valid if Θ is an upper bound on the actual empirical values Θ in the rounds of Algorithm 1. This allows for some of the K machines to at times deliver better or worse accuracy (e.g., if a slow local machine is stopped early during a specific round, to avoid the others needing to wait).*

With this general framework in place, we next discuss two variants of our framework, CoCoA-Primal and CoCoA-Dual. In running either the primal or dual variant of our framework, the goal will always be to solve objective (A) in a distributed fashion. The main difference will be whether this objective is viewed as the primal or dual of the input problem (I). If we map the input (I) to objective (A), then (A) will be viewed as the primal. If we map (I) to (B), the objective (A) will be viewed as the dual. We make this mapping technique precise and discuss its implications in the following subsections (Sections 3.2–3.4).

3.2 Primal Distributed Optimization

In the primal distributed version of the framework (Algorithm 2), the framework is run by mapping the initial problem (I) directly to objective (A) and then applying the generalized CoCoA framework described in Algorithm 1. In other words, we view problem (A) as the primal objective, and solve this problem directly.

From a theoretical perspective, viewing (A) as the primal will allow us to consider non-strongly convex regularizers, since we allow the terms g_i to be non-strongly convex. This setting was not covered in earlier work of Yang (2013); Jaggi et al. (2014); Ma et al. (2015b); and Ma et al. (2015a), and we discuss it in detail in Section 4, as additional machinery must be introduced to develop primal-dual rates for this setting.

Running the primal version of the framework has important practical implications in the distributed setting, as it typically implies that the data is distributed by feature rather than by training point. In this setting, the amount of communication at every outer iteration will be $O(\#$ of training points). When the number of features is high (as is common when using sparsity-inducing regularizers) this can help to reduce communication and improve overall performance, as we demonstrate in Section 6.

Algorithm 2 CoCoA-Primal (Mapping Problem (I) to (A))

1: **Map:** Input problem (I) to objective (A)
2: **Distribute:** Dataset A by columns (here typically features) according to partition $\{\mathcal{P}_k\}_{k=1}^K$
3: **Run:** Algorithm 1 with aggregation parameter γ and subproblem parameter σ'

3.3 Dual Distributed Optimization

In the dual distributed version of the framework (Algorithm 3), we run the framework by mapping the original problem (I) to objective (B), and then solve the problem by running Algorithm 1 on the dual (A). In other words, we view problem (B) as the primal, and solve this problem via the dual (A).

This version of the framework will allow us to consider non-smooth losses, such as the hinge loss or absolute deviation loss, since the terms g_i^* can be non-smooth. From a practical perspective, this version of the framework will typically imply that the data is distributed by training point, and for a vector $O(\#$ of features) to be communicated at every outer iteration. This variant may therefore be preferable when the number of training points exceeds the number of features.

Algorithm 3 CoCoA-Dual (Mapping Problem (I) to (B))

1: **Map:** Input problem (I) to objective (B)
2: **Distribute:** Dataset A by columns (here typically training points) according to partition $\{\mathcal{P}_k\}_{k=1}^K$
3: **Run:** Algorithm 1 with aggregation parameter γ and subproblem parameter σ'

3.4 Primal vs. Dual

In Table 2, we revisit the three cases from Section 2, showing how the primal and dual variants of CoCoA can be applied to various input problems $\ell(\mathbf{u}) + r(\mathbf{u})$, depending on properties of the functions ℓ and r. In particular, in the setting where ℓ is smooth and r is strongly convex, the user may choose whether to run the framework in the primal (Algorithm 2), or in the dual (Algorithm 3). Intuitively, Algorithm 2 will be preferable as r loses strong convexity, and Algorithm 3 will be preferable as ℓ loses smoothness. However, there are also systems-related aspects to consider. In Algorithm 2, we typically distribute the data by feature, and in Algorithm 3, by training point (this distribution depends on how the terms n and d are defined in our mapping, see Section 5). Depending on whether the number of features or number of training points is the dominating term, we may chose to run Algorithm 2 or Algorithm 3, respectively, in order to reduce communication costs. We validate these ideas empirically in Section 6 by comparing the performance of each variant (primal vs. dual) on real distributed datasets.

Table 2: Criteria for Running Algorithms 2 vs. 3.

	Smooth ℓ	Non-smooth and separable ℓ
Strongly convex r	Case I: Alg.2 or 3	Case III: Alg.3
Non-strongly convex and separable r	Case II: Alg.2	–

In the following two subsections, we provide greater insight into the form of the generalized CoCoA framework and its relation to prior work. An extended discussion on related work is available in Section 7.

3.5 Interpretation of CoCoA in the Context of Classical Parallelization Schemes

There are numerous methods that have been developed to solve (A) and (B) in parallel and distributed environments. We describe related work in detail in Section 7, and here briefly highlight a major algorithmic difference between CoCoA and other widely-used parallelized methods. In particular, we contrast CoCoA with mini-batch and batch methods commonly used in distributed computing environments, such as mini-batch stochastic gradient descent or coordinate descent, gradient descent, and quasi-Newton methods.

CoCoA is similar to these methods in that they are all *iterative*, i.e., they make progress towards the optimal solution by updating the parameter vector $\boldsymbol{\alpha}$ according to some function

$h : \mathbb{R}^n \to \mathbb{R}^n$ at each iteration t:

$$\boldsymbol{\alpha}^{(t+1)} = h(\boldsymbol{\alpha}^{(t)}) \quad t = 0, 1, \ldots,$$

until convergence is reached. From a coordinate-wise perspective, two approaches for updating the parameter vector $\boldsymbol{\alpha}$ in an iterative fashion include the Jacobi method, in which updates made to coordinates of $\boldsymbol{\alpha}$ do not take into account the most recent updates to the other coordinates, and Gauss-Seidel, in which the most recent information is used (Bersekas and Tsitsiklis, 1989). In particular, these two paradigms make the following updates to a coordinate i at iteration $t + 1$:

$$\begin{aligned} \text{Jacobi:} \quad & \boldsymbol{\alpha}_i^{(t+1)} = h_i(\boldsymbol{\alpha}_1^{(t)}, \ldots, \boldsymbol{\alpha}_n^{(t)}), \quad i = 1, \ldots, n, \\ \text{Gauss-Seidel:} \quad & \boldsymbol{\alpha}_i^{(t+1)} = h_i(\boldsymbol{\alpha}_1^{(t+1)}, \ldots, \boldsymbol{\alpha}_{i-1}^{(t+1)}, \boldsymbol{\alpha}_i^{(t)}, \ldots, \boldsymbol{\alpha}_n^{(t)}), \quad i = 1, \ldots, n. \end{aligned}$$

The Jacobi method does not require information from the other coordinates to update coordinate i, which makes this style of method well-suited for parallelization. However, the Gauss-Seidel style method tends to converge faster in terms of iterations, since it is able to incorporate information from the other coordinates more quickly. This difference is well-known and evident in single machine solvers, where stochastic methods (benefiting from fresh updates) tend to outperform their batch counterparts.

Typical mini-batch methods, e.g., mini-batch coordinate descent, perform a Jacobi-style update on a subset of the coordinates at each iteration. This makes these methods amenable to high levels of parallelization. However, they are unable to incorporate information as quickly as their serial counterparts in terms of number of data points accessed, because they must wait for a synchronization step to update the coordinates. As the size of the mini-batch grows, this can slow them down in terms of overall runtime, and can even lead to divergence in practice (Takáč et al., 2013; Takáč et al., 2015; Richtárik and Takáč, 2016; Marecek et al., 2015).

CoCoA instead attempts to combine attractive properties of both of these update paradigms. It performs Jacobi-style parallel updates to *blocks* of the coordinates of $\boldsymbol{\alpha}$ to parallelize the method, while allowing for (though not necessarily requiring) faster Gauss-Seidel style updates on each machine. This change in parallelization scheme is one of the major reasons for improved performance over simpler mini-batch or batch style methods.

CoCoA incorporates an additional level of flexibility by allowing an *arbitrary number* of Gauss-Seidel iterations (or any other local solver for that matter) to be performed on each machine, which lets the framework scale from very low-communication environments, where more iterations will be made before communicating, to higher communication environments, where fewer internal iterations are necessary. We will see in Section 6 that this communication flexibility also greatly improves the overall runtime in practice.

3.6 Comparison to ADMM

Finally, in this subsection we provide a direct comparison between CoCoA and ADMM (Boyd et al., 2010). Alternating direction method of multipliers (ADMM) is a well-established framework for distributed optimization. Similar to CoCoA, ADMM differs from the methods discussed in the previous section in that it defines a subproblem for each problem to solve

in parallel, rather than parallelizing a global batch or mini-batch update. It also leverages duality structure, similar to that presented in Section 2.

For consensus ADMM, the objective (B) is decomposed with a re-parameterization:

$$\max_{\mathbf{w}_1,\dots\mathbf{w}_K,\mathbf{w}} \quad \sum_{k=1}^{K}\sum_{i\in\mathcal{P}_k} g^*(-\mathbf{x}_i^\top \mathbf{w}_k) + f^*(\mathbf{w})$$
$$s.t. \quad \mathbf{w}_k = \mathbf{w},\ k = 1,\dots,K.$$

This problem is then solved by constructing the augmented Lagrangian, which yields the following decomposable updates:

$$\mathbf{w}_k^{(t)} = \arg\min_{\mathbf{w}_k} \sum_{i\in\mathcal{P}_k} g^*(-\mathbf{x}_i^\top \mathbf{w}_k) + \frac{\rho}{2}\|\mathbf{w}_k - \left(\mathbf{w}^{(t-1)} - \mathbf{u}_k^{(t-1)}\right)\|^2, \tag{14}$$
$$\mathbf{w}^{(t)} = \arg\min_{\mathbf{w}} \ f^*(\mathbf{w}) + \rho\sum_{k=1}^{K}\mathbf{u}_k^\top(\mathbf{w}_k - \mathbf{w}) + \frac{\rho}{2}\sum_{k=1}^{K}\|\mathbf{w}_k - \mathbf{w}\|^2,$$
$$\mathbf{u}_k^{(t)} = \mathbf{u}_k^{(t-1)} + \mathbf{w}_k^{(t)} - \mathbf{w}^{(t)},$$

where ρ is a penalty parameter that must be tuned for best performance. When running CoCoA in the dual (Algorithm 3) and setting $f(\cdot) = \frac{1}{2}\|\cdot\|_2^2$, we can derive a similar subproblem for updating $\mathbf{w}_k$ in the CoCoA framework. In particular, the following subproblem can be found by unrolling the CoCoA update and viewing the dual subproblem in its primal formulation:

$$\min_{\mathbf{w}_k} \sum_{i\in\mathcal{P}_k} g_i^*(-\mathbf{x}_i^\top \mathbf{w}_k) + \frac{\tau}{2\sigma'}\left\|\mathbf{w}_k - \left(\mathbf{w}^{(t-1)} + \gamma\Delta\mathbf{v}^{(t-1)}\right)\right\|^2. \tag{15}$$

Comparing (14) and (15) we can see that in the specific case where $f(\cdot) = \frac{1}{2}\|\cdot\|_2^2$ and we solve the problem in the dual (according to Algorithm 3), ADMM and CoCoA consider a similar subproblem on each machine, but where the parameter ρ is explicitly set in CoCoA as $\frac{\tau}{\sigma'}$. However, there are major differences between the methods even in this setting. First, CoCoA has a more direct and simplified scheme for updating the global weight vector $\mathbf{w}$. Second, and most importantly, in the CoCoA method and theory, we allow for the subproblem to be solved approximately, rather than requiring a full batch update as in ADMM. We will see in our experiments that these differences have a large impact in practice (Section 6). We provide a full derivation of the comparison to ADMM for reference in Appendix C.

4. Convergence Analysis

In this section, we provide convergence rates for the proposed framework and introduce an important theoretical technique in analyzing non-strongly convex terms in the primal-dual setting. For simplicity of presentation, we assume in the analysis that the data partitioning is balanced; i.e., $n_k = n/K$ for all k. Furthermore, we assume that the columns of A satisfy $\|\mathbf{x}_i\| \le 1$ for all $i \in [n]$. We present rates for the case where $\gamma := 1$ in Algorithm 1, and where the subproblems (10) are defined using the corresponding safe bound $\sigma' := K$. This

case will guarantee convergence while delivering our fastest rates in the distributed setting, which in particular don't degrade as the number of machines K grows and n remains fixed.

4.1 Proof Strategy: Relating Subproblem Approximation to Global Progress

To guarantee convergence, it is critical to show how progress made on the local subproblems (10) relates to the global objective $\mathcal{O}_A$. Our first lemma provides exactly this information. In particular, we see that if the aggregation and subproblem parameters are selected according to Definition 5, the sum of the subproblem objectives, $\sum_{k=1}^K \mathcal{G}_k^{\sigma'}$, will form a block-separable upper bound on the global objective $\mathcal{O}_A$.

Lemma 1. *For any weight vector $\boldsymbol{\alpha}, \Delta\boldsymbol{\alpha} \in \mathbb{R}^n$, $\mathbf{v} = \mathbf{v}(\boldsymbol{\alpha}) := A\boldsymbol{\alpha}$, and real values γ, σ' satisfying* (11)*, it holds that*

$$\mathcal{O}_A\Big(\boldsymbol{\alpha} + \gamma \sum_{k=1}^K \Delta\boldsymbol{\alpha}_{[k]}\Big) \leq (1-\gamma)\mathcal{O}_A(\boldsymbol{\alpha}) + \gamma \sum_{k=1}^K \mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]}; \mathbf{v}, \boldsymbol{\alpha}_{[k]})\,. \tag{16}$$

A proof of Lemma 1 is provided in Appendix D. We use this main lemma, in combination with our assumption on the quality of the subproblem approximations (Assumption 1), to deliver our global convergence rates.

4.2 Rates for General Convex g_i, L-Lipschitz g_i^*

Our first main theorem provides convergence guarantees for objectives with general convex g_i (or, equivalently, L-Lipschitz g_i^*), including models with non-strongly convex regularizers such as lasso and sparse logistic regression, or models with non-smooth losses, such as the hinge loss support vector machine.

Providing primal-dual rates and globally defined primal-dual accuracy certificates for these objectives may require an important theoretical technique that we introduce below, in which we show how to satisfy the notion of L-bounded support for g_i, as stated in Definition 2.

Theorem 2. *Consider Algorithm 1 with $\gamma := 1$, and let Θ be the quality of the local solver as in Assumption 1. Let g_i have L-bounded support, and let f be $(1/\tau)$-smooth. Then after T iterations, where*

$$\begin{aligned} T &\geq T_0 + \max\{\Big\lceil \frac{1}{1-\Theta} \Big\rceil, \frac{4L^2n^2}{\tau\epsilon_G(1-\Theta)}\}\,, \\ T_0 &\geq t_0 + \Big[\frac{2}{1-\Theta}\left(\frac{8L^2n^2}{\tau\epsilon_G} - 1\right)\Big]_+\,, \\ t_0 &\geq \max(0, \Big\lceil \tfrac{1}{(1-\Theta)} \log\left(\tfrac{\tau(\mathcal{O}_A(\boldsymbol{\alpha}^{(0)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star))}{2L^2Kn}\right)\Big\rceil)\,, \end{aligned} \tag{17}$$

we have that the expected duality gap satisfies

$$\mathbb{E}\big[\mathcal{O}_A(\overline{\boldsymbol{\alpha}}) - (-\mathcal{O}_B(\mathbf{w}(\overline{\boldsymbol{\alpha}})))\big] \leq \epsilon_G\,,$$

where $\overline{\boldsymbol{\alpha}}$ is the averaged iterate: $\frac{1}{T-T_0}\sum_{t=T_0+1}^{T-1} \boldsymbol{\alpha}^{(t)}$.

4.2.1 Bounded support modification

As mentioned earlier, additional work is necessary if Theorem 2 is to be applied to non-strongly convex regularizers such as the L_1 norm, which do not have L-bounded support for each g_i, and thus violate the assumptions of the theorem. Note for example that the conjugate function of $g_i = |\cdot|$, which is the indicator function of an interval, is not defined globally over $\mathbb{R}$, and thus (without further modification) the duality gap $G(\boldsymbol{\alpha}) := \mathcal{O}_A(\boldsymbol{\alpha}) - (-\mathcal{O}_B(\mathbf{w}(\boldsymbol{\alpha})))$ is not even defined at all points $\boldsymbol{\alpha}$.

Smoothing. To address this problem, existing approaches typically use a simple smoothing technique (as in Nesterov, 2005; Shalev-Shwartz and Zhang, 2014): by adding a small amount of L_2 to the objective g_i, the functions g_i become strongly convex. Followed by this change, the algorithms are then run on the dual of instead of the original primal problem at hand. While this modification satisfies the necessary assumptions for convergence of our framework, this Nesterov smoothing technique is often undesirable in practice, as it changes the iterates, the algorithms at hand, the convergence rate, and the tightness of the resulting duality gap compared to the original objective. Further, the amount of smoothing can be difficult to tune and can have a large influence on the performance of the method at hand. We show practical examples of these difficulties in Section 6.

Bounded support modification. In contrast to smoothing, our approach preserves all solutions of the original objective, leaves the iterate sequence unchanged, and allows for direct reusability of existing solvers for the original g_i objectives (such as L_1 solvers). It also removes the need for tuning a smoothing parameter. To achieve this, we modify the function g_i by imposing an additional weak constraint that is inactive in our region of interest. Formally, we replace $g_i(\alpha_i)$ by the following modified function:

$$\bar{g}_i(\alpha_i) := \begin{cases} g_i(\alpha_i) & : \alpha_i \in [-B, B] \\ +\infty & : \text{otherwise.} \end{cases} \tag{18}$$

For large enough B, this problem yields the same solution as the original objective. Note also that this only affects convergence theory, in that it allows us to present a strong primal-dual rate (Theorem 2 for $L{=}B$). The modification of g_i does not affect the algorithms for the original problems. Whenever a monotone optimizer is used, we will never leave the level set defined by the objective at the starting point.

Using the resulting modified function will allow us to apply the results of Theorem 2 for general convex functions g_i. This technique can also be thought of as "Lipschitzing" the dual g_i^*, because of the general result that g_i^* is L-Lipschitz if and only if g_i has L-bounded support (Rockafellar, 1997, Corollary 13.3.3). We derive the conjugate function $\bar{g}_i^*$ for completeness in Appendix B (Lemma 6). In Section 5, we show how to leverage this technique for a variety of application input problems. See also Dünner et al. (2016) for a follow-up discussion of this technique in the non-distributed case.

4.3 Rates for Strongly Convex g_i, Smooth g_i^*

For the case of objectives with strongly convex g_i (or, equivalently, smooth g_i^*), e.g., elastic net regression or logistic regression, we obtain the following faster *linear* convergence rate.

Theorem 3. *Consider Algorithm 1 with $\gamma := 1$, and let Θ be the quality of the local solver as in Assumption 1. Let g_i be μ-strongly convex $\forall i \in [n]$, and let f be $(1/\tau)$-smooth. Then after T iterations where*

$$T \geq \frac{1}{(1-\Theta)} \frac{\mu\tau + n}{\mu\tau} \log \frac{n}{\epsilon_{\mathcal{O}_A}}, \tag{19}$$

it holds that

$$\mathbb{E}\big[\mathcal{O}_A(\boldsymbol{\alpha}^{(T)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)\big] \leq \epsilon_{\mathcal{O}_A}.$$

Furthermore, after T iterations with

$$T \geq \frac{1}{(1-\Theta)} \frac{\mu\tau + n}{\mu\tau} \log \left(\frac{1}{(1-\Theta)} \frac{\mu\tau + n}{\mu\tau} \frac{n}{\epsilon_G} \right),$$

we have the expected duality gap

$$\mathbb{E}\big[\mathcal{O}_A(\boldsymbol{\alpha}^{(T)}) - (-\mathcal{O}_B(\mathbf{w}(\boldsymbol{\alpha}^{(T)}))\big] \leq \epsilon_G.$$

We provide proofs of both Theorem 2 and Theorem 3 in Appendix D.

4.4 Convergence Cases

Revisiting Table 1 from Section 2, we summarize our convergence guarantees for the three cases of input problems (I) in the following table. In particular, we see that for cases II and III, we obtain a sublinear convergence rate, whereas for case I we can obtain a faster linear rate, as provided in Theorem 3.

Table 3: Applications of Convergence Rates.

	Smooth ℓ	Non-smooth, separable ℓ
Strongly convex r	Case I: Theorem 3	Case III: Theorem 2
Non-strongly convex, separable r	Case II: Theorem 2	–

4.5 Recovering Earlier Work as a Special Case

As a special case, the proposed framework and rates directly apply to L_2-regularized loss-minimization problems, including those presented in the earlier work of Jaggi et al. (2014) and Ma et al. (2015b).

Remark 3. *If we run Algorithm 3 (mapping* (I) *to* (B)*), restrict $f^*(\cdot) := \frac{\lambda}{2}\|\cdot\|^2$ (so that $\tau = \lambda$), and let $g_i^* := \frac{1}{n}\ell_i^*$, Theorem 2 recovers as a special case the* CoCoA$^+$ *rates for general L-Lipschitz ℓ_i^* losses (see Ma et al., 2015b, Corollary 9). The earlier work of* CoCoA*-v1 (Jaggi et al., 2014) did not provide rates for L-Lipschitz ℓ_i^* losses.*

These cases follow since g_i^* is L-Lipschitz if and only if g_i has L-bounded support (Rockafellar, 1997, Corollary 13.3.3).

Remark 4. *If we run Algorithm 3 (mapping* (I) *to* (B)*), restrict $f^*(\cdot) := \frac{\lambda}{2}\|\cdot\|^2$ (so that $\tau = \lambda$), and scale $g_i^* := \frac{1}{n}\ell_i^*$, Theorem 3 recovers as a special case the* CoCoA$^+$ *rates for $(1/\ell_i^*)$-smooth losses (see Ma et al., 2015b, Corollary 11). The earlier rates of* CoCoA*-v1 can be obtained by setting $\gamma{:=}\frac{1}{K}$ and $\sigma'{=}1$ in Algorithm 1 (Jaggi et al., 2014, Theorem 2).*

These cases follow since g_i^* is μ-strongly convex if and only if g_i is $(1/\mu)$-smooth (Hiriart-Urruty and Lemaréchal, 2001, Theorem 4.2.2).

5. Applications

In this section we provide a detailed treatment of example applications that can be cast within the general CoCoA framework. For each example, we describe the primal-dual setup and algorithmic details, discuss the convergence properties our framework for the application, and include practical concerns such as information on state-of-the-art local solvers. We discuss examples according to the three cases defined in Table 1 of Section 2 for finding a minimizer of the general objective $\ell(\mathbf{u}) + r(\mathbf{u})$, and provide a summary of these common examples in Table 4.

Table 4: Common Losses and Regularizers.

(i) Losses

Loss	Obj	f / g^*
Least Squares	(A)	$f{=}\frac{1}{2}\|A\boldsymbol{\alpha}-\mathbf{b}\|_2^2$
	(B)	$g^*{=}\frac{1}{2}\|A^\top\mathbf{w}-\mathbf{b}\|_2^2$
Logistic Reg.	(A)	$f{=}\frac{1}{d}\sum_j\log(1{+}\exp(b_j\mathbf{x}_j^\top\boldsymbol{\alpha}))$
	(B)	$g^*{=}\frac{1}{n}\sum_i\log(1{+}\exp(b_i\mathbf{x}_i^\top\mathbf{w}))$
SVM	(B)	$g^*{=}\frac{1}{n}\sum_i\max(0,1{-}y_i\mathbf{x}_i^\top\mathbf{w})$
Absolute Dev.	(B)	$g^*=\frac{1}{n}\sum_i\|\mathbf{x}_i^\top\mathbf{w}-y_i\|$

(ii) Regularizers

Regularizer	Obj	g / f^*
Elastic Net	(A)	$g{=}\lambda(\eta\|\boldsymbol{\alpha}\|_1{+}\frac{1-\eta}{2}\|\boldsymbol{\alpha}\|_2^2)$
	(B)	$f^*{=}\lambda(\eta\|\mathbf{w}\|_1{+}\frac{1-\eta}{2}\|\mathbf{w}\|_2^2)$
L_2	(A)	$g{=}\frac{\lambda}{2}\|\boldsymbol{\alpha}\|_2^2$
	(B)	$f^*{=}\frac{\lambda}{2}\|\mathbf{w}\|_2^2$
L_1	(A)	$g{=}\lambda\|\boldsymbol{\alpha}\|_1$
Group Lasso	(A)	$g{=}\lambda\sum_p\|\boldsymbol{\alpha}_{\mathcal{I}_p}\|_2$, $\mathcal{I}_p\subseteq[n]$

5.1 Case I: Smooth ℓ, Strongly convex r

For input problems (I) with smooth ℓ and strongly convex r, Theorem 3 from Section 4 gives a global linear (geometric) convergence rate. Smooth loss functions can be mapped either to the function f in objective (A), or g^* in (B). Similarly, strongly convex regularizers can be mapped either to function g in objective (A), or f^* in (B). To illustrate the role of f as a smooth loss function and g as a strongly convex regularizer in objective (A), contrasting with their traditional roles in prior work (Yang, 2013; Jaggi et al., 2014; Ma et al., 2015b,a), we consider the following examples. Note that mapping to objective (B) instead will follow trivially assuming that the loss is separable across training points (see Table 4).

For the examples in this subsection, we use nonstandard definitions of the number of training points as d and the number of features as n. These definitions are intentionally used so that we can present both the primal and dual variations of our framework (Algorithms 2 and 3) with a single abstracted method (Algorithm 1).

Smooth ℓ: least squares loss. Let $\mathbf{b}\in\mathbb{R}^d$ be labels or response values, and consider the least squares objective, $f(\mathbf{v}) := \frac{1}{2}\|\mathbf{v}-\mathbf{b}\|_2^2$, which is 1-smooth. We obtain the familiar least-squares regression objective in our optimization problem (A), using

$$f(A\boldsymbol{\alpha}) := \tfrac{1}{2}\|A\boldsymbol{\alpha}-\mathbf{b}\|_2^2\,. \tag{20}$$

Observing that the gradient of f is $\nabla f(\mathbf{v}) = \mathbf{v} - \mathbf{b}$, the primal-dual mapping is given by: $\mathbf{w}(\boldsymbol{\alpha}) := \nabla f(\mathbf{v}(\boldsymbol{\alpha})) = A\boldsymbol{\alpha} - \mathbf{b}$, which is well known as the *residual vector* in least-squares regression.

Smooth ℓ: logistic regression loss. For classification problems, we consider a logistic regression model with d training examples, $\mathbf{y}_j \in \mathbb{R}^n$ for $j \in [d]$, collected as the rows of the data matrix A. For each training example, we are given a binary label, which we collect in the vector $\mathbf{b} \in \{-1, 1\}^d$. Formally, the objective is defined as $f(\mathbf{v}) := \sum_{j=1}^{d} \log\left(1 + \exp\left(-b_j v_j\right)\right)$, which is again a separable function. The classifier loss is given by

$$f(A\boldsymbol{\alpha}) := \sum_{j=1}^{d} \log\left(1 + \exp\left(-b_j \mathbf{y}_j^\top \boldsymbol{\alpha}\right)\right), \tag{21}$$

where $\boldsymbol{\alpha} \in \mathbb{R}^n$ is the parameter vector. It is not hard to show that f is 1-smooth if the labels satisfy $b_j \in [-1, 1]$. The primal-dual mapping $\mathbf{w}(\boldsymbol{\alpha}) := \nabla f(\mathbf{v}(\boldsymbol{\alpha})) = \nabla f(A\boldsymbol{\alpha})$ is given by $w_j(\boldsymbol{\alpha}) := \frac{-b_j}{1+\exp\left(b_j \mathbf{y}_j^\top \boldsymbol{\alpha}\right)}$.

Strongly convex r: elastic net regularizer. An application we can consider for a strongly convex regularizer, g in (A) or f^* in (B), is elastic net regularization, $\eta\lambda\|\mathbf{u}\|_1 + (1-\eta)\frac{\lambda}{2}\|\mathbf{u}\|_2^2$, for fixed parameter $\eta \in (0, 1]$. This can be obtained in (A) by setting

$$g(\boldsymbol{\alpha}) = \sum_{i=1}^{n} g_i(\alpha_i) := \sum_{i=1}^{n} \eta\lambda|\alpha_i| + (1-\eta)\tfrac{\lambda}{2}\alpha_i^2. \tag{22}$$

For the special case $\eta = 1$, we obtain the L_1-norm, and for $\eta = 0$, we obtain the L_2-norm. The conjugate of g_i is given by: $g_i^*(x) := \frac{1}{2(1-\eta)}\left(\left[|x| - \eta\right]_+\right)^2$, where $[.]_+$ is the positive part operator, $[s]_+ = s$ for $s > 0$, and zero otherwise.

5.2 Case II: Smooth ℓ, Non-Strongly Convex Separable r

In case II, we consider mapping the input problem (I) to objective (A), where ℓ is assumed to be smooth, and r non-strongly convex and separable. For smooth losses in (A), we can consider as examples those provided in Subsection 5.1, e.g., the least squares loss or logistic loss. For an example of a non-strongly convex regularizer, we consider the important case of L_1 regularization below. Again, we note that this application cannot be realized by objective (B), where it is assumed that the regularization term f^* is strongly convex.

Non-strongly convex r: L_1 regularizer. L_1 regularization is obtained in objective (A) by letting $g_i(\cdot) := \lambda|\cdot|$. However, an additional modification is necessary to obtain primal-dual convergence and certificates for this setting. In particular, we employ the modification introduced in Section 4, which will guarantee L-bounded support. Formally, we replace $g_i(\cdot) = |\cdot|$ by

$$\bar{g}(\alpha) := \begin{cases} |\alpha| & : \alpha \in [-B, B], \\ +\infty & : \text{otherwise.} \end{cases}$$

For large enough B, this problem yields the same solution as the original L_1-objective. Note that this only affects convergence theory, in that it allows us to present a strong primal-dual

rate (Theorem 2 for $L{=}B$). With this modified L_1-regularizer, the optimization problem (A) with regularization parameter λ becomes

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \; f(A\boldsymbol{\alpha}) + \lambda \sum_{i=1}^{n} \bar{g}(\alpha_i)\,. \tag{23}$$

For large enough choice of the value B, this problems yields the same solution as the original objective:

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \quad \Big\{\; \mathcal{O}_A(\boldsymbol{\alpha}) :=\; f(A\boldsymbol{\alpha}) + \lambda \sum_{i=1}^{n} |\alpha_i| \;\Big\}. \tag{24}$$

The modified $\bar{g}$ is simply a constrained version of the absolute value to the interval $[-B, B]$. Therefore by setting B to a large enough value that the values of α_i will never reach it, $\bar{g}^*$ will be continuous and at the same time make (23) equivalent to (24).

Formally, a simple way to obtain a large enough value of B, so that all solutions of (24) are unaffected, is the following: If we start the algorithm at $\boldsymbol{\alpha} = \mathbf{0}$, for every solution encountered during execution, the objective values will never become worse than $\mathcal{O}_A(\mathbf{0})$. Formally, under the assumption that f is non-negative, we will have that (for each i):

$$\lambda|\alpha_i| \leq f(\mathbf{0}) = \mathcal{O}_A(\mathbf{0}) \implies |\alpha_i| \leq \frac{f(\mathbf{0})}{\lambda}.$$

We can therefore safely set the value of B as $\frac{f(\mathbf{0})}{\lambda}$. For the modified $\bar{g}_i$, the conjugate $\bar{g}_i^*$ is given by:

$$\bar{g}_i^*(x) := \begin{cases} 0 & : x \in [-1, 1], \\ B(|x| - 1) & : \text{otherwise.} \end{cases}$$

We provide a proof of this in Appendix B (Lemma 6).

Non-strongly convex r: group lasso. The group lasso penalty can be mapped to objective (A), with:

$$g(\boldsymbol{\alpha}) := \lambda \sum_{p=1}^{P} \|\boldsymbol{\alpha}_{\mathcal{I}_p}\|_2 \quad \text{with} \quad \bigcup_{p=1}^{P} \mathcal{I}_p = \{1, \ldots, n\}\,, \tag{25}$$

where the disjoint sets $\mathcal{I}_p \subseteq \{1, \ldots, n\}$ represent a partitioning of the total set of variables. This penalty can be viewed as an intermediate between a pure L_1 or L_2 penalty, performing variable selection only at the group level. The term $\boldsymbol{\alpha}_{\mathcal{I}_p} \in \mathbb{R}^{|\mathcal{I}_p|}$ denotes part of the vector $\boldsymbol{\alpha}$ with indices $\mathcal{I}_p$. The conjugate is given by:

$$g^*(\mathbf{w}) = I_{\{\mathbf{w} \mid \max_{\mathcal{I}_p \in [n]} \|\boldsymbol{\alpha}_{\mathcal{I}_p}\|_2 \leq \lambda\}}(\mathbf{w}).$$

For details, see, e.g., Dünner et al. (2016) or Boyd and Vandenberghe (2004, Example 3.26).

5.3 Case III: Non-Smooth Separable ℓ, Strongly Convex r

Finally, in case III, we consider mapping the input problem (I) to objective (B), where ℓ is assumed to be non-smooth and separable, and r strongly convex. We discuss two common cases of general non-smooth losses ℓ, including the the hinge loss for classification and absolute deviation loss for regression. When paired with a strongly convex regularizer, the regularizer via f gives rise to the primal-dual mapping, and Theorem 2 provides a sublinear convergence rate for objectives of this form. We note that these losses cannot be realized directly by objective (A), where it is assumed that the data fit term f is smooth.

Non-smooth ℓ: hinge loss. For classification problems, we can consider a hinge loss support vector machine model, on n training points in $\mathbb{R}^d$, given with the loss:

$$g^*(-A^\top \mathbf{w}) = \sum_{i=1}^{n} g_i^*(-\mathbf{x}_i^\top \mathbf{w}) := \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i \mathbf{x}_i^\top \mathbf{w}\}. \tag{26}$$

The conjugate function of the hinge loss $\phi(a) = \max\{0, 1-b\}$ is given by $\phi^*(b) = \{b$ if $b \in [-1, 0]$, else ∞ .$\}$. When using the L_2 norm for regularization in this problem: $f^*(\mathbf{w}) := \lambda \|\mathbf{w}\|_2^2$, a primal-dual mapping is given by: $\mathbf{w}(\boldsymbol{\alpha}) := \frac{1}{\lambda n} A\boldsymbol{\alpha}$.

Non-smooth ℓ: absolute deviation loss. The absolute deviation loss, used, e.g., in quantile regression or least absolute deviation regression, can be realized in objective (B) by setting:

$$g^*(-A^\top \mathbf{w}) = \sum_{i=1}^{n} g_i^*(-\mathbf{x}_i^\top \mathbf{w}) := \frac{1}{n}\sum_{i=1}^{n} \left| \mathbf{x}_i^\top \mathbf{w} - y_i \right|. \tag{27}$$

The conjugate function of the absolute deviation loss $\phi(a) = |a - y_i|$ is given by $\phi^*(-b) = -by_i$, with $b \in [-1, 1]$.

5.4 Local Solvers

As discussed in Section 3, the subproblems solved on each machine in the CoCoA framework are appealing in that they are very similar in structure to the global problem (A), with the main difference being that they are defined on a smaller (local) subset of the data, and have a simpler dependence on the term f. Therefore, solvers which have already proven their value in the single machine or multicore setting can be easily leveraged within the framework. We discuss some specific examples of local solvers below, and point the reader to Ma et al. (2015a) for an empirical exploration of these choices.

In the primal setting (Algorithm 2), the local subproblem (10) becomes a simple quadratic problem on the local data, with regularization applied only to local variables $\boldsymbol{\alpha}_{[k]}$. For the L_1 examples discussed, existing fast L_1-solvers for the single-machine case, such as GLMNET variants (Friedman et al., 2010) or BLITZ (Johnson and Guestrin, 2015) can be directly applied to each local subproblem $\mathcal{G}_k^{\sigma'}(\cdot\,; \mathbf{v}, \boldsymbol{\alpha}_{[k]})$ within Algorithm 1. The sparsity induced on the subproblem solutions of each machine naturally translates into the sparsity of the global solution, since the local variables $\boldsymbol{\alpha}_{[k]}$ will be concatenated.

In terms of the approximation quality parameter Θ for the local problems (Assumption 1), we can apply existing recent convergence results from the single machine case. For

example, for randomized coordinate descent (as part of GLMNET), Lu and Xiao (2013, Theorem 1) gives a $O(1/t)$ approximation quality for any separable regularizer, including L_1 and elastic net; see also Tappenden et al. (2015) and Shalev-Shwartz and Tewari (2011).

In the dual setting (Algorithm 3) for the discussed examples, the losses are applied only to local variables $\boldsymbol{\alpha}_{[k]}$, and the regularizer is approximated via a quadratic term. Current state of the art for the problems of the form in (B) are variants of randomized coordinate ascent—Stochastic Dual Coordinate Ascent (SDCA) (Shalev-Shwartz and Zhang, 2013a). This algorithm and its variants are increasingly used in practice (Wright, 2015), and extensions such as accelerated and parallel versions can directly be applied (Shalev-Shwartz and Zhang, 2014; Fan et al., 2008) in our framework. For non-smooth losses such as SVMs, the analysis of Shalev-Shwartz and Zhang (2013a) provides a $O(1/t)$ rate, and for smooth losses, a faster linear rate. There have also been recent efforts to derive a linear convergence rate for problems like the hinge-loss support vector machine that could be applied, e.g., by using error bound conditions (Necoara and Nedelcu, 2014; Wang and Lin, 2014), weak strong convexity conditions (Ma et al., 2015c; Necoara, 2015) or by considering Polyak-Łojasiewicz conditions (Karimi et al., 2016).

6. Experiments

In this section we demonstrate the empirical performance of CoCoA in the distributed setting. We first compare CoCoA to competing methods for two common machine learning applications: lasso regression (Section 6.1) and support vector machine (SVM) classification (Section 6.2). We then explore the performance of CoCoA in the primal versus the dual directly by solving an elastic net regression model with both variants (Section 6.3). Finally, we illustrate general properties of the CoCoA method empirically in Section 6.4.

Experimental setup. We compare CoCoA to numerous state-of-the-art general-purpose methods for large-scale optimization, including:

- Mb-SGD: Mini-batch stochastic gradient. For our experiments with lasso, we compare against Mb-SGD with an L_1-prox.
- GD: Full gradient descent. For lasso we use the proximal version, Prox-GD.
- L-BFGS: Limited-memory quasi-Newton method. For lasso, we use OWL-QN (orthant-wise limited quasi-Newton).
- ADMM: Alternating direction method of multipliers. We use conjugate gradient internally for the lasso experiments, and SDCA for SVM experiments.
- Mb-CD: Mini-batch parallel coordinate descent. For SVM experiments, we implement Mb-SDCA (mini-batch stochastic dual coordinate ascent).

The first three methods are optimized and implemented in Apache Spark's MLlib (v1.5.0) (Meng et al., 2016). We test the performance of each method in large-scale experiments fitting lasso, elastic net regression, and SVM models to the datasets shown in Table 5. In comparing to other methods, we plot the distance to the optimal primal solution. This optimal value is calculated by running all methods for a large number of iterations (until progress has stalled), and then selecting the smallest primal value amongst the re-

sults. All code is written in Apache Spark and experiments are run on public-cloud Amazon EC2 m3.xlarge machines with one core per machine. Our code is publicly available at `github.com/gingsmith/proxcocoa`.

Table 5: Datasets for Empirical Study.

Dataset	**Training Size**	**Feature Size**	**Sparsity**
url	2 M	3 M	3.5e-5
epsilon	400 K	2 K	1.0
kddb	19 M	29 M	9.8e-7
webspam	350 K	16 M	2.0e-4

We carefully tune each competing method in our experiments for best performance. ADMM requires the most tuning, both in selecting the penalty parameter ρ and in solving the subproblems. Solving the subproblems to completion for ADMM is prohibitively slow, and we thus use an iterative method internally and improve performance by allowing early stopping. We also use a varying penalty parameter ρ — practices described in Boyd et al. (2010, Sections 4.3, 8.2.3, 3.4.1). For Mb-SGD, we tune the step size and mini-batch size parameters. For Mb-CD and Mb-SDCA, we scale the updates at each round by $\frac{\beta}{b}$ for mini-batch size b and $\beta \in [1, b]$, and tune both parameters b and β. Further implementation details for all methods are given in Section 6.5.

For simplicity of presentation and comparison, in all of the following experiments, we restrict CoCoA to only use simple coordinate descent as the local solver. We note that even stronger empirical results for CoCoA could be obtained by plugging in state of the art local solvers for each application at hand.

6.1 CoCoA in the Primal

We first demonstrate the performance of CoCoA in the primal (Algorithm 2) by applying CoCoA to a lasso regression model (8) fit to the distributed datasets in Table 5. We use stochastic coordinate descent as a local solver for CoCoA, and select the number of local iterations H (a proxy for subproblem approximation quality, Θ) from several options with best performance.

We compare CoCoA to the general methods listed above, including Mb-SGD with an L_1-prox, Prox-GD, OWL-QN, ADMM and Mb-CD. A comparison with Shotgun (Bradley et al., 2011), a popular method for solving L_1-regularized problems in the multicore environment, is provided as an extreme case to highlight the detrimental effects of frequent communication in the distributed environment. For Mb-CD, Shotgun, and CoCoA in the primal, datasets are distributed by feature, whereas for Mb-SGD, Prox-GD, OWL-QN and ADMM they are distributed by training point.

In analyzing the performance of each algorithm (Figure 1), we measure the improvement to the primal objective given in (A) ($\mathcal{O}_A(\boldsymbol{\alpha})$) in terms of wall-clock time in seconds. We see that both Mb-SGD and Mb-CD are slow to converge, and come with the additional burden of having to tune extra parameters (though Mb-CD makes clear improvements over Mb-SGD). As expected, naively distributing Shotgun (single coordinate updates per

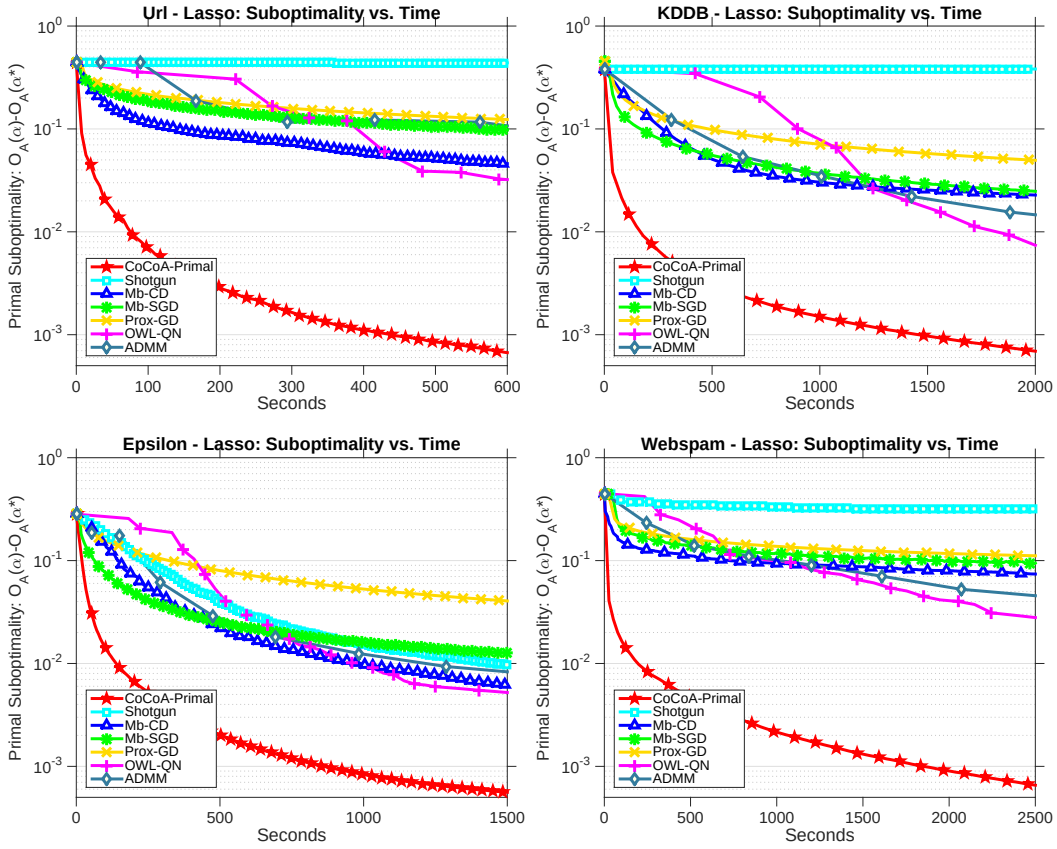

Figure 1: Suboptimality in terms of $\mathcal{O}_A(\boldsymbol{\alpha})$ for fitting a lasso regression model to four datasets: url (K=4, λ=1e-4), kddb (K=4, λ=1e-6), epsilon (K=8, λ=1e-5), and webspam (K=16, λ=1e-5) datasets. CoCoA applied to the primal formulation converges more quickly than all other compared methods in terms of the time in seconds.

machine) does not perform well, as it is tailored to shared-memory systems and requires communicating too frequently. OWL-QN performs the best of all compared methods, but is still much slower to converge than CoCoA, and converges, e.g., 50× more slowly for the webspam dataset. The optimal performance of CoCoA is particularly evident in datasets with large numbers of features (e.g., url, kddb, webspam), which are exactly the datasets of interest for L_1 regularization.

Results are shown for regularization parameters λ such that the resulting weight vector $\boldsymbol{\alpha}$ is sparse. However, our results are robust to varying values of λ as well as to various problem settings, as we illustrate in Figure 2.

A case against smoothing. We additionally motivate the use of CoCoA in the primal by showing how it improves upon CoCoA in the dual (Yang, 2013; Jaggi et al., 2014; Ma et al., 2015b,a) for non-strongly convex regularizers. First, CoCoA in the dual cannot be included in the set of experiments in Figure 1 because it cannot be directly applied to the lasso objective (recall that Algorithm 3 only allows for strongly convex regularizers).

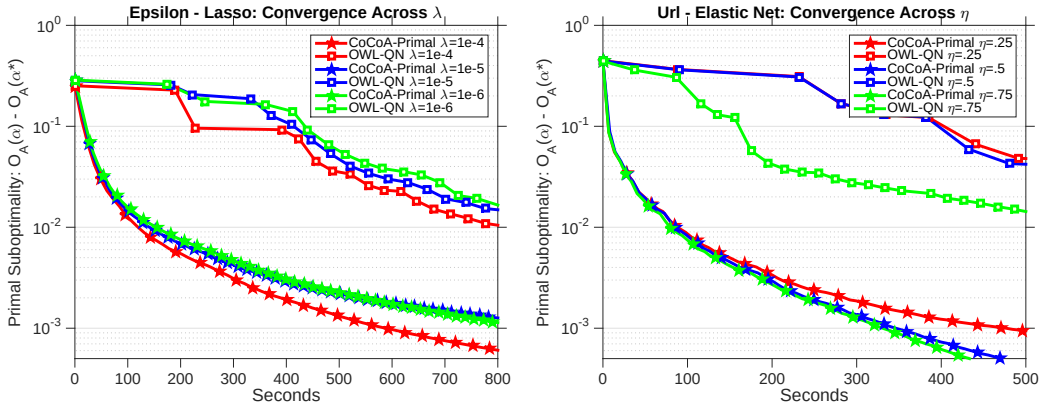

Figure 2: Suboptimality in terms of $\mathcal{O}_A(\boldsymbol{\alpha})$ for solving lasso for the epsilon dataset (left, K=8) and elastic net for the url dataset, (right, K=4, λ=1E-4). Speedups are robust over different regularizers λ (left), and across problem settings, including varying η parameters of elastic net regularization (right).

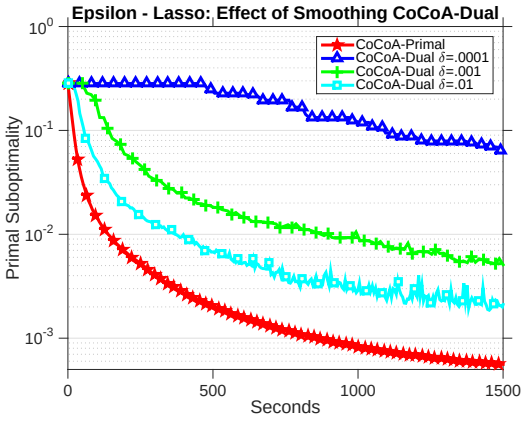

Table 6: Sparsity of Final Iterates.

Method	**Sparsity**
CoCoA-Primal	0.6030
CoCoA-Dual: $\delta = 0.0001$	0.6035
CoCoA-Dual: $\delta = 0.001$	0.6240
CoCoA-Dual: $\delta = 0.01$	0.6465

Figure 3 & Table 6: For pure L_1 regularization, Nesterov smoothing is not an effective option for CoCoA in the dual. It either modifies the solution (Figure 3) or slows convergence (Table 6). This motivates running CoCoA instead on the primal for these problems.

To get around this requirement, previous work has suggested implementing the Nesterov smoothing technique used in, e.g., Shalev-Shwartz and Zhang (2014); Zhang and Lin (2015) — adding a small amount of strong convexity $\delta\|\boldsymbol{\alpha}\|_2^2$ to the objective for lasso regression. In Figure 3 we demonstrate the issues with this approach, comparing CoCoA in the primal on a pure L_1-regularized regression problem to CoCoA in the dual for decreasing levels of δ. The smaller we set δ, the less smooth the problem becomes. As δ decreases, the final sparsity of running CoCoA in the dual starts to match that of running pure L_1 (Table 6), but the performance also degrades (Figure 3). We note that by using CoCoA in the primal with the modification presented in Section 4, we can deliver strong rates without having to make these fundamental alterations to the problem of interest.

6.2 CoCoA in the Dual

Next we present results on CoCoA in the dual against competing methods, for an SVM model (9) on the datasets in Table 5. We use stochastic dual coordinate ascent (SDCA) as a local solver for CoCoA in this setting, again selecting the number of local iterations H from several options with best performance. We compare CoCoA to the general methods listed above, including Mb-SGD, GD, L-BFGS, ADMM, and Mb-SDCA. All datasets are distributed by training point for these methods.

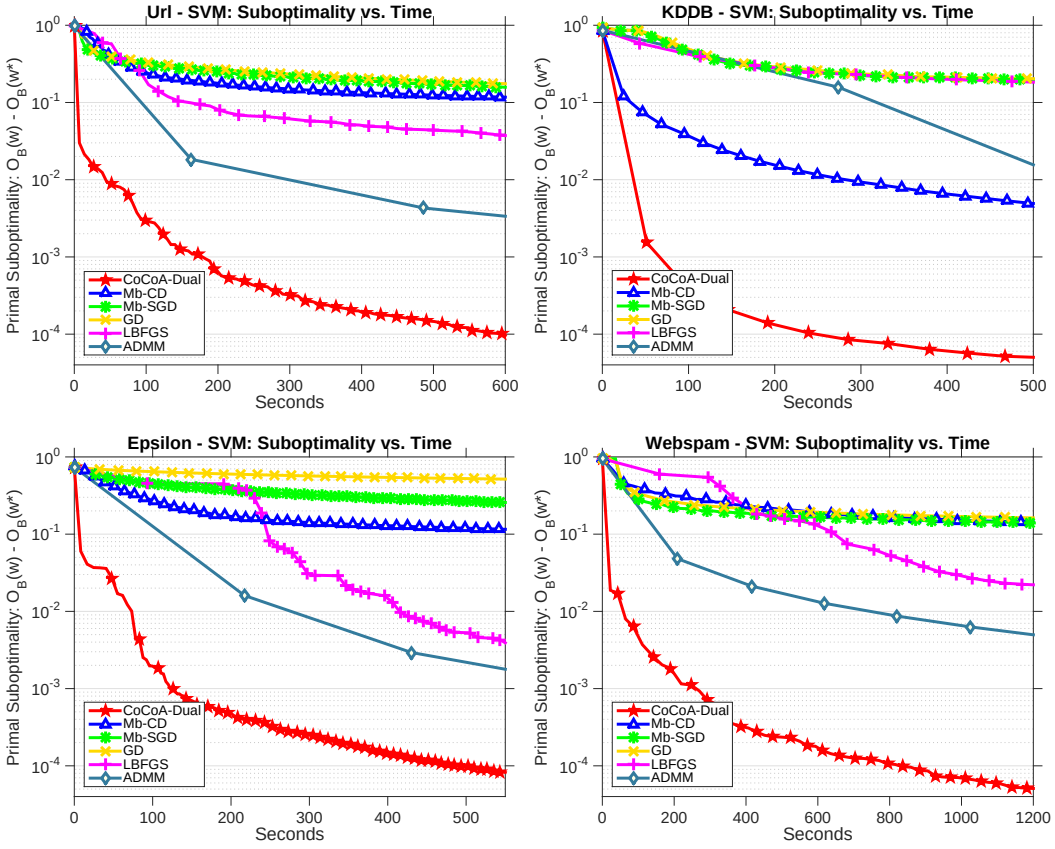

Figure 3: Suboptimality in terms of $\mathcal{O}_B(\mathbf{w})$ for solving a hinge-loss support vector machine model: url (K=4, λ=1e-4), kddb (K=4, λ=1e-6), epsilon (K=8, λ=1e-5), and webspam (K=16, λ=1e-5) datasets. CoCoA applied to the dual formulation converges more quickly than all other compared methods in terms of the time in seconds.

In analyzing the performance the methods in this setting (Figure 3), we measure the improvement to the primal objective given in (B) ($\mathcal{O}_B(\mathbf{w})$) in terms of wall-clock time in seconds. We see again that Mb-SGD and Mb-CD are slow to converge, and come with the additional burden of having to tune extra parameters. ADMM performs the best of the methods other than CoCoA, followed by L-BFGS. However, both are still much slower to converge than CoCoA in the dual. ADMM was in particular affected by the fact that many internal iterations of SDCA were necessary in order to guarantee convergence. In contrast,

CoCoA is able to incorporate arbitrary amounts of work locally and still converge. We note that although CoCoA, ADMM and Mb-SDCA run in the dual, the plots in Figure 3 mark progress towards the primal objective, $\mathcal{O}_B(\mathbf{w})$.

6.3 Primal vs. Dual

To understand the effect of primal versus dual optimization for CoCoA, we compare the performance of both variants by fitting an elastic net regression model (7) to two datasets. For comparability of the methods, we use coordinate descent (with closed-form updates) as the local solver in both variants. From the results in Figure 4, we see that CoCoA in the dual tends to perform better on datasets with a large number of training points (relative to the number of features), and that as expected, the performance deteriorates as the strong convexity in the problem disappears. In contrast, CoCoA in the primal performs well on datasets with a large number of features relative to training points, and is robust to changes in strong convexity. These changes in performance are to be expected, as we have already discussed that CoCoA in the primal is more suited for non-strongly convex regularizers (Section 6.1), and that the feature size dominates communication for CoCoA in the dual, as compared to the training point size for CoCoA in the primal (Section 3.4).

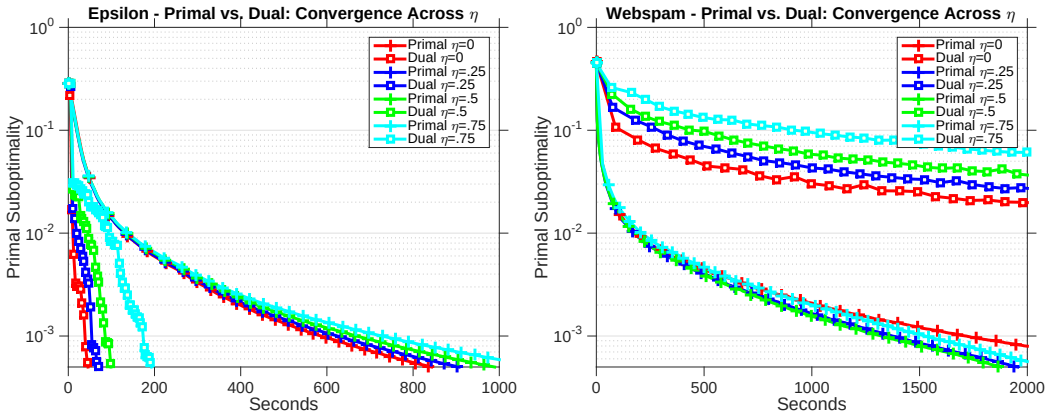

Figure 4: The convergence of CoCoA in the primal versus dual for various values of η in an elastic net regression model. CoCoA in dual performs better on the Epsilon dataset, where the training point size is the dominating term, and CoCoA in the primal performs better on the Webspam dataset, where the feature size is the dominating term. In both datasets, CoCoA in the dual performs better as the problem becomes more strongly convex ($\eta \to 0$), whereas CoCoA in the primal is robust to changes in strong convexity.

6.4 General Properties: Effect of Communication

Finally, we note that in contrast to the compared methods from Sections 6.1 and 6.2, CoCoA comes with the benefit of having only a single parameter to tune: the subproblem approximation quality, Θ, which we control in our experiments via the number of local subproblem iterations, H, for the example of local coordinate descent. We further explore the effect of this parameter in Figure 5, and provide a general guideline for choosing it in prac-

tice (see Remark 1). In particular, we see that while increasing H always results in better performance in terms of the number of communication rounds, smaller or larger values of H may result in better performance in terms of wall-clock time, depending on the cost of communication and computation. The flexibility to fine-tune H is one of the reasons for CoCoA's significant performance gains.

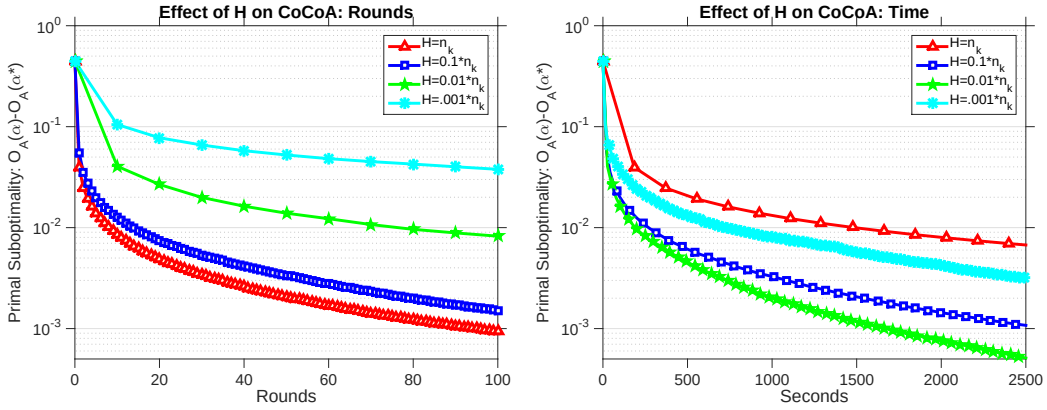

Figure 5: Suboptimality in terms of $\mathcal{O}_A(\boldsymbol{\alpha})$ for solving lasso for the webspam dataset (K=16, λ=1e-5). Here we illustrate how the work spent in the local subproblem (given by H) influences the total performance of CoCoA in terms of number of rounds as well as wall time.

6.5 Experiment Details

In this subsection we provide thorough details on the experimental setup and methods used in our comparison. All experiments are run on Amazon EC2 clusters of m3.xlarge machines, with one core per machine. The code for each method is written in Apache Spark, v1.5.0. Our code is open source and publicly available at `github.com/gingsmith/proxcocoa`.

ADMM. Alternating Direction Method of Multipliers (ADMM) (Boyd et al., 2010) is a popular method that lends itself naturally to the distributed environment. For lasso regression, implementing ADMM for the problems of interest requires solving a large linear system $C\mathbf{x} = \mathbf{d}$ on each machine, where $C \in \mathbb{R}^{n \times n}$ with n scaling beyond 10^7 for the datasets in Table 5, and with C being possibly dense. It is prohibitively slow to solve this directly on each machine, and we therefore employ the iterative method of conjugate gradient with early stopping (see, e.g., Boyd et al., 2010, Section 4.3). For SVM classification, we use stochastic dual coordinate ascent as an internal optimizer, which is shown in Zhang et al. (2012) to have superior performance. We further improve performance by using a varying rather than constant penalty parameter, as suggested in Boyd et al. (2010, Section 3.4.1).

Mini-batch SGD and proximal GD. Mini-batch SGD is a standard and widely used method for parallel and distributed optimization. We use the optimized code provided in Spark's machine learning library, MLlib, v1.5.0 (Meng et al., 2016). We tune both the size of the mini-batch and the SGD step size using grid search. For lasso, we use the proximal version of the method. Full gradient descent can be seen as a specific setting of mini-batch

SGD, where the mini-batch size is equal to the total number of training points. We thus also use the implementation in MLlib for full GD, and tune the step size parameter using grid search.

Mini-batch CD and SDCA. Mini-batch CD (for lasso) and SDCA (for SVM) aim to improve mini-batch SGD by employing coordinate descent, which has theoretical and practical justifications (Shalev-Shwartz and Tewari, 2011; Takáč et al., 2015; Fercoq and Richtárik, 2015; Tappenden et al., 2015; Takáč et al., 2013). We implement mini-batch CD and SDCA in Spark and scale the updates made at each round by $\frac{\beta}{b}$ for mini-batch size b and $\beta \in [1, b]$, tuning both parameters b and β via grid search. For the case of lasso regression, we implement Shotgun (Bradley et al., 2011), which is a popular method for parallel optimization. Shotgun can be seen an extreme case of mini-batch CD where the mini-batch is set to K, i.e., there is a single update made by each machine per round. We see in the experiments that communicating this frequently becomes prohibitively slow in the distributed environment.

OWL-QN. OWN-QN (Yu et al., 2010) is a quasi-Newton method optimized in Spark's spark.ml package (Meng et al., 2016). Outer iterations of OWL-QN make significant progress towards convergence, but the iterations themselves can be slow because they require processing the entire dataset. CoCoA, the mini-batch methods, and ADMM with early stopping all improve on this by allowing the flexibility of only a subset of the dataset to be processed at each iteration. CoCoA and ADMM have even greater flexibility by allowing internal methods to process the dataset more than once. CoCoA makes this approximation quality explicit, both in theoretical convergence rates and by providing general guidelines for setting the parameter.

CoCoA. We implement CoCoA with coordinate descent as the local solver. We note that since the framework and theory allow any internal solver to be used, CoCoA could benefit even beyond the results shown, e.g., by using existing fast L_1-solvers for the single-machine case, such as GLMNET variants (Friedman et al., 2010) or BLITZ (Johnson and Guestrin, 2015) or SVM solvers like LIBLINEAR (Fan et al., 2008). The only parameter necessary to tune for CoCoA is the level of approximation quality, which we parameterize in the experiments through H, the number of local iterations of the iterative method run locally. Our theory relates local approximation quality to global convergence (Section 4), and we provide a guideline for how to choose this value in practice that links the parameter to the systems environment at hand (Remark 1).

7. Related Work

Single-machine coordinate solvers. For strongly convex regularizers, the current state-of-the-art for empirical loss minimization is randomized coordinate ascent on the dual (SDCA) (Shalev-Shwartz and Zhang, 2013a) and its accelerated variants (e.g., Shalev-Shwartz and Zhang, 2014). In contrast to primal stochastic gradient descent (SGD) methods, the SDCA family is often preferred as it is free of learning-rate parameters and has faster (geometric) convergence guarantees. Interestingly, a similar trend in coordinate solvers has been observed in the recent literature on the lasso, but with the roles of primal and dual reversed. For those problems, coordinate descent methods on the primal have become state-

of-the-art, as in GLMNET (Friedman et al., 2010) and extensions (Yuan et al., 2012); see, e.g., the overview in Yuan et al. (2010). However, primal-dual convergence rates for unmodified coordinate algorithms have to our knowledge only been obtained for strongly convex regularizers to date (Shalev-Shwartz and Zhang, 2014; Zhang and Lin, 2015).

Coordinate descent on L_1-regularized problems (i.e., (A) with $g(\cdot) = \lambda\|\cdot\|_1$) can be interpreted as the iterative minimization of a quadratic approximation of the smooth part of the objective (as in a one-dimensional Newton step), followed by a shrinkage step resulting from the L_1 part. In the single-coordinate update case, this is at the core of GLMNET (Friedman et al., 2010; Yuan et al., 2010), and widely used in, e.g., solvers based on the primal formulation of L_1-regularized objectives (Shalev-Shwartz and Tewari, 2011; Yuan et al., 2012; Bian et al., 2013; Fercoq and Richtárik, 2015; Tappenden et al., 2015). When changing more than one coordinate at a time, again employing a quadratic upper bound on the smooth part, this results in a two-loop method as in GLMNET for the special case of logistic regression. This idea is crucial for the distributed setting. When the set of active coordinates coincides with the ones on the local machine, these single-machine approaches closely resemble the distributed framework proposed here.

Parallel methods. For the general regularized loss minimization problems of interest, methods based on stochastic subgradient descent (SGD) are well-established. Several variants of SGD have been proposed for parallel computing, many of which build on the idea of asynchronous communication (Niu et al., 2011; Duchi et al., 2013). Despite their simplicity and competitive performance on shared-memory systems, the downside of this approach in the distributed environment 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 one 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.

For the specific case of L_1-regularized objectives, parallel coordinate descent (with and without using mini-batches) was proposed in Bradley et al. (2011) (Shotgun) and generalized in Bian et al. (2013), and is among the best performing solvers in the parallel setting. Our framework reduces to Shotgun as a special case when the internal solver is a single-coordinate update on the subproblem (10), $\gamma = 1$, and for a suitable σ'. However, Shotgun is not covered by our convergence theory, since it uses a potentially unsafe upper bound of β instead of σ', which isn't guaranteed to satisfy our condition for convergence (11). We compare empirically with Shotgun in Section 6 to highlight the detrimental effects of running this high-communication method in the distributed environment.

One-shot communication schemes. At the other extreme, there are distributed methods that use only a single round of communication, such as Mann et al. (2009); Zinkevich et al. (2010); Zhang et al. (2013); McWilliams et al. (2014); and Heinze et al. (2016). 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 Shamir et al. (2014). Additional relevant lower bounds on

the minimum number of communication rounds necessary for a given approximation quality are presented in Balcan et al. (2012) and Arjevani and Shamir (2015).

Mini-batch methods. Mini-batch methods (which use updates from several training points or features per round) are more flexible and lie within the two extremes of parallel and one-shot communication schemes. However, mini-batch versions of both SGD and coordinate descent (CD) (e.g., Takáč et al., 2013; Shalev-Shwartz and Zhang, 2013b; Qu et al., 2015; Richtárik and Takáč, 2016) 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 more difficult to tune, as it can lie anywhere in the order of mini-batch size. The optimal choice is often either unknown or too challenging to compute in practice. In the CoCoA framework there is no need to tune parameters, as the aggregation parameter and subproblem parameters can be set directly using the safe bound discussed in Section 3 (Definition 5).

Batch solvers. ADMM (Boyd et al., 2010), gradient descent, and quasi-Newton methods such as L-BFGS and are also often used in distributed environments because of their relatively low communication requirements. However, they require at least a full (distributed) batch gradient computation at each round, and therefore do not allow the gradual trade-off between communication and computation provided by CoCoA. In Section 6, we include experimental comparisons with ADMM, gradient descent, and L-BFGS variants, including orthant-wise limited memory quasi-Newton (OWL-QN) for the L_1 setting (Andrew and Gao, 2007).

Finally, we note that while the convergence rates provided for CoCoA mirror the convergence class of classical batch gradient methods in terms of the number of outer rounds, existing batch gradient methods come with a weaker theory, as they do not allow general inexactness Θ for the local subproblem (10). In contrast, our convergence rates incorporate this approximation directly, and, moreover, hold for arbitrary local solvers of much cheaper cost than batch methods (where in each round, every machine has to process exactly a full pass through the local data). This makes CoCoA more flexible in the distributed setting, as it can adapt to varied communication costs on real systems. We have seen in Section 6 that this flexibility results in significant performance gains over the competing methods.

Distributed solvers. By making use of the primal-dual structure in the line of work of Yu et al. (2012); Pechyony et al. (2011); Yang (2013); Yang et al. (2013) and Lee and Roth (2015), the CoCoA-v1 and CoCoA$^+$ frameworks (which are special cases of the presented framework, CoCoA) are the first to allow the use of any local solver—of weak local approximation quality—in each round in the distributed setting. The practical variant of the DisDCA (Yang, 2013), called DisDCA-p, allows for additive updates in a similar manner to CoCoA, but is restricted to coordinate decent (CD) being the local solver, and was initially proposed without convergence guarantees. DisDCA-p, CoCoA-v1, and CoCoA$^+$ are all limited to strongly convex regularizers, and therefore are not as general as the CoCoA framework discussed in this work.

In the L_1-regularized setting, an approach related to our framework includes distributed variants of GLMNET as in Mahajan et al. (2014). Inspired by GLMNET and Yuan et al.

(2012), the works of Bian et al. (2013) and Mahajan et al. (2014) introduced the idea of a block-diagonal Hessian upper approximation in the distributed L_1 context. The later work of Trofimov and Genkin (2014) specialized this approach to sparse logistic regression.

If hypothetically each of our quadratic subproblems $\mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]})$ as defined in (10) were to be minimized exactly, the resulting steps could be interpreted as block-wise Newton-type steps on each coordinate block k, where the Newton-subproblem is modified to also contain the L_1-regularizer (Mahajan et al., 2014; Yuan et al., 2012; Qu et al., 2016). While Mahajan et al. (2014) allows a fixed accuracy for these subproblems, but not arbitrary approximation quality Θ as in our framework, the works of Trofimov and Genkin (2014); Yuan et al. (2012); and Yen et al. (2015) assume that the quadratic subproblems are solved exactly. Therefore, these methods are not able to freely trade off communication and computation. Also, they do not allow the re-use of arbitrary local solvers. On the theoretical side, the convergence rate results provided by Mahajan et al. (2014); Trofimov and Genkin (2014); and Yuan et al. (2012) are not explicit convergence rates but only asymptotic, as the quadratic upper bounds are not explicitly controlled for safety as with our σ'.

8. Discussion

To enable large-scale machine learning, we have developed, analyzed, and evaluated a general-purpose framework for communication-efficient primal-dual optimization in the distributed environment. Our framework, CoCoA, takes a unique approach by using duality to derive subproblems for each machine to solve in parallel. These subproblems closely match the global problem of interest, which allows for state-of-the-art single-machine solvers to easily be re-used in the distributed setting. Further, by allowing the local solvers to find solutions of arbitrary approximation quality to the subproblems on each machine, our framework permits a highly flexible communication scheme. In particular, as the local solvers make updates directly to their local parameters, the need to communicate reduces and can be adapted to the system at hand, which helps to manage the communication bottleneck in the distributed setting.

We analyzed the impact of the local solver approximation quality and derived global primal-dual convergence rates for our framework that are agnostic to the specifics of the local solvers. We have taken particular care in extending our framework to the case of non-strongly convex regularizers, where we introduced a bounded-support modification technique to provide robust convergence guarantees. Finally, we demonstrated the efficiency of our framework in an extensive experimental comparison with state-of-the-art distributed solvers. Our framework achieves up to a 50× speedup over other widely-used methods on real-world distributed datasets.

Acknowledgments

We thank Michael P. Friedlander, Jakub Konečný, and Peter Richtárik for their help and for fruitful discussions.

Appendix A. Convex Conjugates

The convex conjugate of a function $f : \mathbb{R}^d \to \mathbb{R}$ is defined as

$$f^*(\mathbf{v}) := \max_{\mathbf{u}\in\mathbb{R}^d} \mathbf{v}^\top \mathbf{u} - f(\mathbf{u})\,. \tag{28}$$

Below we list several useful properties of conjugates (see, e.g., Boyd and Vandenberghe, 2004, Section 3.3.2):

- Double conjugate: $(f^*)^* = f$ if f is closed and convex.
- Value Scaling: (for $\alpha > 0$) $f(\mathbf{v}) = \alpha g(\mathbf{v}) \quad \Rightarrow \quad f^*(\mathbf{w}) = \alpha g^*(\mathbf{w}/\alpha)\,.$
- Argument Scaling: (for $\alpha \neq 0$) $f(\mathbf{v}) = g(\alpha \mathbf{v}) \quad \Rightarrow \quad f^*(\mathbf{w}) = g^*(\mathbf{w}/\alpha)\,.$
- Conjugate of a separable sum: $f(\mathbf{v}) = \sum_i \phi_i(v_i) \quad \Rightarrow \quad f^*(\mathbf{w}) = \sum_i \phi_i^*(w_i)\,.$

Lemma 4 (Duality between Lipschitzness and L-Bounded Support, (Rockafellar, 1997, Corollary 13.3.3))**.** *Given a proper convex function f, it holds that f is L-Lipschitz if and only if f^* has L-bounded support.*

Lemma 5 (Duality between Smoothness and Strong Convexity, (Hiriart-Urruty and Lemaréchal, 2001, Theorem 4.2.2))**.** *Given a closed convex function f, it holds that f is μ strongly convex w.r.t. the norm $\|\cdot\|$ if and only if f^* is $(1/\mu)$-smooth w.r.t. the dual norm $\|\cdot\|_*$.*

Appendix B. Proofs of Primal-Dual Relationships

In the following subsections we provide derivations of the primal-dual relationship of the general objectives (A) and (B), and then show how to derive the conjugate of the modified L_1-norm, as an example of the bounded-support modification introduced in Section 4.

B.1 Primal-Dual Relationship

The relation of our original formulation (A) to its dual formulation (B) is standard in convex analysis, and is a special case of the concept of Fenchel Duality. Using the combination with the linear map A as in our case, the relationship is called Fenchel-Rockafellar Duality, see e.g. Borwein and Zhu (2005, Theorem 4.4.2) or Bauschke and Combettes (2011, Proposition 15.18). For completeness, we illustrate this correspondence with a self-contained derivation of the duality.

Starting with the original formulation (A), we introduce an auxiliary vector $\mathbf{v} \in \mathbb{R}^d$ representing $\mathbf{v} = A\boldsymbol{\alpha}$. Then optimization problem (A) becomes:

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \quad f(\mathbf{v}) + g(\boldsymbol{\alpha}) \quad \text{such that } \mathbf{v} = A\boldsymbol{\alpha}\,. \tag{29}$$

Introducing Lagrange multipliers $\mathbf{w} \in \mathbb{R}^d$, the Lagrangian is given by:

$$L(\boldsymbol{\alpha}, \mathbf{v}; \mathbf{w}) := f(\mathbf{v}) + g(\boldsymbol{\alpha}) + \mathbf{w}^\top (A\boldsymbol{\alpha} - \mathbf{v})\,.$$

The dual problem of (A) follows by taking the infimum with respect to both $\boldsymbol{\alpha}$ and $\mathbf{v}$:

$$\begin{aligned}\inf_{\boldsymbol{\alpha},\mathbf{v}} L(\mathbf{w},\boldsymbol{\alpha},\mathbf{v}) &= \inf_{\mathbf{v}}\left\{f(\mathbf{v}) - \mathbf{w}^\top\mathbf{v}\right\} + \inf_{\boldsymbol{\alpha}}\left\{g(\boldsymbol{\alpha}) + \mathbf{w}^\top A\boldsymbol{\alpha}\right\} \\ &= -\sup_{\mathbf{v}}\left\{\mathbf{w}^\top\mathbf{v} - f(\mathbf{v})\right\} - \sup_{\boldsymbol{\alpha}}\left\{(-\mathbf{w}^\top A)\boldsymbol{\alpha} - g(\boldsymbol{\alpha})\right\} \\ &= -f^*(\mathbf{w}) - g^*(-A^\top\mathbf{w})\,. \end{aligned} \tag{30}$$

We change signs and turn the maximization of the dual problem (30) into a minimization, thereby arriving at the dual formulation (B) as claimed:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \quad \Big[\ \mathcal{O}_B(\mathbf{w}) := g^*(-A^\top\mathbf{w}) + f^*(\mathbf{w})\ \Big]\,.$$

B.2 Continuous Conjugate Modification for Indicator Functions

Lemma 6 (Conjugate of the modified L_1-norm)**.** *The convex conjugate of the bounded support modification of the L_1-norm, as defined in* (18)*, is:*

$$\bar{g}_i^*(x) := \begin{cases} 0 & : x \in [-1,1], \\ B(|x|-1) & : \textit{otherwise,} \end{cases}$$

and is B-Lipschitz.

Proof. We start by applying the definition of convex conjugate:

$$\bar{g}_i(\alpha) = \sup_{x\in\mathbb{R}}\left[\alpha x - \bar{g}_i^*(x)\right]\,.$$

We begin by looking at the case in which $\alpha \geq B$; in this case it's easy to see that when $x \to +\infty$, we have:

$$\alpha x - B(|x|-1) = (\alpha - B)x - B \to +\infty\,,$$

as $\alpha - B \geq 0$. The case $\alpha \leq -B$ holds analogously. We'll now look at the case $\alpha \in [0, B]$; in this case it is clear we must have $x^\star \geq 0$. It also must hold that $x^\star \leq 1$, since

$$\alpha x - B(x-1) < \alpha x\,,$$

for every $x > 1$. Therefore the maximization becomes

$$\bar{g}_i(\alpha) = \sup_{x\in[0,1]} \alpha x\,,$$

which has maximum α at $x = 1$. The remaining $\alpha \in [-B, 0]$ case follows in similar fashion.

Lipschitz continuity of $\bar{g}_i^*$ follows directly, or alternatively also from the general result that g_i^* is L-Lipschitz if and only if g_i has L-bounded support (Rockafellar, 1997, Corollary 13.3.3) or (Dünner et al., 2016, Lemma 5). □

Appendix C. Comparison to ADMM

Here we derive the comparison of ADMM and CoCoA discussed in Section 3.6, following the line of reasoning in Yang (2013). For consensus ADMM, the objective (B) is decomposed using the following re-parameterization:

$$\begin{aligned} \max_{\mathbf{w}_1,\dots\mathbf{w}_K,\mathbf{w}} \quad & \sum_{k=1}^{K}\sum_{i\in\mathcal{P}_k} g^*(-\mathbf{x}_i^\top \mathbf{w}_k) + f^*(\mathbf{w}) \\ s.t. \quad & \mathbf{w}_k = \mathbf{w},\ k = 1,\dots,K. \end{aligned}$$

To solve this problem, we construct the augmented Lagrangian:

$$\begin{aligned} L_\rho(\mathbf{w}_1,\dots,\mathbf{w}_k,\mathbf{u}_1,\dots,\mathbf{u}_k,\mathbf{w}) := &\sum_{k=1}^{K}\sum_{i\in\mathcal{P}_k} g^*(-\mathbf{x}_i^\top \mathbf{w}_k) \\ &+ f^*(\mathbf{w}) + \rho\sum_{k=1}^{K}\mathbf{u}_k^\top(\mathbf{w}_k-\mathbf{w}) + \frac{\rho}{2}\sum_{k=1}^{K}\|\mathbf{w}_k-\mathbf{w}\|^2, \end{aligned}$$

which yields the following decomposable updates:

$$\begin{aligned} \mathbf{w}_k^{(t)} &= \arg\min_{\mathbf{w}_k} \sum_{i\in\mathcal{P}_k} g^*(-\mathbf{x}_i^\top \mathbf{w}_k) + \frac{\rho}{2}\|\mathbf{w}_k - \mathbf{w}^{(t-1)} + \mathbf{u}_k^{(t-1)}\|^2, \\ \mathbf{w}^{(t)} &= \arg\min_{\mathbf{w}} f^*(\mathbf{w}) + \rho\sum_{k=1}^{K}\mathbf{u}_k^\top(\mathbf{w}_k-\mathbf{w}) + \frac{\rho}{2}\sum_{k=1}^{K}\|\mathbf{w}_k-\mathbf{w}\|^2, \\ \mathbf{u}_k^{(t)} &= \mathbf{u}_k^{(t-1)} + \mathbf{w}_k^{(t)} - \mathbf{w}^{(t)}. \end{aligned}$$

To compare this to the proposed framework, recall that the subproblem (10) (excluding the extraneous term $f(\mathbf{v})$) can be written as:

$$\min_{\boldsymbol{\alpha}_{[k]}\in\mathbb{R}^n} \sum_{i\in\mathcal{P}_k} g_i(\boldsymbol{\alpha}_{[k]_i}) + \mathbf{w}^\top A\boldsymbol{\alpha}_{[k]} + \frac{\sigma'}{2\tau}\left\|A_{[k]}\boldsymbol{\alpha}_{[k]}\right\|^2.$$

We can further reformulate by completing the square:

$$\min_{\boldsymbol{\alpha}_{[k]}\in\mathbb{R}^n} \sum_{i\in\mathcal{P}_k} g_i((\boldsymbol{\alpha}_{[k]})_i) + \frac{\tau}{2\sigma'}\left\|\ \mathbf{w} + \frac{\sigma'}{\tau}A_{[k]}\boldsymbol{\alpha}_{[k]}\right\|^2.$$

Assuming for the time being that $f(\cdot) = \frac{1}{2}\|\cdot\|_2^2$ such that $\mathbf{w} = \nabla f(\mathbf{v}) = \mathbf{v}$, we can unroll the update as follows, using $\gamma\Delta\mathbf{v}^{(t-1)} = \gamma\sum_{i=1}^{K}\Delta\mathbf{v}_k^{(t-1)}$:

$$\min_{\boldsymbol{\alpha}_{[k]}\in\mathbb{R}^n} \sum_{i\in\mathcal{P}_k} g_i((\boldsymbol{\alpha}_{[k]})_i) + \frac{\tau}{2\sigma'}\left\|\ \mathbf{w}^{(t-1)} + \gamma\Delta\mathbf{v}^{(t-1)} + \frac{\sigma'}{\tau}A_{[k]}\boldsymbol{\alpha}_{[k]}\right\|^2.$$

We will show that the above objective has the following primal form for each machine k:

$$\min_{\mathbf{w}} \sum_{i\in\mathcal{P}_k} g_i^*(-\mathbf{x}_i^\top \mathbf{w}) + \frac{\tau}{2\sigma'}\left\|\mathbf{w} - \left(\mathbf{w}^{(t-1)} + \gamma\Delta\mathbf{v}^{(t-1)}\right)\right\|^2. \tag{31}$$

Indeed, suppressing the subscript k for simplicity, we have:

$$\begin{aligned}
&\min_{\mathbf{w}} \sum_i g_i^*(-\mathbf{x}_i^\top \mathbf{w}) + \frac{\tau}{2\sigma'} \Big\| \mathbf{w} - \Big(\mathbf{w}^{(t-1)} + \gamma \Delta \mathbf{v}^{(t-1)} \Big) \Big\|^2 \\
=& \min_{\mathbf{w}} \sum_i \max_{\alpha_i} -\mathbf{x}_i^\top \mathbf{w} \alpha_i - g_i(\alpha_i) + \frac{\tau}{2\sigma'} \Big\| \mathbf{w} - \Big(\mathbf{w}^{(t-1)} + \gamma \Delta \mathbf{v}^{(t-1)} \Big) \Big\|^2 \\
=& \max_{\boldsymbol{\alpha}} \min_{\mathbf{w}} \sum_i -\mathbf{x}_i^\top \mathbf{w} \alpha_i - g_i(\alpha_i) + \frac{\tau}{2\sigma'} \Big\| \mathbf{w} - \Big(\mathbf{w}^{(t-1)} + \gamma \Delta \mathbf{v}^{(t-1)} \Big) \Big\|^2 .
\end{aligned}$$

Solving the minimization yields: $\mathbf{w} = \mathbf{w}^{t-1} + \gamma \Delta \mathbf{v}^{(t-1)} + \frac{\sigma'}{\tau} A\boldsymbol{\alpha}$. Plugging this back in yields:

$$\begin{aligned}
=& \max_{\alpha} \sum_i -g_i(\alpha_i) - (A\boldsymbol{\alpha})^\top \mathbf{w}^{(t-1)} - (A\boldsymbol{\alpha})^\top \gamma \Delta \mathbf{v}^{(t-1)} - \frac{\sigma'}{\tau} \|A\boldsymbol{\alpha}\|^2 + \frac{\tau}{2\sigma'} \Big\| \frac{\sigma'}{\tau} A\boldsymbol{\alpha} \Big\|^2 \\
=& \max_{\boldsymbol{\alpha}} \sum_i -g_i(\alpha_i) - (A\boldsymbol{\alpha})^\top \mathbf{w}^{(t-1)} - (A\boldsymbol{\alpha})^\top \gamma \Delta \mathbf{v}^{(t-1)} - \frac{\sigma'}{2\tau} \|A\boldsymbol{\alpha}\|^2 \\
=& \min_{\boldsymbol{\alpha}} \sum_i g_i(\alpha_i) + (A\boldsymbol{\alpha})^\top \mathbf{w}^{(t-1)} + (A\boldsymbol{\alpha})^\top \gamma \Delta \mathbf{v}^{(t-1)} + \frac{\sigma'}{2\tau} \|A\boldsymbol{\alpha}\|^2 \\
=& \min_{\boldsymbol{\alpha}} \sum_i g_i(\alpha_i) + \frac{\tau}{2\sigma'} \Big\| \mathbf{w}^{(t-1)} + \gamma \Delta \mathbf{v}^{(t-1)} + \frac{\sigma'}{\tau} A\boldsymbol{\alpha} \Big\|^2 .
\end{aligned}$$

Appendix D. Convergence Proofs

In this section we provide proofs of our main convergence results. The arguments follow the reasoning in Ma et al. (2015b,a), but where we have generalized them to be applicable directly to (A). We provide full details of Lemma 1 as a proof of concept, but omit details in later proofs that can be derived using the arguments in Ma et al. (2015b) or earlier work of Shalev-Shwartz and Zhang (2013a), and instead outline the proof strategy and highlight sections where the theory deviates.

D.1 Approximation of $\mathcal{O}_A(\cdot)$ by the Local Subproblems $\mathcal{G}_k^{\sigma'}(\cdot)$

Our first lemma in the overall proof of convergence helps to relate progress on the local subproblems to the global objective $\mathcal{O}_A(\cdot)$.

Lemma' 1. *For any dual variables $\boldsymbol{\alpha}, \Delta\boldsymbol{\alpha} \in \mathbb{R}^n$, $\mathbf{v} = \mathbf{v}(\boldsymbol{\alpha}) := A\boldsymbol{\alpha}$, and real values γ, σ' satisfying* (11)*, it holds that*

$$\mathcal{O}_A\Big(\boldsymbol{\alpha} + \gamma \sum_{k=1}^K \Delta\boldsymbol{\alpha}_{[k]}\Big) \leq (1-\gamma)\mathcal{O}_A(\boldsymbol{\alpha}) + \gamma \sum_{k=1}^K \mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]}; \mathbf{v}, \boldsymbol{\alpha}_{[k]}) \,. \tag{32}$$

Proof. In this proof we follow the line of reasoning in Ma et al. (2015b, Lemma 4) with a more general $(1/\tau)$ smoothness assumption on $f(\cdot)$. An outer iteration of CoCoA performs

the following update:

$$\mathcal{O}_A(\boldsymbol{\alpha}+\gamma\sum_{k=1}^{K}\Delta\boldsymbol{\alpha}_{[k]}) = \underbrace{f(\mathbf{v}(\boldsymbol{\alpha}+\gamma\sum_{k=1}^{K}\Delta\boldsymbol{\alpha}_{[k]}))}_{A} + \underbrace{\sum_{i=1}^{n} g_i(\alpha_i+\gamma(\sum_{k=1}^{K}\Delta\boldsymbol{\alpha}_{[k]})_i)}_{B}\,. \tag{33}$$

We bound A and B separately. First we bound A using $(1/\tau)$-smoothness of f:

$$\begin{aligned}
A &= f\Big(\mathbf{v}(\boldsymbol{\alpha}+\gamma\sum_{k=1}^{K}\Delta\boldsymbol{\alpha}_{[k]})\Big) = f\Big(\mathbf{v}(\boldsymbol{\alpha})+\gamma\sum_{k=1}^{K}\mathbf{v}(\Delta\boldsymbol{\alpha}_{[k]})\Big)\\
&\overset{\text{smoothness of } f \text{ as in (3)}}{\leq} f(\mathbf{v}(\boldsymbol{\alpha})) + \sum_{k=1}^{K}\gamma\nabla f(\mathbf{v}(\boldsymbol{\alpha}))^\top\mathbf{v}(\Delta\boldsymbol{\alpha}_{[k]}) + \frac{\gamma^2}{2\tau}\|\sum_{k=1}^{K}\mathbf{v}(\boldsymbol{\alpha}_{[k]})\|^2\\
&\overset{\text{definition of } \mathbf{w} \text{ as in (5)}}{\leq} f(\mathbf{v}(\boldsymbol{\alpha})) + \sum_{k=1}^{K}\gamma\mathbf{v}(\Delta\boldsymbol{\alpha}_{[k]})^\top\mathbf{w}(\boldsymbol{\alpha}) + \frac{\gamma^2}{2\tau}\|\sum_{k=1}^{K}\mathbf{v}(\boldsymbol{\alpha}_{[k]})\|^2\\
&\overset{\text{safe choice of } \sigma' \text{ as in (11)}}{\leq} f(\mathbf{v}(\boldsymbol{\alpha})) + \sum_{k=1}^{K}\gamma\mathbf{v}(\Delta\boldsymbol{\alpha}_{[k]})^\top\mathbf{w}(\boldsymbol{\alpha}) + \frac{1}{2\tau}\gamma\sigma'\sum_{k=1}^{K}\|\mathbf{v}(\boldsymbol{\alpha}_{[k]})\|^2\,.
\end{aligned}$$

Next we use Jensen's inequality to bound B:

$$\begin{aligned}
B = \sum_{k=1}^{K}\left(\sum_{i\in\mathcal{P}_k} g_i(\alpha_i+\gamma(\Delta\boldsymbol{\alpha}_{[k]})_i)\right) &= \sum_{k=1}^{K}\left(\sum_{i\in\mathcal{P}_k} g_i((1-\gamma)\alpha_i+\gamma(\boldsymbol{\alpha}+\Delta\boldsymbol{\alpha}_{[k]})_i)\right)\\
&\leq \sum_{k=1}^{K}\left(\sum_{i\in\mathcal{P}_k}(1-\gamma)g_i(\alpha_i)+\gamma g_i(\boldsymbol{\alpha}_i+\Delta\boldsymbol{\alpha}_{[k]_i})\right)\,.
\end{aligned}$$

Plugging A and B back into (33) yields:

$$\begin{aligned}
\mathcal{O}_A\Big(\boldsymbol{\alpha}+\gamma\sum_{k=1}^{K}\Delta\boldsymbol{\alpha}_{[k]}\Big) \leq\; & f(\mathbf{v}(\boldsymbol{\alpha})) \pm \gamma f(\mathbf{v}(\boldsymbol{\alpha})) + \sum_{k=1}^{K}\gamma\mathbf{v}(\Delta\boldsymbol{\alpha}_{[k]})^\top\mathbf{w}(\boldsymbol{\alpha}) + \frac{1}{2\tau}\gamma\sigma'\sum_{k=1}^{K}\|\mathbf{v}(\boldsymbol{\alpha}_{[k]})\|^2\\
&+\sum_{k=1}^{K}\sum_{i\in\mathcal{P}_k}(1-\gamma)g_i(\alpha_i)+\gamma g_i(\boldsymbol{\alpha}_i+\Delta\boldsymbol{\alpha}_{[k]_i})\\
=\; & \underbrace{(1-\gamma)f(\mathbf{v}(\boldsymbol{\alpha})) + \sum_{k=1}^{K}\left(\sum_{i\in\mathcal{P}_k}(1-\gamma)g_i(\alpha_i)\right)}_{(1-\gamma)\mathcal{O}_A(\boldsymbol{\alpha})}\\
&+\gamma\sum_{k=1}^{K}\left(\frac{1}{K}f(\mathbf{v}(\boldsymbol{\alpha})) + \mathbf{v}(\Delta\boldsymbol{\alpha}_{[k]})^\top\mathbf{w}(\boldsymbol{\alpha}) + \frac{\sigma'}{2\tau}\|\mathbf{v}(\boldsymbol{\alpha}_{[k]})\|^2 + \sum_{i\in\mathcal{P}_k} g_i(\boldsymbol{\alpha}_i+\Delta\boldsymbol{\alpha}_{[k]_i})\right)\\
\overset{(10)}{=}\; & (1-\gamma)\mathcal{O}_A(\boldsymbol{\alpha}) + \gamma\sum_{k=1}^{K}\mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]};\mathbf{v})\,,
\end{aligned}$$

where the last equality is by the definition of the subproblem objective $\mathcal{G}_k^{\sigma'}(.)$ as in (10). $\square$

D.2 Proof of Main Convergence Result (Theorem 2)

Before proving the main convergence results, we introduce several useful quantities, and establish the following lemma, which characterizes the effect of iterations of Algorithm 1 on the duality gap for any chosen local solver of approximation quality Θ.

Lemma 7. *Let g_i be strongly convex [2] with convexity parameter $\mu \geq 0$ with respect to the norm $\|\cdot\|$, $\forall i \in [n]$. Then at each iteration of Algorithm 1 under Assumption 1, and any $s \in [0,1]$, it holds that*

$$\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^{(t+1)})] \geq \gamma(1-\Theta)\Big(sG(\boldsymbol{\alpha}^{(t)}) - \frac{\sigma' s^2}{2\tau} R^{(t)}\Big)\,, \tag{34}$$

where

$$R^{(t)} := -\tfrac{\tau\mu(1-s)}{\sigma' s}\|\mathbf{u}^{(t)} - \boldsymbol{\alpha}^{(t)}\|^2 + \textstyle\sum_{k=1}^K \|A_{[k]}(\mathbf{u}^{(t)} - \boldsymbol{\alpha}^{(t)})_{[k]}\|^2\,, \tag{35}$$

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

$$u_i^{(t)} \in \partial g_i^*(-\mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha}^{(t)}))\,. \tag{36}$$

Proof. This proof is motivated by Shalev-Shwartz and Zhang (2013a, Lemma 19) and follows Ma et al. (2015b, Lemma 5), with a difference being the extension to our generalized subproblems $\mathcal{G}_k^{\sigma'}(\cdot; \mathbf{v}, \boldsymbol{\alpha}_{[k]})$ along with the mappings $\mathbf{w}(\boldsymbol{\alpha}) := \nabla f(\mathbf{v}(\boldsymbol{\alpha}))$ with $\mathbf{v}(\boldsymbol{\alpha}) := A\boldsymbol{\alpha}$.

For simplicity, we write $\boldsymbol{\alpha}$ instead of $\boldsymbol{\alpha}^{(t)}$, $\mathbf{v}$ instead of $\mathbf{v}(\boldsymbol{\alpha}^{(t)})$, $\mathbf{w}$ instead of $\mathbf{w}(\boldsymbol{\alpha}^{(t)})$ and $\mathbf{u}$ instead of $\mathbf{u}^{(t)}$. We can estimate the expected change of the objective $\mathcal{O}_A(\boldsymbol{\alpha})$ as follows. Starting from the definition of the update $\boldsymbol{\alpha}^{(t+1)} := \boldsymbol{\alpha}^{(t)} + \gamma \sum_k \Delta\boldsymbol{\alpha}_{[k]}$ from Algorithm 1, we apply Lemma 1, which relates the local approximation $\mathcal{G}_k^{\sigma'}(\boldsymbol{\alpha}; \mathbf{v}, \boldsymbol{\alpha}_{[k]})$ to the global objective $\mathcal{O}_A(\boldsymbol{\alpha})$, and then bound this using the notion of quality of the local solver (Θ), as in Assumption 1. This gives us:

$$\begin{aligned}
\mathbb{E}\big[\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^{(t+1)})\big] &= \mathbb{E}\Big[\mathcal{O}_A(\boldsymbol{\alpha}) - \mathcal{O}_A\Big(\boldsymbol{\alpha} + \gamma \sum_{k=1}^K \Delta\boldsymbol{\alpha}_{[k]}\Big)\Big] \\
&\geq \gamma(1-\Theta)\left(\underbrace{\mathcal{O}_A(\boldsymbol{\alpha}) - \sum_{k=1}^K \mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}^\star_{[k]}; \mathbf{v}, \boldsymbol{\alpha}_{[k]})}_{C}\right)\,.
\end{aligned} \tag{37}$$

We next upper bound the C term, denoting $\Delta\boldsymbol{\alpha}^\star = \sum_{k=1}^K \Delta\boldsymbol{\alpha}^\star_{[k]}$. We first plug in the definition of the objective $\mathcal{O}_A$ in (A) and the local subproblems (10), and then substitute

2. Note that the case of weakly convex $g_i(.)$ is explicitly allowed here as well, as the Lemma holds for the case $\mu = 0$.

$s(u_i - \alpha_i)$ for $\Delta\boldsymbol{\alpha}_i^\star$ and apply the μ-strong convexity of the g_i terms. This gives us:

$$\begin{aligned} C &= \sum_{i=1}^{n} \left(g_i(\alpha_i) - g_i(\alpha_i + \Delta\boldsymbol{\alpha}_i^\star)\right) - (A\Delta\boldsymbol{\alpha}^\star)^\top \mathbf{w}(\boldsymbol{\alpha}) - \sum_{k=1}^{K} \frac{\sigma'}{2\tau} \left\| A_{[k]} \Delta\boldsymbol{\alpha}_{[k]}^\star \right\|^2 \\ &\geq \sum_{i=1}^{n} \left(s g_i(\alpha_i) - s g_i(u_i) + \frac{\mu}{2}(1-s)s(u_i - \alpha_i)^2 \right) \\ &\qquad - A(s(\mathbf{u} - \boldsymbol{\alpha}))^\top \mathbf{w}(\boldsymbol{\alpha}) - \sum_{k=1}^{K} \frac{\sigma'}{2\tau} \left\| A_{[k]} (s(\mathbf{u} - \boldsymbol{\alpha})_{[k]}) \right\|^2 . \end{aligned} \tag{38}$$

From the definition of the optimization problems (A) and (B), and definition of convex conjugates, we can write the duality gap as:

$$\begin{aligned} G(\boldsymbol{\alpha}) := \mathcal{O}_A(\boldsymbol{\alpha}) - (-\mathcal{O}_B(\mathbf{w}(\boldsymbol{\alpha})) &\overset{\text{(A),(B)}}{=} \sum_{i=1}^{n} \left(g_i^*(-\mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha})) + g_i(\alpha_i) \right) + f^*(\mathbf{w}(\boldsymbol{\alpha})) + f(A\boldsymbol{\alpha}) \\ &= \sum_{i=1}^{n} \left(g_i^*(-\mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha})) + g_i(\alpha_i) \right) + f^*(\nabla f(A\boldsymbol{\alpha})) + f(A\boldsymbol{\alpha}) \\ &= \sum_{i=1}^{n} \left(g_i^*(-\mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha})) + g_i(\alpha_i) \right) + (A\boldsymbol{\alpha})^\top \mathbf{w}(\boldsymbol{\alpha}) \\ &= \sum_{i=1}^{n} \left(g_i^*(-\mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha})) + g_i(\alpha_i) + \alpha_i \mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha}) \right) . \end{aligned} \tag{39}$$

The convex conjugate maximal property from (36) implies that

$$g_i(u_i) = u_i(-\mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha})) - g_i^*(-\mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha})) . \tag{40}$$

Using (40) and (39), we therefore have:

$$\begin{aligned} C &\overset{(40)}{\geq} \sum_{i=1}^{n} \left(s g_i(\alpha_i) - s u_i(-\mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha})) + s g_i^*(-\mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha})) + \frac{\mu}{2}(1-s)s(u_i - \alpha_i)^2 \right) \\ &\qquad - A(s(\mathbf{u} - \boldsymbol{\alpha}))^\top \mathbf{w}(\boldsymbol{\alpha}) - \sum_{k=1}^{K} \frac{\sigma'}{2\tau} \left\| A_{[k]} (s(\mathbf{u} - \boldsymbol{\alpha})_{[k]}) \right\|^2 \\ &= \sum_{i=1}^{n} \left[s g_i(\alpha_i) + s g_i^*(-\mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha})) + s \mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha}) \alpha_i \right] - \sum_{i=1}^{n} \left[s \mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha})(\alpha_i - u_i) - \frac{\mu}{2}(1-s)s(u_i - \alpha_i)^2 \right] \\ &\qquad - A(s(\mathbf{u} - \boldsymbol{\alpha}))^\top \mathbf{w}(\boldsymbol{\alpha}) - \sum_{k=1}^{K} \frac{\sigma'}{2\tau} \left\| A_{[k]} (s(\mathbf{u} - \boldsymbol{\alpha})_{[k]}) \right\|^2 \\ &\overset{(39)}{=} s G(\boldsymbol{\alpha}) + \frac{\mu}{2}(1-s)s\|\mathbf{u} - \boldsymbol{\alpha}\|^2 - \frac{\sigma' s^2}{2\tau} \sum_{k=1}^{K} \|A_{[k]}(\mathbf{u} - \boldsymbol{\alpha})_{[k]}\|^2 . \end{aligned} \tag{41}$$

The claimed improvement bound (34) then follows by plugging (41) into (37). □

The following Lemma provides a uniform bound on $R^{(t)}$:

Lemma 8. *If g_i^* are L-Lipschitz continuous for all $i \in [n]$, then*

$$\forall t : R^{(t)} \leq 4L^2 \underbrace{\sum_{k=1}^{K} \sigma_k n_k}_{=:\sigma} , \tag{42}$$

where

$$\sigma_k := \max_{\boldsymbol{\alpha}_{[k]} \in \mathbb{R}^n} \frac{\|A_{[k]}\boldsymbol{\alpha}_{[k]}\|^2}{\|\boldsymbol{\alpha}_{[k]}\|^2} . \tag{43}$$

Proof. (Ma et al., 2015b, Lemma 6). For general convex functions, the strong convexity parameter is $\mu = 0$, and hence the definition (35) of the complexity constant $R^{(t)}$ becomes

$$R^{(t)} = \sum_{k=1}^{K} \|A_{[k]}(\mathbf{u}^{(t)} - \boldsymbol{\alpha}^{(t)})_{[k]}\|^2 \overset{(43)}{\leq} \sum_{k=1}^{K} \sigma_k \|(\mathbf{u}^{(t)} - \boldsymbol{\alpha}^{(t)})_{[k]}\|^2 \leq \sum_{k=1}^{K} \sigma_k |\mathcal{P}_k| 4L^2 .$$

Here the last inequality follows from (Shalev-Shwartz and Zhang, 2013a, Lemma 21), which shows that for $g_i^* : \mathbb{R} \to \mathbb{R}$ being L-Lipschitz, it holds that for any real value a with $|a| > L$ one has that $g_i(a) = +\infty$. □

Remark 5. *(Ma et al., 2015b, Remark 7) If the data points $\mathbf{x}_i$ are normalized such that $\|\mathbf{x}_i\| \leq 1$, $\forall i \in [n]$, then $\sigma_k \leq |\mathcal{P}_k| = n_k$. Furthermore, if we assume that the data partition is balanced, i.e., that $n_k = n/K$ for all k, then $\sigma \leq n^2/K$. This can be used to bound the constants $R^{(t)}$, above, as $R^{(t)} \leq \frac{4L^2 n^2}{K}$.*

Theorem 9. *Consider Algorithm 1, using a local solver of quality Θ (See Assumption 1). Let $g_i^*(\cdot)$ be L-Lipschitz continuous, and $\epsilon_G > 0$ be the desired duality gap (and hence an upper-bound on suboptimality $\epsilon_{\mathcal{O}_A}$). Then after T iterations, where*

$$T \geq T_0 + \max\{\left\lceil \frac{1}{\gamma(1-\Theta)} \right\rceil, \frac{4L^2\sigma\sigma'}{\tau\epsilon_G\gamma(1-\Theta)}\} , \tag{44}$$

$$T_0 \geq t_0 + \left[\frac{2}{\gamma(1-\Theta)}\left(\frac{8L^2\sigma\sigma'}{\tau\epsilon_G} - 1\right)\right]_+ , \quad t_0 \geq \max(0, \left\lceil \tfrac{1}{\gamma(1-\Theta)} \log\left(\tfrac{\tau(\mathcal{O}_A(\boldsymbol{\alpha}^{(0)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star))}{2L^2\sigma\sigma'}\right)\right\rceil) ,$$

we have that the expected duality gap satisfies

$$\mathbb{E}[\mathcal{O}_A(\overline{\boldsymbol{\alpha}}) - (-\mathcal{O}_B(\mathbf{w}(\overline{\boldsymbol{\alpha}})))] \leq \epsilon_G$$

at the averaged iterate

$$\overline{\boldsymbol{\alpha}} := \tfrac{1}{T-T_0}\textstyle\sum_{t=T_0+1}^{T-1} \boldsymbol{\alpha}^{(t)} . \tag{45}$$

Proof. We begin by estimating the expected change of feasibility for $\mathcal{O}_A$. We can bound this above by using Lemma 7 and the fact that the $\mathcal{O}_B(\cdot)$ is always a lower bound for $-\mathcal{O}_A(\cdot)$, and then applying (42) to find:

$$\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(t+1)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)] \leq (1 - \gamma(1-\Theta)s)\,(\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)) + \gamma(1-\Theta)\tfrac{\sigma' s^2}{2\tau} 4L^2\sigma . \tag{46}$$

Using (46) recursively we have

$$\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)] \leq (1-\gamma(1-\Theta)s)^t \left(\mathcal{O}_A(\boldsymbol{\alpha}^{(0)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)\right) + s\frac{4L^2\sigma\sigma'}{2\tau}\,. \tag{47}$$

Choosing $s=1$ and $t=t_0 := \max\{0, \lceil \frac{1}{\gamma(1-\Theta)} \log(2(\mathcal{O}_A(\boldsymbol{\alpha}^{(0)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star))/(4L^2\sigma\sigma')) \rceil\}$ leads to

$$\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)] \leq (1-\gamma(1-\Theta))^{t_0} \left(\mathcal{O}_A(\boldsymbol{\alpha}^{(0)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)\right) + \frac{4L^2\sigma\sigma'}{2\tau} \leq \frac{4L^2\sigma\sigma'}{\tau}\,. \tag{48}$$

Next, we show inductively that

$$\forall t \geq t_0 : \mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)] \leq \frac{4L^2\sigma\sigma'}{\tau(1+\frac{1}{2}\gamma(1-\Theta)(t-t_0))}\,. \tag{49}$$

Clearly, (48) implies that (49) holds for $t=t_0$. Assuming that it holds for any $t \geq t_0$, we show that it must also hold for $t+1$. Indeed, using

$$s = \frac{1}{1+\frac{1}{2}\gamma(1-\Theta)(t-t_0)} \in [0,1]\,, \tag{50}$$

we obtain

$$\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(t+1)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)] \leq \frac{4L^2\sigma\sigma'}{\tau} \underbrace{\left(\frac{1+\frac{1}{2}\gamma(1-\Theta)(t-t_0) - \frac{1}{2}\gamma(1-\Theta)}{(1+\frac{1}{2}\gamma(1-\Theta)(t-t_0))^2}\right)}_{D}$$

by applying the bounds (46) and (49), plugging in the definition of s (50), and simplifying. We upper bound the term D using the fact that geometric mean is less or equal to arithmetic mean:

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

If $\overline{\boldsymbol{\alpha}}$ is defined as (45), we apply the results of Lemma 7 and Lemma 8 to obtain

$$\begin{aligned} \mathbb{E}[G(\overline{\boldsymbol{\alpha}})] = \mathbb{E}\left[G\left(\sum_{t=T_0}^{T-1} \tfrac{1}{T-T_0}\boldsymbol{\alpha}^{(t)}\right)\right] &\leq \tfrac{1}{T-T_0}\mathbb{E}\left[\sum_{t=T_0}^{T-1} G\left(\boldsymbol{\alpha}^{(t)}\right)\right] \\ &\leq \frac{1}{\gamma(1-\Theta)s}\frac{1}{T-T_0}\mathbb{E}\left[\mathcal{O}_A(\boldsymbol{\alpha}^{(T_0)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)\right] + \tfrac{4L^2\sigma\sigma' s}{2\tau}\,. \end{aligned} \tag{51}$$

If $T \geq \lceil \frac{1}{\gamma(1-\Theta)} \rceil + T_0$ such that $T_0 \geq t_0$ we have

$$\mathbb{E}[G(\overline{\boldsymbol{\alpha}})] \overset{(51),(49)}{\leq} \frac{1}{\gamma(1-\Theta)s}\frac{1}{T-T_0}\left(\frac{4L^2\sigma\sigma'}{\tau(1+\frac{1}{2}\gamma(1-\Theta)(T_0-t_0))}\right) + \frac{4L^2\sigma\sigma' s}{2\tau}$$
$$= \frac{4L^2\sigma\sigma'}{\tau}\left(\frac{1}{\gamma(1-\Theta)s}\frac{1}{T-T_0}\frac{1}{1+\frac{1}{2}\gamma(1-\Theta)(T_0-t_0)} + \frac{s}{2}\right). \tag{52}$$

Choosing

$$s = \frac{1}{(T-T_0)\gamma(1-\Theta)} \in [0,1] \tag{53}$$

gives us

$$\mathbb{E}[G(\overline{\boldsymbol{\alpha}})] \overset{(52),(53)}{\leq} \frac{4L^2\sigma\sigma'}{\tau}\left(\frac{1}{1+\frac{1}{2}\gamma(1-\Theta)(T_0-t_0)} + \frac{1}{(T-T_0)\gamma(1-\Theta)}\frac{1}{2}\right). \tag{54}$$

To have right hand side of (54) smaller then ϵ_G it is sufficient to choose T_0 and T such that

$$\frac{4L^2\sigma\sigma'}{\tau}\left(\frac{1}{1+\frac{1}{2}\gamma(1-\Theta)(T_0-t_0)}\right) \leq \frac{1}{2}\epsilon_G\,, \tag{55}$$

$$\frac{4L^2\sigma\sigma'}{\tau}\left(\frac{1}{(T-T_0)\gamma(1-\Theta)}\frac{1}{2}\right) \leq \frac{1}{2}\epsilon_G\,. \tag{56}$$

Hence if $T_0 \geq t_0 + \frac{2}{\gamma(1-\Theta)}\left(\frac{8L^2\sigma\sigma'}{\tau\epsilon_G} - 1\right)$ and $T \geq T_0 + \frac{4L^2\sigma\sigma'}{\tau\epsilon_G\gamma(1-\Theta)}$ then (55) and (56) are satisfied. □

The following main theorem simplifies the results of Theorem 9 and is a generalization of Ma et al. (2015b, Corollary 9) for general $f^*(\cdot)$ functions:

Theorem' 2. *Consider Algorithm 1 with $\gamma := 1$, using a local solver of quality Θ (see Assumption 1). Let $g_i^*(\cdot)$ be L-Lipschitz continuous, and assume that the columns of A satisfy $\|\mathbf{x}_i\| \leq 1$, $\forall i \in [n]$. Let $\epsilon_G > 0$ be the desired duality gap (and hence an upper-bound on primal sub-optimality). Then after T iterations, where*

$$T \geq T_0 + \max\{\left\lceil \frac{1}{1-\Theta} \right\rceil, \frac{4L^2n^2}{\tau\epsilon_G(1-\Theta)}\}\,, \tag{57}$$
$$T_0 \geq t_0 + \left[\frac{2}{1-\Theta}\left(\frac{8L^2n^2}{\tau\epsilon_G} - 1\right)\right]_+\,,$$
$$t_0 \geq \max(0, \left\lceil \tfrac{1}{(1-\Theta)} \log\left(\tfrac{\tau(\mathcal{O}_A(\boldsymbol{\alpha}^{(0)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star))}{2L^2Kn}\right)\right\rceil)\,,$$

we have that the expected duality gap satisfies

$$\mathbb{E}[\mathcal{O}_A(\overline{\boldsymbol{\alpha}}) - (-\mathcal{O}_B(\mathbf{w}(\overline{\boldsymbol{\alpha}})))] \leq \epsilon_G\,,$$

where $\overline{\boldsymbol{\alpha}}$ is the averaged iterate returned by Algorithm 1.

Proof. Plug in parameters $\gamma := 1$, $\sigma' := \gamma K = K$ to the results of Theorem 9, and note that for balanced datasets we have $\sigma \leq \frac{n^2}{K}$ (see Remark 5). We can further simplify the rate by noting that $\tau = 1$ for the 1-smooth losses (least squares and logistic) given as examples in this work. □

D.3 Proof of Convergence Result for Strongly Convex g_i

Our second main theorem follows reasoning in Shalev-Shwartz and Zhang (2013a) and is a generalization of Ma et al. (2015b, Corollary 11). We first introduce a lemma to simplify the proof.

Lemma 10. *Assume that $g_i(0) \in [0,1]$ for all $i \in [n]$, then for the zero vector $\boldsymbol{\alpha}^{(0)} := \mathbf{0} \in \mathbb{R}^n$, we have*

$$\mathcal{O}_A(\boldsymbol{\alpha}^{(0)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star) = \mathcal{O}_A(\mathbf{0}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star) \leq n\,. \tag{58}$$

Proof. For $\boldsymbol{\alpha} := \mathbf{0} \in \mathbb{R}^n$, we have $\mathbf{w}(\boldsymbol{\alpha}) = A\boldsymbol{\alpha} = \mathbf{0} \in \mathbb{R}^d$. Therefore, since the dual $-\mathcal{O}_A(\cdot)$ is always a lower bound on the primal $\mathcal{O}_B(\cdot)$, and by definition of the objective $\mathcal{O}_A$ given in (A),

$$0 \leq \mathcal{O}_A(\boldsymbol{\alpha}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star) \leq \mathcal{O}_A(\boldsymbol{\alpha}) - (-\mathcal{O}_B(\mathbf{w}(\boldsymbol{\alpha}))) \overset{\text{(A)}}{\leq} n\,. \qquad \square$$

Theorem 11. *Assume that g_i are μ-strongly convex $\forall i \in [n]$. We define $\sigma_{\max} = \max_{k\in[K]} \sigma_k$. Then after T iterations of Algorithm 1, with*

$$T \geq \tfrac{1}{\gamma(1-\Theta)} \tfrac{\mu\tau + \sigma_{\max}\sigma'}{\mu\tau} \log \tfrac{n}{\epsilon_{\mathcal{O}_A}}\,,$$

it holds that

$$\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(T)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)] \leq \epsilon_{\mathcal{O}_A}\,.$$

Furthermore, after T iterations with

$$T \geq \tfrac{1}{\gamma(1-\Theta)} \tfrac{\mu\tau + \sigma_{\max}\sigma'}{\mu\tau} \log\left(\tfrac{1}{\gamma(1-\Theta)} \tfrac{\mu\tau + \sigma_{\max}\sigma'}{\mu\tau} \tfrac{n}{\epsilon_G}\right)\,,$$

we have the expected duality gap

$$\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(T)}) - (-\mathcal{O}_B(\mathbf{w}(\boldsymbol{\alpha}^{(T)})))] \leq \epsilon_G\,.$$

Proof. Given that $g_i(.)$ is μ-strongly convex with respect to the $\|\cdot\|$ norm, we can apply (35) and the definition of σ_k to find:

$$\begin{aligned} R^{(t)} &\leq -\tfrac{\tau\mu(1-s)}{\sigma' s}\|\mathbf{u}^{(t)} - \boldsymbol{\alpha}^{(t)}\|^2 + \textstyle\sum_{k=1}^K \sigma_k \|\mathbf{u}^{(t)} - \boldsymbol{\alpha}^{(t)}_{[k]}\|^2 \\ &\leq \left(-\tfrac{\tau\mu(1-s)}{\sigma' s} + \sigma_{\max}\right)\|\mathbf{u}^{(t)} - \boldsymbol{\alpha}^{(t)}\|^2\,, \end{aligned} \tag{59}$$

where $\sigma_{\max} = \max_{k\in[K]} \sigma_k$. If we plug the following value of s

$$s = \frac{\tau\mu}{\tau\mu + \sigma_{\max}\sigma'} \in [0,1] \tag{60}$$

into (59) we obtain that $\forall t : R^{(t)} \leq 0$. Putting the same s into (34) will give us

$$\begin{aligned} \mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^{(t+1)})] &\overset{(34),(60)}{\geq} \gamma(1-\Theta)\frac{\tau\mu}{\tau\mu + \sigma_{\max}\sigma'} G(\boldsymbol{\alpha}^{(t)}) \\ &\geq \gamma(1-\Theta)\frac{\tau\mu}{\tau\mu + \sigma_{\max}\sigma'}(\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star))\,. \end{aligned} \tag{61}$$

Using the fact that $\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(t)})-\mathcal{O}_A(\boldsymbol{\alpha}^{(t+1)})] = \mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^\star)-\mathcal{O}_A(\boldsymbol{\alpha}^{(t+1)})]+\mathcal{O}_A(\boldsymbol{\alpha}^{(t)})-\mathcal{O}_A(\boldsymbol{\alpha}^\star)$ we have

$$\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^\star) - \mathcal{O}_A(\boldsymbol{\alpha}^{(t+1)})] + \mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star) \overset{(61)}{\geq} \gamma(1-\Theta)\frac{\tau\mu}{\tau\mu+\sigma_{\max}\sigma'}(\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star))\,,$$

which is equivalent to

$$\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(t+1)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)] \leq \left(1-\gamma(1-\Theta)\frac{\tau\mu}{\tau\mu+\sigma_{\max}\sigma'}\right)(\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star))\,. \tag{62}$$

Therefore if we denote $\epsilon_{\mathcal{O}_A}^{(t)} = \mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)$ we have recursively that

$$\begin{aligned}\mathbb{E}[\epsilon_{\mathcal{O}_A}^{(t)}] &\overset{(62)}{\leq} \left(1-\gamma(1-\Theta)\frac{\tau\mu}{\tau\mu+\sigma_{\max}\sigma'}\right)^t \epsilon_{\mathcal{O}_A}^{(0)} \overset{(58)}{\leq} \left(1-\gamma(1-\Theta)\frac{\tau\mu}{\tau\mu+\sigma_{\max}\sigma'}\right)^t n \\ &\leq \exp\left(-t\gamma(1-\Theta)\frac{\tau\mu}{\tau\mu+\sigma_{\max}\sigma'}\right) n\,.\end{aligned}$$

The right hand side will be smaller than some $\epsilon_{\mathcal{O}_A}$ if

$$t \geq \frac{1}{\gamma(1-\Theta)}\frac{\tau\mu+\sigma_{\max}\sigma'}{\tau\mu}\log\frac{n}{\epsilon_{\mathcal{O}_A}}\,.$$

Moreover, to bound the duality gap, we have

$$\gamma(1-\Theta)\frac{\tau\mu}{\tau\mu+\sigma_{\max}\sigma'}G(\boldsymbol{\alpha}^{(t)}) \overset{(61)}{\leq} \mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^{(t+1)})] \leq \mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)]\,.$$

Thus, $G(\boldsymbol{\alpha}^{(t)}) \leq \frac{1}{\gamma(1-\Theta)}\frac{\tau\mu+\sigma_{\max}\sigma'}{\tau\mu}\epsilon_{\mathcal{O}_A}^{(t)}$. Hence if $\epsilon_{\mathcal{O}_A} \leq \gamma(1-\Theta)\frac{\tau\mu}{\tau\mu+\sigma_{\max}\sigma'}\epsilon_G$ then $G(\boldsymbol{\alpha}^{(t)}) \leq \epsilon_G$. Therefore after

$$t \geq \frac{1}{\gamma(1-\Theta)}\frac{\tau\mu+\sigma_{\max}\sigma'}{\tau\mu}\log\left(\frac{1}{\gamma(1-\Theta)}\frac{\tau\mu+\sigma_{\max}\sigma'}{\tau\mu}\frac{n}{\epsilon_G}\right)$$

iterations we have obtained a duality gap less than ϵ_G. □

Theorem' 3. *Consider Algorithm 1 with $\gamma := 1$, using a local solver of quality Θ (see Assumption 1). Let $g_i(\cdot)$ be μ-strongly convex, $\forall i \in [n]$, and assume that the columns of A satisfy $\|\mathbf{x}_i\| \leq 1\ \forall i \in [n]$. Then we have that T iterations are sufficient for suboptimality $\epsilon_{\mathcal{O}_A}$, with*

$$T \geq \tfrac{1}{\gamma(1-\Theta)}\tfrac{\tau\mu+n}{\tau\mu}\log\tfrac{n}{\epsilon_{\mathcal{O}_A}}\,.$$

Furthermore, after T iterations with

$$T \geq \tfrac{1}{\gamma(1-\Theta)}\tfrac{\tau\mu+n}{\tau\mu}\log\left(\tfrac{1}{\gamma(1-\Theta)}\tfrac{\tau\mu+n}{\tau\mu}\tfrac{n}{\epsilon_G}\right)\,,$$

we have the expected duality gap

$$\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(T)}) - (-\mathcal{O}_B(\mathbf{w}(\boldsymbol{\alpha}^{(T)})))] \leq \epsilon_G\,.$$

Proof. Plug in parameters $\gamma := 1$, $\sigma' := \gamma K = K$ to the results of Theorem 11 and note that for balanced datasets we have $\sigma_{\max} \leq \frac{n}{K}$ (see Remark 5). We can further simplify the rate by noting that $\tau = 1$ for the 1-smooth losses (least squares and logistic) given as examples in this work. □

References

G. Andrew and J. Gao. Scalable training of L1-regularized log-linear models. In *International Conference on Machine Learning*, 2007.

Y. Arjevani and O. Shamir. Communication complexity of distributed convex learning and optimization. In *Neural Information Processing Systems*, 2015.

M.-F. Balcan, A. Blum, S. Fine, and Y. Mansour. Distributed learning, communication complexity and privacy. In *Conference on Learning Theory*, 2012.

H. H. Bauschke and P. L. Combettes. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. Springer Science & Business Media, New York, NY, 2011.

D. P. Bersekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, Englewood Cliffs, NJ, 1989.

Y. Bian, X. Li, Y. Liu, and M.-H. Yang. Parallel coordinate descent Newton method for efficient ℓ_1-regularized minimization. *arXiv.org*, 2013.

J. M. Borwein and Q. Zhu. *Techniques of Variational Analysis*. Springer Science & Buisness Media, New York, NY, 2005.

S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, UK, 2004.

S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2010.

J. K. Bradley, A. Kyrola, D. Bickson, and C. Guestrin. Parallel coordinate descent for l1-regularized loss minimization. In *International Conference on Machine Learning*, 2011.

J. Duchi, M. I. Jordan, and B. McMahan. Estimation, optimization, and parallelism when data is sparse. *Neural Information Processing Systems*, 2013.

C. Dünner, S. Forte, M. Takáč, and M. Jaggi. Primal-dual rates and certificates. In *International Conference on Machine Learning*, 2016.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

O. Fercoq and P. Richtárik. Accelerated, parallel, and proximal coordinate descent. *SIAM Journal on Optimization*, 25(4):1997–2023, 2015.

S. Forte. Distributed Optimization for Non-Strongly Convex Regularizers. Master's thesis, ETH Zürich, 2015.

J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.

C. Heinze, B. McWilliams, and N. Meinshausen. DUAL-LOCO: Distributing statistical estimation using random projections. In *International Conference on Artificial Intelligence and Statistics*, 2016.

J.-B. Hiriart-Urruty and C. Lemaréchal. *Fundamentals of convex analysis*. Springer–Verlag, Berlin, 2001.

M. Jaggi, V. Smith, M. Takáč, J. Terhorst, S. Krishnan, T. Hofmann, and M. I. Jordan. Communication-efficient distributed dual coordinate ascent. In *Neural Information Processing Systems*, 2014.

T. Johnson and C. Guestrin. Blitz: A principled meta-algorithm for scaling sparse optimization. In *International Conference on Machine Learning*, 2015.

H. Karimi, J. Nutini, and M. Schmidt. Linear convergence of gradient and proximal-gradient methods under the Polyak-łojasiewicz condition. In *European Conference on Machine Learning*, 2016.

C.-P. Lee and D. Roth. Distributed box-constrained quadratic optimization for dual linear SVM. In *International Conference on Machine Learning*, 2015.

Z. Lu and L. Xiao. On the complexity analysis of randomized block-coordinate descent methods. *arXiv.org*, 2013.

C. Ma, J. Konečný, M. Jaggi, V. Smith, M. Jordan, P. Richtárik, and M. Takáč. Distributed optimization with arbitrary local solvers. *arXiv.org*, 2015a.

C. Ma, V. Smith, M. Jaggi, M. I. Jordan, P. Richtárik, and M. Takáč. Adding vs. averaging in distributed primal-dual optimization. In *International Conference on Machine Learning*, 2015b.

C. Ma, R. Tappenden, and M. Takáč. Linear convergence of the randomized feasible descent method under the weak strong convexity assumption. *arXiv.org*, 2015c.

D. Mahajan, S. S. Keerthi, and S. Sundararajan. A distributed block coordinate descent method for training l_1 regularized linear classifiers. *arXiv.org*, 2014.

G. Mann, R. McDonald, M. Mohri, N. Silberman, and D. D. Walker. Efficient large-scale distributed training of conditional maximum entropy models. *Neural Information Processing Systems*, 2009.

J. Marecek, P. Richtárik, and M. Takáč. *Distributed Block Coordinate Descent for Minimizing Partially Separable Functions*, volume 134 of *Springer Proceedings in Mathematics & Statistics*. Springer International Publishing, Switzerland, 2015.

B. McWilliams, C. Heinze, N. Meinshausen, G. Krummenacher, and H. P. Vanchinathan. LOCO: Distributing ridge regression with random projections. *arXiv.org*, 2014.

X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar. MLlib: Machine learning in apache spark. *Journal of Machine Learning Research*, 17(34):1–7, 2016.

I. Necoara. Linear convergence of first order methods under weak nondegeneracy assumptions for convex programming. *arXiv.org*, 2015.

I. Necoara and V. Nedelcu. Distributed dual gradient methods and error bound conditions. *arXiv.org*, 2014.

Y. Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103(1):127–152, 2005.

F. Niu, B. Recht, C. Ré, and S. J. Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *Neural Information Processing Systems*, 2011.

D. Pechyony, L. Shen, and R. Jones. Solving large scale linear SVM with distributed block minimization. In *International Conference on Information and Knowledge Management*, 2011.

Z. Qu, P. Richtárik, and T. Zhang. Quartz: Randomized dual coordinate ascent with arbitrary sampling. In *Neural Information Processing Systems*, 2015.

Z. Qu, P. Richtárik, M. Takáč, and O. Fercoq. SDNA: Stochastic dual Newton ascent for empirical risk minimization. In *International Conference on Machine Learning*, 2016.

P. Richtárik and M. Takáč. Distributed coordinate descent method for learning with big data. *Journal of Machine Learning Research*, 17:1–25, 2016.

R. T. Rockafellar. *Convex Analysis*. Princeton University Press, Princeton, NJ, 1997.

S. Shalev-Shwartz and A. Tewari. Stochastic methods for l_1-regularized loss minimization. *Journal of Machine Learning Research*, 12:1865–1892, 2011.

S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14:567–599, 2013a.

S. Shalev-Shwartz and T. Zhang. Accelerated mini-batch stochastic dual coordinate ascent. In *Neural Information Processing Systems*, 2013b.

S. Shalev-Shwartz and T. Zhang. Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization. *Mathematical Programming*, Series A:1–41, 2014.

O. Shamir, N. Srebro, and T. Zhang. Communication-efficient distributed optimization using an approximate newton-type method. In *International Conference on Machine Learning*, 2014.

V. Smith, S. Forte, M. I. Jordan, and M. Jaggi. L1-Regularized Distributed Optimization: A Communication-Efficient Primal-Dual Framework. *arXiv.org*, 2015.

M. Takáč, A. Bijral, P. Richtárik, and N. Srebro. Mini-batch primal and dual methods for SVMs. In *International Conference on Machine Learning*, 2013.

M. Takáč, P. Richtárik, and N. Srebro. Distributed mini-batch SDCA. *arXiv.org*, 2015.

R. Tappenden, M. Takáč, and P. Richtárik. On the complexity of parallel coordinate descent. *arXiv.org*, 2015.

I. Trofimov and A. Genkin. Distributed coordinate descent for l1-regularized logistic regression. *arXiv.org*, 2014.

P.-W. Wang and C.-J. Lin. Iteration complexity of feasible descent methods for convex optimization. *Journal of Machine Learning Research*, 15(1):1523–1548, 2014.

S. J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

T. Yang. Trading computation for communication: Distributed stochastic dual coordinate ascent. In *Neural Information Processing Systems*, 2013.

T. Yang, S. Zhu, R. Jin, and Y. Lin. On theoretical analysis of distributed stochastic dual coordinate ascent. *arXiv.org*, Dec. 2013.

I. E.-H. Yen, S.-W. Lin, and S.-D. Lin. A dual augmented block minimization framework for learning with limited memory. In *Neural Information Processing Systems*, 2015.

H.-F. Yu, C.-J. Hsieh, K.-W. Chang, and C.-J. Lin. Large linear classification when data cannot fit in memory. *ACM Transactions on Knowledge Discovery from Data*, 5(4):1–23, 2012.

J. Yu, S. Vishwanathan, S. Günter, and N. N. Schraudolph. A quasi-Newton approach to nonsmooth convex optimization problems in machine learning. *Journal of Machine Learning Research*, 11:1145–1200, 2010.

G.-X. Yuan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. A comparison of optimization methods and software for large-scale l1-regularized linear classification. *Journal of Machine Learning Research*, 11:3183–3234, 2010.

G.-X. Yuan, C.-H. Ho, and C.-J. Lin. An improved GLMNET for L1-regularized logistic regression. *Journal of Machine Learning Research*, 13:1999–2030, 2012.

C. Zhang, H. Lee, and K. G. Shin. Efficient distributed linear classification algorithms via the alternating direction method of multipliers. In *Artificial Intelligence and Statistics Conference*, 2012.

Y. Zhang and X. Lin. Stochastic primal-dual coordinate method for regularized empirical risk minimization. In *International Conference on Machine Learning*, 2015.

Y. Zhang, J. C. Duchi, and M. J. Wainwright. Communication-efficient algorithms for statistical optimization. *Journal of Machine Learning Research*, 14:3321–3363, 2013.

M. A. Zinkevich, M. Weimer, A. J. Smola, and L. Li. Parallelized stochastic gradient descent. *Neural Information Processing Systems*, 2010.