Approximate Steepest Coordinate Descent

Sebastian U. Stich [1] **Anant Raj** [2] **Martin Jaggi** [1]

Abstract

We propose a new selection rule for the coordinate selection in coordinate descent methods for huge-scale optimization. The efficiency of this novel scheme is provably better than the efficiency of uniformly random selection, and can reach the efficiency of steepest coordinate descent (SCD), enabling an acceleration of a factor of up to n, the number of coordinates. In many practical applications, our scheme can be implemented at no extra cost and computational efficiency very close to the faster uniform selection. Numerical experiments with Lasso and Ridge regression show promising improvements, in line with our theoretical guarantees.

1. Introduction

Coordinate descent (CD) methods have attracted a substantial interest the optimization community in the last few years (Nesterov, 2012; Richtárik & Takáč, 2016). Due to their computational efficiency, scalability, as well as their ease of implementation, these methods are the state-of-the-art for a wide selection of machine learning and signal processing applications (Fu, 1998; Hsieh et al., 2008; Wright, 2015). This is also theoretically well justified: The complexity estimates for CD methods are in general better than the estimates for methods that compute the full gradient in one batch pass (Nesterov, 2012; Nesterov & Stich, 2017).

In many CD methods, the active coordinate is picked at random, according to a probability distribution. For smooth functions it is theoretically well understood how the sampling procedure is related to the efficiency of the scheme and which distributions give the best complexity estimates (Nesterov, 2012; Zhao & Zhang, 2015; Allen-Zhu et al., 2016; Qu & Richtárik, 2016; Nesterov & Stich, 2017). For nonsmooth and composite functions — that appear in many machine learning applications — the picture is less clear. For instance in (Shalev-Shwartz & Zhang, 2013; Friedman et al., 2007; 2010; Shalev-Shwartz & Tewari, 2011) uniform sampling (UCD) is used, whereas other papers propose adaptive sampling strategies that change over time (Papa et al., 2015; Csiba et al., 2015; Osokin et al., 2016; Perekrestenko et al., 2017).

A very simple deterministic strategy is to move along the direction corresponding to the component of the gradient with the maximal absolute value (steepest coordinate descent, SCD) (Boyd & Vandenberghe, 2004; Tseng & Yun, 2009). For smooth functions this strategy yields always better progress than UCD, and the speedup can reach a factor of the dimension (Nutini et al., 2015). However, SCD requires the computation of the whole gradient vector in each iteration which is prohibitive (except for special applications, cf. Dhillon et al. (2011); Shrivastava & Li (2014)).

In this paper we propose approximate steepest coordinate descent (ASCD), a novel scheme which combines the best parts of the aforementioned strategies: (i) ASCD maintains an approximation of the *full* gradient in each iteration and selects the active coordinate among the components of this vector that have large absolute values — similar to SCD; and (ii) in many situations the gradient approximation can be updated cheaply at no extra cost — similar to UCD. We show that regardless of the errors in the gradient approximation (even if they are infinite), ASCD performs always better than UCD.

Similar to the methods proposed in (Tseng & Yun, 2009) we also present variants of ASCD for composite problems. We confirm our theoretical findings by numerical experiments for Lasso and Ridge regression on a synthetic dataset as well as on the RCV1 (binary) dataset.

Structure of the Paper and Contributions. In Sec. 2 we review the existing theory for SCD and (i) extend it to the setting of smooth functions. We present (ii) a novel lower bound, showing that the complexity estimates for SCD and UCD can be equal in general. We (iii) introduce ASCD and the save selection rules for both smooth (Sec. 3) and to composite functions (Sec. 5). We prove that (iv) ASCD performs always better than UCD (Sec. 3) and (v) it can reach the performance of SCD (Sec. 6). In Sec. 4 we discuss important applications where the gradient estimate can efficiently be maintained. Our theory is supported by nu-

[1]EPFL [2]Max Planck Institute for Intelligent Systems. Correspondence to: Sebastian U. Stich <sebastian.stich@epfl.ch>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

merical evidence in Sec. 7, which reveals that (vi) ASCD performs extremely well on real data.

Notation. Define $[\boldsymbol{x}]_i := \langle \boldsymbol{x}, \boldsymbol{e}_i \rangle$ with $\boldsymbol{e}_i$ the standard unit vectors in $\mathbb{R}^n$. We abbreviate $\nabla_i f := [\nabla f]_i$. A convex function $f\colon \mathbb{R}^n \to \mathbb{R}$ with coordinate-wise L_i-Lipschitz continuous gradients[1] for constants $L_i > 0$, $i \in [n] := \{1, \dots, n\}$, satisfies by the standard reasoning

$$f(\boldsymbol{x} + \eta \boldsymbol{e}_i) \leq f(\boldsymbol{x}) + \eta \nabla_i f(\boldsymbol{x}) + \tfrac{L_i}{2} \eta^2 \tag{1}$$

for all $\boldsymbol{x} \in \mathbb{R}^n$ and $\eta \in \mathbb{R}$. A function is coordinate-wise L-smooth if $L_i \leq L$ for $i = 1, \dots, n$. For an optimization problem $\min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x})$ define $X^\star := \arg\min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x})$ and denote by $\boldsymbol{x}^\star \in \mathbb{R}^n$ an arbitrary element $\boldsymbol{x}^\star \in X^\star$.

2. Steepest Coordinate Descent

In this section we present SCD and discuss its theoretical properties. The functions of interest are composite convex functions $F\colon \mathbb{R}^n \to \mathbb{R}$ of the form

$$F(\boldsymbol{x}) := f(\boldsymbol{x}) + \Psi(\boldsymbol{x}) \tag{2}$$

where f is coordinate-wise L-smooth and Ψ convex and separable, that is that is $\Psi(\boldsymbol{x}) = \sum_{i=1}^n \Psi_i([\boldsymbol{x}]_i)$. In the first part of this section we focus on smooth problems, i.e. we assume that $\Psi \equiv 0$.

Coordinate descent methods with constant step size generate a sequence $\{\boldsymbol{x}_t\}_{t \geq 0}$ of iterates that satisfy the relation

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \tfrac{1}{L} \nabla_{i_t} f(\boldsymbol{x}) \boldsymbol{e}_{i_t} \,. \tag{3}$$

In UCD the active coordinate i_t is chosen uniformly at random from the set $[n]$, $i_t \in_{u.a.r.} [n]$. SCD chooses the coordinate according to the Gauss-Southwell (GS) rule:

$$i_t = \arg\max_{i \in [n]} \nabla_i \left| f(\boldsymbol{x}_t) \right| \,. \tag{4}$$

2.1. Convergence analysis

With the quadratic upper bound (1) one can easily get a lower bound on the one step progress

$$\mathbb{E}\left[f(x_t) - f(x_{t+1}) \mid x_t \right] \geq \mathbb{E}_{i_t}\left[\tfrac{1}{2L} \left| \nabla_{i_t} f(\boldsymbol{x}_t) \right|^2 \right] \,. \tag{5}$$

For UCD and SCD the expression on the right hand side evaluates to

$$\begin{aligned} \tau_{\mathrm{UCD}}(\boldsymbol{x}_t) &:= \tfrac{1}{2nL} \left\| \nabla f(\boldsymbol{x}_t) \right\|_2^2 \\ \tau_{\mathrm{SCD}}(\boldsymbol{x}_t) &:= \tfrac{1}{2L} \left\| \nabla f(\boldsymbol{x}_t) \right\|_\infty^2 \end{aligned} \tag{6}$$

With Cauchy-Schwarz we find

$$\tfrac{1}{n} \tau_{\mathrm{SCD}}(\boldsymbol{x}_t) \leq \tau_{\mathrm{UCD}}(\boldsymbol{x}_t) \leq \tau_{\mathrm{SCD}}(\boldsymbol{x}_t) \,. \tag{7}$$

Hence, the lower bound on the one step progress of SCD is always at least as large as the lower bound on the one step progress of UCD. Moreover, the one step progress could be even lager by a factor of n. However, it is very difficult to formally prove that this linear speed-up holds for more than one iteration, as the expressions in (7) depend on the (a priori unknown) sequence of iterates $\{\boldsymbol{x}_t\}_{t \geq 0}$.

Strongly Convex Objectives. Nutini et al. (2015) present an elegant solution of this problem for μ_2-strongly convex functions[2]. They propose to measure the strong convexity of the objective function in the 1-norm instead of the 2-norm. This gives rise to the lower bound

$$\tau_{\mathrm{SCD}}(\boldsymbol{x}_t) \geq \tfrac{\mu_1}{L} \left(f(\boldsymbol{x}_t) - f(\boldsymbol{x}^\star) \right) \,, \tag{8}$$

where μ_1 denotes the strong convexity parameter. By this, they get a uniform upper bound on the convergence that does not directly depend on local properties of the function, like for instance $\tau_{\mathrm{SCD}}(\boldsymbol{x}_t)$, but just on μ_1. It always holds $\mu_1 \leq \mu_2$, and for functions where both quantities are equal, SCD enjoys a linear speedup over UCD.

Smooth Objectives. When the objective function f is just smooth (but not necessarily strongly convex), then the analysis mentioned above is not applicable. We here extend the analysis from (Nutini et al., 2015) to smooth functions.

Theorem 2.1. *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be convex and coordinate-wise L-smooth. Then for the sequence $\{\boldsymbol{x}_t\}_{t \geq 0}$ generated by SCD it holds:*

$$f(\boldsymbol{x}_t) - f(\boldsymbol{x}^\star) \leq \frac{2LR_1^2}{t} \,, \tag{9}$$

for $R_1 := \max_{\boldsymbol{x}^\star \in X^\star} \left\{ \max_{\boldsymbol{x} \in \mathbb{R}^n} \left[\left\| \boldsymbol{x} - \boldsymbol{x}^\star \right\|_1 \mid f(\boldsymbol{x}) \leq f(\boldsymbol{x}_0) \right] \right\}$.

Proof. In the proof we first derive a lower bound on the one step progress (Lemma A.1), similar to the analysis in (Nesterov, 2012). The lower bound for the one step progress of SCD can in each iteration differ up to a factor of n from the analogous bound derived for UCD (similar as in (7)). All details are given in Section A.1 in the appendix. □

Note that the R_1 is essentially the diameter of the level set at $f(\boldsymbol{x}_0)$ measured in the 1-norm. In the complexity estimate of UCD, R_1^2 in (9) is replaced by nR_2^2, where R_2 is the diameter of the level at $f(\boldsymbol{x}_0)$ measured in the 2-norm (cf. Nesterov (2012); Wright (2015)). As in (7) we observe with Cauchy-Schwarz

$$\tfrac{1}{n} R_1^2 \leq R_2^2 \leq R_1^2 \,, \tag{10}$$

i.e. SCD can accelerate up to a factor of n over to UCD.

[1] $\left| \nabla_i f(\boldsymbol{x} + \eta \boldsymbol{e}_i) - \nabla_i f(\boldsymbol{x}) \right| \leq L_i \left| \eta \right| \,, \quad \forall \boldsymbol{x} \in \mathbb{R}^n, \eta \in \mathbb{R}$.

[2] A function is μ_p-strongly convex in the p-norm, $p \geq 1$, if $f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{y} - \boldsymbol{x} \rangle + \tfrac{\mu_p}{2} \left\| \boldsymbol{y} - \boldsymbol{x} \right\|_p^2, \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$.

2.2. Lower bounds

In the previous section we provided complexity estimates for the methods SCD and UCD and showed that SCD can converge up to a factor of the dimension n faster than UCD. In this section we show that this analysis is tight. In Theorem 2.2 below we give a function $q\colon \mathbb{R}^n \to \mathbb{R}$, for which the one step progress $\tau_{\mathrm{SCD}}(\boldsymbol{x}_t) \approx \tau_{\mathrm{UCD}}(\boldsymbol{x}_t)$ up to a constant factor, for all iterates $\{\boldsymbol{x}_t\}_{t\geq 0}$ generated by SCD.

By a simple technique we can also construct functions for which the speedup is exactly equal to an arbitrary factor $\lambda \in [1, n]$. For instance we can consider functions with a (separable) low dimensional structure. Fix integers s, n such that $\frac{n}{s} \approx \lambda$, define the function $f\colon \mathbb{R}^n \to \mathbb{R}$ as

$$f(\boldsymbol{x}) := q(\pi_s(\boldsymbol{x})) \tag{11}$$

where π_s denotes the projection to $\mathbb{R}^s$ (being the first s out of n coordinates) and $q\colon \mathbb{R}^s \to \mathbb{R}$ is the function from Theorem 2.2. Then

$$\tau_{\mathrm{SCD}}(\boldsymbol{x}_t) \approx \lambda \cdot \tau_{\mathrm{UCD}}(\boldsymbol{x}_t)\,, \tag{12}$$

for all iterates $\{\boldsymbol{x}_t\}_{t\geq 0}$ generated by SCD.

Theorem 2.2. *Consider the function $q(\boldsymbol{x}) = \frac{1}{2}\langle Q\boldsymbol{x}, \boldsymbol{x}\rangle$ for $Q := I_n - \frac{99}{100n} J_n$, where $J_n = 1_n 1_n^T$, $n > 2$. Then there exists $\boldsymbol{x}_0 \in R^n$ such that for the sequence $\{\boldsymbol{x}_t\}_{t\geq 0}$ generated by SCD it holds*

$$\|\nabla q(\boldsymbol{x}_t)\|_\infty^2 \leq \tfrac{4}{n} \|\nabla q(\boldsymbol{x}_t)\|_2^2\,. \tag{13}$$

Proof. In the appendix we discuss a family of functions defined by matrices $Q := (\alpha - 1)\frac{1}{n} J_n + I_n$ and define corresponding parameters $0 < c_\alpha < 1$ such that for $\boldsymbol{x}_0$ defined as $[\boldsymbol{x}_0]_i = c_\alpha^{i-1}$ for $i = 1, \ldots, n$, SCD cycles through the coordinates, that is, the sequence $\{\boldsymbol{x}_t\}_{t\geq 0}$ generated by SCD satisfies

$$[\boldsymbol{x}_t]_{1+(t-1 \mod n)} = c_\alpha^n \cdot [\boldsymbol{x}_{t-1}]_{1+(t-1 \mod n)}\,. \tag{14}$$

We verify that for this sequence property (13) holds. □

2.3. Composite Functions

The generalization of the GS rule (4) to composite problems (2) with nontrival Ψ is not straight forward. The 'steepest' direction is not always meaningful in this setting; consider for instance a constrained problem where this rule could yield no progress at all when stuck at the boundary.

Nutini et al. (2015) discuss several generalizations of the Gauss-Southwell rule for composite functions. The GS-s rule is defined to choose the coordinate with the most negative directional derivative (Wu & Lange, 2008). This rule is identical to (4) but requires the calculation of subgradients of Ψ_i. However, the length of a step could be arbitrarily small. In contrast, the GS-r rule was defined to pick the coordinate direction that yields the longest step (Tseng & Yun, 2009). The rule that enjoys the best theoretical properties (cf. Nutini et al. (2015)) is the GS-q rule, which is defined as to maximize the progress assuming a quadratic upper bound on f (Tseng & Yun, 2009). Consider the coordinate-wise models

$$V_i(\boldsymbol{x}, y, s) := sy + \tfrac{L}{2} y^2 + \Psi_i([\boldsymbol{x}]_i + y)\,, \tag{15}$$

for $i \in [n]$. The GS-q rule is formally defined as

$$i_{\mathrm{GS-q}} = \operatorname*{arg\,min}_{i\in[n]} \min_{y\in\mathbb{R}} V_i(\boldsymbol{x}, y, \nabla_i f(\boldsymbol{x}))\,. \tag{16}$$

2.4. The Complexity of the GS rule

So far we only studied the iteration complexity of SCD, but we have disregarded the fact that the computation of the GS rule (4) can be as expensive as the computation of the whole gradient. The application of coordinate descent methods is only justified if the complexity to compute one directional derivative is approximately n times cheaper than the computation of the full gradient vector (cf. Nesterov (2012)). By Theorem 2.2 this reasoning also applies to SCD. A class of function with this property is given by functions $F\colon \mathbb{R}^n \to \mathbb{R}$

$$F(\boldsymbol{x}) := f(A\boldsymbol{x}) + \sum_{i=1}^n \Psi_i([\boldsymbol{x}]_i) \tag{17}$$

where A is a $d \times n$ matrix, and where $f\colon \mathbb{R}^d \to \mathbb{R}$, and $\Psi_i\colon \mathbb{R}^n \to \mathbb{R}$ are convex and simple, that is the time complexity T for computing their gradients is linear: $T(\nabla_{\boldsymbol{y}} f(\boldsymbol{y}), \nabla_{\boldsymbol{x}} \Psi(\boldsymbol{x}) = O(d + n)$. This class of functions includes least squares, logistic regression, Lasso, and SVMs (when solved in dual form).

Assuming the matrix is dense, the complexity to compute the full gradient of F is $T(\nabla_{\boldsymbol{x}} F(\boldsymbol{x})) = O(dn)$. If the value $\boldsymbol{w} = A\boldsymbol{x}$ is already computed, one directional derivative can be computed in time $T(\nabla_i F(\boldsymbol{x})) = O(d)$. The recursive update of $\boldsymbol{w}$ after one step needs the addition of one column of matrix A with some factors and can be done in time $O(d)$. However, we note that recursively updating the full gradient vector takes time $O(dn)$ and consequently the computation of the GS rule *cannot* be done efficiently.

Nutini et al. (2015) consider sparse matrices, for which the computation of the Gauss-Southwell rule becomes traceable. In this paper, we propose an alternative approach. Instead of updating the exact gradient vector, we keep track of an approximation of the gradient vector and recursively update this approximation in time $O(n \log n)$. With these updates, the use of coordinate descent is still justified in case $d = \Omega(n)$.

Algorithm 1 Approximate SCD (ASCD)

Input: f, $\boldsymbol{x}_0$, T, δ-gradient oracle g, method $\mathcal{M}$
Initialize $[\tilde{\boldsymbol{g}}_0]_i = 0$, $[\boldsymbol{r}_0]_i = \infty$ for $i \in [n]$.
for $t = 0$ **to** T **do**
  For $i \in [n]$ define *compute u.-and l.-bounds*
  $[\boldsymbol{u}_t]_i := \max\{|[\tilde{\boldsymbol{g}}_t]_i - [\boldsymbol{r}_t]_i|, |[\tilde{\boldsymbol{g}}_t]_i + [\boldsymbol{r}_t]_i|\}$
  $[\boldsymbol{\ell}_t]_i := \min_{y\in\mathbb{R}}\{|y| \mid [\tilde{\boldsymbol{g}}_t]_i - [\boldsymbol{r}_t]_i \le y \le [\tilde{\boldsymbol{g}}_t]_i + [\boldsymbol{r}_t]_i\}$
  $\mathrm{av}(\mathcal{I}) := \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}}[\boldsymbol{\ell}_t]_i^2$ *compute active set*
  $\mathcal{I}_t := \arg\min_{\mathcal{I}} \left|\{\mathcal{I} \subseteq [n] \mid [\boldsymbol{u}_t]_i^2 < \mathrm{av}(\mathcal{I}), \forall i \notin \mathcal{I}\}\right|$
  Pick $i_t \in_{\text{u.a.r.}} \arg\max_{i\in\mathcal{I}_t}\{[\boldsymbol{\ell}]_i\}$ *active coordinate*
  $(\boldsymbol{x}_{t+1}, [\tilde{\boldsymbol{g}}_{t+1}]_{i_t}, [\boldsymbol{r}_{t+1}]_{i_t}) := \mathcal{M}(\boldsymbol{x}_t, \nabla_{i_t} f(\boldsymbol{x}_t))$
  $\gamma_t := [\boldsymbol{x}_{t+1}]_{i_t} - [\boldsymbol{x}_t]_{i_t}$ *update $\nabla f(\boldsymbol{x}_{t+1})$ estimate*
  Update $[\tilde{\boldsymbol{g}}_{t+1}]_j := [\tilde{\boldsymbol{g}}_t]_j + \gamma_t g_{i_t j}(\boldsymbol{x}_t)$, $j \neq i_t$
  Update $[\boldsymbol{r}_{t+1}]_j := [\boldsymbol{r}_t]_j + \gamma_t \delta_{i_t j}$, $j \neq i_t$
end for

3. Algorithm

Is it possible to get the significantly improved convergence speed from SCD, when one is only willing to pay the computational cost of only the much simpler UCD? In this section, we give a formal definition of our proposed approximate SCD method which we denote ASCD.

The core idea of the algorithm is the following: While performing coordinate updates, ideally we would like to efficiently track the evolution of *all* elements of the gradient, not only the one coordinate which is updated in the current step. The formal definition of the method is given in Algorithm 1 for smooth objective functions. In each iteration, only one coordinate is modified according to some arbitrary update rule $\mathcal{M}$. The coordinate update rule $\mathcal{M}$ provides two things: First the new iterate $\boldsymbol{x}_{t+1}$, and secondly also an estimate $\tilde{g}$ of the i_t-th entry of the gradient at the new iterate[3]. Formally,

$$(\boldsymbol{x}_{t+1}, \tilde{g}, r) := \mathcal{M}(\boldsymbol{x}_t, \nabla_{i_t} f(\boldsymbol{x}_t)) \tag{18}$$

such that the quality of the new gradient estimate $\tilde{g}$ satisfies

$$|\nabla_{i_t} f(\boldsymbol{x}_{t+1}) - \tilde{g}| \le r\,. \tag{19}$$

The non-active coordinates are updated with the help of gradient oracles with accuracy $\delta \ge 0$ (see next subsection for details). The scenario of exact updates of all gradient entries is obtained for accuracy parameters $\delta = r = 0$ and in this case ASCD is identical to SCD.

3.1. Safe bounds for gradient evolution

ASCD maintains lower and upper bounds for the absolute values of each component of the gradient ($[\boldsymbol{\ell}]_i \le |\nabla_i f(\boldsymbol{x})| \le [\boldsymbol{u}]_i$). These bounds allow to identify the coordinates on which the absolute values of the gradient are small (and hence cannot be the steepest one). More precisely, the algorithm maintains a set $\mathcal{I}_t$ of active coordinates (similar in spirit as in active set methods, see e.g. Kim & Park (2008); Wen et al. (2012)). A coordinate j is excluded from $\mathcal{I}_t$ if the estimated progress in this direction (cf. (5)) is lower than the average of the estimated progress along coordinate directions in $\mathcal{I}_t$, $[\boldsymbol{u}_t]_j^2 < \frac{1}{|\mathcal{I}_t|}\sum_{i\in\mathcal{I}_t}[\boldsymbol{\ell}_t]_i^2$. The active set $\mathcal{I}_t$ can be computed in $O(n \log n)$ time by sorting. All other operations take linear $O(n)$ time.

Gradient Oracle. The selection mechanism in ASCD crucially relies on the following definition of a δ-gradient oracle. While the update $\mathcal{M}$ delivers the estimated active entry of the new gradient, the additional gradient oracle is used to update all other coordinates $j \neq i_t$ of the gradient; as in the last two lines of Algorithm 1.

Definition 3.1 (δ-gradient oracle)**.** *For a function $f\colon \mathbb{R}^n \to \mathbb{R}$ and indices $i, j \in [n]$, a (i,j)-gradient oracle with error $\delta_{ij} \ge 0$ is a function $g_{ij}\colon \mathbb{R}^n \to \mathbb{R}$ satisfying $\forall \boldsymbol{x} \in \mathbb{R}^n, \forall \gamma \in \mathbb{R}$:*

$$|\nabla_j f(\boldsymbol{x} + \gamma \boldsymbol{e}_i) - \gamma g_{ij}(\boldsymbol{x})| \le |\gamma|\,\delta_{ij}\,. \tag{20}$$

We denote by a δ-gradient oracle a family $\{g_{ij}\}_{i,j\in[n]}$ of δ_{ij}-gradient oracles.

We discuss the availability of good gradient oracles for many problem classes in more detail in Section 4. For example for least squares problems and general linear models, a δ-gradient oracle is for instance given by a scalar product estimator as in (24) below. Note that ASCD can also handle very bad estimates, as long as the property (20) is satisfied (possibly even with accuracy $\delta_{ij} = \infty$).

Initialization. In ASCD the initial estimate $\tilde{\boldsymbol{g}}_0$ of the gradient is just arbitrarily set to $\mathbf{0}$, with uncertainty $\boldsymbol{r}_0 = \infty$. Hence in the worst case it takes $\Theta(n \log n)$ iterations until each coordinate gets picked at least once (cf. Dawkins (1991)) and until corresponding gradient estimates are set to a realistic value. If better estimates of the initial gradient are known, they can be used for the initialization as long as a strong error bound as in (19) is known as well. For instance the initialization can be done with $\nabla f(\boldsymbol{x}_0)$ if one is willing to compute this vector in one batch pass.

Convergence Rate Guarantee. We present our first main result showing that the performance of ASCD is provably between UCD and SCD. First observe that if in Algorithm 1 the gradient oracle is always exact, i.e. $\delta_{ij} \equiv 0$, and if $\tilde{\boldsymbol{g}}_0$ is initialized with $\nabla f(\boldsymbol{x}_0)$, then in each iteration $|\nabla_{i_t} f(\boldsymbol{x}_t)| = \|\nabla f(\boldsymbol{x}_t)\|_\infty$ and ASCD identical to SCD.

Lemma 3.1. *Let $i_{\max} := \arg\max_{i\in[n]} |\nabla_i f(\boldsymbol{x}_t)|$. Then $i_{\max} \in \mathcal{I}_t$, for $\mathcal{I}_t$ as in Algorithm 1.*

[3] For instance, for updates by exact coordinate optimization (line-search), we have $\tilde{g} = r = 0$.

Proof. This is immediate from the definitions of $\mathcal{I}_t$ and the upper and lower bounds. Suppose $i_{\max} \notin \mathcal{I}_t$, then there exists $j \neq i_{\max}$ such that $[\ell_t]_j > [u_t]_{i_{\max}}$, and consequently $|\nabla_j f(\boldsymbol{x}_t)| > |\nabla_{i_{\max}} f(\boldsymbol{x}_t)|$. □

Theorem 3.2. *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be convex and coordinate-wise L-smooth, let $\tau_{\mathrm{UCD}}, \tau_{\mathrm{SCD}}, \tau_{\mathrm{ASCD}}$ denote the expected one step progress (6) of UCD, SCD and ASCD, respectively, and suppose all methods use the same step-size rule $\mathcal{M}$. Then*

$$\tau_{\mathrm{UCD}}(\boldsymbol{x}) \leq \tau_{\mathrm{ASCD}}(\boldsymbol{x}) \leq \tau_{\mathrm{SCD}}(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in \mathbb{R}^n\,. \tag{21}$$

Proof. By (5) we get $\tau_{\mathrm{ASCD}}(\boldsymbol{x}) = \frac{1}{2L|\mathcal{I}|}\sum_{i\in\mathcal{I}} |\nabla_i f(\boldsymbol{x})|^2$, where $\mathcal{I}$ denotes the corresponding index set of ASCD when at iterate $\boldsymbol{x}$. Note that for $j \notin \mathcal{I}$ it must hold that $|\nabla_j f(\boldsymbol{x})|^2 \leq [\boldsymbol{u}]_j^2 < \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}}[\boldsymbol{\ell}]_i^2 \leq \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}}|\nabla_i f(\boldsymbol{x})|^2$ by definition of $\mathcal{I}$. □

Observe that the above theorem holds for all gradient oracles and coordinate update variants, as long as they are used with corresponding quality parameters r (as in (19)) and δ_{ij} (as in (20)) as part of the algorithm.

Heuristic variants. Below also propose three heuristic variants of ASCD. For all these variants the active set $\mathcal{I}_t$ can be computed $O(n)$, but the statement of Theorem 3.2 does not apply. These variants only differ from ASCD in the choice of the active set in Algorithm 1:

u-ASCD: $\mathcal{I}_t := \arg\max_{i\in[n]} [\boldsymbol{u}_t]_i$
ℓ-ASCD: $\mathcal{I}_t := \arg\max_{i\in[n]} [\boldsymbol{\ell}_t]_i$
a-ASCD: $\mathcal{I}_t := \left\{ i \in [n] \mid [\boldsymbol{u}_t]_i \geq \max_{i\in[n]} [\boldsymbol{\ell}_t]_i \right\}$

4. Approximate Gradient Update

In this section we argue that for a large class of objective functions of interest in machine learning, the change in the gradient along every coordinate direction can be estimated efficiently.

Lemma 4.1. *Consider $F\colon \mathbb{R}^n \to \mathbb{R}$ as in (17) with twice-differentiable $f\colon \mathbb{R}^d \to \mathbb{R}$. Then for two iterates $\boldsymbol{x}_t, \boldsymbol{x}_{t+1} \in \mathbb{R}^n$ of a coordinate descent algorithm, i.e. $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \gamma_t \boldsymbol{e}_{i_t}$, there exists a $\tilde{\boldsymbol{x}} \in \mathbb{R}^n$ on the line segment between $\boldsymbol{x}_t$ and $\boldsymbol{x}_{t+1}$, $\tilde{\boldsymbol{x}} \in [\boldsymbol{x}_t, \boldsymbol{x}_{t+1}]$ with*

$$\nabla_i F(\boldsymbol{x}_{t+1}) - \nabla_i F(\boldsymbol{x}_t) = \gamma_t \langle \boldsymbol{a}_i, \nabla^2 f(A\tilde{\boldsymbol{x}})\boldsymbol{a}_{i_t}\rangle \quad \forall i \neq i_t \tag{22}$$

where $\boldsymbol{a}_i$ denotes the i-th column of the matrix A.

Proof. For coordinates $i \neq i_t$ the gradient (or subgradient set) of $\Psi_i([\boldsymbol{x}_t]_i)$ does not change. Hence it suffices to calculate the change $\nabla f(\boldsymbol{x}_{t+1}) - \nabla f(\boldsymbol{x}_t)$. This is detailed in the appendix. □

Least-Squares with Arbitrary Regularizers. The least squares problem is defined as problem (17) with $f(A\boldsymbol{x}) = \frac{1}{2}\|A\boldsymbol{x} - \boldsymbol{b}\|_2^2$ for a $\boldsymbol{b} \in \mathbb{R}^d$. This function is twice differentiable with $\nabla^2 f(A\boldsymbol{x}) = I_n$. Hence (22) reduces to

$$\nabla_i F(\boldsymbol{x}_{t+1}) - \nabla_i F(\boldsymbol{x}_t) = \gamma_t \langle \boldsymbol{a}_i, \boldsymbol{a}_{i_t}\rangle \quad \forall i \neq i_t\,. \tag{23}$$

This formulation gives rise to various gradient oracles (20) for the least square problems. For for $i \neq i_t$ we easily verify that the condition (20) is satisfied:

1. $g^1_{ij} := \langle \boldsymbol{a}_i, \boldsymbol{a}_{i_t}\rangle$; $\delta_{ij} = 0$,
2. $g^2_{ij} := \max\{-\|\boldsymbol{a}_i\|\|\boldsymbol{a}_j\|, \min\{S(i,j), \|\boldsymbol{a}_i\|\|\boldsymbol{a}_j\|\}\}$; $\delta_{ij} = \epsilon\|\boldsymbol{a}_i\|\|\boldsymbol{a}_j\|$, where $S\colon [n]\times[n]$ denotes a function with the property
$$|S(i,j) - \langle \boldsymbol{a}_i, \boldsymbol{a}_j\rangle| \leq \epsilon\|\boldsymbol{a}_i\|\|\boldsymbol{a}_j\|\,, \quad \forall i,j \in [n] \tag{24}$$
3. $g^3_{ij} := 0$; $\delta_{ij} = \|\boldsymbol{a}_i\|\|\boldsymbol{a}_j\|$,
4. $g^4_{ij} \in_{\text{u.a.r.}} [-\|\boldsymbol{a}_i\|\|\boldsymbol{a}_j\|, \|\boldsymbol{a}_i\|\|\boldsymbol{a}_j\|]$; $\delta_{ij} = \|\boldsymbol{a}_i\|\|\boldsymbol{a}_j\|$.

Oracle g^1 can be used in the rare cases where the dot product matrix is accessible to the optimization algorithm without any extra cost. In this case the updates will all be exact. If this matrix is not available, then the computation of each scalar product takes time $O(d)$. Hence, they cannot be recomputed on the fly, as argued in Section 2.4. In contrast, the oracles g^3 and g^4 are extremely cheap to compute, but the error bounds are worse. In the numerical experiments in Section 7 we demonstrate that these oracles perform surprisingly well.

The oracle g^2 can for instance be realized by low-dimensional embeddings, such as given by the Johnson-Lindenstrauss lemma (cf. Achlioptas (2003); Matoušek (2008)). By embedding each vector in a lower-dimensional space of dimension $O\left(\epsilon^{-2}\log n\right)$ and computing the scalar products of the embedding in time $O(\log n)$, relation (24) is satisfied.

Updating the gradient of the active coordinate. So far we only discussed the update of the passive coordinates. For the active coordinate the best strategy depends on the update rule $\mathcal{M}$ from (18). If exact line search is used, then $0 \in \nabla_{i_t} f(\boldsymbol{x}_{t+1})$. For other update rules we can update the gradient $\nabla_{i_t} f(\boldsymbol{x}_{t+1})$ with the same gradient oracles as for the other coordinates, however we need also to take into account the change of the gradient of $\Psi_i([\boldsymbol{x}_t]_i)$. If Ψ_i is simple, like for instance in ridge or lasso, the subgradients at the new point can be computed efficiently.

Bounded variation. In many applications the Hessian $\nabla^2 f(A\tilde{\boldsymbol{x}})$ is not so simple as in the case of square loss. If we assume that the Hessian of f is bounded, i.e. $\nabla^2 f(A\boldsymbol{x}) \preceq M \cdot I_n$ for a constant $M \geq 0$, $\forall \boldsymbol{x} \in \mathbb{R}^n$, then it is easy to see that the following holds :

$$-M\|\boldsymbol{a}_i\|\|\boldsymbol{a}_j\| \leq \langle \boldsymbol{a}_i, \nabla^2 f(A\tilde{\boldsymbol{x}})\boldsymbol{a}_{i_t}\rangle \leq M\|\boldsymbol{a}_i\|\|\boldsymbol{a}_j\|\,.$$

Using this relation, we can define gradient oracles for more general functions, by taking the additional approximation factor M into account. The quality can be improved, if we have access to local bounds on $\nabla^2 f(A\boldsymbol{x})$.

Heuristic variants. By design, ASCD is robust to high errors in the gradient estimations – the steepest descent direction is always contained in the active set. However, instead of using only the very crude oracle g^4 to approximate *all* scalar products, it might be advantageous to compute some scalar products with higher precision. We propose to use a caching technique to compute the scalar products with high precision for all vectors in the active set (and storing a matrix of size $O(\mathcal{I}_t \times n)$). This presumably works well if the active set does not change much over time.

5. Extension to Composite Functions

The key ingredients of ASCD are the coordinate-wise upper and lower bounds on the gradient and the definition of the active set $\mathcal{I}_t$ which ensures that the steepest descent direction is always kept and that only provably bad directions are removed from the active set. These ideas can also be generalized to the setting of composite functions (2). We already discussed some popular GS-$*$ update rules in the introduction in Section 2.3.

Implementing ASCD for the GS-s rule is straight forward, and we comment on the GS-r in the appendix in Sec. D.2. Here we exemplary detail the modification for the GS-q rule (16), which turns out to be the most evolved (the same reasoning also applies to the GSL-q rule from (Nutini et al., 2015)). In Algo. 2 we show the construction — based just on approximations of the gradient of the smooth part f — of the active set $\mathcal{I}$. For this we compute upper and lower bounds $\mathbf{v}, \boldsymbol{w}$ on $\min_{y\in\mathbb{R}} V(\boldsymbol{x}, y, \nabla_i f(\boldsymbol{x}))$, such that

$$[\mathbf{v}]_i \leq \min_{y\in\mathbb{R}} V(\boldsymbol{x}, y, \nabla_i f(x) \leq [\boldsymbol{w}]_i \quad \forall i \in [n]\,. \tag{25}$$

The selection of the active coordinate is then based on these bounds. Similar as in Lemma 3.1 and Theorem 3.2 this set has the property $i_{\mathrm{GS-q}} \in \mathcal{I}$, and directions are only discarded in such a way that the efficiency of ASCD-q cannot drop below the efficiency of UCD. The proof can be found in the appendix in Section D.1.

Algorithm 2 Adaptation of ASCD for GS-q rule

Input: Gradient estimate $\tilde{\boldsymbol{g}}$, error bounds $\boldsymbol{r}$.

For $i \in [n]$ define: *compute u.-and l.-bounds*

$[\boldsymbol{u}]_i := [\tilde{\boldsymbol{g}}]_i + [\boldsymbol{r}]_i$, $[\boldsymbol{\ell}]_i := [\tilde{\boldsymbol{g}}]_i - [\boldsymbol{r}]_i$

$[\boldsymbol{u}^\star]_i := \arg\min_{y\in\mathbb{R}} V(\boldsymbol{x}, y, [\boldsymbol{u}]_i)$ *minimize the model*

$[\boldsymbol{\ell}^\star]_i := \arg\min_{y\in\mathbb{R}} V(\boldsymbol{x}, y, [\boldsymbol{\ell}]_i)$

compute u.-and l. bounds on $\min_{y\in\mathbb{R}} V(\boldsymbol{x}, y, \nabla_i f(\boldsymbol{x}))$

$[\boldsymbol{\omega}_u]_i := V(\boldsymbol{x}, [\boldsymbol{u}^\star]_i, [\boldsymbol{u}]_i) + \max\{0, [\boldsymbol{u}^\star]_i([\boldsymbol{\ell}]_i - [\boldsymbol{u}]_i)\}$

$[\boldsymbol{\omega}_\ell]_i := V(\boldsymbol{x}, [\boldsymbol{\ell}^\star]_i, [\boldsymbol{\ell}]_i) + \max\{0, [\boldsymbol{\ell}^\star]_i([\boldsymbol{u}]_i - [\boldsymbol{\ell}]_i)\}$

$[\mathbf{v}]_i := \min\{V(\boldsymbol{x}, [\boldsymbol{u}^\star]_i, [\boldsymbol{u}]_i), V(\boldsymbol{x}, [\boldsymbol{\ell}^\star]_i, [\boldsymbol{\ell}]_i)\}$

$[\boldsymbol{w}]_i := \min\{[\boldsymbol{\omega}_u]_i, [\boldsymbol{\omega}_\ell]_i, \Psi_i([\boldsymbol{x}]_i)\}$

$\mathrm{av}(\mathcal{I}) := \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} [\boldsymbol{w}]_i$ *compute active set*

$\mathcal{I}_t := \arg\min_{\mathcal{I}} |\{\mathcal{I} \subseteq [n] \mid [\mathbf{v}]_i > \mathrm{av}(\mathcal{I}), \forall i \notin \mathcal{I}\}|$

6. Analysis of Competitive Ratio

In Section 3 we derived in Thm. 3.2 that the one step progress of ASCD is between the bounds on the onestep progress of UCD and SCD. However, we know that the efficiency of the latter two methods can differ much, up to a factor of n. In this section we will argue that in certain cases where SCD performs much better than UCD, ASCD will accelerate as well. To measure this effect, we could for instance consider the ratio:

$$\varrho_t := \frac{\left|\left\{i \in \mathcal{I}_t \mid |\nabla_i f(\boldsymbol{x}_t)| \geq \frac{1}{2}\|\nabla f(\boldsymbol{x}_t)\|_\infty\right\}\right|}{|\mathcal{I}_t|}\,, \tag{26}$$

For general functions this expression is a bit cumbersome to study, therefore we restrict our discussion to the class of objective functions (11) as introduced in Sec. 2.2. Of course not all real-world objective functions will fall into this class, however this problem class is still very interesting in our study, as we will see in the following, because it will highlight the ability (or disability) of the algorithms to eventually identify the right set of 'active' coordinates.

For the functions with the structure (11) (and q as in Thm. 2.2), the active set falls into the first s coordinates. Hence it is reasonable to approximate ϱ_t by the competitive ratio

$$\rho_t := \frac{|\mathcal{I}_t \cap [s]|}{|\mathcal{I}_t|}\,. \tag{27}$$

It is also reasonable to assume that in the limit, $(t \to \infty)$, a constant fraction of the $[s]$ will be contained in the active set $\mathcal{I}_t$ (it might not hold $[s] \subseteq \mathcal{I}_t\ \forall t$, as for instance with exact line search the directional derivative vanishes just after the update). In the following theorem we calculate ρ_t for $(t \to \infty)$, the proof is given in the appendix.

Theorem 6.1. *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be of the form (11). For indices $i \notin [s]$ define $\mathcal{K}_i := \{t \mid i \notin \mathcal{I}_t, i \in \mathcal{I}_{t-1}\}$. For $j \in \mathcal{K}_i$ define $T_j^i := \min\{t - j \mid i \in \mathcal{I}_{j+t}\}$, i.e. the number of iterations outside the active set, $T_\infty^i := \lim_{t\to\infty} \mathbb{E}_{j\in\mathcal{K}_i}\left[T_j^i \mid j > k\right]$, and the average $T_\infty := \mathbb{E}_{i\notin[s]}\left[T_\infty^i\right]$. If there exists a constant $c > 0$ such that $\lim_{t\to\infty} |[s] \cap \mathcal{I}_t| = cs$, then (with the notation $\rho_\infty := \lim_{t\to\infty} \mathbb{E}\,[\rho_t]$),*

$$\rho_\infty \geq \frac{2cs}{cs + n - s - T_\infty + \sqrt{\theta}}\,, \tag{28}$$

where $\theta \equiv \theta := n^2 + (c-1)^2 s^2 + 2n((c-1)s - T_\infty) + 2(1+c)sT_\infty + T_\infty^2$. Especially, $\rho_\infty \geq 1 - \frac{n-s}{T_\infty}$.

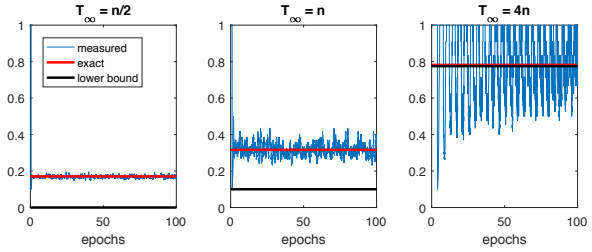


Figure 1. Competitive ratio ρ_t (*blue*) in comparison with ρ_∞ (28) (*red*) and the lower bound $\rho_\infty \geq 1 - \frac{n-s}{T_\infty}$ (*black*). Simulation for parameters $n = 100$, $s = 10$, $c = 1$ and $T_\infty \in \{50, 100, 400\}$.

In Figure 1 we compare the lower bound (28) of the competitive ratio in the limit ($t \to \infty$) with actual measurements of ρ_t for simulated example with parameters $n = 100$, $s = 10$, $c = 1$ and various $T_\infty \in \{50, 100, 400\}$. We initialized the active set $\mathcal{I}_0 = [s]$, but we see that the equilibrium is reached quickly.

6.1. Estimates of the competitive ratio

Based on this Thm. 6.1, we can now estimate the competitive ratio in various scenarios. On the class (11) it holds $c \approx 1$ as we argued before. Hence the competitive ratio (28) just depends on T_∞. This quantity measures how many iterations a coordinate $j \notin [s]$ is in average outside of the active set $\mathcal{I}_t$. From the lower bound we see that the competitive ratio ρ_t approaches a constant for ($t \to \infty$) if $T_\infty = \Theta(n)$, for instance $\rho_\infty \geq 0.8$ if $T_\infty \geq 5n$.

As an approximation to T_∞, we estimate the quantities $T^j_{t_0}$ defined in Thm. 6.1. $T^j_{t_0}$ denotes the number of iterations it takes until coordinate j enters the active set again, assuming it left the active set at iteration $t_0 - 1$. We estimate $T^j_{t_0} \geq \hat{T}$, where $\hat{T}$ denotes maximum number of iterations such that

$$\sum_{t=t_0}^{t_0+\hat{T}} \gamma_t \delta_{i_t j} \leq \frac{1}{s} \sum_{k=1}^{s} \left| \nabla_k f\left(\boldsymbol{x}_{t_0+\hat{T}}\right) \right| \quad \forall j \notin [s]. \tag{29}$$

For smooth functions, the steps $\gamma_t = \Theta\left(|\nabla_{i_t} f(\boldsymbol{x}_t)|\right)$ and if we additionally assume that the errors of the gradient oracle are uniformly bounded $\delta_{ij} \leq \delta$, the sum in (29) simplifies to $\delta \sum_{t=t_0}^{t_0+\hat{T}} |\nabla_{i_t} f(\boldsymbol{x}_t)|$.

For smooth, but not strongly convex function q, the norms of the gradient changes very slowly, with a rate independent of s or n, and we get $\hat{T} = \Theta\left(\frac{1}{\delta}\right)$. Hence, the competitive ratio is constant for $\delta = \Theta\left(\frac{1}{n}\right)$.

For strongly convex function q, the norm of the gradient decreases linearly, say $\|\nabla f(\boldsymbol{x}_t)\|_2^2 \propto e^{\kappa t}$ for $\kappa \approx \frac{1}{s}$. I.e. it decreases by half after each $\Theta(s)$ iterations. Therefore to guarantee $\hat{T} = \Theta(n)$ it needs to hold $\delta = e^{-\Theta\left(\frac{n}{s}\right)}$. This result seems to indicate that the use of ACDM is only justified if s is large, for instance $s \geq \frac{1}{4}n$. Otherwise the convergence on q is too fast, and the gradient approximations are too weak. However, notice that we assumed δ to be an uniform bound on all errors. If the errors have large discrepancy the estimates become much better (this holds for instance on datasets where the norm data vectors differs much, or when caching techniques as mentioned in Sec. 4 are employed).

7. Empirical Observations

In this section we evaluate the empirical performance of ASCD on synthetic and real datasets. We consider the following regularized general linear models:

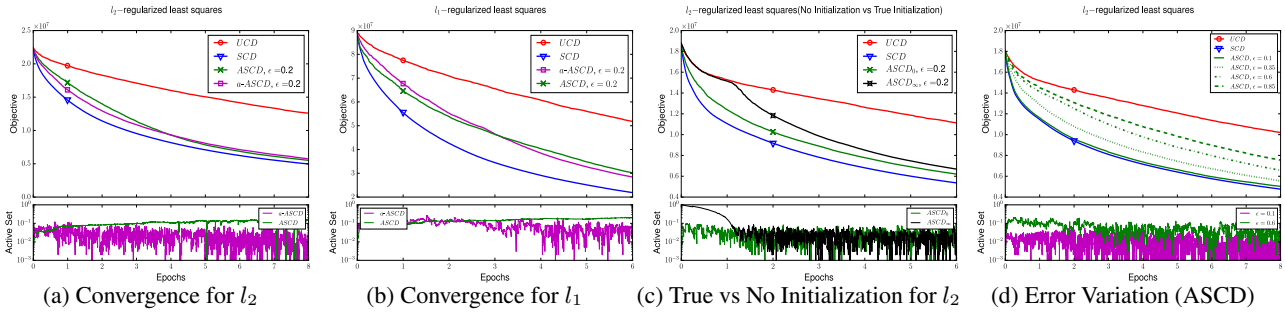
$$\min_{\boldsymbol{x} \in \mathbb{R}^n} \tfrac{1}{2}\|A\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \tfrac{\lambda}{2}\|\boldsymbol{x}\|_2^2\,, \tag{30}$$

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} \tfrac{1}{2}\|A\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \lambda\|\boldsymbol{x}\|_1\,, \tag{31}$$

that is, l_2-regularized least squares (30) as well as l_1-regularized linear regression (Lasso) in (31), respectively.

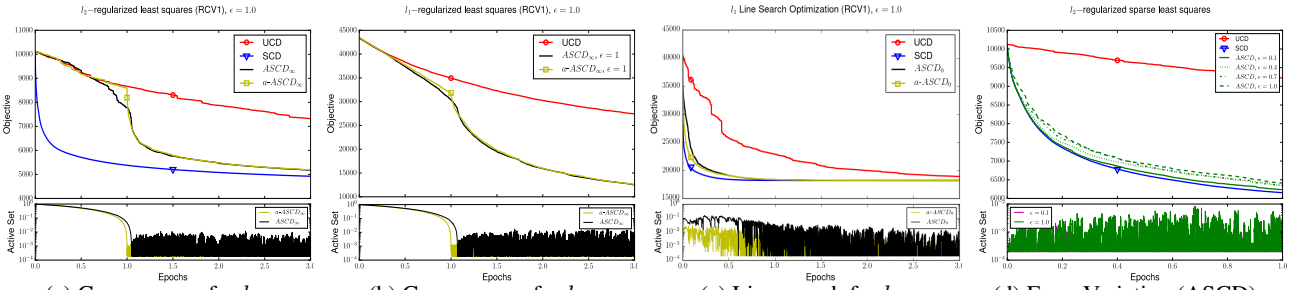
Datasets. The datasets $A \in \mathbb{R}^{d \times n}$ in problems (30) and (31) were chosen as follows for our experiments. For the synthetic data, we follow the same generation procedure as described in (Nutini et al., 2015), which generates very sparse data matrices. For completeness, full details of the data generation process are also provided in the appendix in Sec. E. For the synthetic data we choose $n = 5000$ for problem (31) and $n = 1000$ for problem (30). Dimension $d = 1000$ is fixed for both cases.
For real datasets, we perform the experimental evaluation on RCV1 (binary,training), which consists of $20,242$ samples, each of dimension $47,236$ (Lewis et al., 2004). We use the un-normalized version with all non-zeros values set to 1 (bag-of-words features).

Gradient oracles and implementation details. On the RCV1 dataset, we approximate the scalar products with the oracle g^4 that was introduced in Sec. 4. This oracle is extremely cheap to compute, as the norms $\|\boldsymbol{a}_i\|$ of the columns of A only need to be computed once.
On the synthetic data, we simulate the oracle g^2 for various precisions values ϵ. For this, we sample a value uniformly at random from the allowed error interval (24). Figs. 2d and 3d show the convergence for different accuracies.
For the l_1-regularized problems, we used ASCD with the GS-s rule (the experiments in (Nutini et al., 2015) revealed almost identical performance of the different GS-$*$ rules). We compare the performance of UCD, SCD and ASCD. We also implement the heuristic version a-ASCD that was introduced in Sec. 3. All algorithm variants use the same step size rule (i.e. the method $\mathcal{M}$ in Algorithm 1). We use exact line search for the experiment in Fig. 3c, for all others we used a fixed step size rule (the convergence is slower



(a) Convergence for l_2 (b) Convergence for l_1 (c) True vs No Initialization for l_2 (d) Error Variation (ASCD)

Figure 2. Experimental results on synthetically generated datasets



(a) Convergence for l_2 (b) Convergence for l_1 (c) Line search for l_1 (d) Error Variation (ASCD)

Figure 3. Experimental results on the RCV1-binary dataset

for all algorithms, but the different effects of the selection of the active coordinate is more distinctly visible).
ASCD is either initialized with the true gradient (Figs. 2a, 2b, 2d, 3c, 3d) or arbitrarely (with error bounds $\delta = \infty$) in Figs. 3a and 3b (Fig. 2c compares both initializations).
Fig. 2 shows results on the synthetic data, Fig. 3 on the RCV1 dataset. All plots show also the size of the active set $\mathcal{I}_t$. The plots 3c and 3d are generated on a subspace of RCV1, with 10000 and 5000 randomly chosen columns, respectively.

Here are the highlights of our experimental study:

1. **No initialization needed.** We observe (see e.g. Figs. 2c,3a, 3b) that initialization with the true gradient values is *not* needed at beginning of the optimization process (the cost of the initialization being as expensive as one epoch of ASCD). Instead, the algorithm performs strong in terms of learning the active set on its own, and the set converges very fast after just one epoch.
2. **High errors toleration.** The gradient oracle g^4 gives very crude approximations, however the convergence of ASCD is excellent on RCV1 (Fig. 3). Here the size of the true active set is very small (in the order of 0.1% on RCV1) and ASCD is able to identify this set. Fig. 3d shows that almost nothing can be gained from more precise (and more expensive) oracles.
3. **Heuristic a-ASCD performs well.** The convergence behavior of ASCD follows theory. For the heuristic version a-ASCD (which computes the active set slightly faster, but Thm. 3.2 does not hold) performs identical to ASCD in practice (cf. Figs. 2, 3), and sometimes slightly better. This is explained by the active set used in ASCD typically being larger than the active set of a-ASCD (Figs. 2a,2b, 3a, 3b).

8. Concluding Remarks

We proposed ASCD, a novel selection mechanism for the active coordinate in CD methods. Our scheme enjoys three favorable properties: (i) its performance can reach the performance steepest CD — both in theory and practice, (ii) the performance is never worse than uniform CD, (iii) in many important applications, the scheme it can be implemented at no extra cost per iteration.

ASCD calculates the active set in a safe manner, and picks the active coordinate uniformly at random from this smaller set. It seems possible that an adaptive sampling strategy on the active set could boost the performance even further. Here we only study CD methods where a single coordinate gets updated in each iteration. ASCD can immediately also be generalized to block-coordinate descent methods. However, the exact implementation in a distributed setting can be challenging.

Finally, it is an interesting direction to extend ASCD also to the stochastic gradient descent setting (not only heuristically, but with the same strong guarantees as derived in this paper).

References

Achlioptas, Dimitris. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66(4):671 – 687, 2003.

Allen-Zhu, Z, Qu, Z, Richtarik, P, and Yuan, Y. Even faster accelerated coordinate descent using non-uniform sampling. 2016.

Boyd, Stephen P and Vandenberghe, Lieven. *Convex optimization*. Cambridge University Press, 2004.

Csiba, Dominik, Qu, Zheng, and Richtárik, Peter. Stochastic Dual Coordinate Ascent with Adaptive Probabilities. In *ICML 2015 - Proceedings of the 32th International Conference on Machine Learning*, 2015.

Dawkins, Brian. Siobhan's problem: The coupon collector revisited. *The American Statistician*, 45(1):76–82, 1991.

Dhillon, Inderjit S, Ravikumar, Pradeep, and Tewari, Ambuj. Nearest Neighbor based Greedy Coordinate Descent. In *NIPS 2014 - Advances in Neural Information Processing Systems 27*, 2011.

Friedman, Jerome, Hastie, Trevor, Höfling, Holger, and Tibshirani, Robert. Pathwise coordinate optimization. *The Annals of Applied Statistics*, 1(2):302–332, 2007.

Friedman, Jerome, Hastie, Trevor, and Tibshirani, Robert. Regularization Paths for Generalized Linear Models via Coordinate Descent. *Journal of Statistical Software*, 33(1):1–22, 2010.

Fu, Wenjiang J. Penalized regressions: The bridge versus the lasso. *Journal of Computational and Graphical Statistics*, 7 (3):397–416, 1998.

Hsieh, Cho-Jui, Chang, Kai-Wei, Lin, Chih-Jen, Keerthi, S Sathiya, and Sundararajan, S. A Dual Coordinate Descent Method for Large-scale Linear SVM. In *the 25th International Conference on Machine Learning*, pp. 408–415, New York, USA, 2008.

Kim, Hyunsoo and Park, Haesun. Nonnegative matrix factorization based on alternating nonnegativity constrained least squares and active set method. *SIAM Journal on Matrix Analysis and Applications*, 30(2):713–730, 2008.

Lee, Daniel D and Seung, H Sebastian. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755): 788–791, 1999.

Lewis, David D., Yang, Yiming, Rose, Tony G., and Li, Fan. Rcv1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res.*, 5:361–397, 2004.

Matoušek, Jiří. On variants of the johnsonlindenstrauss lemma. *Random Structures & Algorithms*, 33(2):142–156, 2008.

Nesterov, Yu. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

Nesterov, Yurii and Stich, Sebastian U. Efficiency of the accelerated coordinate descent method on structured optimization problems. *SIAM Journal on Optimization*, 27(1):110–123, 2017.

Nutini, Julie, Schmidt, Mark W, Laradji, Issam H, Friedlander, Michael P, and Koepke, Hoyt A. Coordinate Descent Converges Faster with the Gauss-Southwell Rule Than Random Selection. In *ICML*, pp. 1632–1641, 2015.

Osokin, Anton, Alayrac, Jean-Baptiste, Lukasewitz, Isabella, Dokania, Puneet K., and Lacoste-Julien, Simon. Minding the gaps for block frank-wolfe optimization of structured svms. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pp. 593–602. PMLR, 2016.

Papa, Guillaume, Bianchi, Pascal, and Clémençon, Stéphan. Adaptive Sampling for Incremental Optimization Using Stochastic Gradient Descent. *ALT 2015 - 26th International Conference on Algorithmic Learning Theory*, pp. 317–331, 2015.

Perekrestenko, Dmytro, Cevher, Volkan, and Jaggi, Martin. Faster Coordinate Descent via Adaptive Importance Sampling. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pp. 869–877, Fort Lauderdale, FL, USA, 20–22 Apr 2017. PMLR.

Qu, Zheng and Richtárik, Peter. Coordinate descent with arbitrary sampling i: algorithms and complexity. *Optimization Methods and Software*, 31(5):829–857, 2016.

Richtárik, Peter and Takáč, Martin. Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, 156(1):433–484, 2016.

Shalev-Shwartz, Shai and Tewari, Ambuj. Stochastic Methods for l_1-regularized Loss Minimization. *JMLR*, 12:1865–1892, 2011.

Shalev-Shwartz, Shai and Zhang, Tong. Stochastic Dual Coordinate Ascent Methods for Regularized Loss Minimization. *JMLR*, 14:567–599, 2013.

Shrivastava, Anshumali and Li, Ping. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In *NIPS 2014 - Advances in Neural Information Processing Systems 27*, pp. 2321–2329, 2014.

Tseng, Paul and Yun, Sangwoon. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming*, 117(1):387–423, 2009.

Wen, Zaiwen, Yin, Wotao, Zhang, Hongchao, and Goldfarb, Donald. On the convergence of an active-set method for 1 minimization. *Optimization Methods and Software*, 27(6):1127–1146, 2012.

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

Wu, Tong Tong and Lange, Kenneth. Coordinate descent algorithms for lasso penalized regression. *Ann. Appl. Stat.*, 2(1): 224–244, 2008.

Zhao, Peilin and Zhang, Tong. Stochastic optimization with importance sampling for regularized loss minimization. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *PMLR*, pp. 1–9, Lille, France, 2015. PMLR.