Adaptive Balancing of Gradient and Update Computation Times using Global Geometry and Approximate Subproblems

Sai Praneeth Karimireddy
EPFL

Sebastian U. Stich
EPFL

Martin Jaggi
EPFL

Abstract

First-order optimization methods comprise two important primitives: i) the computation of gradient information and ii) the computation of the update that leads to the next iterate. In practice there is often a wide mismatch between the time required for the two steps, leading to underutilization of resources. In this work, we propose a new framework, Approx Composite Minimization (ACM) that uses *approximate* update steps to ensure balance between the two operations. The accuracy is *adaptively* chosen in an online fashion to take advantage of changing conditions. Our unified analysis for approximate composite minimization generalizes and extends previous work to new settings. Numerical experiments on Lasso regression and SVMs demonstrate the effectiveness of the novel scheme.

1 Introduction

In the last decade, first-order methods and especially stochastic first-order methods have proven to be the superior choice to solve problems of very large size that arise in modern machine learning applications (cf. [27]). The state of the art comprises i) stochastic sub-gradient methods [28], ii) variance reduced stochastic gradient methods [2, 8, 12, 15, 20], iii) coordinate methods (e.g. dual ascent) such as [9, 11, 18, 19, 24, 29, 30].

In a different line of research, distributed algorithm variants have been developed for larger datasets which exceed the capacity of a single machine or device. Here the state-of-the-art algorithms are [16, 31] based on a primal-dual structure of the problem and [10, 13, 22] which work in the primal alone.

All of these methods are iterative algorithms that can be described in the following framework: in each iteration t, the algorithm i) constructs a *model* $m_t(\mathbf{x})\colon \mathbb{R}^d \to \mathbb{R}$ of the objective function (m_t is an approximation of the objective function) and then ii) computes an update step by minimizing this model. The model is typically constructed from gradient information (full gradient, a subset of its coordinates, or a stochastic approximation) and accounting for smoothness assumptions and the structure of the regularizers.

In this work, we are studying the cases when there is a *mismatch* between the time required to compute the gradient information and the time required to compute the update step (i.e. the optimization of the model). We will assume that the cost of the steps involved in computing the gradient information is fixed (i.e. following from the specification of the optimization problem) and outside of our control. However, a parameter that is in our control is the time we spend optimizing the model in each iteration. That is, we will show that it is not required to solve the model *exactly*, but it is enough to compute an *approximate* solution in each iteration. By making the accuracy a *tunable parameter*, we ensure that we do not spend too much time computing the update. Therefore, we optimally balance the two steps.

Whilst the idea of using approximations to speed up the computation is not new (cf. e.g. [7, 26, 33]), the idea of exploiting this for balancing the computational cost is novel. We also extend the framework of [33] to several new settings (Table 1).

Paper outline and contributions. In Section 2 we specify the problem setting and present two motivating examples that exemplify the need of sufficiently complex models—we propose to use models that take *global geometry* (curvature) into account. Section 3 outlines the proposed unified ACM framework and its convergence analysis is given in Section 4. Section 5 extends the framework to the empirical risk minimiza-

Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS) 2018, Lanzarote, Spain. PMLR: Volume 84. Copyright 2018 by the author(s).

Table 1: Summary of the settings where our framework is applicable.

Method	Approximate subproblems	Composite functions	Strongly convex	General convex	Dual version	Parallel block-coordinate updates
PCDM [24]	✗	✓	✓	✓	✗	✓
Inexact [33]	✓	✓	✓	✓	✗	✗
SDNA [21]	✗	✓	✓	✗	✓	✗
PSNM [17]	✗	✗	✓	✗	✗	✓
ACM	§3, (Def 3)	✓	§4 (Thm 1)	§4 (Thm 2)	§5 (Thms 3,4)	§6 (Lem 4 with Thms 1,2,3, and 4)

tion setting where we prove primal-dual guarantees. In Section 6 we demonstrate the generality of the ACM framework by exemplary considering specific solvers for the subproblems and we show how the convergence guarantees of these algorithms easily follow from our analysis. ACM specifically supports *changing accuracies* for different iterations. In Section 7 we capitalize this property by designing mechanisms that *adaptively control* the subproblem accuracy, always balancing the computation times. In Section 8 we present experimental results that show the numerical advantage our schemes and conclude in Section 9. All missing proofs and figures can be found in the appendix.

2 Setup and Motivation

We address optimization problems of the form

$$\min_{\mathbf{x}\in\mathbb{R}^d} \left[F(\mathbf{x}) \overset{\text{def}}{=} f(\mathbf{x}) + g(\mathbf{x}) \right] , \tag{1}$$

where $f\colon \mathbb{R}^d \to \mathbb{R}$ is a smooth convex function and $g\colon \mathbb{R}^d \to \mathbb{R}\cup\{+\infty\}$ is an arbitrary extended-valued convex function. $\mathbf{x}^\star \in \mathbb{R}^d$ denotes an optimal solution, and for $\epsilon > 0$, a point $\mathbf{y} \in \mathbb{R}^d$ with $F(\mathbf{y}) - F(\mathbf{x}^\star) \leq \epsilon$ is an ϵ-approximate solution.

2.1 Imbalance in the computations

In this section we illustrate the main problem we tackle in this paper—inefficiencies caused by mismatches in the computation times of different steps in the optimization algorithm. First-order methods typically optimize an over-approximation $U\colon \mathbb{R}^n \to \mathbb{R}$ of the objective (1), with

$$U(\Delta\mathbf{x}) \overset{\text{def}}{=} u(\mathbf{x}+\Delta\mathbf{x}) + g(\mathbf{x}+\Delta\mathbf{x}) \geq F(\mathbf{x}+\Delta\mathbf{x}) \tag{2}$$

where $u(\mathbf{x}+\Delta\mathbf{x}) \overset{\text{def}}{=} f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \Delta\mathbf{x}\rangle + \frac{L}{2}\left\|\Delta\mathbf{x}\right\|_2^2$ and L is the smoothness parameter of f.

We will now discuss two key examples of a significant imbalance in the gradient computation vs. the optimization of the model (2).

Case A: Gradient computation is slower. Consider the coordinate descent algorithm on the L1-regularized logistic regression problem. Given an $n \times d$ data matrix A, let A_i refer to its i-th column and denote the residual as $\mathbf{v} \overset{\text{def}}{=} A\mathbf{x}$. Then the logistic loss can be written as $F(\mathbf{x}) = \sum_{i=1}^n \log(1+e^{v_i}) + \left\|\mathbf{x}\right\|_1$ and the i-th coordinate of the gradient $\nabla f(\mathbf{x})$ is $\nabla_i f(\mathbf{x}) = \langle A_i, \nabla l_{log}(\mathbf{v})\rangle$, where

$$\nabla l_{log}(\mathbf{v}) = \left(\tfrac{1}{1+e^{-v_1}}, \ldots, \tfrac{1}{1+e^{-v_n}}\right) .$$

Assuming that $\mathbf{v}$ is already stored in active memory, computing the i-th gradient requires i) fetching A_i from memory, ii) performing n exponentiations and divisions, iii) one dot product, and iv) possibly communicating back the update steps to facilitate the next gradient computation. The coordinate update step on the other hand is $\left[\mathbf{x}_i - \frac{1}{L}\nabla_i f(\mathbf{x})\right]_{\lambda/L}$ where $[c]_\gamma$ is the *shrinkage* operator. If $\Delta\mathbf{x}_i$ is the update made to coordinate i, $\mathbf{v}$ can be updated as $\mathbf{v} - \Delta x_i A_i$. In total for minimizing the model we only make i) one coordinate update and ii) one vector addition, in contrast with the more involved gradient coordinate computation. Thus, cache misses, complex operations, as well as communication overhead all lead to the gradient computation being much slower than the update step. Increasing the number of coordinates being updated i.e. block-coordinate updates can alleviate overhead to some extent due to cache misses, though it is ineffective against the other sources of delay.

Case B: Model minimization is slower. When the data is not extremely high dimensional and the loss function f is simple, it is quite fast to compute the gradient. For example, a very common task in image processing is to reconstruct the original image given a corrupted linear measurement y. For such applications, the commonly used loss function is $F(\mathbf{x}) = \frac{1}{2}\left\|A\mathbf{x} - \mathbf{b}\right\|_2^2 + \lambda\left\|\mathbf{x}\right\|_{TV}$ [5]. Even though f is just a quadratic, there is no closed form solution for minimizing $U(\Delta\mathbf{x})$ because of $g(\mathbf{x}) = \left\|\mathbf{x}\right\|_{TV}$. In this case a lot of time is spent in minimizing the model $U(\Delta\mathbf{x})$ whereas the gradient computation is fast.

2.2 A solution: curvature models

When the model is too "simple", i.e. computationally cheap to minimize compared to the expensive gradient information, then traditional methods will waste significant resources for just the gradient information. Hence, the model should be sufficiently sophisticated in order to be able to *extract the maximum progress* using a single gradient computation. In other words, we would prefer that exact minimization of the model takes more time than computing the gradient information. In this case, we can minimize the model approximately to a *tunable degree of accuracy* to ensure that the two essential parts are balanced in terms of computational cost. The model, of course, cannot be arbitrarily hard to optimize either. In particular, it should be at least possible to make non-zero progress on minimizing the model in time less than it takes to compute the gradient information.

One way to create good models is to capture some second-order information of the objective, i.e. the global geometry (curvature) using a $d \times d$ matrix M as for instance in [21, 33]. This means that we replace u in (2) with

$$u_M(\mathbf{x} + \Delta\mathbf{x}) \overset{\text{def}}{=} f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \Delta\mathbf{x} \rangle + \tfrac{1}{2}\Delta\mathbf{x}^\top M \Delta\mathbf{x}\,.$$

For example, in the L1-regularized logistic regression discussed before, we can use $M \overset{\text{def}}{=} \frac{1}{4}A^\top A$.

Related literature. There are a number of algorithms which incorporate curvature information. These include methods based on using the diagonal of the Hessian information to compute the sample probabilities as well as the step size [3, 18, 19, 24]. Recently, a more direct approach to incorporating the Hessian information through *preconditioning* has become more popular [17, 32, 33]. However, such preconditioning means that the computational effort involved in solving the subproblem is significant. To overcome this, one resorts to approximate solutions [32, 33]. The idea of preconditioning has also been extended to dual ascent algorithms as in [21]. However, there exact solutions to the subproblems were required.

2.3 Notation and Definitions

For a fixed positive semi-definite matrix M ($M \succcurlyeq 0$), we use $\|\mathbf{x}\|_M^2 \overset{\text{def}}{=} \mathbf{x}^\top M\mathbf{x}$. With this semi-norm, the regularity assumptions on f can be written as follows.

Definition 1 (M-smoothness). *A differentiable function $h\colon \mathbb{R}^d \to \mathbb{R}$ is M-smooth with respect to a fixed matrix $M \succcurlyeq 0$ if for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$*

$$h(\mathbf{y}) \le h(\mathbf{x}) + \langle \nabla h(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \frac{1}{2}\|\mathbf{y} - \mathbf{x}\|_M^2\,.$$

Definition 2 (λ-strong convexity). *A differentiable function $h\colon \mathbb{R}^d \to \mathbb{R}$ is λ-strongly convex w.r.t. $M \succcurlyeq 0$ and $\lambda > 0$ if for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$*

$$h(\mathbf{y}) \ge h(\mathbf{x}) + \langle \nabla h(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \frac{\lambda}{2}\|\mathbf{y} - \mathbf{x}\|_M^2\,.$$

Note that an M-smooth function can only be λ-strongly convex w.r.t. M for $\lambda \in (0, 1]$.

3 Approx Composite Minimization

In this section we describe the general method as outlined in Algorithm 1. At each time step t, an approximation $m_t(\Delta\mathbf{x}; M) \ge F(\mathbf{x}_t + \Delta\mathbf{x}) - F(\mathbf{x}_t)$ is constructed as follows:

$$\begin{aligned} m_t(\Delta\mathbf{x}; \mathbf{x}_t, M) \overset{\text{def}}{=} \langle \nabla f(\mathbf{x}_t), \Delta\mathbf{x} \rangle + \tfrac{1}{2}\|\Delta\mathbf{x}\|_M^2 \\ +g(\mathbf{x}_t + \Delta\mathbf{x}) - g(\mathbf{x}_t)\,. \end{aligned} \tag{3}$$

When obvious from context, we drop M and $\mathbf{x}_t$ and simply refer it as $m_t(\Delta\mathbf{x})$. Let $m_t(\Delta\mathbf{x}_t^\star) \overset{\text{def}}{=} m_t^\star$ be the minimum of the subproblem obtained at $\Delta\mathbf{x}_t^\star$.

Definition 3. *We denote by $\Theta_t \in (0, 1]$ the relative accuracy to which the subproblem* (3) *is solved at step t, i.e., we compute $\Delta\mathbf{x}_t$ such that*

$$m_t(\Delta\mathbf{x}_t) - m_t^\star \le (1 - \Theta_t)(m_t(\mathbf{0}) - m_t^\star)\,.$$

Here $\Theta_t = 1$ means that we solve the problem exactly. We then update the iterate as $\mathbf{x}_{t+1} := \mathbf{x}_t + \Delta\mathbf{x}_t$. Note that Θ_t can adaptively change with each step.

Algorithm 1: Approx Composite Minimization (ACM)

```
Input: $M$, $\mathbf{x}_0$
for $t = 0, \dots$ do
    Let $m_t(\Delta\mathbf{x}) \overset{\text{def}}{=}$
      $\langle \nabla f(\mathbf{x}_t), \Delta\mathbf{x} \rangle + \frac{1}{2}\|\Delta\mathbf{x}\|_M^2 + g(\mathbf{x}_t + \Delta\mathbf{x}) - g(\mathbf{x}_t)$
    Minimize subproblem: Find $\Delta\mathbf{x}_t$ such that
      $m_t(\Delta\mathbf{x}_t) - m_t^\star \le (1 - \Theta_t)(m_t(\mathbf{0}) - m_t^\star)$
    $\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t + \Delta\mathbf{x}_t$
end
```

4 Covergence Analysis

We will examine the cases when F is strongly convex (Def. 2 is satisfied for $\lambda > 0$) and the general convex case separately (when $\lambda = 0$) and generalize the results in [33] to account for varying approximation factors.

Theorem 1. *Given that f is M-smooth* (1), *and that f and g are λ_f and λ_g strongly convex respectively* (2) *for $\lambda_f + \lambda_g > 0$, then running Algorithm 1 gives linear convergence, i.e. $F(\mathbf{x}_T) - F(\mathbf{x}^\star) \le \epsilon$ for*

$$T \ge \frac{1 + \lambda_g}{(\lambda_f + \lambda_g)\tilde{\Theta}_T} \log\left(\frac{F(\mathbf{x}_0) - F(\mathbf{x}^\star)}{\epsilon}\right),$$

where $\tilde{\Theta}_T = \frac{1}{T}\sum_{t=1}^T \Theta_t$ is the average accuracy.

Remark 1. *The convergence rate from Theorem 1 shows that the rate critically depends on not just λ_f but also λ_g. Thus M must be chosen to not just approximate f but also g, i.e. M must be chosen to approximate the curvature of F as closely as possible. This is an important observation since f and g typically have very different curvatures.*

Our proof hinges on this very useful lemma proved in the appendix.

Lemma 2. *Assuming g is λ_g-strongly convex, for any vector $\mathbf{v}$ and $\lambda \in (0, 1]$, then*

$$\min_{\Delta\mathbf{x}} \; m(\Delta\mathbf{x}; \mathbf{v}, M) \leq \frac{\lambda + \lambda_g}{1 + \lambda_g} \cdot \min_{\Delta\mathbf{x}} \; m(\Delta\mathbf{x}; \mathbf{v}, \lambda M)\,.$$

In the general convex case, we obtain a convergence rate of $O(1/T)$ which matches the rate of standard proximal-gradient algorithms. We again leave the proof of the theorem for the appendix.

Theorem 2. *Given that f is M-smooth (1), running Algorithm 1 such that at every step $m_t(\Delta\mathbf{x}_t) \leq 0$ and $\Theta_t > 0$ ensures*

$$F(\mathbf{x}_T) - F(\mathbf{x}^\star) \leq \epsilon \quad \textit{for} \quad T > \frac{2D}{\tilde{\Theta}_T \epsilon}\,,$$

where D is at most the diameter of the initial level set of F and $\tilde{\Theta}_T$ is the average accuracy:

$$D \overset{\text{def}}{=} \max_{\mathbf{y} | F(\mathbf{y}) \leq F(\mathbf{x}_0)} \|\mathbf{y} - \mathbf{x}^\star\|_M^2 \;\; \textit{and} \;\; \tilde{\Theta}_T \overset{\text{def}}{=} \frac{1}{T} \sum_{t=0}^{T} \Theta_t\,.$$

5 Primal-Dual Guarantees

Suppose our objective function comes with the following additional structure, ubiquitous in machine learning and signal processing models:

$$\min_{\mathbf{w}} \; \sum_{i=1}^{n} l_i(\mathbf{w}^\top A_i) + \psi(\mathbf{w})\,.$$

Let A be a $d \times n$ data matrix. Here l_i is the loss defined for each data-point $A_i \in \mathbb{R}^d$ and ψ is a regularizer. We can define the dual objective in terms of the *Fenchel conjugate* of $\{l_i\}_i$ and ψ, $\{l_i^*\}_i$ and ψ^* respectively [9]:

$$\max_{\boldsymbol{\alpha}} \; \sum_{i=1}^{n} -l_i^*(-\alpha_i) - \psi^*(A\boldsymbol{\alpha})\,.$$

These can be more compactly written as

$$\begin{aligned} \mathcal{O}_A(\boldsymbol{\alpha}) &\overset{\text{def}}{=} f(A\boldsymbol{\alpha}) + g(\boldsymbol{\alpha})\,, \\ \mathcal{O}_B(\mathbf{w}) &\overset{\text{def}}{=} f^*(\mathbf{w}) + g^*(-A^\top\mathbf{w})\,. \end{aligned} \tag{4}$$

Our objective to minimize the *duality gap* defined as $\mathcal{O}_B(\mathbf{w}) + \mathcal{O}_A(\boldsymbol{\alpha})$. We assume that $f(A\boldsymbol{\alpha})$ is M-smooth. Depending on the problem, it may be more convenient and efficient either to cast the problem as $\mathcal{O}_A(\boldsymbol{\alpha})$, or map it to $\mathcal{O}_B(\mathbf{w})$ and solve the dual. We can do this since primal-dual algorithms minimize both $\mathcal{O}_B(\mathbf{w})$ and $\mathcal{O}_A(\boldsymbol{\alpha})$ simultaneously. For more details about the setting and applications, we refer to the discussion in [9].

Algorithm 2: Approximate Dual Ascent (ADA)

```
Input: $M$, $\mathbf{x}_0$
Initialize: $\boldsymbol{\alpha}_0 \leftarrow \mathbf{0} \in \mathbb{R}^n$, $\mathbf{v}_0 \leftarrow \mathbf{0} \in \mathbb{R}^d$
for $t = 1, \dots$ do
    Let $m_t(\Delta\boldsymbol{\alpha}) \overset{\text{def}}{=} \langle \nabla f(\mathbf{v}_t), A\Delta\boldsymbol{\alpha} \rangle +$
      $\frac{1}{2}\|\Delta\boldsymbol{\alpha}\|_M^2 + g(\boldsymbol{\alpha}_t + \Delta\boldsymbol{\alpha}) - g(\boldsymbol{\alpha}_t)$
    Minimize subproblem: Find $\Delta\boldsymbol{\alpha}_t$ such that
        $m_t(\Delta\boldsymbol{\alpha}_t) - m_t^\star \leq (1 - \Theta_t)(m_t(\mathbf{0}) - m_t^\star)$
    Update steps:
        $\boldsymbol{\alpha}_{t+1} \leftarrow \boldsymbol{\alpha}_t + \Delta\boldsymbol{\alpha}_t$
        $\mathbf{v}_{t+1} \leftarrow \mathbf{v}_t + A\Delta\boldsymbol{\alpha}_t$
end
```

Just as in the Section 4, we can state two theorems—one for the case when the objective is strongly convex and one for the general convex case. The proofs of these theorems follow along the lines of [16, 31] with some simplifications.

Theorem 3. *For an objective of the form (4), let us assume that that $f(A\boldsymbol{\alpha})$ is M-smooth (1), and that $f(A\boldsymbol{\alpha})$ and $g(\boldsymbol{\alpha})$ are λ_f and λ_g strongly convex respectively (2). Then running Algorithm 2 gives linear convergence i.e. for $\lambda = \frac{\lambda_g + \lambda_f}{1 + \lambda_g}$ and $\tilde{\Theta}_T = \frac{1}{T}\sum_{t=0}^{T} \Theta_t$,*

$$\mathcal{O}_B(\nabla f(\mathbf{v}_t)) + \mathcal{O}_A(\boldsymbol{\alpha}_t) \leq \epsilon \quad \textit{for}$$

$$T \geq \frac{1}{\lambda \tilde{\Theta}_T} \log\left(\frac{(1 + \lambda_g)(\mathcal{O}_A(\mathbf{0}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star))}{\lambda_g \Theta_T \, \epsilon} \right)\,.$$

Theorem 4. *Given that f is M-smooth (1), running Algorithm 2 ensures convergence at a rate of $O(\frac{1}{T})$ i.e. $\mathcal{O}_B(\mathbf{w}(\bar{\mathbf{v}}_t)) + \mathcal{O}_A(\bar{\boldsymbol{\alpha}}_t) \leq \epsilon$ for*

$$T \geq t_0 + \max\left[\frac{4D}{\epsilon}, \frac{1}{\hat{\Theta}_t} \right], \quad t_0 \geq \frac{4D}{\tilde{\Theta}_{t_0}\epsilon}\,,$$

where $\bar{\boldsymbol{\alpha}}_t \overset{\text{def}}{=} \frac{1}{t - t_0}\sum_{i=t_0+1}^{t} \boldsymbol{\alpha}_i$, $\bar{\mathbf{v}}_t \overset{\text{def}}{=} \frac{1}{t - t_0}\sum_{i=t_0+1}^{t} \mathbf{v}_i$, and $\mathbf{w}(\bar{\mathbf{v}}_t) \overset{\text{def}}{=} \nabla f(\bar{\mathbf{v}}_t)$. D is the diameter of the level set of $\mathcal{O}_A(\boldsymbol{\alpha})$, $Q \overset{\text{def}}{=} \{\boldsymbol{\alpha} \mid \mathcal{O}_A(\boldsymbol{\alpha}) \leq \mathcal{O}_A(\mathbf{0})\}$. $D \overset{\text{def}}{=} \max_{\mathbf{a}, \mathbf{b} \in Q} \|\mathbf{a} - \mathbf{b}\|_M^2$. Further $\tilde{\Theta}_t \overset{\text{def}}{=} \frac{1}{t}\sum_{t'=0}^{t} \Theta_{t'}$ and $\hat{\Theta}_t \overset{\text{def}}{=} \min_{t' \in [t]} \Theta_{t'}$.

Remark 3. *We obtain sharper bounds than in [16, 31] in both settings. In particular, we show that strong convexity constant λ_f of f is also useful for faster convergence. The rate in [31] is equivalent to setting $\lambda_f = 0$. Similarly, in the general convex setting, our rate is again simpler and has better constants.*

6 Extension to Coordinate Updates

Thus far in our discussion we have largely restricted ourselves to the case where the model was constructed using the full gradient. However, when the function $g(\mathbf{x}) = \sum_{j=1}^{d} g_j(x_j)$ is coordinate-wise separable, a popular strategy is to update just a single coordinate in each iteration. This approach leads to state of the art algorithms for several problems [11, 18, 19, 24, 29, 30]. Coordinate methods have been widely successful not just due to their faster convergence, but also because they are less resource intensive and provide faster update times [35]. However, the imbalance between the time for gradient information computation and the time for update computation is often exacerbated in coordinate descent methods. This is because the time to compute one directional derivative of the gradient scales in general linearly with the dimension, whereas computing the one-dimensional update only requires constant time. Updating multiple coordinates simultaneously might reduce some of the overhead—such as the delay due to memory accesses or communication costs—but cannot completely settle this imbalance.

In this section we see how we can extend our ideas from Sections 3 and 5 seamlessly to this scenario and achieve a better balance between the gradient and update computation times. It is clear, that one single step of coordinate descent can be seen as an approximate solution to the full dimensional problem (3). Thus, the convergence results can be directly recovered from the theorems derived in the previous sections. However, there is a small caveat: the model (3) depends on the full gradient, whereas in coordinate descent methods it is sufficient to compute just the directional derivatives of gradient along the descent directions. To capture this more precisely, we will now refine our notation.

Typically, the number of coordinates being updated at each iteration is fixed by external factors—for e.g. when the data is distributed by columns amongst multiple machines, the coordinates being updated are constrained by data available on each machine, or in the single machine case it is constrained by the number of columns which fit in a cache block. For this reason we assume that the number of coordinates being updated is fixed. Further, we will not only support sequential updates, but also updates to multiple blocks of coordinates in *parallel*. This setting is useful when i) the data is distributed over multiple machines which can update a different block of coordinates in parallel, or ii) when there are multiple threads (in a multiprocessor) each with a separate cache. Examples of this setting can be found in [17, 31].

Suppose at iteration t, we have κ machines such that machine $k \in [\kappa]$ can compute the coordinates $\pi_k \subseteq [d]$ of the gradient i.e. it can compute

$$\nabla_{\pi_k} f(\mathbf{x}) \overset{\mathsf{def}}{=} \sum_{j \in \pi_k} \nabla_j f(\mathbf{x}) \mathbf{e}_j \, .$$

Here the sets π_k need not necessarily need to form a complete partition of $[d]$. Note that this notation also captures the case of sequential updates on one single machine: in this case π_1 just denotes the set of coordinates that are updated in this iteration. We further assume that machine k has access to the principal submatrix of M corresponding to the coordinates π_k,

$$M_{\pi_k} \overset{\mathsf{def}}{=} \sum_{i,j \in \pi_k} M_{i,j} \mathbf{e}_i \mathbf{e}_j^\top \, .$$

Then we can form a set of κ subproblems such that each machine $k \in [\kappa]$ can solve $m_t^\sigma(\Delta\mathbf{x}; \pi_k)$ defined as

$$m_t^\sigma(\Delta\mathbf{x}; \pi_k) \overset{\mathsf{def}}{=} \langle \nabla_{\pi_k} f(\mathbf{x}), \Delta\mathbf{x} \rangle + \frac{\sigma}{2} \|\Delta\mathbf{x}\|_{M_{\pi_k}}^2 + \sum_{j \in \pi_k} g_j(x_j + \Delta x_j) - g_j(x_j) \, .$$

Here $\sigma \in [1, \kappa]$ is a parameter which measures the *separability* of the matrix M. Crucially, the problems $m_t^\sigma(\Delta\mathbf{x}; \pi_k)$ for $k \in \{1, \dots, \kappa\}$ can be solved in parallel. The solutions from these subproblems can then be combined as

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \sum_{k=1}^{\kappa} [\Delta\mathbf{x}]_{\pi_k} \, ,$$

where $[\Delta\mathbf{x}]_{\pi_k}$ is a Θ_t-approximate solution to $m_t^\sigma(\Delta\mathbf{x}; \pi_k)$ as in Definition 3. In this manner we can use global geometry (curvature) information encoded in M to better utilize the gradient information we computed even in the block coordinate setting.

We can relate the Θ_t progress on the coordinate models $m_t^\sigma(\Delta\mathbf{x}; \pi_k)$ to the progress on the global model $m_t(\Delta\mathbf{x})$. Recall that $m_t(\Delta\mathbf{x})$ was our shorthand for $m_t(\Delta\mathbf{x}; \mathbf{x}_t, M)$ defined in (3). A more formal study with additional technical details and definitions is relegated to Section A.1.

Lemma 4. *Suppose that $[\Delta\mathbf{x}]_{\pi_k}$ is an Θ_t-approximate solution to $m_t^\sigma(\Delta\mathbf{x}; \pi_k)$ for $k \in [\kappa]$ and $\Delta\mathbf{x}_t = \sum_{k=1}^{\kappa} [\Delta\mathbf{x}]_{\pi_k}$. Then under the conditions specified in Lemma 8 in Section A.1,*

$$\mathbb{E}[m_t(\Delta\mathbf{x}_t) - m_t^\star] \leq \left(1 - \frac{\kappa\Theta_t}{m\sigma\nu}\right)(m_t(\mathbf{0}) - m_t^\star) \, ,$$

where the expectation is over the random selection of the sets $\{\pi_1, \dots, \pi_\kappa\}$, and $m \in [1, d]$ and $\nu \geq 1$ are parameters depending on the sampling (see Definitions 4 and 6 in Section A.1).

Thus, using just the block-coordinate models, we can make progress on the global problem $m_t(\Delta\mathbf{x}_t; \mathbf{x}_t, M)$. We can then combine Lemma 4 with with Theorems 1, 2, 3, and 4 to derive the corresponding parallel-block coordinate versions of the algorithm.

7 Adaptive Accuracy

Requiring only approximate solutions means that we can use iterative methods for the sub-problems which can be much faster and cheaper than exact solvers [33]. More importantly, using iterative algorithms enables us to adaptively control the number of iterations run on each of the subproblem. This allows us to tune the time spent on computing the update, and thus better balance against the time spent on computing gradient information. Intuitively, the more expensive is the cost of computing the gradient information, the more time we should spend trying to utilize it better. The time for the former can however be orders of magnitude different in different real world systems [16]. Moreover, as we will argue later in the section, the 'right' amount of time to be spent computing the update under identical system conditions also varies during the duration of the algorithm. Due to these reasons we propose to use simple strategies to *automatically* and *adaptively* choose the number of iterations, and hence tune the time spent computing the update.

Let us assume that each iteration of the solver takes one unit of time. From the proofs of Theorems 1,2,3, and 4, we know that the progress we make at each step is

$$F(\mathbf{x}_{t+1}) \leq F(\mathbf{x}_t) + \Theta_t m_t^\star \,.$$

Suppose we spent r iterations i.e. r units of times on this subproblem. Since we want to design strategies which control r, let us parameterize the above equation in terms of r. Since $m_t^\star \leq 0$, we denote the progress made as

$$\Theta_t m_t^\star \overset{\mathsf{def}}{=} -p_t(r) \,.$$

Further let c_t be the ratio between the time spent in computing the gradient information and the time required to perform one iteration of the sub-solver. This includes all fixed costs such as communication time, etc. Our objective should to be to pick an r which makes the maximum progress possible per unit of time spent:

$$r_t^\star \overset{\mathsf{def}}{=} \arg\max_r \frac{p_t(r)}{r + c_t} \,. \tag{5}$$

7.1 Fixed strategies for picking r_t

It is reasonable to assume that i) $p_t(r)$ is increasing meaning that running the sub-solver for more iterations leads to improved accuracy, and ii) $p_t(r)$ is *sub-linear* and concave function meaning that doubling the number of iterations results in at most double the accuracy (refer Fig 1). Based on this knowledge, two fixed rules can be formulated to pick a fixed r_t.

One step. The simplest strategy is to just perform one inner step per round. If $c_t = 0$, $\frac{p_t(r)}{r}$ is a decreasing function (Fig 1, left). Thus when the gradient computation cost is low, the one step strategy is optimal. However, this is no more true when c_t is significant (Fig 1, right).

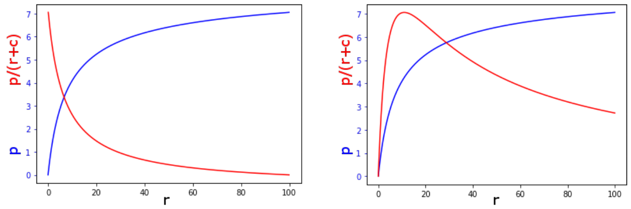


Figure 1: The progress $p(r)$ (in blue) is bounded, increasing, and concave; attaining its maxima at $r = \infty$. Progress per unit time ($\frac{p(r)}{r+c}$) shown in red is i) a decreasing function if $c = 0$ (left) or ii) increases first and then decreases if $c > 0$ (right).

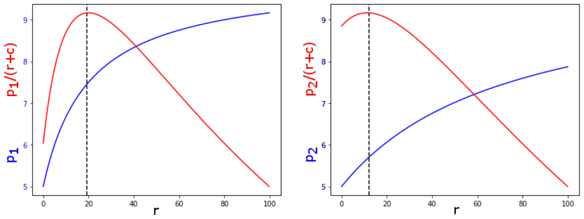


Figure 2: The slope of $p_2(r)$ (in blue, right) is always lesser than that of $p_1(r)$ (in blue, left) and the optimum number or rounds is lesser for p_2 than for p_1.

Comparable steps. 'Best practice' dictates that we should spend roughly equal time performing the update as we take for gradient computation [16]. So this strategy tries to pick $r_t = c_t$. While being a good guiding principle, this strategy fails to take advantage of the trade-offs present in more realistic regimes. For example it ignores the fact that if the local solver is bad or the particular subproblem is especially hard, there might not be much to gain running for c_t steps and it might make sense to terminate early.

7.2 Adaptive strategies for picking r_t

Consider two functions p_1 and p_2 with the corresponding optimum number of iterations being $r_1^\star$ and $r_2^\star$ as per (5). If the slope of $p_2(r)$ is always smaller than that of $p_1(r)$, then the optimum number of rounds typically (but not always) reduces and $r_2^\star < r_1^\star$ (refer Fig 2). Since $p(r)$ denotes the progress made on the subproblem, this means that the optimal number of rounds depends on the *ease of minimizing the subproblem*. During the course of our algorithm, the nature of $p_t(r)$ may change substantially and hence the optimum number of rounds also changes. Thus it is imperative to use strategies which are *adaptive to the hardness of* $p_t(r)$.

Optimal. If we could access the entire function $p_t(r)$, and c_t *beforehand*, it is possible to pick the optimal $r_t^\star$ at every step. However, knowing $p_t(r)$ for all r requires expensive computations. Using an upper bound on $p_t(r)$ obtained from convergence rates is also impractical since it requires knowledge of parameters which are often inaccessible.

Gradient based strategies. At each step, we can assume access to the total time spent $(r_t + c_t)$, $p_t(r_t)$, and $p_t'(r_t)$ as feedback. This is because $p_t(r_t)$ and $p_t'(r_t)$ are just the value of the subproblem $-m_t(\Delta \mathbf{x}_t)$ and the rate at which it decreases, measured at the end. Using this feedback, we can compute the gradient g_t of $\frac{p_t(r)}{r+c_t}$ at r_t:

$$g_t = \frac{p_t'(r_t)(r_t + c_t) - p_t(r_t)}{(r_t + c_t)^2} \,.$$

We can use the gradient g_t as an indicator about the direction we need to change r_t.

1. Additive change. If $g_t > 0$, increase $r_{t+1} \leftarrow r_t + 1$ else $r_{t+1} \leftarrow r_t - 1$.
2. Multiplicative-additive change. If $g_t > 0$, increase $r_{t+1} \leftarrow 2r_t$ else $r_{t+1} \leftarrow r_t - 1$. This is inspired by the TCP protocols used to determine the rate at which to send packets to make maximum use of network capacity [6].
3. Gradient change. Increment with gradient $r_{t+1} \leftarrow r_t + g_t$.

We only focus on very simple strategies to compute r here—many more sophisticated methods based on hyper-parameter optimization or bandit optimization are possible [14].

8 Empirical Evaluation

In this section we present empirical results for our framework, demonstrating the insights gained. Recall from Section 7 that c_t is the ratio between the time taken for computing the gradient and the time taken by the local solver to perform one iteration. In practice, the value of c_t can vary by orders of magnitudes [31]. We artificially set c_t to take values from 0.5 to 1024 in powers of 2 and observe its effect on the different algorithms. This way we can simulate a wide range of real world conditions in our experiments.

8.1 Experimental Setting

We focus on two important tasks in our experiments—Lasso and SVM. We vary the value of c_t and measure the total time it takes to reach a predetermined sub-optimality value. The total time is the sum of the measured update time as well as the simulated gradient computation time. The gradient computation time is calculated as $\frac{t_u c_t}{r_t}$ where t_u is the measured time for update and r_t is the number of iterations performed by the solver.

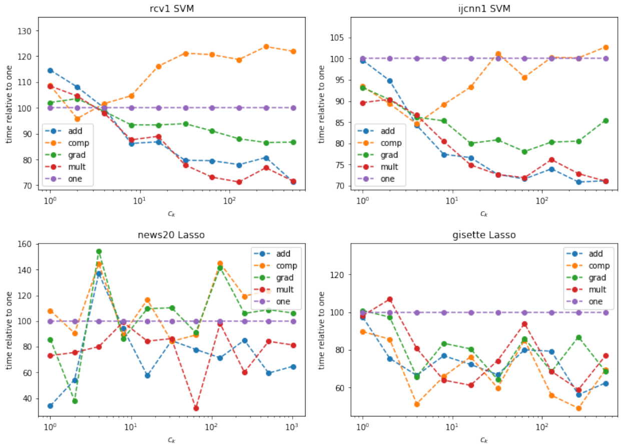


Figure 3: Time taken relative to `one` in percentage, to reach sub optimality (duality gap for SVM) of 1e−4. Here c_t is the ratio between time for gradient and 1 step of subsolver. Adaptive rules nearly always outperform fixed rules `one` and `comp`.

Lasso. For Lasso, our objective function is $F(\mathbf{x}) = f(\mathbf{x}) + g(\mathbf{x})$ where $g(\mathbf{x}) = \lambda \|\mathbf{x}\|_1$ and $f(\mathbf{x}) = \frac{1}{2n}\|A\mathbf{x} - \mathbf{b}\|_2^2$. Here λ is an regularization parameter chosen to be $\frac{1}{n}$ as is standard [25]. We run this on the `gisette` and `news20` datasets.[1] For each value of c_t, we measure the total time it takes for each algorithm to reach an sub-optimality of 1e−4. The minimum value is calculated by letting the algorithm run for 1k effective passes over the data.

SVM. Here we require primal-dual convergence for the hinge loss $\mathcal{O}_B(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^n l(\mathbf{w}^\top A_i, y_i) + \frac{\lambda}{2}\|\mathbf{w}\|^2$, c.f. [9]. The regularization parameter λ is again chosen to be $\frac{1}{n}$. We run this on `rcv1` and `ijcnn1` datasets till we reach a duality gap of 1e−4.

Subproblem solver. To simplify the comparison, we use random coordinate descent as a solver for all the subproblems. Further we create smaller subproblems of dimension 100 using the algorithm from Section 6 (see also Sec. A.1). Hence, one iteration of the subproblem solver consists of 100 random coordinate updates. To estimate the gradient $p_t'(r_t)$, we use the average progress made in the last 10 steps of the solver.

Strategies for r_t. To demonstrate our discussion form the previous section, we compare the different strategies which perform $100r_t$ CD iterations at each step: i) `one` where $r_t = 1$, ii) `comp` where $r_t = c_t$, iii) `add` where r_t is computed using the additive change rule, iv) `mult` where r_t is computed using the multiplicative-additive rule, and v) `grad` where r_t is computed using the gradient rule.

[1] All datasets are available from `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

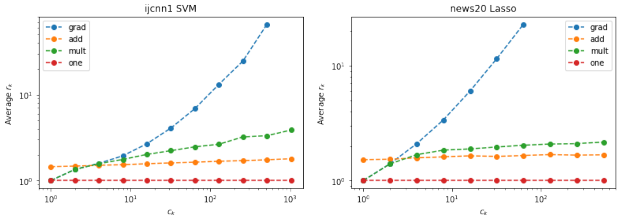


Figure 4: Average value of r chosen by different strategies. It remains relatively constant as c_t increases across problems and strategies.

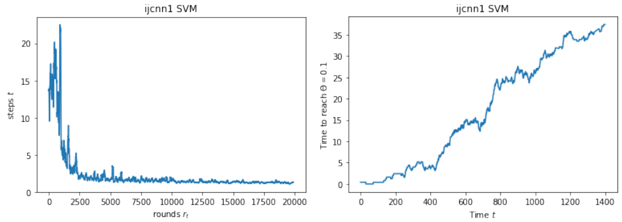


Figure 5: (Left) Time varying r_t for `mult` with $c = 64$ for `ijcnn1`. ((Right) The number of rounds required to solve the subproblem to an accuracy of 0.1. The hardness of the subproblems increase with t and in response, r_t chosen by `mult` decreases. A moving average with window of size 100 is used to aid visualization.

8.2 Discussion

As seen in Fig 3, the adaptive rules based on the gradient consistently scale better for increasing values for the ratio c_t. In almost all the cases the simple `mult` rule performs better than both the standard `one` rule as well the `comp` rule.

For small values of c_t (≤ 2) in the `rcv1` SVM experiment as well as in the `gisette` Lasso example using the `one` rule and setting $r_t = 1$ is effective. However, as c_t increases, the `one` rule becomes much less competitive. This is because for small values of c_t, the function $\frac{p_t(r)}{r+c_t}$ attains its maxima close to 1.

We also plot the average values of r_t over the period of optimization for the `ijcnn1` and the `news20` cases (cf. Fig 4). The plots from the other two datasets were similar. The average values of r_t are pretty identical in both the cases. It also remains close to 1–2, and is mostly independent of c_t for the `mult` and the `add` rule. Thus the improvements we see in Fig 3 for these rules must have been because of the use of *adaptivity*.

To understand how adaptivity affects our algorithms, we look at how r_t varies over time for `ijcnn1` with the `mult` rule with $c_t = 64$ in (Fig 5, left). The strikingly clear downward trend in r_t can be explained via our discussion Section 7.2 about changing hardness of subproblems. The sub-problems in `ijcnn1` are increasing in their difficulty (Fig 5, right), pushing for lower values of r_t.

For `news20` dataset, the value of r_t remain relatively constant (Appendix Section C). This explains why fixed rules such as `comp` perform comparably to the adaptive ones in `news20`. The variation of r_t in `rcv1` is similar to that of `ijcnn1` and so they have similar results. The growing gap between adaptive rules and the fixed rules in `rcv1` and `ijcnn1` clearly demonstrates the effectiveness of adaptive rules in taking advantage of changing conditions. Refer to the Appendix Section C for additional figures and discussion.

Another aspect to notice in Fig 3 is that the `add` and `mult` rules perform quite similarly. This is because during the majority of the runtime, r_t is close to 1. When $r_t = 1$, both the rules are identical. Changing the multiplicative constant could alleviate this issue. Similarly in the `grad` algorithm, due to the highly noisy nature of the gradient signal, we noticed large oscillations. Using step-sizes or a running average instead could stabilize the algorithm. We believe that such fine tuning of parameters would lead to better algorithms and even more gains.

9 Concluding Remarks

We present ACM, a single framework that provides a unified analysis for first-order optimization algorithms for composite problems. The framework allows to incorporate curvature information and only requires the computation of weak approximate solutions to the subproblems in a stochastic sense—hence, it specifically also covers randomized methods such as block-coordinate descent and also parallel algorithms. Moreover, the accuracy parameters are allowed to change over time. This combines and improves upon the results of [16, 17, 21, 33].

We leverage our framework to provide speedups when the gradient computation and the update computation times are unbalanced. In particular, we give a simple adaptive procedure to adaptively tune the accuracy that is required in for the optimization of the model in each iteration. This procedure ensures optimal balance between of the two computations.

The effectiveness of the adaptive scheme is exemplary demonstrated on a set of numerical experiments for Lasso (`gisette` and `news20` dataset) and SVM (`rcv1`, `ijcnn1`) with randomized coordinate descent as subproblem solver. The experiments shows that simply tuning a static accuracy parameter will in general not obtain optimal rate. In contrast, the parameters of our adaptive scheme vary as the optimization progresses, and achieve significant speedups.

Acknowledgements SPK thanks Mojmir Mutny for discussions relating to parallelizing preconditioned block-coordinate methods, as well as Anastasia Koloskova for help with the experiments.

References

[1] N. Agarwal, B. Bullins, and E. Hazan. Second Order Stochastic Optimization in Linear Time. *arXiv:1602.03943 [cs, stat]*, Feb. 2016.

[2] Z. Allen-Zhu. Katyusha: The First Direct Acceleration of Stochastic Gradient Methods. *arXiv:1603.05953 [cs, math, stat]*, Mar. 2016.

[3] Y. Bian, X. Li, Y. Liu, and M.-H. Yang. Parallel Coordinate Descent Newton Method for Efficient \$\ell_1\$-Regularized Minimization. *arXiv:1306.4080 [cs]*, June 2013.

[4] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, New York, NY, USA, 2004.

[5] A. Chambolle, M. Novaga, D. Cremers, and T. Pock. An introduction to total variation for image analysis. In *In Theoretical Foundations and Numerical Methods for Sparse Recovery, De Gruyter*, 2010.

[6] D.-M. Chiu and R. Jain. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Comput. Netw. ISDN Syst.*, 17(1):1–14, June 1989.

[7] A. d'Aspremont. Smooth Optimization with Approximate Gradient. *SIAM Journal on Optimization*, 19(3):1171–1183, Jan. 2008.

[8] A. Defazio, F. Bach, and S. Lacoste-Julien. SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 1646–1654. Curran Associates, Inc., 2014.

[9] C. Dünner, S. Forte, M. Takac, and M. Jaggi. Primal-Dual Rates and Certificates. In *ICML'16 - Proceedings of the 33rd International Conference on International Conference on Machine Learning*, pages 783–792, June 2016.

[10] O. Fercoq, Z. Qu, P. Richtarik, and M. Takac. Fast distributed coordinate descent for non-strongly convex losses. In *2014 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6, Sept. 2014.

[11] O. Fercoq and P. Richtárik. Accelerated, Parallel, and Proximal Coordinate Descent. *SIAM Journal on Optimization*, 25(4):1997–2023, Jan. 2015.

[12] R. Johnson and T. Zhang. Accelerating Stochastic Gradient Descent using Predictive Variance Reduction. In *NIPS - Advances in Neural Information Processing Systems 26*, pages 315–323. Curran Associates, Inc., 2013.

[13] J. D. Lee, Q. Lin, T. Ma, and T. Yang. Distributed Stochastic Variance Reduced Gradient Methods and A Lower Bound for Communication Complexity. *arXiv:1507.07595 [cs, math, stat]*, July 2015.

[14] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *arXiv:1603.06560 [cs, stat]*, Mar. 2016.

[15] Q. Lin, Z. Lu, and L. Xiao. An Accelerated Proximal Coordinate Gradient Method. In *NIPS - Advances in Neural Information Processing Systems 27*, pages 3059–3067. Curran Associates, Inc., 2014.

[16] C. Ma, J. Konecny, M. Jaggi, V. Smith, M. I. Jordan, P. Richtarik, and M. Takac. Distributed optimization with arbitrary local solvers. *Optimization Methods and Software*, 32(4):813–848, July 2017.

[17] M. Mutny and P. Richtarik. Parallel Stochastic Newton Method. *arXiv:1705.02005 [math]*, May 2017.

[18] Y. Nesterov. Efficiency of Coordinate Descent Methods on Huge-Scale Optimization Problems. *SIAM Journal on Optimization*, 22(2):341–362, Jan. 2012.

[19] Y. Nesterov and S. Stich. Efficiency of the Accelerated Coordinate Descent Method on Structured Optimization Problems. *SIAM Journal on Optimization*, 27(1):110–123, Jan. 2017.

[20] L. M. Nguyen, J. Liu, K. Scheinberg, and M. Takac. SARAH: A Novel Method for Machine Learning Problems Using Stochastic Recursive Gradient. *arXiv:1703.00102 [cs, math, stat]*, Feb. 2017.

[21] Z. Qu, P. Richtarik, M. Takac, and O. Fercoq. SDNA: Stochastic Dual Newton Ascent for Empirical Risk Minimization. *arXiv:1502.02268 [cs]*, Feb. 2015.

[22] S. J. Reddi, J. Konecny, P. Richtarik, B. Poczos, and A. Smola. AIDE: Fast and Communication Efficient Distributed Optimization. *arXiv:1608.06879 [cs, math, stat]*, Aug. 2016.

[23] P. Richtarik and M. Takac. Distributed Coordinate Descent Method for Learning with Big Data. *J. Mach. Learn. Res.*, 17(1):2657–2681, Jan. 2016.

[24] P. Richtarik and M. Takac. Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, 156(1-2):433–484, Mar. 2016.

[25] N. L. Roux, M. Schmidt, and F. Bach. A Stochastic Gradient Method with an Exponential Convergence Rate for Finite Training Sets. In *NIPS - Neural Information Processing Systems*, NIPS'12, pages 2663–2671, USA, 2012. Curran Associates Inc.

[26] M. Schmidt, N. L. Roux, and F. R. Bach. Convergence Rates of Inexact Proximal-Gradient Methods for Convex Optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 1458–1466. Curran Associates, Inc., 2011.

[27] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, New York, NY, USA, 2014.

[28] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: Primal estimated sub-gradient solver for SVM. *Mathematical Programming*, 127(1):3–30, Mar. 2011.

[29] S. Shalev-Shwartz and T. Zhang. Stochastic Dual Coordinate Ascent Methods for Regularized Loss. *J. Mach. Learn. Res.*, 14(1):567–599, Feb. 2013.

[30] S. Shalev-Shwartz and T. Zhang. Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization. *Mathematical Programming*, 155(1-2):105–145, Jan. 2016.

[31] V. Smith, S. Forte, C. Ma, M. Takac, M. I. Jordan, and M. Jaggi. CoCoA: A General Framework for Communication-Efficient Distributed Optimization. *arXiv:1611.02189 [cs]*, Nov. 2016.

[32] S. U. Stich, C. L. Müller, and B. Gärtner. Variable metric random pursuit. *Mathematical Programming*, 156(1-2):549–579, Mar. 2016.

[33] R. Tappenden, P. Richtarik, and J. Gondzio. Inexact Coordinate Descent: Complexity and Preconditioning. *Journal of Optimization Theory and Applications*, 170(1):144–176, July 2016.

[34] S. Tu, S. Venkataraman, A. C. Wilson, A. Gittens, M. I. Jordan, and B. Recht. Breaking Locality Accelerates Block Gauss-Seidel. *arXiv:1701.03863 [math]*, Jan. 2017.

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

A Subproblem Solvers

The ACM framework described in Algorithm 1 employs an arbitrary *subproblem solver*, delivering $\Delta\mathbf{x}_t$ of a relative accuracy measured by Definition 3. As noted in [33], the biggest advantage of requiring only approximate solutions to sub-problems means that we can use *iterative* solvers. Apart from direct specialized algorithms for solving the subproblems, we discuss two notable cases: i) when g is separable, we can use *coordinate descent* algorithms and ii) when M is accessible as a sum of outer products, we can even employ *stochastic gradient* methods.

A.1 Parallel Block-Coordinate Updates

Coordinate methods have been widely successful for not just their faster convergence, but also because they are less resource intensive and provide faster update times [35]. To take advantage of this, when g is separable, we can sample a block of coordinates and create smaller subproblems. This is the setting considered in [17, 21, 33].

Coordinate notation. Let $[v]_i$ or simply v_i denote the i-th component of vector $\mathbf{v}$. Similarly $M_{i,j}$ or $M(i,j)$ denotes the (i,j)-th component of matrix M. For a set of indices $\mathcal{I} = (i_1, \ldots, i_t) \subseteq [d]$, $[\mathbf{v}]_{\mathcal{I}}$ or $\mathbf{v}_{\mathcal{I}}$ denotes the t-dimensional vector $(v_{i_1}, \ldots, v_{i_t})$. The matrix equivalent $M_{\mathcal{I}}$ is a $t \times t$ principal submatrix of M whose (t,l)-th entry is $M_{\mathcal{I}}(t,l) = M(i_t, i_l)$.

If we want to denote the d dimensional vector, we use $\vec{\mathbf{v}}_i$ to mean $v_i\mathbf{e}_i$ and $\vec{\mathbf{v}}_{\mathcal{I}} = \sum_{i\in\mathcal{I}} v_i\mathbf{e}_i$ where $(\mathbf{e}_1, \ldots, \mathbf{e}_d)$ is the standard coordinate basis of $\mathbb{R}^d$.

Defining Subproblems. We define a κ-block *unbiased* sampling as a sample where we select κ blocks of coordinates to update in parallel at each step.

Definition 4 (κ-block unbiased sampling)**.** *Let $1 \le \kappa \le d$. A κ-block unbiased sampling with parameter m chooses a set $\mathcal{I}$ of κ blocks of coordinates, $\mathcal{I} = \{\pi_1, \ldots, \pi_\kappa\}$ at random such that for any vector $\mathbf{v} \in \mathbb{R}^d$,*

$$\mathbb{E}\left[\sum_{k\in[\kappa]} \vec{\mathbf{v}}_{\pi_k}\right] = \frac{\kappa}{m}\mathbf{v}\,.$$

Remark 5. *A κ-block unbiased sampling can also be realized as follows: Consider not only one partition of the coordinates, but a collection of partitions $\mathcal{P} = \{P_1, \ldots, P_h\}$, where each $P_j \in \mathcal{P}$ does partition the full $[d]$ coordinates into at least κ blocks. Given P_j, the blocks are therefore disjoint. By i) first sampling a partition $P_j \in \mathcal{P}$ uniformly at random, and ii) picking κ blocks $\mathcal{I} = \{\pi_1, \ldots, \pi_\kappa\} \subseteq P_j$ again uniformly at random, one obtains a κ-block unbiased sampling.*

Remark 6. *A κ-block unbiased sampling need not be confined to choosing from a fixed set of blocks as was [24, 33] (which corresponds to using a single partition only). Thus we can gain speedups by breaking locality [34].*

We also need to quantify how separable the matrix M is using a parameter σ. This dictates how much we gain by performing κ number of block-coordinate updates in parallel.

Definition 5 (σ-separable)**.** *Let $\mathcal{I} = \{\pi_1, \ldots, \pi_\kappa\} \subseteq P_j$ be a κ-block unbiased sample of coordinates. Then M is said to be σ-separable if*

$$\sigma \ge \sigma_{max} \overset{\mathsf{def}}{=} \max_{\mathbf{v}\neq 0} \frac{\mathbb{E}_{\mathcal{I}} \left\|\sum_{k\in[\kappa]} \vec{\mathbf{v}}_{\pi_k}\right\|_M^2}{\mathbb{E}_{\mathcal{I}} \sum_{k\in[\kappa]} \left\|\vec{\mathbf{v}}_{\pi_k}\right\|_M^2}\,.$$

Lemma 7. *$M \succcurlyeq 0$ is κ-separable. I.e. for any κ-block unbiased sampling*

$$\sigma_{max} \le \kappa\,.$$

Lemma 7 gives a bound on how *non-separable* a problem can be and shows that even in the worst case, adding parallelism will never make the rate worse.

Finally, we need to define a constant ν which relates the expected matrix $\mathbb{E}_{\mathcal{I}}[M_{\mathcal{I}}]$ to the full matrix M.

Definition 6 (ν)**.** *Let $\nu \ge 1$ such that*

$$M \preccurlyeq \mathbb{E}_{\mathcal{I}}\left[\frac{m}{\nu\kappa} M_{\mathcal{I}}\right]\,.$$

Parallel updates. Suppose we have a κ-block unbiased sampling $\mathcal{I} = \{\pi_1, \dots, \pi_\kappa\} \subseteq P_j$ and M is σ separable. Then we create κ subproblems which we solve *in parallel.* For each $\pi_k \in \mathcal{I}$, we create the subproblem $m_t^\sigma(\Delta\mathbf{x}; \pi_k, \mathbf{x}_t, M)$:

$$m_t^\sigma(\Delta\mathbf{x}; \pi_k, \mathbf{x}_t, M) \overset{\mathsf{def}}{=} \langle \nabla_{\pi_k} f(\mathbf{x}), (\Delta\mathbf{x})_{\pi_k} \rangle + \frac{\sigma}{2} \left\| \Delta\mathbf{x}_{\pi_k} \right\|_{M_{\pi_k}}^2 + g_{\pi_k}(\mathbf{x}_{\pi_k} + (\Delta\mathbf{x})_{\pi_k}) - g_{\pi_k}(\mathbf{x}_{\pi_k}) \,,$$

and the total subproblem solved at each iteration is

$$m_t^\sigma(\Delta\mathbf{x}; \mathcal{I}, \mathbf{x}_t, M) = \sum_{k \in [\kappa]} m_t^\sigma(\Delta\mathbf{x}; \pi_k, \mathbf{x}_t, M) \,.$$

We can relate Θ_t progress on the coordinate model $m_t^\sigma(\Delta\mathbf{x}; \mathcal{I}, \mathbf{x}_t, M)$ to the progress on the global model $m_t(\Delta\mathbf{x}; \mathbf{x}_t, M)$. Denote $(m_t^\sigma)^\star = \min_{\Delta\mathbf{x}} m_t^\sigma(\Delta\mathbf{x}; \mathcal{I}, \mathbf{x}_t, M)$ and $m_t^\star = \min_{\Delta\mathbf{x}} m_t(\Delta\mathbf{x}; \mathbf{x}_t, M)$.

Lemma 8. *Assuming that M is σ-separable, let $\Delta\mathbf{x}_t$ be such that*

$$m_t^\sigma(\Delta\mathbf{x}_t; \mathcal{I}, \mathbf{x}_t, M) - (m_t^\sigma)^\star \leq (1 - \Theta_t)(m_t^\sigma(\mathbf{0}; \mathcal{I}, \mathbf{x}_t, M) - (m_t^\sigma)^\star) \,.$$

Then

$$\mathbb{E}[m_t(\Delta\mathbf{x}_t; \mathbf{x}_t, M) - m_t^\star] \leq (1 - \frac{\kappa\Theta_t}{m\sigma\nu})(m_t(\mathbf{0}; \mathbf{x}_t, M) - m_t^\star) \,.$$

We can combine Lemma 8 with Theorems 1,2,3, and 4 to obtain the equivalent parallelized block *coordinate* versions. This gives a parallel, approximate block coordinate algorithm with primal-dual guarantees. If $\sigma \ll \kappa$, we gain significant speed-ups due to parallel updates as noted in [17]. In fact, a more careful analysis is possible which requires the constant ν (Definition 6) only in Theorems 1 and 3 (cf. [34]). We however ignore this point to simplify the presentation of our results.

A.2 Stochastic Gradients

The subproblem $m_t(\Delta\mathbf{x}; \mathbf{x}_t, M)$ can often be decomed as a sum of n functions where each function is very simple. For example, let us assume that $M = \frac{1}{n} A^\top A$ for some $n \times d$ matrix A with row $A_{i:}$ and that $f(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n f_i(\mathbf{x})$. Then

$$m_t(\Delta\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n \left[\langle \nabla f_i(\mathbf{x}), \Delta\mathbf{x} \rangle + \langle A_{i:}, \Delta\mathbf{x} \rangle^2 + g(\mathbf{x}) \right] .$$

Then a stochastic estimate of the gradient $\nabla m_t(\Delta\mathbf{x})$ can be very cheaply computed as $\nabla f_i(\mathbf{x}) + (\langle A_{i:}, \Delta\mathbf{x} \rangle) A_{i:}$. Then any variance-reduced stochastic algorithm such as [8, 12, 20] can be applied to minimize $m_t(\Delta\mathbf{x}; \mathbf{x}_t, M)$ to an accuracy of Θ. In this way, second order information M can be efficiently introduced into stochastic gradient algorithms. A similar idea was used in [1] to efficiently perform Newton-like updates in the smooth case.

B Additional Proofs

B.1 Convergence of $F(\mathbf{x})$

Lemma 9. *Let h and V be two convex functions defined over Q such that the following assumption holds:*

$$0 \in Q \text{ and } h(0) = V(0) = 0 \,.$$

For any $\beta \in (0, 1]$, define the following function

$$\zeta(\mathbf{x}, \beta) \overset{\mathsf{def}}{=} h(\mathbf{x}) + \frac{1}{\beta} V^2(\mathbf{x}) \,.$$

Then, ζ has the following properties:

1. $\frac{V^2(\mathbf{x})}{\beta}$ *is jointly convex in $(\beta, \mathbf{x})$.*
2. *$\zeta(\mathbf{x}, \beta)$ defined over $Q \times (0, 1]$ is jointly convex over its domain.*
3. $\min_{\mathbf{x} \in Q} \zeta(\mathbf{x}, \beta) \leq \beta \min_{\mathbf{x} \in Q} \zeta(\mathbf{x}, 1)$.

Proof. **1:** For any x_1, x_2 in Q and $\beta_1, \beta_2 \in (0,1]$,

$$\begin{aligned}((1-\alpha)\beta_1+\alpha\beta_2)&\left(V^2(\mathbf{x}_1)\frac{1-\alpha}{\beta_1}+V^2(\mathbf{x}_2)\frac{\alpha}{\beta_2}\right)\\ &= V^2(\mathbf{x}_1)(1-\alpha)^2+V^2(\mathbf{x}_2)\alpha^2+\alpha(1-\alpha)\left(V^2(\mathbf{x}_1)\frac{\beta_2}{\beta_1}+V^2(\mathbf{x}_2)\frac{\beta_1}{\beta_2}\right)\\ &\ge V^2(\mathbf{x}_1)(1-\alpha)^2+V^2(\mathbf{x}_2)\alpha^2+2\alpha(1-\alpha)V(\mathbf{x}_1)V(\mathbf{x}_2)\\ &= ((1-\alpha)V(\mathbf{x}_1)+\alpha V(\mathbf{x}_2))^2\\ &\ge V^2((1-\alpha)\mathbf{x}_1+\alpha\mathbf{x}_2)\,.\end{aligned}$$

In the first inequality, we used the AM-GM inequality, whereas for the second we used that $V(\mathbf{x})$ is a convex function. Thus we have shown that

$$\frac{1}{(1-\alpha)\beta_1+\alpha\beta_2}V^2((1-\alpha)\mathbf{x}_1+\alpha\mathbf{x}_2)\le V^2(\mathbf{x}_1)\frac{1-\alpha}{\beta_1}+V^2(\mathbf{x}_2)\frac{\alpha}{\beta_2}\,.$$

2: For any x_1, x_2 in Q and $\beta_1, \beta_2 \in (0,1]$,

$$\begin{aligned}\zeta((1-\alpha)x_1+\alpha x_2,(1-\alpha)\beta_1+\alpha\beta_2) &= h((1-\alpha)x_1+\alpha x_2)+\frac{V^2((1-\alpha)x_1+\alpha x_2)}{(1-\alpha)\beta_1+\alpha\beta_2}\\ &\le h(1-\alpha)(\mathbf{x}_1)+\alpha h(\mathbf{x}_2)+\frac{V^2((1-\alpha)x_1+\alpha x_2)}{(1-\alpha)\beta_1+\alpha\beta_2}\\ &\le h(1-\alpha)(\mathbf{x}_1)+\alpha h(\mathbf{x}_2)+(1-\alpha)\frac{V^2(\mathbf{x}_1)}{\beta_1}+\alpha\frac{V^2(\mathbf{x}_2)}{\beta_2}\,.\end{aligned}$$

In the first inequality, we used the convexity of h and in the second inequality we used that $\frac{V(\mathbf{x})}{\beta}$ is jointly convex.
3: Then notice that as we tend β to zero, since $V^2(\mathbf{x})\ge 0$, the term $\frac{V^2(\mathbf{x})}{\beta}$ goes to ∞ unless $V^2(\mathbf{x})=0$. Thus $\lim_{\beta\to 0}\min_{\mathbf{x}}\zeta(\mathbf{x},\beta)=\min_{\mathbf{x}\,\mid\, V^2(\mathbf{x})=0}h(\mathbf{x})\le h(0)=0$. Let $\chi(\beta)=\min_{\mathbf{x}}\zeta(\mathbf{x},\beta)$. Since $\zeta(\mathbf{x},\beta)$ is jointly convex over its parameters, $\chi(\beta)$ is also convex [4]. We can then extend the domain of $\chi(\beta)$ from $(0,1]$ to $[0,1]$ by taking the limit point i.e. $\chi(0)\overset{\mathsf{def}}{=}\lim_{\beta\to 0}\chi(\beta)$. The function $\chi(\beta)$ remains convex over the extended domain [4]. Thus

$$\chi(\beta(1)+(1-\beta)0)\le\beta\chi(1)+(1-\beta)\chi(0)\le\beta\chi(1)\,.$$

□

Proof of Lemma 2

Proof. Let $\beta=\frac{\lambda+\lambda_g}{1+\lambda_g}$, $h(\Delta\mathbf{x})=\langle\nabla f(v),\Delta\mathbf{x}\rangle+g(\Delta\mathbf{x}+v)-g(v)-\lambda_g\left\|\Delta\mathbf{x}\right\|_M^2$ and $V^2(\Delta\mathbf{x})=\frac{\lambda+\lambda_g}{2}\left\|\Delta\mathbf{x}\right\|_M^2$ and $Q=\{\chi-v\}$. Observe the following:

1. h is convex in $\Delta\mathbf{x}$ since g is λ_g-strongly convex. V is also convex since $\left\|\Delta\mathbf{x}\right\|_M$ is a convex function.
2. Since $v\in\chi$, $0\in\{\chi-v\}$.
3. $h(0)=V(0)=0$.

We apply Lemma 9 to prove the required result. □

Proof of Theorem 1

$$\begin{aligned}\mathbb{E}_t[F(\mathbf{x}_{t+1})] &\overset{(1)}{\le}\mathbb{E}_t\left[f(\mathbf{x}_t)+\langle\nabla f(\mathbf{x}_t),\mathbf{x}_{t+1}-\mathbf{x}_t\rangle+\frac{1}{2}\|\mathbf{x}_{t+1}-\mathbf{x}_t\|_M{}^2+g(\mathbf{x}_{t+1})\right]\\ &= F(\mathbf{x}_t)+\mathbb{E}_t[m_t(\Delta\mathbf{x}_t)]\\ &= F(\mathbf{x}_t)+\mathbb{E}_t[m_t(\Delta\mathbf{x}_t)-m_t^\star]+m_t^\star\\ &\le F(\mathbf{x}_t)+(1-\Theta_t)[\underbrace{m_t(\mathbf{0})}_{=0}-m_t^\star]+m_t^\star\\ &= F(\mathbf{x}_t)+\Theta_t\min_{\Delta\mathbf{x}\in\chi-\mathbf{x}_t}m(\Delta\mathbf{x};\mathbf{x}_t,M)\,.\end{aligned}$$

Using Lemma 2, we can relate progress made on $m(\Delta\mathbf{x};\mathbf{x}_t,M)$ to the progress made with $m(\Delta\mathbf{x};\mathbf{x}_t,\lambda M)$.

$$\begin{aligned}\mathbb{E}_t[F(\mathbf{x}_{t+1})] &\le F(\mathbf{x}_t) + \Theta_t\lambda \min_{\Delta\mathbf{x}\in\chi-\mathbf{x}_t} m(\Delta\mathbf{x};\mathbf{x}_t,\lambda M)\\ &\le (1-\Theta_t\lambda)F(\mathbf{x}_t) + \Theta_t\lambda\left[F(\mathbf{x}_t) + m(\mathbf{x}^\star - \mathbf{x}_t;\mathbf{x}_t,\lambda M)\right]\\ &\overset{(2)}{\le} (1-\Theta_t\lambda)F(\mathbf{x}_t) + \Theta_t\lambda(F(\mathbf{x}^\star))\,.\end{aligned}$$

Subtracting $F(\mathbf{x}^\star)$ on both sides gives

$$\begin{aligned}\mathbb{E}[F(\mathbf{x}_T) - F(\mathbf{x}^\star)] &= \mathbb{E}[\mathbb{E}_t[F(\mathbf{x}_T) - F(\mathbf{x}^\star)]]\\ &\le (1-\Theta_T\lambda)\,\mathbb{E}[F(\mathbf{x}_{T-1}) - F(\mathbf{x}^\star)]\\ &\le \prod_{t=0}^{T}(1-\Theta_t\lambda)\,\mathbb{E}[F(\mathbf{x}_0) - F(\mathbf{x}^\star)]\\ &\le \exp\left(-\sum_{t=0}^{T}\Theta_t\lambda\right)\mathbb{E}[F(\mathbf{x}_0) - F(\mathbf{x}^\star)]\,.\end{aligned}$$

Thus for $T\tilde{\Theta}_T\lambda \ge \log\left(\frac{F(\mathbf{x}_0)-F(\mathbf{x}^\star)}{\epsilon}\right)$, $\mathbb{E}[F(\mathbf{x}_{t+1}) - F(\mathbf{x}^\star)] \le \epsilon$. □

Proof of Theorem 2. We assumed that $m_t(\Delta\mathbf{x}_t) \le 0$ for all steps. Using smoothness, $F(\mathbf{x}_{t+1}) \le F(\mathbf{x}_t) + m_t(\Delta\mathbf{x}_t) \le F(\mathbf{x}_t)$. Thus for all t, $F(\mathbf{x}_t) \le F(\mathbf{x}_0)$ and $\mathbf{x}_t \in \{y \mid F(y) \le F(\mathbf{x}_0)\}$.

$$\begin{aligned}\mathbb{E}_t[F(\mathbf{x}_{t+1})] &\overset{(1)}{\le} \mathbb{E}_t\left[f(\mathbf{x}_t) + \langle\nabla f(\mathbf{x}_t), \mathbf{x}_{t+1} - \mathbf{x}_t\rangle + \frac{1}{2}\|\mathbf{x}_{t+1} - \mathbf{x}_t\|_M{}^2 + g(\mathbf{x}_{t+1})\right]\\ &= F(\mathbf{x}_t) + \mathbb{E}_t[m_t(\Delta\mathbf{x}_t)]\\ &= F(\mathbf{x}_t) + \mathbb{E}_t[m_t(\Delta\mathbf{x}_t) - m_t^\star] + m_t^\star\\ &\le F(\mathbf{x}_t) + (1-\Theta_t)[\underbrace{m_t(\mathbf{0})}_{=0} - m_t^\star] + m_t^\star\\ &\le (1-\Theta_t)F(\mathbf{x}_t) + \Theta_t \min_{\alpha\in[0,1]}\left[F(\mathbf{x}_t) + m(\alpha(\mathbf{x}^\star - \mathbf{x}_t);\mathbf{x}_t,M)\right]\\ &= (1-\Theta_t)F(\mathbf{x}_t) + \Theta_t \min_{\alpha\in[0,1]}[f(\mathbf{x}_t) + \alpha\langle\nabla f(\mathbf{x}_t), \mathbf{x}^\star - \mathbf{x}_t\rangle\\ &\qquad + \frac{\alpha^2}{2}\|\mathbf{x}^\star - \mathbf{x}_t\|_M{}^2 + g(\alpha\mathbf{x}^\star + (1-\alpha)\mathbf{x}_t)]\\ &\le (1-\Theta_t)F(\mathbf{x}_t) + \Theta_t \min_{\alpha\in[0,1]}\left[F(\mathbf{x}_t + (1-\alpha)\mathbf{x}^\star) + \frac{\alpha^2 D}{2}\right]\\ &\le (1-\Theta_t)F(\mathbf{x}_t) + \Theta_t \min_{\alpha\in[0,1]}\left[(1-\alpha)F(\mathbf{x}_t) + \alpha F(\mathbf{x}^\star) + \frac{\alpha^2 D}{2}\right]\,.\end{aligned}$$

Let $\delta_t \overset{\text{def}}{=} \mathbb{E}[F(\mathbf{x}_t) - F(\mathbf{x}^\star)]$ denote the suboptimality of F at t-th step. Substituting $\alpha = 0$ in the previous equation gives that $\mathbb{E}[F(\mathbf{x}_{t+1})] \le F(\mathbf{x}_t)$ i.e. δ_t is a non-increasing sequence. Subtracting $F(\mathbf{x}^\star)$ in the same equation,

$$\begin{aligned}\delta_{t+1} &\le \min_{\alpha\in[0,1]}(1-\Theta_t\alpha)\delta_t + \frac{\alpha^2 D}{2}\\ &= \delta_t - \frac{\Theta_t}{2D}\delta_t^2\\ &\le \delta_t - \frac{\Theta_t}{2D}\delta_t\delta_{t+1}\,.\end{aligned}$$

The last step used that δ_t is a non-increasing sequence. Let us assume that $\delta_t \neq 0$ (since otherwise we are done). Dividing both sides of the equation with $\delta_t\delta_{t+1}$,

$$\frac{1}{\delta_{t+1}} \ge \frac{1}{\delta_t} + \frac{\Theta_t}{2D} \ge \frac{1}{\delta_0} + \frac{\sum_{i=0}^{t}\Theta_i}{2D} \ge \frac{t\tilde{\Theta}_t}{2D}\,.$$

Recall the definition of $\tilde{\Theta}_t$ from the statement of Theorem 2:

$$t\tilde{\Theta}_t \overset{\text{def}}{=} \sum_{i=0}^{t} \Theta_i\,.$$

□

B.2 Primal-dual convergence proofs

Instead of using the SDCA lemma as [21] uses, we will use Lemma 7 from the CoCoA analysis [31]. This allows us to generalize for the non-strongly convex case, as well as deal with approximate solutions to the subproblem.

Lemma 10 ([31, Lemma 7])**.** *Let g be λ_g-strongly convex with respect to norm $\|\cdot\|_M$. Then at each iteration if Definition 3 is satisfied, then for any $s \in [0,1]$ it holds that*

$$\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}_t) - \mathcal{O}_A(\boldsymbol{\alpha}_{t+1})] \geq \Theta_t\left(sG(\boldsymbol{\alpha}_t) - \frac{s^2}{2}R_t\right)\,,$$

where

$$R_t \overset{\text{def}}{=} \left(1 - \frac{\lambda_g(1-s)}{s}\right)\|u_t - \boldsymbol{\alpha}_t\|_M^2$$

for $\mathbf{v}_t = A\boldsymbol{\alpha}_t$, $u_t \in \partial g^*(-A^\top \nabla f(\mathbf{v}_t))$ and $G(\boldsymbol{\alpha}_t) \overset{\text{def}}{=} \mathcal{O}_B(\nabla f(\mathbf{v}_t)) + \mathcal{O}_A(\boldsymbol{\alpha}_t)$.

Proof of Theorem 3. By our assumption $f(A\boldsymbol{\alpha})$ is M smooth and λ_f strongly convex. Further $g(\boldsymbol{\alpha})$ is λ_g strongly convex. This means that we can apply Theorem 1 to $\mathcal{O}_A(\boldsymbol{\alpha})$ and so for $\lambda = \frac{\lambda_f + \lambda_g}{1+\lambda_g}$,

$$\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}_t) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)] \leq \exp\left(-t\tilde{\Theta}_t\lambda\right)\left(\mathcal{O}_A(\boldsymbol{\alpha}_0) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)\right).$$

Also Lemma 10 is valid for any $s \in [0,1]$. We will pick s such that $R_t = 0$. For this $s = \frac{\lambda_g}{1+\lambda_g}$ suffices. Putting these together with the fact that $\boldsymbol{\alpha}_0 = 0$,

$$\begin{aligned}\frac{\lambda_g}{1+\lambda_g}\,\mathbb{E}[G(\boldsymbol{\alpha}_t)] &\leq \frac{1}{\Theta_t}\,\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}_t) - \mathcal{O}_A(\boldsymbol{\alpha}_{t+1})]\\ &\leq \frac{1}{\Theta_t}\,\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}_t) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)]\\ &\leq \frac{1}{\Theta_t}\exp\left(-t\tilde{\Theta}_t\lambda\right)\left(\mathcal{O}_A(\mathbf{0}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)\right).\end{aligned}$$

For $T \geq \frac{1}{\lambda\tilde{\Theta}_T}\log\left(\frac{((1+\lambda_g)(\mathcal{O}_A(\mathbf{0})-\mathcal{O}_A(\boldsymbol{\alpha}^\star))}{\lambda_g\Theta_t\epsilon}\right)$, $\mathbb{E}[G(\boldsymbol{\alpha}_t)] \leq \epsilon$. □

Lemma 11. *For all t, assuming the conditions in Theorem 4, $R_t \leq 4D$.*

Proof. As we saw in the proof of Theorem 2, the assumption that at every step $m_t(\Delta\alpha_t) \leq m_t(\mathbf{0})$ means that the iterates α_t produced by our algorithm are bounded in the domain $Q \overset{\text{def}}{=} \{\boldsymbol{\alpha} \mid \mathcal{O}_A(\boldsymbol{\alpha}) \leq \mathcal{O}_A(\mathbf{0})\}$ with squared diameter $D = \max_{a,b\in Q}\|a-b\|_M^2$. Thus we can use the *Lipschitzing trick* on $g^*(-A^\top w)$ similar to [31, Lemma 8] or [9, Theorem 6] and restrict the domain of $g(\boldsymbol{\alpha})$ to Q. □

Proof of Theorem 4. By our assumption that $f(A\boldsymbol{\alpha})$ is M smooth, we can apply Theorem 2 to $\mathcal{O}_A(\boldsymbol{\alpha})$ i.e. for $D' = \max_{\mathbf{y}\in Q}\|\mathbf{y} - \mathbf{x}^\star\|_M^2 \leq D$, so

$$\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}_t) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)] \leq \frac{2D}{\tilde{\Theta}_t t}\,.$$

Substituting Lemma 11 in Lemma 10, and combining with the above statement,

$$\begin{aligned}\mathbb{E}[G(\boldsymbol{\alpha}_t)] &\leq \frac{1}{s\Theta_t}\,\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}_t) - \mathcal{O}_A(\boldsymbol{\alpha}_{t+1})] + \frac{\Theta_t s}{2}R_t\\ &\leq \frac{1}{s\Theta_t}\,\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}_t) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)] + 2\Theta_t sD\\ &\leq \frac{2D}{s\Theta_t\tilde{\Theta}_t t} + 2\Theta_t sD\,.\end{aligned}$$

Since the above statement holds for any $s \in [0,1]$ and $t > 0$, for $t \geq \frac{1}{\Theta_t^2 \tilde{\Theta}_t}$, we can use $s = \frac{1}{\Theta_t \sqrt{\tilde{\Theta}_t t}}$ and obtain a convergence rate

$$G(\boldsymbol{\alpha}_t) \leq \frac{4D}{\sqrt{\tilde{\Theta}_t t}}\,.$$

The above statement gives us the convergence proportional to $\frac{1}{\sqrt{t}}$ for the last iterate $\boldsymbol{\alpha}_t$ *without averaging.* While this is a very useful result in itself, if we are willing to take an average iterate $\tilde{\boldsymbol{\alpha}}_t$, we can show a convergence proportional to $\frac{1}{t}$. Let $\hat{\Theta}_t = \min_{i=0}^t \Theta_i$,

$$\begin{aligned}
\mathbb{E}[G(\tilde{\boldsymbol{\alpha}}_t)] = \mathbb{E}\left[G\left(\frac{1}{t-t_0}\sum_{i=t_0}^{t-1}\boldsymbol{\alpha}_i\right)\right] &\leq \frac{1}{t-t_0}\,\mathbb{E}\left[\sum_{i=t_0}^{t-1} G(\boldsymbol{\alpha}_i)\right] \\
&\leq \frac{1}{s\hat{\Theta}_t(t-t_0)}\sum_{i=t_0}^{t-1}\mathbb{E}\left[\mathcal{O}_A(\boldsymbol{\alpha}_i) - \mathcal{O}_A(\boldsymbol{\alpha}_{i+1})\right] + 2\hat{\Theta}_t s D \\
&\leq \frac{1}{s\hat{\Theta}_t(t-t_0)}\,\mathbb{E}\left[\mathcal{O}_A(\boldsymbol{\alpha}_{t_0}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)\right] + 2\hat{\Theta}_t s D \\
&\leq \frac{2D}{s\tilde{\Theta}_{t_0}\hat{\Theta}_t(t-t_0)t_0} + 2\hat{\Theta}_t s D\,.
\end{aligned}$$

For $t \geq t_0 + \frac{1}{\hat{\Theta}_t}$, set $s = \frac{1}{(t-t_0)\hat{\Theta}_t} \leq 1$. Then the above equation becomes

$$\mathbb{E}[G(\tilde{\boldsymbol{\alpha}}_t)] \leq \frac{2D}{\tilde{\Theta}_{t_0} t_0} + \frac{2D}{t-t_0}\,.$$

Taking $T \geq t_0 + \max\left[\frac{4D}{\epsilon}, \frac{1}{\hat{\Theta}_t}\right]$ and $t_0 \geq \frac{4D}{\tilde{\Theta}_{t_0}\epsilon}$ ensures that $\mathbb{E}[G(\tilde{\boldsymbol{\alpha}}_t)] \leq \epsilon$.

B.3 Parallel block coordinate updates

Proof of Lemma 7. Recall the definition of $\sigma_{\max}$.

$$\sigma_{max} \overset{\mathsf{def}}{=} \max_{\mathbf{v}\neq 0} \frac{\mathbb{E}_{\mathcal{I}}\left\|\sum_{k\in[\kappa]} \vec{\mathbf{v}}_{\pi_k}\right\|_M^2}{\mathbb{E}_{\mathcal{I}}\sum_{k\in[\kappa]}\left\|\vec{\mathbf{v}}_{\pi_k}\right\|_M^2}\,.$$

This follows because for any vector $\mathbf{v} \neq 0$, using the convexity of $\|\cdot\|_M$ and Jensen's inequality,

$$\left\|\sum_{k\in[\kappa]}\vec{\mathbf{v}}_{\pi_k}\right\|_M^2 \leq \kappa \sum_{k\in[\kappa]}\left\|\vec{\mathbf{v}}_{\pi_k}\right\|_M^2\,.$$

Proof of Lemma 8. This lemma follows from the notion of expected separability over-approximation (ESO) [23, 24] using which we can bound the over-approximation.

Using the definition of σ-separable on the matrix M, for any partition $P_j \in \mathcal{P}$

$$\mathbb{E}_{\mathcal{I}}\left\|\Delta\mathbf{x}_t\right\|_M^2 = \mathbb{E}_{\mathcal{I}}\left\|\sum_{k\in[\kappa]}\left(\vec{\Delta\mathbf{x}}_t\right)_{\pi_k}\right\|_M^2 \leq \sigma\,\mathbb{E}_{\mathcal{I}}\sum_{k\in[\kappa]}\left\|\left(\vec{\Delta\mathbf{x}}_t\right)_{\pi_k}\right\|_M^2\,.$$

Recall the definition of the subproblems $m_t^\sigma(\Delta\mathbf{x}; \pi_k, \mathbf{x}_t, M)$,

$$m_t^\sigma(\Delta\mathbf{x}; \mathcal{I}, \mathbf{x}_t, M) \overset{\mathsf{def}}{=} \sum_{k\in\kappa}\langle\nabla_{\pi_k} f(\mathbf{x}), (\Delta\mathbf{x})_{\pi_k}\rangle + \frac{\sigma}{2}\left\|\Delta\mathbf{x}_{\pi_k}\right\|_{M_{\pi_k}}^2 + g_{\pi_k}(\mathbf{x}_{\pi_k} + (\Delta\mathbf{x})_{\pi_k}) - g_{\pi_k}(\mathbf{x}_{\pi_k})\,.$$

The only difference between $\mathbb{E}[m_t^\sigma(\Delta\mathbf{x};\mathcal{I},\mathbf{x}_t,M)]$ and $m_t(\Delta\mathbf{x}_t)$ is the quadratic term. However this is taken care by our previous inequality involving σ. Hence we have,

$$\begin{aligned}
\mathbb{E}_{\mathcal{I}}[m_t(\Delta\mathbf{x}_t)] &\le \mathbb{E}\left[\langle\nabla f(\mathbf{x}_t),\Delta\mathbf{x}_t\rangle + \frac{\sigma}{2}\sum_{k\in[\kappa]}\left\|\Delta\vec{\mathbf{x}}_{t\pi_k}\right\|_M^2 + g(\mathbf{x}_t+\Delta\mathbf{x}_t) - g(\mathbf{x}_t)\right] \\
&= \mathbb{E}_{\mathcal{I}}\left[\sum_{k\in[\kappa]} m_t^\sigma(\Delta\mathbf{x}_t;\pi_k,\mathbf{x}_t,M)\right] \\
&\le \Theta\,\mathbb{E}_{\mathcal{I}}\left[\min_{\Delta\mathbf{x}}\sum_{k\in[\kappa]} m_t^\sigma(\Delta\mathbf{x};\pi_k,\mathbf{x}_t,M)\right]\,.
\end{aligned}$$

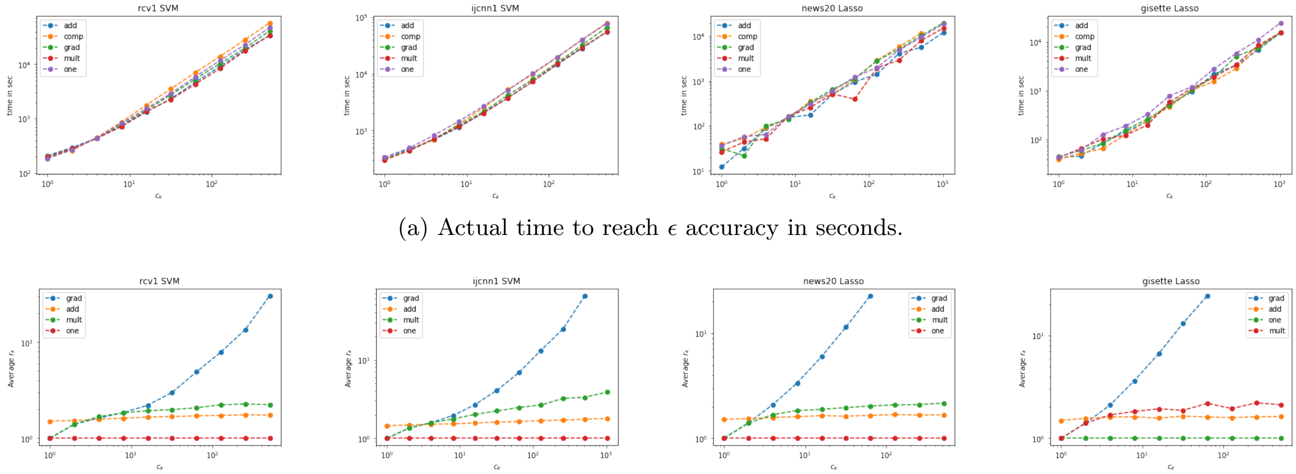
The last inequality followed from the assumption that the subproblem was solved to Θ accuracy.

Now since min is a concave function, using Jensen's inequality we can exchange the min and the expectation. Then we apply Lemma 9 to change the constant from σ to 1.

$$\begin{aligned}
\mathbb{E}_{\mathcal{I}}[m_t(\Delta\mathbf{x}_t)] &\le \Theta\,\mathbb{E}_{\mathcal{I}}\left[\min_{\Delta\mathbf{x}}\sum_{k\in[\kappa]} m_t^\sigma(\Delta\mathbf{x};\pi_k,\mathbf{x}_t,M)\right] \\
&\le \Theta\min_{\Delta\mathbf{x}}\mathbb{E}_{\mathcal{I}}\left[\sum_{k\in[\kappa]} m_t^\sigma(\Delta\mathbf{x};\pi_k,\mathbf{x}_t,M)\right] \\
&= \Theta\min_{\Delta\mathbf{x}}\mathbb{E}_{\mathcal{I}}\left[\sum_{k\in[\kappa]}\Big(\langle\nabla_{\pi_k}f(\mathbf{x}),(\Delta\mathbf{x})_{\pi_k}\rangle + \frac{\sigma}{2}\left\|\Delta\mathbf{x}_{\pi_k}\right\|_{M_{\pi_k}}^2 + g_{\pi_k}(\mathbf{x}_{\pi_k}+(\Delta\mathbf{x})_{\pi_k}) - g_{\pi_k}(\mathbf{x}_{\pi_k})\Big)\right] \\
&= \frac{\kappa\Theta}{m\sigma}\min_{\Delta\mathbf{x}}\left[\langle\nabla f(\mathbf{x}_t),\Delta\mathbf{x}\rangle + g(\mathbf{x}_t+\Delta\mathbf{x}) - g(\mathbf{x}_t) + \frac{1}{2}(\Delta\mathbf{x})^\top\,\mathbb{E}_{\mathcal{I}}\left[\frac{m}{\kappa}M_{\mathcal{I}}\right](\Delta\mathbf{x})\right] \\
&\le \frac{\kappa\Theta}{m\nu\sigma}\min_{\Delta\mathbf{x}}\left[\langle\nabla f(\mathbf{x}_t),\Delta\mathbf{x}\rangle + g(\mathbf{x}_t+\Delta\mathbf{x}) - g(\mathbf{x}_t) + \frac{1}{2}(\Delta\mathbf{x})^\top M(\Delta\mathbf{x})\right] \\
&= \frac{\kappa\Theta}{m\nu\sigma}\min_{\Delta\mathbf{x}} m_t(\Delta\mathbf{x})\,.
\end{aligned}$$

In the last inequality we used the definition of ν and to obtain the equality before that, we used the assumption that the sampling scheme was unbiased.

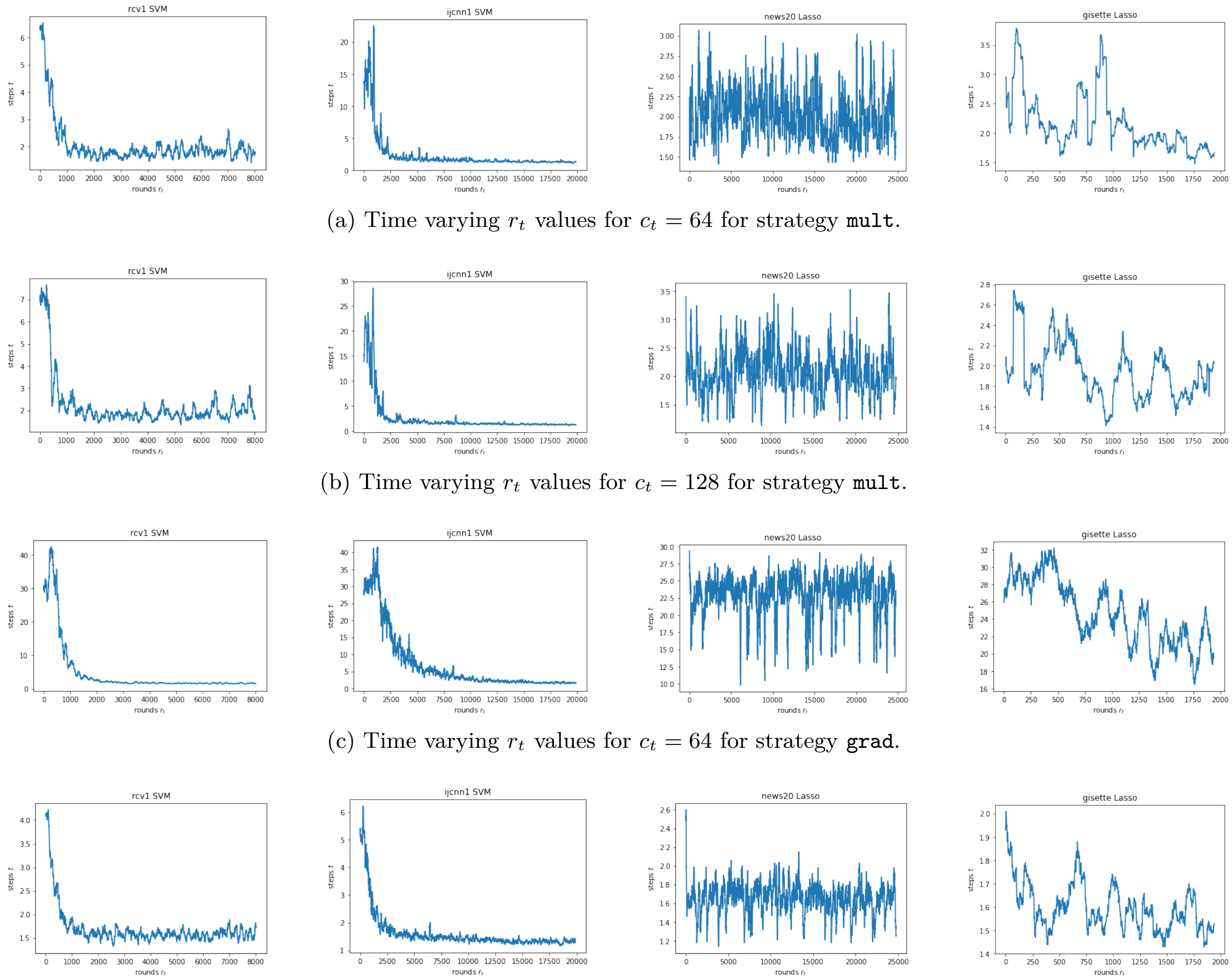
C Additional Experiments



(a) Actual time to reach ϵ accuracy in seconds.

(b) Average values of r_t for varying values of c_t by different strategies.

Figure 6: Figures missed from the main paper of which only a subset were displayed.



(a) Time varying r_t values for $c_t = 64$ for strategy **`mult`**.

(b) Time varying r_t values for $c_t = 128$ for strategy **`mult`**.

(c) Time varying r_t values for $c_t = 64$ for strategy **`grad`**.

(d) Time varying r_t values for $c_t = 64$ for strategy **`add`**.

Figure 7: Plots showing uniform behavior of strategies across values of c_t, and depending only on the dataset. This strengthens our claim that the *adaptive* behavior of r_t is predominantly influenced by the dataset aka the *hardness of the subproblem*.