Convex Optimization for Big Data

Volkan Cevher[*], Stephen Becker[†], and Mark Schmidt[‡]

September 2014

This article reviews recent advances in convex optimization algorithms for Big Data, which aim to reduce the computational, storage, and communications bottlenecks. We provide an overview of this emerging field, describe contemporary approximation techniques like first-order methods and randomization for scalability, and survey the important role of parallel and distributed computation. The new Big Data algorithms are based on surprisingly simple principles and attain staggering accelerations even on classical problems.

Convex optimization in the wake of Big Data

Convexity in signal processing dates back to the dawn of the field, with problems like least-squares being ubiquitous across nearly all sub-areas. However, the importance of convex formulations and optimization has increased even more dramatically in the last decade due to the rise of new theory for structured sparsity and rank minimization, and successful statistical learning models like support vector machines. These formulations are now employed in a wide variety of signal processing applications including compressive sensing, medical imaging, geophysics, and bioinformatics [1–4].

There are several important reasons for this explosion of interest, with two of the most obvious ones being the existence of efficient algorithms for computing globally optimal solutions and the ability to use convex geometry to prove useful properties about the solution [1, 2]. A unified convex formulation also transfers useful knowledge across different disciplines, such as sampling and computation, that focus on different aspects of the same underlying mathematical problem [5].

However, the renewed popularity of convex optimization places convex algorithms under tremendous pressure to accommodate increasingly large data sets and to solve problems in unprecedented dimensions. Internet, text, and imaging problems (among a myriad of other examples) no longer produce data sizes from megabytes to gigabytes, but rather from terabytes to exabytes. Despite

[*]École Polytechnique Fédérale de Lausanne (EPFL)—Laboratory for Information and Inference Systems (LIONS)
[†]University of Colorado—Applied Math
[‡]University of British Columbia—Laboratory for Computational Intelligence

the progress in parallel and distributed computing, the practical utility of classical algorithms like interior point methods may not go beyond discussing the theoretical tractability of the ensuing optimization problems [3].

In response, convex optimization is reinventing itself for Big Data where the data and parameter sizes of optimization problems are too large to process locally, and where even basic linear algebra routines like Cholesky decompositions and matrix-matrix or matrix-vector multiplications that algorithms take for granted are prohibitive. In stark contrast, convex algorithms also no longer need to seek high-accuracy solutions since Big Data models are necessarily simple or inexact [6].

The basics

We describe the fundamentals of Big Data optimization via the following composite formulation

$$F^* \stackrel{\text{def}}{=} \min_x \left\{ F(x) \stackrel{\text{def}}{=} f(x) + g(x) : x \in \mathbb{R}^p \right\}, \tag{1}$$

where f and g are convex functions. We review efficient numerical methods to obtain an optimal solution $x^\star$ of (1) as well as required assumptions on f and g. Such composite convex minimization problems naturally arise in signal processing when we estimate unknown parameters $x_0 \in \mathbb{R}^p$ from data $y \in \mathbb{R}^n$. In *maximum a posteriori* estimation, for instance, we regularize a smooth data likelihood function as captured by f typically with a non-smooth prior term g that encodes parameter complexity [1].

A basic understanding of Big Data optimization algorithms for (1) rests on three key pillars:

- **First-order methods** (Section I): First order methods obtain low- or medium-accuracy numerical solutions by using only first-order oracle information from the objective, such as gradient estimates. They can also handle the important *non-smooth* variants of (1) by making use of the *proximal mapping* principle. These methods feature nearly dimension-independent convergence rates, they are theoretically robust to the approximations of their oracles, and they typically rely on computational primitives that are ideal for distributed and parallel computation.

- **Randomization** (Section II): Randomization techniques particularly stand out among many other approximation techniques to enhance the scalability of first-order methods since we can control their expected behavior. Key ideas include random partial updates of optimization variables, replacing the deterministic gradient and proximal calculations with cheap statistical estimators, and speeding up basic linear algebra routines via randomization.

- **Parallel and distributed computation** (Section III): First-order methods naturally pro-

vide a flexible framework to distribute optimization tasks and perform computations in parallel. Surprisingly, we can further augment these methods with approximations for increasing levels of scalability, from idealized synchronous parallel algorithms with centralized communications to enormously-scalable *asynchronous* algorithms with *decentralized* communications.

The three concepts above complement each other to offer surprising scalability benefits for Big Data optimization. For instance, *randomized* first-order methods can exhibit significant acceleration over their deterministic counterparts since they can generate a good quality solution with high probability by inspecting only a negligibly small fraction of the data [3]. Moreover, since the computational primitives of such methods are inherently approximate, we can often obtain near linear speed-ups with a large number of processors [7, 8], which is a difficult feat when exact computation is required.

A motivation for first-order methods

A main source of Big Data problems is the ubiquitous linear observation model in many disciplines:

$$y = \Phi x_0 + z, \tag{2}$$

where x_0 is an unknown parameter, $\Phi \in \mathbb{R}^{n \times p}$ is a known matrix and $z \in \mathbb{R}^n$ encodes unknown perturbations or noise—modeled typically with zero-mean iid Gaussian entries with variance σ^2. Linear observations sometimes arise directly from the basic laws of physics as in magnetic resonance imaging and geophysics problems. Other times, (2) is an approximate model for more complicated nonlinear phenomena as in recommender systems and phase retrieval applications.

The linear model (2) along with low-dimensional signal models on x_0, such as sparsity, low total-variation, and low-rankness, has been an area of intense research activity in signal processing. Hence, it is instructive to first study the choice of convex formulations and their scalability implications here. The classical convex formulation in this setting has always been the least squares (LS) estimator

$$\widehat{x}_{\mathrm{LS}} = \operatorname*{argmin}_{x \in \mathbb{R}^p} \left\{ F(x) := \frac{1}{2} \|y - \Phi x\|_2^2 \right\}, \tag{3}$$

which can be efficiently solved by Krylov subspace methods using only matrix-vector multiplications. An important variant to (3) is the ℓ_1-regularized least absolute shrinkage and selection operator (LASSO), which features the composite form (1)

$$\widehat{x}_{\mathrm{LASSO}} = \operatorname*{argmin}_{x \in \mathbb{R}^p} \left\{ F(x) := \frac{1}{2} \|y - \Phi x\|_2^2 + \lambda \|x\|_1 \right\}, \tag{4}$$

where λ controls the strength of the regularization. Compared to the LS estimator, the LASSO

Dimension	Time		Error $\|\widehat{x} - x_0\|^2/\sigma^2$		Iterations	
	SDPT3	TFOCS	SDPT3	TFOCS	SDPT3	TFOCS
128	0.3 s	0.3 s	1.2	1.2	10	94
512	2.2 s	0.3 s	2.3	2.3	11	121
1024	16.0 s	0.5 s	2.4	2.4	12	157
2048	145.0 s	0.7 s	2.8	2.8	12	234
4096	N/A	1.0 s	N/A	3.3	N/A	281
16384	N/A	2.9 s	N/A	3.7	N/A	527
131072	N/A	40.2 s	N/A	4.4	N/A	1265
1048576	N/A	838.5 s	N/A	5.1	N/A	3440

Table 1: A numerical comparison of the default first-order method implemented in TFOCS [9] versus the interior point method SDPT3 implemented in CVX [10] for the LASSO problem (4) with $\lambda = 2\sigma\sqrt{2\log p}$. In the linear observation model (2), the matrix Φ is a randomly sub-sampled DCT matrix with $n = p/2$, the signal x_0 has $s = p/25$ non-zero coefficients with norm $\|x_0\|_2^2 \approx s$, and the noise z has variance $\sigma^2 = 10^{-4}$.

estimator has the advantage of producing *sparse* solutions (i.e., $\widehat{x}_{\text{LASSO}}$ has mostly zero entries), but its numerical solution is essentially harder since the regularizing term is non-smooth.

It turns out that the ℓ_1-regularization in sparse signal recovery with the linear model (2) is indeed critical when we are data deficient (i.e., $n < p$). Otherwise, the LASSO formulation imparts only a denoising effect to the solution when $n \geq p$. Theoretical justifications of the LASSO (4) over the LS estimation (3) come from statistical analysis and the convex geometry of (4), and readily apply to many other low-dimensional signal models and their associated composite formulations [1–3].

Table 1 illustrates key hallmarks of the first-order methods with the LASSO problem against the classical interior point method: nearly dimension-independent convergence and the ability to exploit implicit linear operators (e.g., the DCT transform). In contrast, interior point methods require much larger space and have near cubic dimension dependence due to the application of dense matrix-matrix multiplications or Cholesky decompositions in finding the Newton-like search directions. Surprisingly, the LASSO formulation possesses additional structures that provably enhance the convergence of the first-order methods [1], making them competitive in accuracy even to the interior point method.

Figure 1 shows that we can scale radically better than even the conjugate gradients (CG) method for the LS formulation when $n \gg p$ if we exploit stochastic approximation within first-order methods. We take the simplest optimization method, namely gradient descent with fixed step-size, and replace its gradient calculations with their cheap statistical estimates (cf., Section II for the recipe). The resulting *stochastic* gradient algorithm already obtains a strong baseline performance with access to only a fraction of the rows of Φ while the conjugate gradient method requires many more full accesses.

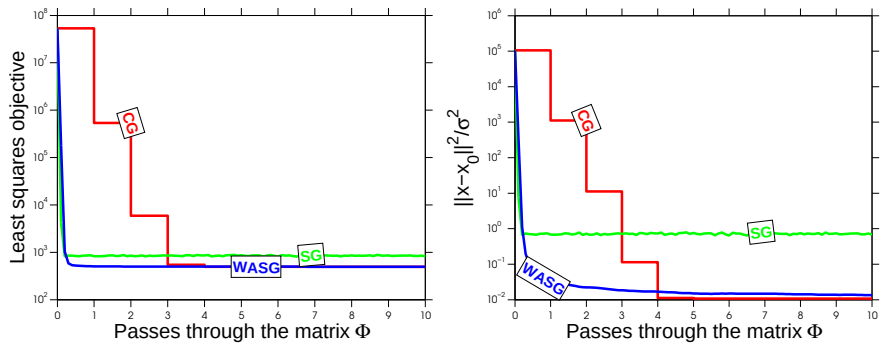

Figure 1: A numerical comparison of the CG method versus the stochastic gradient (SG) method and the weighted averaged SG (WASG) method for the LS problem (3), showing the objective (a) and the normalized estimation error (b). The matrix Φ has standard normal entries with dimensions $n = 10^5$ and $p = 10^3$. The noise variance is $\sigma^2 = 10^{-2}$ whereas $\|x_0\|_2^2 \approx p$. At a fractional access to the matrix Φ, the stochastic methods obtain a good relative *accuracy on the signal estimate. Finally, the SG method has an optimization error due to our step-size choice; cf., Section II.B for an explanation.*

1 First-Order Methods for Smooth and Non-Smooth Convex Optimization

As the LASSO formulation highlights, non-smooth regularization can play an indispensable role in solution quality. By using the powerful *proximal gradient* framework, we will see that many of these non-smooth problems can be solved nearly as efficiently as their smooth counterparts, a point not well-understood until the mid 2000s. To this end, this section describes first-order methods within this context, emphasizing specific algorithms with global convergence guarantees. In the sequel, we will assume that the readers have some familiarity with basic notions of convexity and complexity.

1.1 Smooth objectives

We begin our exposition with an important special case of (1), where the objective F only consists of a differentiable convex function f. The elementary first-order technique for this case is the *gradient* method, which uses only the local gradient $\nabla f(x)$ and iteratively performs the following update:

$$x^{k+1} = x^k - \alpha_k \nabla f(x^k), \tag{5}$$

where k is the iteration count and α_k is an appropriate step-size that ensures convergence.

For smooth minimization, we can certainly use several other *faster* algorithms such as Newton-like methods. By faster, we mean that these methods require fewer iterations than the gradient method to reach a target accuracy: i.e., $F(x^k) - F^* \leq \varepsilon$. However, we do not focus on these useful methods since they either require additional information from the function F, more expensive

computations, or do not generalize easily to constrained and non-smooth problems.

Fortunately, the low per-iteration cost of the gradient method can more than make up for its drawbacks in iteration count. For instance, computing the gradient dominates the per-iteration cost of the method, which consists of matrix-vector multiplications with Φ and its adjoint Φ^T when applied to the LS problem (3). Hence, we can indeed perform *many* gradient iterations for the cost of a single iteration of more complicated methods, potentially taking a shorter time to reach the same level of accuracy ε.

Surprisingly, by making simple assumptions about f, we can rigorously analyze how many iterations the gradient method will in fact need to reach an ε-accurate solution. A common assumption that holds in many applications is that the gradient of f is Lipschitz continuous, meaning that

$$\forall x, y \in \mathbb{R}^p,\ \|\nabla f(x) - \nabla f(y)\|_2 \leq L\|x - y\|_2,$$

for some constant L. When f is twice-differentiable, a sufficient condition is the eigenvalues of its Hessian $\nabla^2 f(x)$ are bounded above by L. Hence, we can trivially estimate $L = \|\Phi\|_2^2$ for (3).

If we simply set the step-size $\alpha_k = 1/L$ or alternatively use a value that decreases f the most, then the iterates of the gradient method for any convex f with a Lipschitz-continuous gradient obey

$$f(x^k) - f^\star \leq \frac{2L}{k+4} d_0^2, \tag{6}$$

where $d_0 = \|x^0 - x^\star\|_2$ is the distance of the initial iterate x^0 to an optimal solution $x^\star$ [11, Cor. 2.1.2]. Hence, the gradient method needs $\mathcal{O}(1/\varepsilon)$-iterations for an ε-accurate solution in the worst-case.

Unfortunately, this convergence rate does not attain the known complexity lower-bound

$$f(x^k) - f^\star \geq \frac{3Ld_0^2}{32(k+1)^2},$$

which holds for all functions f with Lipschitz-continuous gradients. That is, in the worst case *any* iterative method based only on function and gradient evaluations cannot hope for a better accuracy than $\Omega(1/k^2)$ at iteration k for $k < p$ [11]. Amazingly, a minor modification by Nesterov achieves this *optimal* convergence by the simple step-size choice $\alpha_k = 1/L$ and an extra-momentum step with a parameter $\beta_k = \frac{k}{k+3}$ [11]:

Algorithm 1 Nesterov's accelerated gradient method for unconstrained minimization ($v^0 = x^0$) [11]

1: $x^{k+1} = v^k - \alpha_k \nabla f(v^k)$
2: $v^{k+1} = x^{k+1} + \beta_k (x^{k+1} - x^k)$

The *accelerated* gradient method in Algorithm 1 achieves the best possible worst-case error rate, and hence, it is typically referred to as an *optimal* first-order method.

Many functions also feature additional structures useful for numerical optimization. Among them, *strong convexity* deserves special attention since this structure provably offers key benefits such as the existence of a unique minimizer and improved optimization efficiency. A function f is called strongly convex if the function $x \mapsto f(x) - \frac{\mu}{2}\|x\|_2^2$ is convex for some positive value μ. Perhaps not so obvious is the fact that even non-smooth functions can have strong convexity by this definition (i.e., $f(x) = \|x\|_1 + \mu/2\|x\|_2^2$).

Indeed, we can transform any convex problem into a strongly-convex problem by simply adding a squared ℓ_2-regularization term. For instance, when we have $n < p$ in (3), then the classic Tikhonov regularization results in a strongly convex objective with $\mu = \lambda$:

$$\widehat{x}_{\text{ridge}} = \underset{x \in \mathbb{R}^p}{\operatorname{argmin}} \left\{ F(x) := \frac{1}{2}\|y - \Phi x\|_2^2 + \frac{\lambda}{2}\|x\|_2^2 \right\},$$

The solution above is known as the ridge estimator and offers statistical benefits [1]. When f is twice-differentiable, a sufficient condition for strong convexity is that the eigenvalues of its Hessian $\nabla^2 f(x)$ are bounded below by μ for all x. For the LS problem (3), strong convexity simply requires Φ to have independent columns.

For strongly-convex problems with Lipschitz gradient, such as the ridge estimator, the gradient method geometrically converges to the unique minimizer when the step-size is chosen as $\alpha_k = 1/L$:

$$\left\| x^k - x^\star \right\|_2 \leq \left(1 - \frac{\mu}{L}\right)^k \left\| x^0 - x^\star \right\|_2. \tag{7}$$

This convergence improves slightly when we instead use $\alpha_k = 2/(\mu + L)$ [11]. Beside the obvious convergence rate difference, we highlight a subtlety between (6) and (7): guarantees due to the Lipschitz assumption such as (6) does not necessarily imply convergence in iterates x^k, while for stronlgy-convex functions we obtain guarantees on the convergence of both $f(x^k)$ and x^k.

It turns out that the accelerated-gradient method can also benefit from strong-convexity with an appropriate choice of the momentum term β_k. For example, if we set $\beta_k = (L - \mu)/(L + \mu)$, the accelerated gradient method obtains a near-optimal convergence rate given its assumptions [11, Thm. 2.2.3]. In contrast, the gradient method automatically exploits strong convexity without any

Algorithm	Convex	Strongly-Convex
[Proximal]-Gradient	$\mathcal{O}(Ld_0^2/\varepsilon)$	$\mathcal{O}\left(\frac{L}{\mu}\log(d_0^2/\varepsilon)\right)$
Accelerated-[Proximal]-Gradient	$\mathcal{O}(\sqrt{Ld_0^2/\varepsilon})$	$\mathcal{O}\left(\sqrt{\frac{L}{\mu}}\log(d_0^2/\varepsilon)\right)$

Table 2: Total number of iterations to reach ε-accurate solutions for first-order optimization methods. L and μ denote the Lipschitz and strong convexity constants and $d_0 = \|x^0 - x^\star\|_2$.

knowledge of μ.

Table 2 summarizes the number of iterations to reach an accuracy of ϵ for the different configurations discussed in this section. Note however that there are numerous practical enhancements, such as step-size selection rules for α_k and adaptive restart of the momentum parameter β_k [12] that add only a small computational cost and do not rely on knowledge of the Lipschitz constant L or the strong-convexity parameter μ. While such tricks-of-the-trade do not rigorously improve the worst-case convergence rates, they often lead to superior empirical convergence (cf., Figure 2) and similarly apply to their important *proximal* counterparts for solving (1) that we discuss next.

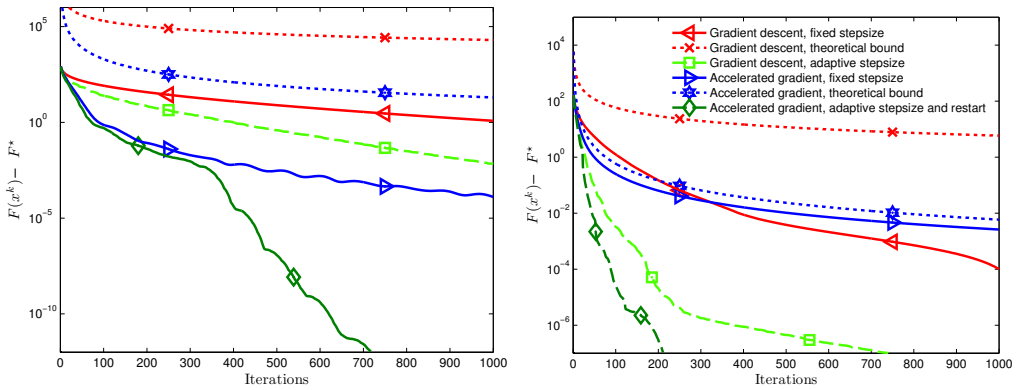

Figure 2: The performance of first-order methods can improve significantly with practical enhancements. We demonstrate how the objective $F(x^k)$ progresses as a function of iterations k for solving (Left) the LS formulation (3), and (Right) the LASSO formulation (4), both with $p = 5000$ and $n = 2500$, for four methods: (proximal)-gradient descent with fixed step-size $\alpha = 1/L$ and adaptive step-size, accelerated (proximal)-gradient descent with fixed step-size, and accelerated (proximal)-gradient descent with the adaptive step-size and restart scheme in TFOCS [9]. For the LS formulation, the basic methods behave qualitatively the same as their theoretical upper-bounds predict but dramatically improve with the enhancements. In the LASSO formulation, gradient descent automatically benefits from sparsity of the solution and actually outperforms the basic accelerated method in high-accuracy regime, but adding the adaptive restart enhancement allows the accelerated method to also benefit from sparsity.

Finally, the fast gradient algorithms described here also apply to non-smooth minimization problems using Nesterov's smoothing technique [11]. In addition, rather than assuming Lipschitz-

continuity of the gradient of the objective function, recent work has considered efficient gradient methods for smooth self-concordant functions, which naturally emerge in Poisson imaging, graph learning, and quantum tomography problems [13].

1.2 Composite objectives

We now consider the canonical composite problem (1), where the objective F consists of a differentiable convex function f and a non-smooth convex function g as in (4).

In general, the non-differentiability of g seems to substantially reduce the efficiency of first-order methods. This was indeed the conventional wisdom since generic non-smooth optimization methods, such as subgradient and bundle methods, require $\mathcal{O}(1/\varepsilon^2)$ iterations to reach ε-accurate solutions [11]. While strong convexity helps to improve this rate to $\mathcal{O}(1/\varepsilon)$, the resulting rates are slower than first-order methods for smooth objective.

Fortunately, composite objectives are far from generic non-smooth convex optimization problems. The *proximal-gradient* methods specifically take advantage of the composite structure in order to retain the same convergence rates of the gradient method for the smooth problem classes in Table 2 [14]. It becomes apparent that these algorithms are in fact natural extensions of the gradient method when we view the gradient method's iterations (5) as an optimization problem:

$$x^{k+1} = \operatorname*{argmin}_{y\in\mathbb{R}^p} \left\{ f(x^k) + \nabla f(x^k)^T (y - x^k) + \frac{1}{2\alpha_k} \left\| y - x^k \right\|^2 \right\}, \tag{8}$$

which is based on a simple local quadratic approximation of f. Note that when $\alpha_k \leq 1/L$, the objective function above is a quadratic upper bound on f. Proximal-gradient methods use the same approximation of f, but simply include the non-smooth term g in an explicit fashion:

$$x^{k+1} = \operatorname*{argmin}_{y\in\mathbb{R}^p} \left\{ f(x^k) + \nabla f(x^k)^T (y - x^k) + \frac{1}{2\alpha_k} \left\| y - x^k \right\|^2 + g(y) \right\}. \tag{9}$$

For $\alpha_k \leq 1/L$, the objective is an upper bound on F in (1).

The optimization problem (9) is the update rule of the proximal-gradient method:

$$x^{k+1} = \operatorname{prox}_{\alpha_k g}(x^k - \alpha_k \nabla f(x^k)),$$

where the *proximal map* or *proximal operator* is defined as

$$\operatorname{prox}_g(y) \stackrel{\text{def}}{=} \operatorname*{argmin}_x \left\{ g(x) + \frac{1}{2}\|x - y\|_2^2 \right\}. \tag{10}$$

The *accelerated* proximal-gradient method is defined analogously:

Algorithm 2 Accelerated proximal gradient method to solve (1) [11, 15]. Set $v^0 = x^0$.

1: $x^{k+1} = \text{prox}_{\alpha_k g}\left(v^k - \alpha_k \nabla f(v^k)\right)$
2: $v^{k+1} = x^{k+1} + \beta_k(x^{k+1} - x^k)$

An interesting special case of the proximal-gradient algorithm arises if we consider the indicator function on a convex set $\mathcal{C}$, which is an elegant way of incorporating constraints into (1)

$$g(x) = \begin{cases} 0 & x \in \mathcal{C} \\ \infty & x \notin \mathcal{C} \end{cases}.$$

Then, the proximal-gradient method yields the classic projected-gradient method for constrained optimization.

These method's fast convergence rates can also be preserved under approximate proximal maps [14]. Proximal operators offer a flexible computational framework to incorporate a rich set of signal priors in optimization. For instance, we can often represent a signal x_0 as a linear combination of atoms $a \in \mathcal{A}$ from some *atomic set* $\mathcal{A} \subseteq \mathbb{R}^p$ as $x_0 = \sum_{a \in \mathcal{A}} c_a a$, where c_a are the representation coefficients. Examples of atomic sets include structured sparse vectors, sign-vectors, low-rank matrices, and many more. The geometry of these sets can facilitate perfect recovery even from underdetermined cases of the linear observations (2) with sharp sample complexity characterizations [2].

To promote the structure of the set $\mathcal{A}$ in convex optimization, we can readily exploit its gauge function: $g_{\mathcal{A}}(x) \stackrel{\text{def}}{=} \inf\{\rho > 0 \mid x \in \rho \cdot \overline{\text{conv}}(\mathcal{A})\}$, where $\overline{\text{conv}}(\mathcal{A})$ is the convex hull of the set $\mathcal{A}$. The corresponding proximal operator of the gauge function has the following form

$$\text{prox}_{\gamma g_{\mathcal{A}}}(u) = u - \underset{v \in \mathbb{R}^d}{\text{argmin}} \left\{ \|u - v\|_2^2 : \langle a, v \rangle \leq \gamma, \forall a \in \mathcal{A} \right\}, \tag{11}$$

which involves a quadratic program in general but can be explicitly calculated in many cases [2]. Intriguingly, (11) also has mathematical connections to *discrete* submodular minimization.

By and large, whenever the computation of the proximal map is efficient, so are the proximal-gradient algorithms. For instance, when $g(x) = \lambda\|x\|_1$ as in the LASSO formulation (4), the proximal operator is the efficient soft thresholding operator. Against intuition, a set with an infinite number of atoms can admit an efficient proximal map, such as the set of rank-1 matrices with unit Frobenius norm whose proximal map is given by singular value thresholding. On the other hand, a set with a finite number of atoms need not, such as rank-1 matrices with ± 1 entries whose proximal operator is intractable. Numerous other examples exist [2, 4].

When g represents the indicator function of a compact set, the Frank-Wolfe method solves (9) without the quadratic term and can achieve an $O(1/\epsilon)$ convergence rate in the convex case [16].

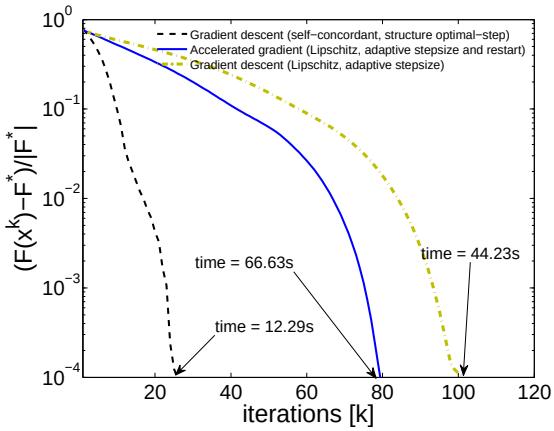

Figure 3: Choosing the correct smoothness structure on f in composite minimization is key to the numerical efficiency of the first-order methods. The convergence plot here simply demonstrates this with a composite objective, called the heteroskedastic LASSO (hLASSO) from [13] where $n = 1.5 \times 10^4$ and $p = 5 \times 10^4$. hLASSO features a self-concordant but not Lipschitz gradient smooth part f, and obtains sparse solutions while simultaneously estimating the unknown noise variance σ^2 in the linear model (2). The simplest first-order method, when matched to correct self-concordant smoothness structure, can calculate its step-sizes optimally, and hence, significantly outperforms the standard first-order methods based on the Lipschitz gradient assumption even with the enhancements we discussed. Surprisingly, the accelerated gradient method takes longer than the gradient method to reach the same accuracy since it relies heavily on the Lispchitz gradient assumption in its momentum steps, which lead to costlier step-size adaptation.

This linear sub-problem may be easier to solve, and the method gives sparse iterations since each method only modifies a single element of the atomic set.

Finally, proximal-gradient methods that optimally exploit the self-concordance properties of f are also explored in [13] (c.f., Figure 3 for an example). Interestingly, many self-concordant functions themselves have tractable proximity operators as well—a fact that proves useful next.

1.3 Proximal objectives

For many applications, the first-order methods we have covered so far are not directly applicable. As a result, we will find it useful here to view the composite form (1) in the following guise

$$\min_{x,z\in\mathbb{R}^p} \left\{ F(x,z) \stackrel{\text{def}}{=} h(x) + g(z) : \Phi z = x \right\}, \tag{12}$$

and only posit that the proximity operators of h and g are both efficient.

This seemingly innocuous reformulation can simultaneously enhance our modeling and computational capabilities. First, (12) can address non-smooth and non-Lipschitz objective functions that commonly occur in many applications [13, 17], such as robust principal component analysis (RPCA), graph learning, and Poisson imaging, *in addition* to the composite objectives we have

covered so far. Second, we can apply a simple algorithm, called the alternating direction method of multipliers (ADMM) for its solutions, which leverages powerful augmented Lagrangian and dual decomposition techniques [4, 18]:

Algorithm 3 ADMM to solve (12); $\gamma > 0, z^0 = u^0 = 0$

1: $x^{k+1} = \text{argmin}_x \ \gamma h(x) + \frac{1}{2}\|x - \Phi z^k + u^k\|_2^2 = \text{prox}_{\gamma h}(\Phi z^k - u^k)$
2: $z^{k+1} = \text{argmin}_z \ \gamma g(z) + \frac{1}{2}\|x^{k+1} - \Phi z + u^k\|_2^2$
3: $u^{k+1} = u^k + x^{k+1} - \Phi z^{k+1}$

Algorithm (3) is well-suited for distributed optimization and turns out to be equivalent or closely related to many other algorithms, such as Douglas-Rachford splitting and Bregman iterative algorithms [18]. ADMM requires a penalty parameter γ as input and produces a sequence of iterates that approach feasibility and produce the optimal objective value in the limit. An overview of ADMM, its convergence, enhancements, parameter selection, and stopping criteria can be found here [18].

We highlight two caveats for ADMM. First, we have to numerically solve step 2 in Algorithm 3 in general except when $\Phi^T\Phi$ is efficiently diagonalizable. Fortunately, many notable applications support these features, such as matrix completion where Φ models sub-sampled matrix entries, image deblurring where Φ is a convolution operator, and total variation regularization where Φ is a differential operator with periodic boundary conditions. Secondly, the *naïve* extension of ADMM to problems with more than two objective terms no longer has convergence guarantees.

Algorithm 4 Primal-Dual Hybrid Gradient algorithm to solve (12); $\gamma > 0$ and $\tau \leq 1/\|\Phi\|^2$

1: $x^{k+1} = \text{prox}_{\gamma h}(\Phi z^k - u^k)$
2: $z^{k+1} = \text{prox}_{\gamma\tau g}(z^k + \tau\Phi^T(x^{k+1} - \Phi z^k + u^k))$
3: $u^{k+1} = u^k + x^{k+1} - \Phi z^{k+1}$

Several solutions address the two drawbacks above. For the former, we can update z^{k+1} inexactly by using a single step of the proximal gradient method, which leads to the method shown in Algorithm 4 which was motivated in [19] as a preconditioned variant of ADMM and then analyzed in [20] in a more general framework. Interestingly, when $h(x)$ has a difficult proximal operator in Algorithm 3 but also has a Lipschitz gradient, we can replace $h(x)$ in Step 1 with its quadratic surrogate as in (8) to obtain the *linearized* ADMM [21]. Surprisingly, these inexact update-steps can be as fast to converge as the full ADMM in certain applications [19]. We refer the readers to [18, 20, 21] for the parameter selection of these variants as well as their convergence and generalizations.

For the issue regarding objectives with more than two terms, we can use dual decomposition

techniques to treat the multiple terms in the objective of (12) as separate problems and simultaneously solve them *in parallel*. We defer this to Section III and Algorithm 8.

2 Big Data scaling via randomization

In theory, first-order methods are well-positioned to address very large-scale problems. *In practice*, however, the exact numerical computations demanded by their iterations can make even these simple methods infeasible as the problem dimensions grows. Fortunately, it turns out that first-order methods are quite robust to using approximations of their optimization primitives, such as gradient and proximal calculations [14]. This section describes emerging randomized approximations that increase the reach of first-order methods to extraordinary scales.

To deliver an example of the key ideas, we will focus only on the smooth and strongly convex F as objectives and point out extensions when possible. Many notable Big Data problems indeed satisfy this assumption. For instance, Google's PageRank problem measures the importance of nodes in a given graph via its incidence matrix $M \in \mathbb{R}^{p\times p}$ and p is on the order of *tens of billions*. Assuming that more important nodes have more connections, the problem in fact aims to find the top singular vector of the stochastic matrix $\Phi = M\mathrm{diag}(M^T\mathbf{1}_p)^{-1}$, where $\mathbf{1}_p \in \mathbb{R}^p$ is the vector of all 1's.

The PageRank algorithm simply solves this basic linear algebra problem (i.e., find $x^* \geq 0$ such that $\Phi x^* = x^*$ and $\mathbf{1}_p^T x^* = 1$) with the *power method*. However, we can well-approximate this goal using a least squares problem when we relax the constraints with a penalty parameter $\gamma > 0$ [22]:

$$\min_{x\in\mathbb{R}^p} \left\{ F(x) \stackrel{\text{def}}{=} \frac{1}{2}\|x - \Phi x\|_2^2 + \frac{\gamma}{2}\left(\mathbf{1}_p^T x - 1\right)^2 \right\}, \tag{13}$$

Note that we can work with a constrained version of this problem that includes positivity, but since the PageRank formulation itself is not exact model of reality, the simpler problem can be preferable for obvious computational reasons. Clearly, we would like to minimize the number of operations involving the matrix Φ in any solution method.

2.1 Coordinate descent methods

Calculating the full gradient for the PageRank problem formulation requires a matrix-vector operation at each iteration. A cheaper vector-only operation would be to pick a coordinate i of x and only modify the corresponding variable x_i to improve the objective function. This idea captures the essence of *coordinate descent methods*, which have a long history in optimization [23] and are related to classic methods like the Gauss-Seidel cyclic reduction strategy for solving linear systems.

The general form of coordinate descent methods is illustrated in Algorithm 5, where e_i is the ith canonical coordinate vector and $\nabla_i F(\cdot)$ is the ith coordinate of the gradient.

Algorithm 5 Coordinate descent to minimize F over $\mathbb{R}^p$

1: Choose an index $i_k \in \{1, 2, \ldots, p\}$ *(see the main text for possible selection schemes)*

2: $x^{k+1} = x^k - \alpha \nabla_{i_k} F(x^k) e_{i_k}$

The key design consideration across all coordinate descent methods is the choice of the coordinate i at each iteration. A simple strategy amenable to analysis is to greedily pick the coordinate with the largest *directional derivative* $\nabla_i F$. This selection with $\alpha = 1/L_{\max}$ or optimizing the variable exactly leads to a convergence rate of

$$F(x^k) - F(x^*) \leq \left(1 - \frac{\mu}{pL_{\max}}\right)^k (F(x^0) - F(x^\star)), \tag{14}$$

where $L_{\max} \stackrel{\text{def}}{=} \max_i L_i$ is the maximum across the Lipschitz constants of $\nabla_i F(x)$ [22]. This configuration indeed seeks the best reduction in the objective per iteration we can hope for under this setting.

The example above underlines the fundamental difficulty in coordinate descent methods. Finding the best coordinate to update, the maximum of the gradient element's magnitudes, can require a computational effort as high as the gradient calculation itself. However, the incurred cost is not justified since the method's convergence is provably slower than the gradient method due to the basic relationship $L_i \leq L \leq pL_i$. An alternative proposal is to cycle through all coordinates sequentially. This is the cheapest coordinate selection strategy we can hope for but it results in a substantially slower convergence rate.

Surprisingly, randomization of the coordinate choice can achieve the best of both worlds. Suppose we choose the coordinate i uniformly at random among the set $\{1, 2, \ldots, p\}$. This selection can be done with a cost independent of p, but surprisingly nevertheless achieves the same convergence rate (14) in expectation [22]. The randomized algorithm's variance around its expected performance is well-controlled.

We also highlight two salient features of coordinate descent methods. First, they are perhaps most useful for objectives of the form $F(Ax)$ with $A \in \mathbb{R}^{n \times p}$, where evaluating the (not necessarily smooth) F costs $\mathcal{O}(n)$. By tracking the product Ax^k with incremental updates, we can then perform coordinate descent updates in linear time. Second, if we importance sample the coordinates proportional to their Lipschitz constants L_i, then the convergence rate of the randomized method

improves to

$$F(x^k) - F(x^\star) \leq \left(1 - \frac{\mu}{pL_{\text{mean}}}\right)^k (F(x^0) - F(x^\star)), \tag{15}$$

where L_{mean} is the mean across the L_i. Hence, this non-uniform random sampling strategy improves the speed by only adding an $\mathcal{O}(\log(p))$ importance sampling cost to the algorithm [22].

Finally, accelerated and composite versions of coordinate descent methods have also recently been explored, although accelerated methods often do not preserve the cheap iteration cost of the non-accelerated versions [3]: cf., [22] for a numerical example of these methods on the PageRank problem (13).

2.2 Stochastic gradient methods

In contrast to randomized coordinate descent methods, which update a single coordinate at a time with its exact gradient, stochastic gradient methods update all coordinates simultaneously but use approximate gradients. They are best suited for minimizing decomposable objective functions F

$$\min_{x \in \mathbb{R}^p} \left\{ F(x) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{j=1}^{n} F_j(x) \right\}, \tag{16}$$

where each F_j measures the data misfit for a single data point. This includes models as simple as least squares and also more elaborate models like conditional random fields.

Algorithm 6 Stochastic gradient descent to minimize F over $\mathbb{R}^p$

1: Choose an index $j_k \in \{1, 2, \ldots, n\}$ *uniformly at random*

2: $x^{k+1} = x^k - \alpha_k \nabla F_{j_k}(x^k)$

Stochastic gradient iterations heavily rely on the decomposability of (16) as shown in Algorithm 6. Similar to the coordinate descent methods, the crucial design problem in the stochastic gradient methods is the selection of the data points j at each iteration. Analogously, we obtain better convergence rates by choosing the j uniformly at random rather than cycling through the data [24]. In contrast, per iteration cost of the algorithm now depends only on p but not n.

Interestingly, a random data point selection results in an unbiased gradient estimate when we view (16) as the empirical observation of a *expected risk function* that governs the optimization problem

$$\min_{x \in \mathbb{R}^p} \left\{ F(x) \stackrel{\text{def}}{=} \mathbb{E}_\xi[F_\xi(x)] \right\}, \tag{17}$$

where the expectation is taken over the sampling distribution for the indices ξ. Indeed, if we can sample from the true underlying distribution, then stochastic gradient methods directly optimize the

expected risk minimization problem and result in provable generalization capabilities in machine learning applications [6]. The unbiased gradient estimation idea enables the stochastic gradient descent method to handle convex objectives beyond the decomposable form (16) [3].

The general SG method has classically used a decreasing sequence of step-sizes $\{\alpha_k\}$. However, this unfortunately leads to the same slow $\mathcal{O}(1/\sqrt{\epsilon})$ and $\mathcal{O}(1/\epsilon)$ convergence rates of the sub-gradient method. But interestingly, if we still use a constant step-size at each iteration for the stochastic gradient method the algorithm is known to quickly reduce the initial error, even if it has a non-vanishing optimization error [24]. We have observed this for the stochastic gradient descent example in Figure 1. Indeed, while stochastic gradient descent methods have historically been notoriously hard to tune, recent results show that using large step sizes and weighted averaging of the iterates (cf., Figure 1) allows us to achieve optimal convergence rates while being robust to the setting of the step size and the modeling assumptions [24, 25]. For example, recent work has shown [25] that an averaged stochastic gradient iteration with a constant step-size achieves an $\mathcal{O}(1/\varepsilon)$ convergence rate even without strong convexity under joint self-concordance-like and Lipschitz gradient assumptions. Another interesting recent development has been stochastic algorithms that achieve linear convergence rates for strongly-convex problems of the form (16) in the special case where the data size is finite [26].

2.3 Randomized linear algebra

For Big Data problems, basic linear algebra operations, such as matrix decompositions (e.g., eigenvalue, singular value, and Cholesky) and matrix-matrix multiplications can be major computational bottlenecks due to their superlinear dependence on dimensions. However, when the relevant matrix objects have low-rank representations (i.e., $M = LR^T$ with $L \in \mathbb{R}^{p\times r}$ and $L \in \mathbb{R}^{p\times r}$ where $r \ll p$), the efficiency of these methods uniformly improves. For instance, the corresponding singular value decomposition (SVD) of M would only cost $\mathcal{O}(pr^2 + r^3)$ flops.

The idea behind randomized linear algebra methods is either to approximate $M \approx Q(Q^T M)$ with $Q \in \mathbb{R}^{p\times r}$, or to construct a low-rank representation by column or row subset selection in order to speed up computation. And indeed, doing this in a *randomized* fashion gives us control over the distribution of the errors [27, 28]. This idea generalizes to matrices of any dimensions and has the added benefit of exploiting mature computational routines in nearly all programming languages. Hence, they immediately lend themselves well to modern distributed architectures.

We describe three impacts of randomizing linear algebra routines in optimization here. First, we can accelerate computation of the proximity operators of functions that depend on spectral values of a matrix. For instance, the proximity operator of the nuclear norm, used in matrix completion and RPCA problems, requires a partial SVD. This is traditionally done with the Lanczos algorithm

which does not parallelize easily due to synchronization and re-orthogonalization issues. However, with the randomized approach, the expected error in the computation is bounded and can be used to maintain rigorous guarantees for the convergence of the whole algorithm [29].

Secondly, the idea also works in obtaining unbiased gradient estimates for matrix objects, when randomization is chosen appropriately, and hence applies to virtually all stochastic gradient algorithms. Finally, the randomized approach can be used to *sketch* objective functions, i.e., to approximate them in order to obtain much cheaper iterations with exact first-order methods while retaining accuracy guarantees for the true objective [28].

Algorithm 7 Randomized low-rank approximation

Require: $M \in \mathbb{R}^{p\times p}$, integer r

Draw $\Omega \in \mathbb{R}^{p\times r}$ iid $\mathcal{N}(0,1)$

$W = M\Omega$ // Matrix multiply, cost is $\mathcal{O}(p^2 r)$

$QR = W$ // QR algorithm, e.g., Gram-Schmidt, cost is $\mathcal{O}(pr^2)$

$U = M^T Q$ // Matrix multiply, cost is $\mathcal{O}(p^2 r)$

return $\widehat{M}_{(r)} = QU^T$ // Rank r

Algorithm 7 is an example of a randomized low-rank approximation, which is simply a single step of the classical QR iteration, using a random initial value. Surprisingly, the error in approximating M is nearly as good as the *best* rank-r approximation, where $\ell = r + \rho$ and ρ is small. Specifically, for $r \geq 2, \rho \geq 2$ and $\ell \leq p$, [27] provides the bound

$$\mathbb{E}\,\|\widehat{M}_{(\ell)} - M\|_F \leq \sqrt{1 + \frac{r}{\rho - 1}}\|M - M_{(r)}\|_F$$

where the expectation is taken with respect to the randomization, $M_{(r)}$ is the best rank-r approximation of M, which only keeps the first r terms in the SVD and sets the rest to zero; furthermore, [27] shows a deviation bound showing that the error concentrates tightly around the expectation. Thus, the approximation can be very accurate if the spectrum of the matrix decays to zero rapidly. For additional randomized linear algebra schemes and their corresponding guarantees, including using a power iteration to improve on this bound, we refer the readers to [27].

Figure (4) illustrates the numerical benefits of such randomization over the classical Lanczos method. Since the randomized routine can perform all the multiplications in blocks, it benefits significantly from parallelization.

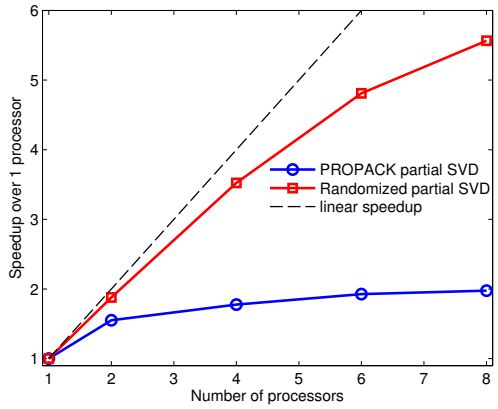

Figure 4: Computing the top 5 singular vectors of a 10^9 entry matrix using varying number of computer cores. The matrix is a dense 61440×17784 matrix (8.1 GB RAM) generated from video sequences from `http://perception.i2r.a-star.edu.sg/bk_model/bk_index.html`*. Such partial SVDs are used in the proximity operator for the nuclear norm term that arises in robust PCA formulations of video background subtraction. The randomized factorization happens to be faster than the Lanczos-based SVD from the PROPACK software even with a single core, but more importantly, the randomized method scales better as the parallelism increases. The accuracies of the two methods are indistinguishable.*

3 The role of parallel and distributed computation

Thanks to Moore's law of scaling silicon density, raw computational throughput and storage capacity have increased at exponential rates up until the mid 2000's, thereby giving convex optimization algorithms commensurate performance boosts. However, while Moore's law is expected to continue for years to come, transistor *efficiencies* have plateaued. As dictated by Dennard's law, scaling silicon density now results in unprecedented levels of power consumption. To handle the massive computational and storage resources demanded by Big Data at reasonable power costs, we must hence increasingly rely on parallel and distributed computation.

While first-order methods seem ideally suited for near-optimal performance speed-ups, two issues block us when using distributed and heterogeneous hardware:

- **Communication**: Uneven or faulty communication links between computers and within local memory hierarchy can significantly reduce the overall numerical efficiency of first-order methods. Two approaches broadly address such drawbacks. First, we can specifically design algorithms that minimize communication. Second, we can eliminate a master vector x^k and instead work with a local copy in each machine that each lead to a consensus $x^\star$ at convergence.

- **Synchronization**: To *exactly* perform the computations in a distributed fashion, first-order methods must coordinate the activities of different computers whose numerical primitives depend on the same vector x^k at each iteration. However, this procedure slows down even when a single machine takes much longer than the others. To alleviate this quintessential

synchronization problem, *asynchronous* algorithms allow updates using outdated versions of their parameters.

In this section we describe several key developments related to first-order methods within this context. Due to lack of space, we will gloss over many important issues that impact the practical performance of these methods, such as latency and multi-hop communication schemes.

3.1 Embarrassingly parallel first-order methods

First-order methods can significantly benefit from parallel computing. These computing systems are typified by uniform processing nodes that are in close proximity and have reliable communications. Indeed, the expression *embarrassingly parallel* refers to an ideal scenario for parallelization where we split the job into independent calculations that can be simultaneously performed in a predictable fashion.

In parallel computing, the formulation of the convex problem makes a great deal of difference. An important embarrassingly parallel example is the computation of the gradient vector when the objective naturally decomposes as in (16). Here, we can process each F_i with one of m computers using only $\mathcal{O}(n/m)$ local computation. Each machine also stores data locally with the corresponding $\mathcal{O}(n/m)$-data samples since each F_i directly corresponds to a data point. Each processor then communicates with the central location to form the final gradient and achieve the ideal linear speed-up.

Algorithm 8 Decomposition algorithm (aka, consensus ADMM) [30] to solve (18); $\gamma > 0$, $x_i^0 = 0$ for $i = 1, \ldots, n$.

1: $z^{k+1} = \frac{1}{n}\sum_{i=1}^{n} \operatorname{prox}_{\gamma F_i}(x_{(i)}^k)$
2: **for** $i = 1$ **to** n **do**
3: $\quad x_{(i)}^{k+1} = 2z^{k+1} - z^k + x_{(i)}^k - \operatorname{prox}_{\gamma F_i}(x_{(i)}^k)$
4: **end for**

Beyond parallelizing the basic gradient method for smooth problems, an embarrassingly parallel *distribute and gather* framework for non-smooth problems results from an artificial reformulation of (16) so that we can apply decomposition techniques, such as Algorithm 8:

$$\min_{x, x_{(i)}: i=1,\ldots,n} \left\{ \frac{1}{n} \sum_{i=1}^{n} F_i(x_{(i)}) : x_{(i)} = x, i = 1, \ldots, n \right\}. \tag{18}$$

Indeed, the decomposition idea above forms the basis of the massively parallel consensus ADMM algorithm, which provides an extremely scalable optimization framework for $n > 2$. See [18, 21, 30] for convergence analysis and further variants that include additional linear operators.

Fortunately, we have access to many computer programming models to put these ideas immediately into action. Software frameworks, such as MapReduce, Hadoop, Spark, Mahout, MADlib, SystemML, and BigInsights, and corresponding high-level languages such as Pig, Hive and Jaql, can govern the various optimization tasks in parallel while managing all communications and data transfers within the computing system, and seamlessly provide for redundancy and fault tolerance in communications.

3.2 First-order methods with reduced or decentralized communications

In large systems, communicating the gradient or its elements to a central location may create a communication bottleneck. In this setting, coordinate descent methods provide a principled approach to reduce communications. There is indeed substantial work on developing parallel versions of these methods, dating back to work on the Jacobi algorithm for solving linear systems. The basic idea is simply to apply several coordinate descent updates at the same time in parallel. The advantage of this strategy in terms of communication is that each processor only needs to communicate a single coordinate update, while it only needs to receive the updates from the coordinates that have changed.

When the objective is decomposable, this is simply an embarrassingly parallel version of the serial algorithm. Furthermore, classical work shows that this strategy is convergent, although it may require a smaller step size than the serial variant. However, it does not necessarily lead to a speed increase for non-separable functions. Recent work has sought to precisely characterize the conditions under which parallel coordinate descent methods still obtain a large speed-up [8].

Surprisingly, we can also decentralize the communication requirements of gradient methods for decomposable objectives with only minor modifications [31]. The resulting algorithm performs a modified gradient update to the average of the parameter vectors only among the neighbors it communicates with. This strategy in fact achieves similar convergence rates to the gradient method with central communications, where the rate degradation depends on the graph Laplacian of the underlying communication network.

3.3 Asynchronous first-order methods with decentralized communications

The gradient and the decomposition methods above still require a global synchronization to handle decomposable problems such as (16). For instance, the gradient algorithm computes the gradient *exactly* with respect to one (or more) examples at x^k and then synchronizes in sequence to update x^{k+1} in a standard implementation. In contrast, stochastic gradient algorithms that address (16) only use a crude approximation of the gradient. Hence, we expect these algorithm to be robust to

outdated information, which can happen in asynchronous settings.

A variety of recent works have shown that this is indeed the case. We highlight the work [7], which models a lock-free shared-memory system where stochastic gradient updates are independently performed by each processor. While the lock-free stochastic gradient still keeps a global vector x, processors are free to update it without any heed to other processors and continue their standard motions using the cached x. Under certain conditions this asynchronous procedure preserves the convergence of stochastic gradient methods, and results in substantial speed-ups when many cores are available. The same memory lock-free model also applies to stochastic parallel coordinate descent methods [8]. Finally, first-order algorithms with randomization can be effective even in asynchronous and decentralized settings with the possibility of communication failures [32].

4 Outlook for convex optimization

Big data problems necessitate a fundamental overhaul of how we design convex optimization algorithms, and suggest unconventional computational choices. To solve increasingly larger convex optimization problems with relatively modest growth in computational resources, this article makes it clear that we must identify key structure-dependent algorithmic approximation trade-offs.

Since the synchronization and communication constraints of the available hardware naturally dictates the choice of the algorithms, we expect that new approximation tools will continue to be discovered that ideally adapt convex algorithms to the heterogeneity of computational platforms. We also predict an increased utilization of composite models and the corresponding proximal mapping principles for non-smooth Big Data problems to cope with noise and other constraints. For example, the LASSO formulation in (4) has estimation guarantees that are quantitatively stronger than the guarantees of the LS estimator when the signal x_0 has at most k non-zero entries and Φ obeys certain assumptions [1]. That is to say, in order to get more out of the same data, we must use composite models. This also invites the question of whether we can use composite models to get the same information out but do it faster, an issue which has been discussed [5, 6] but not yet had an impact in practice.

Acknowledgements

Volkan Cevher's work is supported in part by the European Commission under grants MIRG-268398 and ERC Future Proof and by the Swiss Science Foundation under grants SNF 200021-132548, SNF 200021-146750, and SNF CRSII2-147633. During the preparation of the work, Stephen Becker was supported as a Goldstein Fellow at the IBM T. J. Watson research center, and Mark Schmidt was supported by the Natural Language Laboratory at Simon Fraser University.

References

[1] M. J. Wainwright, "Structured regularizers for high-dimensional problems: Statistical and computational issues," *Annual Review of Statistics and Its Application*, vol. 1, pp. 233–253, 2014.

[2] V. Chandrasekaran, B. Recht, P. A. Parrilo, and A. S. Willsky, "The convex geometry of linear inverse problems," *Found. Comput. Math.*, vol. 12, no. 6, pp. 805–849, 2012.

[3] Y. Nesterov and A. Nemirovski, "On first-order algorithms for l1/nuclear norm minimization," *Acta Numerica*, vol. 22, pp. 509–575, 2013.

[4] P. L. Combettes and J.-C. Pesquet, "Proximal splitting methods in signal processing," in *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, H. H. Bauschke, R. S. Burachik, P. L. Combettes, V. Elser, D. R. Luke, and H. Wolkowicz, Eds. New York: Springer-Verlag, 2011, pp. 185–212.

[5] V. Chandrasekaran and M. I. Jordan, "Computational and statistical tradeoffs via convex relaxation," *Proceedings of the National Academy of Sciences*, vol. 110, no. 13, pp. E1181–E1190, 2013.

[6] L. Bottou and O. Bousquet, "The tradeoffs of large scale learning." in *NIPS*, vol. 4, 2007, p. 2.

[7] F. Niu, B. Recht, C. Ré, and S. J. Wright, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent," *Advances in Neural Information Processing Systems*, vol. 24, pp. 693–701, 2011.

[8] P. Richtárik and M. Takáč, "Parallel coordinate descent methods for big data optimization," *arXiv preprint arXiv:1212.0873*, 2012.

[9] S. Becker, E. J. Candès, and M. Grant, "Templates for convex cone problems with applications to sparse signal recovery," *Math. Prog. Comp.*, vol. 3, no. 3, 2011, http://cvxr.com/tfocs.

[10] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 2.1," 2014, http://cvxr.com/cvx.

[11] Y. Nesterov, *Introductory lectures on convex optimization: a basic course*, ser. Applied Optimization. Kluwer Academic Publishers, 2004, vol. 87.

[12] B. O'Donoghue and E. J. Candès, "Adaptive restart for accelerated gradient schemes," *Fond. Comp. Math.*, 2013.

[13] Q. Tran-Dinh, A. Kyrillidis, and V. Cevher, "Composite self-concordant minimization," *arXiv preprint arXiv:1308.2867*, 2013.

[14] M. Schmidt, N. L. Roux, and F. Bach, "Convergence rates of inexact proximal-gradient methods for convex optimization," in *Advances in Neural Information Processing Systems (NIPS)*, 2011.

[15] M. Beck, A. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM J. Imaging Sci.*, vol. 2, no. 1, pp. 183–202, 2009.

[16] M. Jaggi, "Revisiting Frank-Wolfe: Projection-free sparse convex optimization," *International Conference on Machine Learning*, 2013.

[17] M. B. McCoy, V. Cevher, Q. T. Dinh, A. Asaei, and L. Baldassarre, "Convexity in source separation: Models, geometry, and algorithms," *arXiv preprint arXiv:1311.0258*, 2013.

[18] 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*, vol. 3, no. 1, pp. 1–122, 2011.

[19] E. Esser, X. Zhang, and T. Chan, "A general framework for a class of first order primal-dual algorithms for TV minimization," UCLA, Center for Applied Math, Tech. Rep. 09-67, 2009.

[20] A. Chambolle and T. Pock, "A first-order primal-dual algorithm for convex problems with applications to imaging," *J. Math. Imaging Vision*, vol. 40, no. 1, pp. 120–145, 2010.

[21] L. Condat, "A primal-dual splitting method for convex optimization involving Lipschitzian, proximable and linear composite terms," *J. Optim. Theory Appl.*, pp. 460–479, 2013.

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

[23] Z.-Q. Luo and P. Tseng, "On the convergence of the coordinate descent method for convex differentiable minimization," *Journal of Optimization Theory and Applications*, vol. 72, no. 1, pp. 7–35, 1992.

[24] A. Nedic and D. Bertsekas, "Convergence rate of incremental subgradient algorithms," in *Stochastic Optimization: Algorithms and Applications.* Kluwer Academic, 2000, pp. 263–304.

[25] F. Bach and E. Moulines, "Non-strongly-convex smooth stochastic approximation with convergence rate $O(1/n)$," *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[26] N. Le Roux, M. Schmidt, and F. Bach, "A stochastic gradient method with an exponential convergence rate for finite training sets," *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[27] N. Halko, P.-G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM review*, vol. 53, no. 2, pp. 217–288, 2011.

[28] M. Mahoney, "Randomized algorithms for matrices and data," *Found. Trends Machine Learning*, vol. 3, no. 2, 2011.

[29] S. Becker, V. Cevher, and A. Kyrillidis, "Randomized singular value projection," in *Sampling Theory and Approximation (SampTA)*, Bremen, Germany, 2013.

[30] P. L. Combettes and J.-C. Pesquet, "A proximal decomposition method for solving convex variational inverse problems," *Inverse Problems*, vol. 24, no. 6, p. 27, 2008.

[31] W. Shi, Q. Ling, G. Wu, and W. Yin, "Extra: An exact first-order algorithm for decentralized consensus optimization," *arXiv preprint arXiv:1404.6264*.

[32] A. Agarwal and J. C. Duchi, "Distributed delayed stochastic optimization," *Advances in Neural Information Processing Systems (NIPS)*, 2011.