Efficient Greedy Coordinate Descent for Composite Problems

Sai Praneeth Karimireddy* EPFL
Anastasia Koloskova* EPFL
Sebastian U. Stich EPFL
Martin Jaggi EPFL

Abstract

Coordinate descent with random coordinate selection is the current state of the art for many large scale optimization problems. However, greedy selection of the steepest coordinate on smooth problems can yield convergence rates independent of the dimension n, and requiring up to n times fewer iterations.

In this paper, we consider greedy updates that are based on subgradients for a class of non-smooth composite problems, which includes $L1$-regularized problems, SVMs and related applications. For these problems we provide (i) the first linear rates of convergence independent of n, and show that our greedy update rule provides speedups similar to those obtained in the smooth case. This was previously conjectured to be true for a stronger greedy coordinate selection strategy.

Furthermore, we show that (ii) our new selection rule can be mapped to instances of maximum inner product search, allowing to leverage standard nearest neighbor algorithms to speed up convergence. We demonstrate the validity of the approach through extensive numerical experiments.

1 Introduction

In recent years, there has been increased interest in coordinate descent (CD) methods due to their simplicity, low cost per iteration, and efficiency (Wright, 2015). Algorithms based on coordinate descent are the state of the art for many optimization problems (Nesterov, 2012; Shalev-Shwartz and Zhang, 2013b; Lin et al., 2014; Shalev-Shwartz and Zhang, 2013a, 2016; Richtarik and Takac, 2016; Fercoq and Richtárik, 2015; Nesterov and Stich, 2017).Most of the CD methods draw their coordinates from a fixed distribution—for instance from the uniform distribution as in uniform coordinate descent (UCD). However, it is clear that significant improvements can be achieved by choosing more *important* coordinates more frequently (Nesterov, 2012; Nesterov and Stich, 2017; Stich et al., 2017a,b; Perekrestenko et al., 2017). In particular, we could greedily choose the 'best' coordinate at each iteration i.e. the greedy or steepest coordinate descent (GCD).

GCD for composite problems. Consider the smooth quadtratic function $f(\boldsymbol{\alpha}) \overset{\text{def}}{=} \frac{1}{2}\|A\boldsymbol{\alpha} - \boldsymbol{b}\|_2^2$. There are three natural notions of the 'best' coordinate.[1] One could choose (i) `GS-s`: the *steepest* coordinate direction based on (sub)-gradients, (ii) `GS-r`: the coordinate which allows us to take the largest step, and (iii) `GS-q`: the coordinate that allows us to minimize the function value the most. For our example (and in general for smooth functions), the three rules are equivalent. When we add an additional non-smooth function to f, such as $g(\boldsymbol{\alpha}) = \lambda\|\boldsymbol{\alpha}\|_1$, however, the three notions are no more equivalent. The performance of greedy coordinate descent in this composite setting is not well understood, and is the focus of this work.

Iteration complexity of GCD. If the objective f decomposes into n identical separable problems, then clearly GCD is identical to UCD. In all but such extreme cases, Nutini et al. (2015) give a refined analysis of GCD for smooth functions and show that it outperforms UCD. This lead to a renewed interest in greedy methods (e.g. (Karimi et al., 2016; You et al., 2016; Dünner et al., 2017; Song et al., 2017; Nutini et al., 2017; Stich et al., 2017a; Locatello et al., 2018; Lu et al., 2018)). However, for the composite case the analysis in (Nutini et al., 2015) of GCD methods for any of the three rules mentioned earlier falls back to that of UCD. Thus they fail to demonstrate the advantage of greedy methods for the composite case. In fact it is claimed that the rate of the `GS-s` greedy rule may

Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS) 2019, Naha, Okinawa, Japan. PMLR: Volume 89.

* Equal contribution.

[1] Following standard notation (cf. (Nutini et al., 2015)) we call them the Gauss-Southwell (GS) rules.

even be worse than that of UCD. In this work we provide a refined analysis of GCD for a certain class of composite problems, and show that all three strategies (GS-s, GS-r, and GS-q) converge on composite problems at a rate similar to GCD in the smooth case. Thus for these problems too, greedy coordinate algorithms are provably faster than UCD other than in extreme cases.

Efficiency of GCD. A naïve implementation of GCD would require computing the full gradient at a cost roughly n times more than just computing one coordinate of the gradient as required by UCD. This seems to negate any potential gain of GCD over UCD. The working principle behind *approximate GCD* methods is to trade-off exactness of the greedy direction against the time spent to decide the steepest direction (e.g. (Stich et al., 2017a)). For smooth problems, Dhillon et al. (2011) show that *approximate nearest neighbor search* algorithms can be used to provide in *sublinear time* an approximate steepest descent direction. We build upon these ideas and extend the framework to non-smooth composite problems, thereby capturing a significantly larger class of input problems. In particular we show how to efficiently map the GS-s rule to an instance of maximum inner product search (MIPS).

Contributions. We analyze and advocate the use of the GS-s greedy rule to compute the update direction for composite problems. Our main contributions are:

i) We show that on a class of composite problems, greedy coordinate methods achieve convergence rates which are very similar to those obtained for smooth functions, thereby extending the applicability of GCD. This class of problems covers several important applications such as SVMs (in its dual formulation), Lasso regression, $L1$-regularized logistic regression among others. With this we establish that greedy methods significantly outperform UCD also on composite problems, except in extreme cases (cf. Remark 4).

ii) We show that both the GS-s as well as the GS-r rules achieve convergence rates which are (other than in extreme cases) faster than UCD. This sidesteps the negative results by Nutini et al. (2015) for these methods through a more fine-grained analysis. We also study the effect of approximate greedy directions on the convergence.

iii) Algorithmically, we show how to precisely map the GS-s direction computation as a special instance of a maximum inner product search problem (MIPS). Many standard nearest neighbor algorithms such as e.g. Locality Sensitive Hashing (LSH) can therefore be used to efficiently run GCD on composite optimization problems.

iv) We perform extensive numerical experiments to study the advantages and limitations of our greedy descent combined with a current state-of-the-art MIPS implementation (Boytsov and Naidan, 2013).

Related Literature. Coordinate descent, being one of the earliest known optimization methods, has a rich history (e.g. (Bickley, 1941; Warga, 1963; Bertsekas and Tsitsiklis, 1989, 1991)). A significant renewal in interest followed the works of Nesterov (2012) who provided a simple analysis of the convergence of UCD, and (Shalev-Shwartz and Zhang, 2013b) who apply UCD on the dual problems (called SDCA). In practice, many solvers (e.g. (Ndiaye et al., 2015; Massias et al., 2018)) combine UCD with active set heuristics where attention is restricted to a subset of *active* coordinates. These methods are orthogonal to, and hence can be combined with, the greedy rules studied here.

Greedy coordinate methods can also be viewed as an 'extreme' version of adaptive importance sampling (Stich et al., 2017a; Perekrestenko et al., 2017). For the smooth case, greedy methods choose the coordinate of the gradient with the largest absolute value while importance sampling methods sample coordinates proportional to the absolute value of the gradient coordinate (or its power). Thus, new greedy algorithms can directly be translated into new importance sampling schemes. However unlike greedy methods, even in the smooth case, there are no easily characterized function classes for which the importance sampling schemes or the active set methods are provably faster than UCD. The work closest to ours, other than the already discussed Nutini et al. (2015), would be that of Dhillon et al. (2011). The latter show a sublinear $O(1/t)$ convergence rate for GS-r on composite problems. They also propose a practical variant for $L1$-regularized problems which essentially ignores the regularizer and is hence not guaranteed to converge.

2 Setup

We consider composite optimization problems of the structure

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \Big[F(\boldsymbol{\alpha}) := f(\boldsymbol{\alpha}) + \sum_{i=1}^{n} g_i(\alpha_i)\Big], \tag{1}$$

where n is the number of coordinates, $f\colon \mathbb{R}^n \to \mathbb{R}$ is convex and smooth, and the $g_i\colon \mathbb{R} \to \mathbb{R}$, $i \in [n]$ are convex and possibly non-smooth. In this exposition, we further restrict the function $g(\boldsymbol{\alpha}) := \sum_{i=1}^n g_i(\alpha_i)$ to either enforce a *box constraint* or an $L1$ *regularizer*. This comprises many important problem classes, for instance dual SVM or Lasso regression, see Appendix A.3.

We further assume that the smooth component f is

coordinate-wise L smooth: for any $\boldsymbol{\alpha}$, γ and i,

$$f(\boldsymbol{\alpha}+\gamma\mathbf{e}_i) \leq f(\boldsymbol{\alpha}) + \nabla_i f(\boldsymbol{\alpha})\gamma + \frac{L\gamma^2}{2}\,. \tag{2}$$

Sometimes we will assume that f is in addition also strongly convex with respect to the $\|\cdot\|_p$ norm, $p \in \{1,2\}$, that is,

$$f(\boldsymbol{\alpha}+\Delta\boldsymbol{\alpha}) \geq f(\boldsymbol{\alpha}) + \langle \nabla f(\boldsymbol{\alpha}), \Delta\boldsymbol{\alpha}\rangle + \frac{\mu_p}{2}\|\Delta\boldsymbol{\alpha}\|_p^2 \tag{3}$$

for any $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}+\Delta\boldsymbol{\alpha}$ in the domain of F. In general it holds $\mu_1 \in [\mu_2/n, \mu_2]$. See Nutini et al. (2015) for a detailed comparison of the two constants.

3 GCD for Non-Smooth Problems

Here we briefly recall the definitions of the `GS-s`, `GS-r` and `GS-q` coordinate selection rules and introduce the approximate `GS-s` rule that we will consider in detail.

$$\texttt{GS-s}(\boldsymbol{\alpha}) := \arg\max_j \left[\min_{s\in\partial g_j} |\nabla_j f(\boldsymbol{\alpha}) + s| \right], \tag{4}$$

$$\texttt{GS-r}(\boldsymbol{\alpha}) := \arg\max_{j\in[n]} |\gamma_j|\,, \tag{5}$$

$$\texttt{GS-q}(\boldsymbol{\alpha}) := \arg\max_{j\in[n]} |\chi_j(\boldsymbol{\alpha})|\,, \tag{6}$$

for an iterate $\boldsymbol{\alpha} \in \mathbb{R}^n$, $\nabla_j f(\boldsymbol{\alpha}) := \langle \nabla f(\boldsymbol{\alpha}), \mathbf{e}_j\rangle$ for standard unit vector $\mathbf{e}_j$. Here $\chi_j(\boldsymbol{\alpha})$ and γ_j are defined as the minimum value and minimizer respectively of

$$\min_\gamma \left[\gamma\nabla_j f(\boldsymbol{\alpha}) + \frac{L\gamma^2}{2} + g_j(\alpha_j+\gamma) - g_j(\alpha_j) \right].$$

We relax the requirement for an exact steepest selection, and define an approximate `GS-s` rule.

Definition 1 (Θ-approximate `GS-s`)**.** *For given $\boldsymbol{\alpha}$, the coordinate j is considered to be a Θ-approximate steepest direction for $\Theta \in (0,1]$ if*

$$\min_{s\in\partial g_j} |\nabla_j f(\boldsymbol{\alpha}) + s| \geq \Theta \max_i \left[\min_{s\in\partial g_i} |\nabla_i f(\boldsymbol{\alpha}) + s| \right].$$

3.1 GCD for $L1$-regularized problems

We now discuss the `GS-s` rule for the concrete example of $L1$ problems, and collect some observations that we will use later to define the mapping to the MIPS instance. A similar discussion is included for box constrained problems in Appendix B.

Consider $L1$-regularized problems of the form

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \left[F(\boldsymbol{\alpha}) := f(\boldsymbol{\alpha}) + \lambda\|\boldsymbol{\alpha}\|_1 \right]. \tag{7}$$

The `GS-s` steepest rule (4) and update rules can be simplified for such functions. Let $\operatorname{sign}(x)$ denote the sign function, and define $S_\lambda(x)$ as the shrinkage operator

$$S_\lambda(x) := \begin{cases} x - \operatorname{sign}(x)\lambda, & \text{if } |x| \geq \lambda \\ 0 & \text{otherwise}\,. \end{cases}$$

Further, for any $\boldsymbol{\alpha}$, let us define $\boldsymbol{s}(\boldsymbol{\alpha})$ as

$$s(\boldsymbol{\alpha})_i := \begin{cases} S_\lambda(\nabla_i f(\boldsymbol{\alpha})), & \text{if } \alpha_i = 0 \\ \nabla_i f(\boldsymbol{\alpha}) + \operatorname{sign}(\alpha_i)\lambda & \text{otherwise}\,. \end{cases} \tag{8}$$

Lemma 1. *For any $\boldsymbol{\alpha}$, the `GS-s` rule is equivalent to*

$$\max_i \left[\min_{s\in\partial g_i} |\nabla_i f(\boldsymbol{\alpha}) + \boldsymbol{s}| \right] \equiv \max_i |s(\boldsymbol{\alpha})_i|\,. \tag{9}$$

Our analysis of `GS-s` rule requires bounding the number of '`bad`' steps (to be detailed in Section 4). For this, we will slightly modify the update of the coordinate descent method. Note that we still always follow the `GS-s` direction, but will sometimes not perform the standard proximal coordinate update along this direction. To update the i_t-th coordinate, we either rely on the standard proximal step on the coordinate,

$$\alpha_{i_t}^+ := S_{\frac{\lambda}{L}}\left(\alpha_{i_t}^{(t)} - \frac{1}{L}\nabla_{i_t} f(\boldsymbol{\alpha}^{(t)}) \right). \tag{10}$$

or we perform line-search

$$\alpha_{i_t}^+ := \arg\min_\gamma F\big(\boldsymbol{\alpha}^{(t)} + (\gamma - \alpha_{i_t}^{(t)})\mathbf{e}_{i_t}\big) \tag{11}$$

Finally, the i_t-th coordinate is updated as

$$\alpha_i^{(t+1)} := \begin{cases} \alpha_i^+, & \text{if } \alpha_i^+\alpha_i^{(t)} \geq 0 \\ 0, & \text{otherwise} \end{cases}. \tag{12}$$

Our modification or 'post-processing' step (12) ensures that the coordinate α_i can never 'cross' the origin. This small change will later on help us bound the precise number of steps needed in our convergence rates (Sec. 4). The details are summarized in Algorithm 1.

Algorithm 1 $L1$ Greedy Coordinate Descent

1: **Initialize:** $\boldsymbol{\alpha}_0 := \mathbf{0} \in \mathbb{R}^n$.
2: **for** $t = 0, 1, \ldots,$ until convergence **do**
3: Select coordinate i_t as in `GS-s`, `GS-r`, or `GS-q`.
4: Find $\alpha_{i_t}^+$ via gradient (10) or line-search (11).
5: Compute $\alpha_{i_t}^{(t+1)}$ as in (12).
6: **end for**

3.2 GCD for Box-Constrained Problems

Using similar ideas, we can also derive the greedy coordinate update for problems with box constraints, such as for the dual SVM. The detailed approach is provided in Appendix B.

4 Convergence Rates

In this section, we present our main convergence results. We illustrate the novelty of our results in the important $L1$-regularized case: For strongly convex functions f, we provide the first linear rates of convergence independent of n for greedy coordinate methods over $L1$-regularized problems, matching the rates in the smooth case. In particular, for `GS-s` this was conjectured to be impossible (Nutini et al., 2015, Section

H.5, H.6) (see Remark 4). We also show the sublinear convergence of the three rules in the non-strongly convex setting. Similar rates also hold for box-constrained problems.

4.1 Linear convergence for strongly convex f

Theorem 1. *Consider an $L1$-regularized optimization problem (7), with f being coordinate-wise L smooth, and μ_1 strongly convex with respect to the $L1$ norm. After t steps of Algorithm 1 where the coordinate i_t is chosen using either the* GS-s *,* GS-r*, or* GS-q *rule,*

$$F(\boldsymbol{\alpha}^{(t)}) - F(\boldsymbol{\alpha}^\star) \leq \left(1 - \frac{\mu_1}{L}\right)^{\lceil t/2 \rceil} \left(F(\boldsymbol{\alpha}^{(0)}) - F(\boldsymbol{\alpha}^\star)\right).$$

Remark 2. *The linear convergence rate of Theorem 1 also holds for the Θ-approximate* GS-s *rule as in Definition 1. In this case the μ_1 will be multiplied by Θ^2.*

Remark 3. *All our linear convergence rates can be extended to objective functions which only satisfy the weaker condition of proximal-PL strong convexity (Karimi et al., 2016).*

Remark 4. *The standard analysis (e.g. in Nesterov (2012)) of UCD gives a convergence rate of*

$$\mathbb{E}\left[F(\boldsymbol{\alpha}^{(t)})\right] - F(\boldsymbol{\alpha}^\star) \leq \left(1 - \frac{\mu_2}{nL}\right)^t \left(F(\boldsymbol{\alpha}^{(0)}) - F(\boldsymbol{\alpha}^\star)\right).$$

Here μ_2 is the strong convexity constant with respect to the $L2$ norm, which satisfies $\mu_1 \in [\mu_2/n, \mu_2]$. The left boundary $\mu_1 = \mu_2/n$ marks the worst-case for GCD, resulting in convergence slower than UCD. It is shown in Nutini et al. (2015) that this occurs only in extreme examples (e.g. when f consists of n identical separable functions). For all other situations when $\mu_1 \geq 2\mu_2/n$, our result shows that GCD is faster.

Remark 5. *Our analysis in terms of μ_1 works for all three selection rules* GS-s*,* GS-r*, or* GS-q *rules. In (Nutini et al., 2015, Section H5, H6) it was conjectured (but not proven) that this linear convergence rate holds for* GS-q*, but that it cannot hold for* GS-s *or* GS-r*. Example functions were constructed where it was shown that the single step progress of* GS-s *or* GS-r *is much smaller than $1-\mu_2/(nL)$. However these example steps were all* **bad** *steps, as we will define in the following proof sketch, whose number we show can be bounded.*

We state an analogous linear rate for the box-constrained case too, but refer to Appendix B for the detailed algorithm and proof.

Theorem 2. *Suppose that f is coordinate-wise L smooth, and μ_1 strongly convex with respect to the $L1$ norm, for problem (1) with g encoding a box-constraint. After t steps of Algorithm 2 (the box analogon of Algorithm 1) where the coordinate i_t is chosen using the* GS-s *,* GS-r*, or* GS-q *rule, then*

$$f(\boldsymbol{\alpha}^{(t)}) - f(\boldsymbol{\alpha}^\star)$$

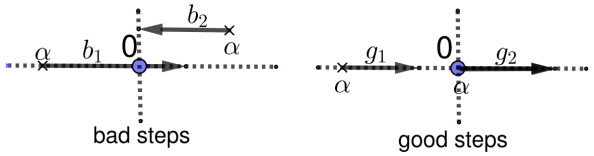

Figure 1: The arrows represent proximal coordinate updates $S_{\frac{\lambda}{L}}(\alpha_i - \frac{1}{L}\nabla_i f(\boldsymbol{\alpha}))$ from different starting points $\boldsymbol{\alpha}$. Updates which 'cross' (b_1) or 'end at' (b_2) the origin are **bad**, whereas the rest (g_1, g_2) are **good**.

$$\leq \left(1 - \max\left(\frac{1}{2n}, \frac{\mu_1}{L}\right)\right)^{t/2} \left(f(\boldsymbol{\alpha}^{(0)}) - f(\boldsymbol{\alpha}^\star)\right).$$

While the proof shares ideas with the $L1$-case, there are significant differences, e.g. the division of the steps into three categories: i) **good** steps which give a $(1-\mu_1/L)$ progress, ii) **bad** steps which may not give much progress but are bounded in number, and a third iii) **cross** steps which give a $(1-1/n)$ progress.

Remark 6. *For the box case, the greedy methods converge faster than UCD if $\mu_1 \geq 2\mu_2/n$, as before, and if $\mu_2/L \leq 1/4$. Typically, μ_2/L is much smaller than 1 and so the second condition is almost always satisfied. Hence we can expect greedy to be much faster in the box case, just as in the unconstrained smooth case. It remains unclear if the $1/n$ term truly affects the rate of convergence. For example, in the separated quadratic case considered in (Nutini et al., 2015, Sec. 4.1), $\mu_1/L \leq 1/n$ and so we can ignore the $1/n$ term in the rate (see Remark 16 in the Appendix).*

Proof sketch. While the full proofs are given in the appendix, we here give a sketch of the convergence of Algorithm 1 for $L1$-regularized problems in the strongly convex case, as in Theorem 1. The key idea is to partition the iterates into two sets: **good** and **bad** steps depending on whether they make (provably) sufficient progress. Then we show that the modification to the update we made in (12) ensures that we do not have too many **bad** steps. Since Algorithm 1 is a descent method, we can focus only on the **good** steps and describe its convergence. The "contradiction" to the convergence of GS-s provided in (Nutini et al., 2015, Section H.5, H.6) are in fact instances of **bad** steps.

The definitions of **good** and **bad** steps are explained in Fig. 1 (and formally in Def. 11). The core technical lemma below shows that in a **good** step, the update along the GS-s direction has an alternative characterization. For the sake of simplicity, let us assume that $\Theta = 1$ and that we use the exact GS-s coordinate.

Lemma 2. *Suppose that iteration t of Algorithm 1 updates coordinate i and that it was a* **good** *step. Then*

$$F(\boldsymbol{\alpha}^{(t+1)}) - F(\boldsymbol{\alpha}^{(t)}) \leq \min_{\boldsymbol{w} \in \mathbb{R}^n} \left\{ \left\langle \nabla f(\boldsymbol{\alpha}^{(t)}), \boldsymbol{w} - \boldsymbol{\alpha}^{(t)} \right\rangle \right.$$

$$+ \frac{L}{2}\|\boldsymbol{w}-\boldsymbol{\alpha}^{(t)}\|_1^2 + \lambda(\|\boldsymbol{w}\|_1 - \|\boldsymbol{\alpha}^{(t)}\|_1)\Big\}.$$

Proof sketch. We will only examine the case when $\boldsymbol{\alpha} > 0$ here for the sake of simplicity. Combining this with the assumption that iteration t was a `good` step gives that both $\alpha_i > 0$, $\alpha_i^+ > 0$, and $\alpha_i^+ = \alpha_i - \frac{1}{L}(\nabla_j f(\boldsymbol{\alpha}) + \lambda)$. Further if $\boldsymbol{\alpha} > 0$, the `GS-s` rule simplifies to $\arg\max_{j\in[n]} |\nabla_j f(\boldsymbol{\alpha}) + \lambda|$.

Since f is coordinate-wise smooth (2),

$$\begin{aligned}
&F(\boldsymbol{\alpha}^+) - F(\boldsymbol{\alpha}) \leq \\
&\quad (\nabla_j f(\boldsymbol{\alpha}))(\alpha_i^+ - \alpha_i) + \frac{L}{2}(\alpha_i^+ - \alpha_i)^2 + \lambda(|\alpha_i^+| - |\alpha_i|) \\
&\quad = -\frac{1}{L}(\nabla_j f(\boldsymbol{\alpha}))(\nabla_j f(\boldsymbol{\alpha}) + \lambda) + \\
&\qquad \frac{L}{2L^2}(\nabla_j f(\boldsymbol{\alpha}) + \lambda)^2 - \frac{\lambda}{L}(\nabla_j f(\boldsymbol{\alpha}) + \lambda) \\
&\quad = -\frac{1}{2L}(\nabla_j f(\boldsymbol{\alpha}) + \lambda)^2 .
\end{aligned}$$

But the `GS-s` rule exactly maximizes the last quantity. Thus we can continue:

$$\begin{aligned}
F(\boldsymbol{\alpha}^+) - F(\boldsymbol{\alpha}) &\leq -\frac{1}{2L}\|\nabla f(\boldsymbol{\alpha}) + \lambda \mathbf{1}\|_\infty^2 \\
&= \min_{\boldsymbol{w}\in\mathbb{R}^n} \left\{ \langle \nabla f(\boldsymbol{\alpha}) + \lambda\mathbf{1}, \boldsymbol{w}-\boldsymbol{\alpha}\rangle + \frac{L}{2}\|\boldsymbol{w}-\boldsymbol{\alpha}\|_1^2 \right\} \\
&= \min_{\boldsymbol{w}\in\mathbb{R}^n} \Big\{ \langle \nabla f(\boldsymbol{\alpha}), \boldsymbol{w}-\boldsymbol{\alpha}\rangle + \frac{L}{2}\|\boldsymbol{w}-\boldsymbol{\alpha}\|_1^2 \\
&\qquad + \lambda(\langle \mathbf{1}, \boldsymbol{w}\rangle - \langle \mathbf{1}, \boldsymbol{\alpha}\rangle) \Big\} .
\end{aligned}$$

Recall that $\boldsymbol{\alpha} > 0$ and so $\|\boldsymbol{\alpha}\|_1 = \langle \boldsymbol{\alpha}, \mathbf{1}\rangle$. Further for any $x \in \mathbb{R}$, $|x| \geq x$ and so $\langle \mathbf{1}, \boldsymbol{w}\rangle \leq \|\boldsymbol{w}\|_1$. This means

$$\lambda(\langle \mathbf{1}, \boldsymbol{w}\rangle - \langle \mathbf{1}, \boldsymbol{\alpha}\rangle) \leq \lambda(\|\boldsymbol{w}\|_1 - \|\boldsymbol{\alpha}\|_1) .$$

Plugging this into our previous equation gives us the lemma. See Lemma 8 for the full proof. □

If $\lambda = 0$ (i.e. F is smooth), Lemma 2 reduces to the 'refined analysis' of Nutini et al. (2015). We can now state the rate obtained in the strongly convex case.

Proof sketch for Theorem 1. Notice that if $\alpha_{i_t} = 0$, the step is necessarily `good` by definition (see Fig. 1). Since we start at the origin $\mathbf{0}$, the first time each coordinate is picked is a `good` step. Further, if some step t is `bad`, this implies that $\alpha_{i_t}^+$ 'crosses' the origin. In this case our modified update rule (12) sets the coordinate α_{i_t} to 0. The next time coordinate i_t is picked, the step is sure to be `good`. Thus in t steps, we have at least $\lceil t/2 \rceil$ `good` steps.

As per Lemma 2, every `good` step corresponds to optimizing the upper bound with the $L1$-squared regularizer. We can finish the proof:

$$\begin{aligned}
F(\boldsymbol{\alpha}^{(t+1)}) - F(\boldsymbol{\alpha}^{(t)}) &\leq \min_{\boldsymbol{w}\in\mathbb{R}^n} \Big\{ \Big\langle \nabla f(\boldsymbol{\alpha}^{(t)}, \boldsymbol{w}-\boldsymbol{\alpha}^{(t)} \Big\rangle \\
&\quad + \frac{L}{2}\|\boldsymbol{w}-\boldsymbol{\alpha}^{(t)}\|_1^2 + \lambda(\|\boldsymbol{w}\|_1 - \Big\|\boldsymbol{\alpha}^{(t)}\Big\|_1) \Big\} \\
&\overset{(a)}{\leq} \frac{\mu_1}{L} \min_{\boldsymbol{w}\in\mathbb{R}^n} \Big\{ \Big\langle \nabla f(\boldsymbol{\alpha}^{(t)}, \boldsymbol{w}-\boldsymbol{\alpha}^{(t)} \Big\rangle \\
&\quad + \frac{\mu_1}{2}\|\boldsymbol{w}-\boldsymbol{\alpha}^{(t)}\|_1^2 + \lambda(\|\boldsymbol{w}\|_1 - \Big\|\boldsymbol{\alpha}^{(t)}\Big\|_1) \Big\} \\
&\overset{(b)}{\leq} \frac{\mu_1}{L}\Big(F(\boldsymbol{\alpha}^\star) - F(\boldsymbol{\alpha}^{(t)})\Big) .
\end{aligned}$$

Inequality (a) follows from Karimireddy et al. (2018, Lemma 9), and (b) from strong convexity of f. Rearranging the terms above gives us the required linear rate of convergence. □

4.2 Sublinear convergence for general f

A sublinear convergence rate independent of n for GCD can be obtained when f is not strongly convex.

Theorem 3. *Suppose that F is coordinate-wise L smooth and convex, for g being an $L1$-regularizer or a box-constraint. Also let $\mathcal{Q}^\star$ be the set of minima of F with a minimum value $F^\star$. After t steps of Algorithm 1 or Algorithm 2 respectively, where the coordinate i_t is chosen using the `GS-s`, `GS-r`, or `GS-q` rule,*

$$F(\boldsymbol{\alpha}^{(t)}) - F^\star \leq \mathcal{O}\Big(\frac{LD^2}{t}\Big),$$

where D is the $L1$-diameter of the level set. For the set of minima $\mathcal{Q}^\star$,

$$D = \max_{\boldsymbol{w}\in\mathbb{R}^n} \min_{\boldsymbol{\alpha}^\star\in\mathcal{Q}^\star} \left\{ \|\boldsymbol{w}-\boldsymbol{\alpha}^\star\|_1 \,\middle|\, F(\boldsymbol{w}) \leq F(\boldsymbol{\alpha}^{(0)}) \right\}.$$

While a similar convergence rate was known for the `GS-r` rule (Dhillon et al., 2011), we here establish it for all three rules—even for approximate `GS-s`.

5 Maximum Inner Product Search

We now shift the focus from the theoretical rates to the actual implementation. A very important observation—as pointed out by Dhillon et al. (2011)—is that finding the steepest descent direction is closely related to a geometric problem. As an example consider the function $f(\boldsymbol{\alpha}) \overset{\text{def}}{=} \frac{1}{2}\|A\boldsymbol{\alpha} - \boldsymbol{b}\|^2$ for a data matrix $A \in \mathbb{R}^{d\times n}$. The gradient takes the form $\nabla f(\boldsymbol{\alpha}) = A^\top \boldsymbol{q}$ for $\boldsymbol{q} = (A\boldsymbol{\alpha} - \boldsymbol{b})$ and thus finding steepest coordinate direction is equal to finding the datapoint with the largest (in absolute value) inner product $\langle \boldsymbol{A}_i, \boldsymbol{q}\rangle$ with the query vector $\boldsymbol{q}$, which a priori requires the evaluation of all n scalar products. However, when we have to perform multiple similar queries (such as over the iterations of GCD), it is possible to pre-process the dataset A to speed up the query time. Note that we do not require the columns $\boldsymbol{A}_i$ to be normalized.

For the more general set of problems we consider here, we need the following slightly stronger primitive.

Definition 7 (S-MIPS)**.** *Given a set of m, d-dimensional points $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_m \in \mathbb{R}^d$, the* Subset Maxi-

mum Inner Product Search *or* S-MIPS *problem is to pre-process the set* $\mathcal{P}$ *such that for any query vector* $\boldsymbol{q}$ *and any subset of the points* $\mathcal{B} \subseteq [m]$*, the best point* $j \in \mathcal{B}$*,*

$$\text{S-MIPS}_{\mathcal{P}}(\boldsymbol{q}; \mathcal{B}) := \arg\max_{j \in \mathcal{B}} \{\langle \boldsymbol{p}_j, \boldsymbol{q} \rangle\} ,$$

can be computed with $o(m)$ *scalar product evaluations.*

State-of-the-art algorithms relax the exactness assumption and compute an approximate solution in time equivalent to a *sublinear* number of scalar product evaluations, i.e. $o(n)$ (e.g. (Charikar, 2002; Lv et al., 2007; Shrivastava and Li, 2014; Neyshabur and Srebro, 2015; Andoni et al., 2015)). We consciously refrain from stating more precise running times, as these will depend on the actual choice of the algorithm and the parameters chosen by the user. Our approach in this paper is transparent to the actual choice of S-MIPS algorithm, we only show how GCD steps can be *exactly cast* as such instances. By employing an arbitrary solver one thus gets a sublinear time approximate GCD update. An important caveat is that in subsequent queries, we will *adaptively* change the subset $\mathcal{B}$ based on the solution to the previous query. Hence the known theoretical guarantees shown for LSH do not directly apply, though the practical performance does not seem to be affected by this (see Appendix Fig. 13, 17). Practical details of efficiently solving S-MIPS are provided in Section 7.

6 Mapping GS-s to MIPS

We now move to our next contribution and show how the GS-s rule can be *efficiently* implemented. We aim to cast the problem of computing the GS-s update as an instance of MIPS (Maximum Inner Product Search), for which very fast query algorithms exist. In contrast, the GS-r and GS-q rules do not allow such a mapping. In this section, we will only consider objective functions of the following special structure:

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^n} \left\{ F(\boldsymbol{\alpha}) \overset{\text{def}}{=} \underbrace{l(A\boldsymbol{\alpha}) + \boldsymbol{c}^\top \boldsymbol{\alpha}}_{f(\boldsymbol{\alpha})} + \sum_{i=1}^{n} g_i(\alpha_i) \right\} . \quad (13)$$

The usual problems such as Lasso, dual SVM, or logistic regression, etc. have such a structure (see Appendix A.3).

Difficulty of the Greedy Rules. This section will serve to strongly motivate our choice of using the GS-s rule over the GS-r or GS-q. Let us pause to examine the three greedy selection rules and compare their relative difficulty. As a warm-up, consider again the smooth function $f(\boldsymbol{\alpha}) \overset{\text{def}}{=} \frac{1}{2} \|A\boldsymbol{\alpha} - \boldsymbol{b}\|^2$ for a data matrix $A \in \mathbb{R}^{d \times n}$ as introduced above in Section 5. We have observed that the steepest coordinate direction is

$$\arg\max_{j \in [n]} |\nabla_j f(\boldsymbol{\alpha})| \equiv \arg\max_{j \in [n]} \max_{s \in \{-1,1\}} \langle s\boldsymbol{A}_j, \boldsymbol{v} \rangle . \quad (14)$$

The formulation on the right is an instance of MIPS over the $2n$ vectors $\pm \boldsymbol{A}_i$. Now consider a non-smooth problem of the form $F(\boldsymbol{\alpha}) \overset{\text{def}}{=} \frac{1}{2} \|A\boldsymbol{\alpha} - \boldsymbol{b}\|^2 + \lambda \|\boldsymbol{\alpha}\|_1$. For simplicity, let us assume $\boldsymbol{\alpha} > 0$ and $L = 1$. In this case, the subgradient is $A^\top \boldsymbol{v} + \lambda \boldsymbol{1}$ and the GS-s rule is

$$\arg\max_{j \in [n]} \min_{s \in \delta |\alpha_j|} |\nabla_i f(\boldsymbol{\alpha}) + s| \equiv \arg\max_{j \in [n]} \max_{s \in \{-1,1\}} \langle s\boldsymbol{A}_j, \boldsymbol{v} \rangle + s\lambda . \quad (15)$$

The rule (15) is clearly not much harder than (14), and can be cast as a MIPS problem with minor modifications (see details in Sec. 6.1).

Let α_j^+ denote the proximal coordinate update along the j-th coordinate. In our case, $\alpha_j^+ = S_\lambda(\alpha_j - \langle \boldsymbol{A}_j, \boldsymbol{v} \rangle)$. The GS-r rule can now be 'simplified' as:

$$\arg\max_{j \in [n]} \begin{Bmatrix} \alpha_j, & \text{if } |\alpha_j - \langle \boldsymbol{A}_j, \boldsymbol{v} \rangle| \leq \lambda \\ \text{sign}(\alpha_j - \langle \boldsymbol{A}_j, \boldsymbol{v} \rangle), & \text{otherwise} . \end{Bmatrix} \quad (16)$$

It does not seem easy to cast (16) as a MIPS instance. It is even less likely that the GS-q rule which reads

$$\arg\min_{j \in [n]} \left\{ \nabla_j f(\boldsymbol{\alpha})(\alpha_j^+ - \alpha_j) + \tfrac{1}{2}(\alpha_j^+ - \alpha_j)^2 + \lambda(|\alpha_j^+| - \alpha_j) \right\}$$

can be mapped as to MIPS. This highlights the simplicity and usefulness of the GS-s rule.

6.1 Mapping $L1$-Regularized Problems

Here we focus on problems of the form (13) where $g(\boldsymbol{\alpha}) = \lambda \|\boldsymbol{\alpha}\|_1$. Again, we have $\nabla f(\boldsymbol{\alpha}) = A^\top \nabla l(\boldsymbol{v}) + \boldsymbol{c}$ where $\boldsymbol{v} = A\boldsymbol{\alpha}$.

For simplicity, let $\boldsymbol{\alpha} \neq 0$. Then the GS-s rule in (9) is

$$\arg\max_{j \in [n]} |s(\boldsymbol{\alpha})_j| = \arg\max_{j \in [n]} |\langle \boldsymbol{A}_j, \nabla l(\boldsymbol{v}) \rangle + c_j + \text{sign}(\alpha_j)\lambda| = \arg\max_{j \in [n]} \max_{s \in \pm 1} s \left[\langle \boldsymbol{A}_j, \nabla l(\boldsymbol{v}) \rangle + c_j + \text{sign}(\alpha_j)\lambda \right] . \quad (17)$$

We want to map the problem of the above form to a S-MIPS instance. Define for some $\beta > 0$, vectors

$$\tilde{\boldsymbol{A}}_j^{\pm} := \begin{pmatrix} \pm\beta, & \beta c_j, & \boldsymbol{A}_j \end{pmatrix}^\top , \quad (18)$$

and form a query vector $\boldsymbol{q}$ as

$$\boldsymbol{q} := \begin{pmatrix} \frac{\lambda}{\beta}, & \frac{1}{\beta}, & \nabla l(\boldsymbol{v}) \end{pmatrix}^\top . \quad (19)$$

A simple computation shows that the problem in (17) is equivalent to

$$\arg\max_{j \in [n]} \max_{s \in \pm 1} \left\langle s\tilde{\boldsymbol{A}}_j^{\text{sign}(\alpha_j)}, \boldsymbol{q} \right\rangle .$$

Thus by searching over a subset of vectors in $\{\pm \tilde{\boldsymbol{A}}_j^{\pm}\}$,

we can compute the GS-s direction. Dealing with the case where $\alpha_j = 0$ goes through similar arguments, and the details are outlined in Appendix E. Here we only state the resulting mapping.

The constant β in (18) and (19) is chosen to ensure that the entry is of the same order of magnitude on average as the rest of the coordinates of $\boldsymbol{A}_i$. The need for β only arises out of the performance concerns about the underlying algorithm to solve the S-MIPS instance. For example, β has no effect if we use exact search.

Formally, define the set $\mathcal{P} := \{\pm\tilde{\boldsymbol{A}}_j^{\pm} : j \in [n]\}$. Then at any iteration t with current iterate $\boldsymbol{\alpha}^{(t)}$, we also define $\mathcal{B}_t$ as $\mathcal{B}_t = \mathcal{B}_t^1 \cup \mathcal{B}_t^2 \cup \mathcal{B}_t^3 \cup \mathcal{B}_t^4$, where

$$\mathcal{B}_t^1 = \left\{\tilde{\boldsymbol{A}}_j^+ : \alpha_j^{(t)} > 0\right\}, \qquad \mathcal{B}_t^2 = \left\{-\tilde{\boldsymbol{A}}_j^+ : \alpha_j^{(t)} \geq 0\right\},$$
$$\mathcal{B}_t^3 = \left\{\tilde{\boldsymbol{A}}_j^- : \alpha_j^{(t)} \leq 0\right\}, \qquad \mathcal{B}_t^4 = \left\{-\tilde{\boldsymbol{A}}_j^- : \alpha_j^{(t)} < 0\right\}. \tag{20}$$

Lemma 3. *At any iteration t, for $\mathcal{P}$ and $\mathcal{B}_t$ as defined in (20), the query vector $\boldsymbol{q}_t$ as in (19), and $s(\boldsymbol{\alpha})$ as in (9) then the following are equivalent for $f(\boldsymbol{\alpha})$ is of the form $l(A\boldsymbol{\alpha}) + \boldsymbol{c}^\top\boldsymbol{\alpha}$:*

$$\text{S-MIPS}_{\mathcal{P}}(\boldsymbol{q}_t; \mathcal{B}_t) = \arg\max_i |s(\boldsymbol{\alpha})_i| .$$

The sets $\mathcal{B}_t$ and $\mathcal{B}_{t+1}$ differ in at most four points since $\boldsymbol{\alpha}^{(t)}$ and $\boldsymbol{\alpha}^{(t+1)}$ differ only in a single coordinate. This makes it computationally very efficient to incrementally maintain $\mathcal{B}_{t+1}$ and $\boldsymbol{\alpha}^{(t+1)}$ for $L1$-regularized problems.

6.2 Mapping Box-Constrained Problems

Using similar ideas, we demonstrate how to efficiently map problems of the form (13) where g enforces box constraints, such as for the dual SVM. The detailed approach is provided in Appendix B.1.

7 Experimental Results

Our experiments focus on the standard tasks of Lasso regression, as well as SVM training (on the dual objective). We refer the reader to Appendix A.3 for definitions. Lasso regression is performed on the rcv1 dataset while SVM is performed on w1a and the ijcnn1 datasets. All columns of the dataset (features for Lasso, datapoints for SVM) are normalized to unit length, allowing us to use the standard cosine-similarity algorithms nmslib (Boytsov and Naidan, 2013) to efficiently solve the S-MIPS instances. Note however that our framework is applicable without any normalization, if using a general MIPS solver instead.

We use the hnsw algorithm of the nmslib library with the default hyper-parameter value M and other parameters as in Table 1, selected by grid-search.[2] More details such as the meaning of these parameters can be found in the nmslib manual (Naidan and Boytsov, 2015, pp. 61). We exclude the time required for preprocessing of the datasets since it is amortized over the multiple experiments run on the same dataset (say for hyper-parameter tuning etc.). All our experiments are run on an Intel Xeon CPU E5-2680 v3 (2.50GHz, 30 MB cache) with 48 cores and 256GB RAM.

Table 1: Datasets and hyper-parameters: Lasso is run on rcv1, and SVM on w1a and ijcnn1. (**d**, **n**) is dataset size, the constant β from (18), (19) is set to $50/\sqrt{n}$, nmslib hyper-parameter M is set as a default, $efC = 100$. ρ is the density of the optimal solution.

Dataset	n	d	ρ	efS	post
rcv1, $\lambda = 1$	47,236	15,564	19%	100	2
rcv1, $\lambda = 10$	47,236	15,564	3%	400	2
w1a	2,477	300		100	0
ijcnn1	49,990	22		50	0

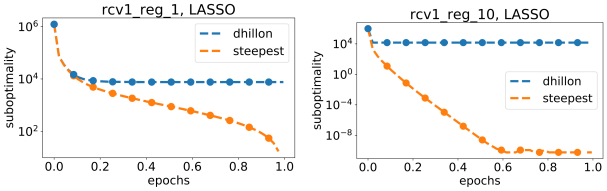

Figure 2: Evaluating dhillon: steepest which is based on the GS-s rule outperforms dhillon which quickly stagnates. Increasing the regularization, it stagnates in even fewer iterations.

First we compare the practical algorithm (dhillon) of Dhillon et al. (2011), which disregards the regularization part in choosing the next coordinate, and Algorithm 1 with GS-s rule (steepest) for Lasso regression. Note that dhillon is not guaranteed to converge. To compare the selection rules without biasing on the choice of the library, we perform exact search to answer the MIPS queries. As seen from Fig. 2, steepest significantly outperforms dhillon. In fact dhillon stagnates (though it does not diverge), once the error $f(\boldsymbol{\alpha})$ becomes small and the $L1$ regularization term starts playing a significant role. Increasing the regularization λ further worsens its performance. This is understandable since the rule used by dhillon ignores the $L1$ regularizer.

Next we compare our steepest strategy (Algorithms 1 and 2 using the GS-s rule), and the corresponding nearest-neighbor-based approximate versions (steepest-nn) against uniform, which picks coordinates uniformly at random. In all these experiments,

[2] A short overview of how to set these hyper-parameters can be found at https://github.com/nmslib/nmslib/blob/master/python_bindings/parameters.md.

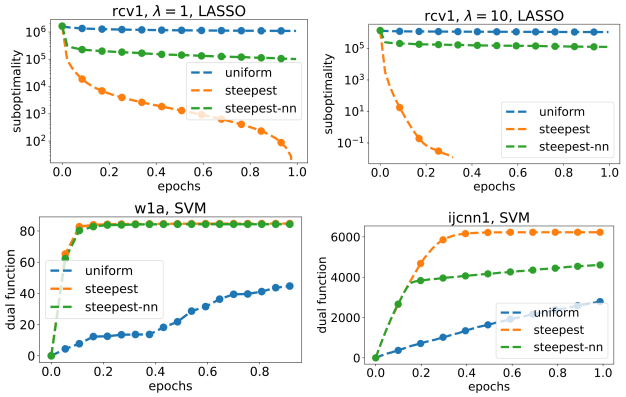

Figure 3: `steepest` as well `steepest-nn` significantly outperform `uniform` in number of iterations.

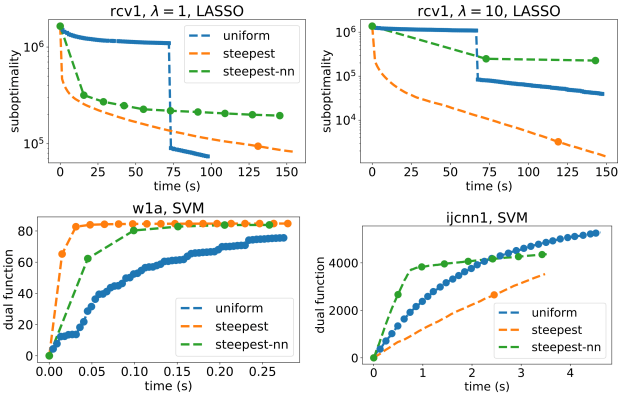

Figure 4: `steepest-nn` is very competitive and sometimes outperforms `uniform` even in terms of wall time especially towards the beginning. However eventually the performance of `uniform` is better than `steepest-nn`. This is because `nmslib` performs poorly as the norm of the gradient becomes small.

$\lambda \in \{1, 10\}$ for Lasso and at $1/n$ for SVM. Fig. 3 shows the clearly superior performance in terms of iterations of the `steepest` strategy as well as `steepest-nn` over `uniform` for both the Lasso as well as SVM problems. However, towards the end of the optimization i.e. in high accuracy regimes, `steepest-nn` fails to find directions substantially better than `uniform`. This is because towards the end, all components of the gradient $\nabla f(\boldsymbol{\alpha})$ become small, meaning that the query vector is nearly orthogonal to all points—a setting in which the employed nearest neighbor library `nmslib` performs poorly (Boytsov and Naidan, 2013).

Fig. 4 compares the wall-time performance of the `steepest`, `steepest-nn` and `uniform` strategies. This includes all the overhead of finding the descent direction. In all instances, the `steepest-nn` algorithm is competitive with `uniform` at the start, compensating for the increased time per iteration by increased progress per iteration. However towards the end `steepest-nn` gets comparable progress per iteration at a significantly larger cost, making its performance worse. With increasing sparsity of the solution (see Table 1 for sparsity levels), exact `steepest` rule starts to outperform `uniform` and `steepest-nn`.

Wall-time experiments (Fig. 4) show that `steepest-nn` always shows a significant performance gain in the important early phase of optimization, but in the later phase loses out to `uniform` due to the query cost and poor performance of `nmslib`. In practice, the recommended implementation is to use `steepest-nn` algorithm in the early optimization regime, and switch to uniform once the iteration cost outweighs the gain. In the Appendix (Fig. 13) we further investigate the poor quality of the solution provided by `nmslib`.

Repeating our experiments with other datasets, or using `FALCONN` (Andoni et al., 2015), another popular library for MIPS, yielded comparable results, provided in Appendix G.

8 Concluding Remarks

In this work we advocate the use of approximate `GS-s` selection rule for coordinate descent, and show its convergence for several important classes of problems for the first time, furthering our understanding of steepest descent on non-smooth problems. Our results demonstrate that significant speedups can be achieved by using the subgradient (4) to efficiently select the most 'important' coordinate at each iteration. Prior to our work, the analysis of greedy coordinate methods for composite functions mostly reverted to that of UCD or made unreasonable assumptions (e.g. Nutini et al. (2015); Zhang and Xiao (2017); Lei et al. (2017)). This is in stark contrast with smooth functions where many algorithms provably have rates better than UCD (e.g. Nutini et al. (2015, 2017); Nesterov and Stich (2017)). We believe our proof techniques can help bridge this gap between the analysis of greedy methods on smooth and non-smooth functions.

Our extensive numerical experiments also showcase the strengths and weaknesses of current state-of-the-art libraries for computing a Θ-approximate `GS-s` direction. As n grows, the cost per iteration for `nmslib` remains comparable to that of UCD, while the progress made per iteration increases. This means that as problem sizes grow, `GS-s` implemented via S-MIPS becomes an increasingly attractive approach. However, when the norm of the gradient becomes small, current state-of-the-art methods struggle to find directions substantially better than uniform. Alleviating this, and leveraging some of the very active development of recent alternatives to LSH as subroutines for our method is a promising direction for future work.

Acknowledgements. We thank Ludwig Schmidt for numerous useful discussions on LSH and using `FALCONN`. We also thank Vyacheslav Alipov for help with `nmslib`, Hadi Daneshmand for algorithmic insights, Mikkel Thorup for discussions on using hashing schemes in practice, and Mathurin Massias for feedback on our writeup. The feedback from many anonymous reviewers has also helped significantly improve the presentation.

References

Andoni, A., Indyk, P., Laarhoven, T., Razenshteyn, I., and Schmidt, L. (2015). Practical and Optimal LSH for Angular Distance. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 1225–1233. Curran Associates, Inc.

Bertsekas, D. P. and Tsitsiklis, J. N. (1989). *Parallel and distributed computation: numerical methods*, volume 23. Prentice hall Englewood Cliffs, NJ.

Bertsekas, D. P. and Tsitsiklis, J. N. (1991). Some aspects of parallel and distributed iterative algorithms—a survey. *Automatica*, 27(1):3–21.

Bickley, W. (1941). Relaxation methods in engineering science: A treatise on approximate computation.

Boytsov, L. and Naidan, B. (2013). Engineering efficient and effective Non-Metric Space Library. In Brisaboa, N., Pedreira, O., and Zezula, P., editors, *Similarity Search and Applications*, volume 8199 of *Lecture Notes in Computer Science*, pages 280–293. Springer Berlin Heidelberg.

Charikar, M. S. (2002). Similarity Estimation Techniques from Rounding Algorithms. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 380–388, New York, NY, USA. ACM.

Dhillon, I. S., Ravikumar, P. K., and Tewari, A. (2011). Nearest Neighbor based Greedy Coordinate Descent. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 24*, pages 2160–2168. Curran Associates, Inc.

Dünner, C., Parnell, T., and Jaggi, M. (2017). Efficient use of limited-memory accelerators for linear learning on heterogeneous systems. In *Advances in Neural Information Processing Systems*, pages 4258–4267.

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

Karimi, H., Nutini, J., and Schmidt, M. (2016). Linear convergence of gradient and proximal-gradient methods under the polyak-łojasiewicz condition. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 795–811. Springer.

Karimireddy, S. P. R., Stich, S., and Jaggi, M. (2018). Adaptive balancing of gradient and update computation times using global geometry and approximate subproblems. In *International Conference on Artificial Intelligence and Statistics*, pages 1204–1213.

Lei, Q., Yen, I. E., Wu, C.-y., Dhillon, I. S., and Ravikumar, P. (2017). Doubly greedy primal-dual coordinate descent for sparse empirical risk minimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2034–2042. JMLR. org.

Lin, Q., Lu, Z., and Xiao, L. (2014). An Accelerated Proximal Coordinate Gradient Method. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 3059–3067. Curran Associates, Inc.

Locatello, F., Raj, A., Karimireddy, S. P., Rätsch, G., Schölkopf, B., Stich, S., and Jaggi, M. (2018). On matching pursuit and coordinate descent. In *International Conference on Machine Learning*, pages 3204–3213.

Lu, H., Freund, R. M., and Mirrokni, V. (2018). Accelerating greedy coordinate descent methods. *arXiv preprint arXiv:1806.02476*.

Lv, Q., Josephson, W., Wang, Z., Charikar, M., and Li, K. (2007). Multi-probe lsh: efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd international conference on Very large data bases*, pages 950–961. VLDB Endowment.

Massias, M., Gramfort, A., and Salmon, J. (2018). Celer: a fast solver for the lasso with dual extrapolation. In *International Conference on Machine Learning*, pages 3321–3330.

Naidan, B. and Boytsov, L. (2015). Non-metric space library manual. *arXiv preprint arXiv:1508.05470*.

Ndiaye, E., Fercoq, O., Gramfort, A., and Salmon, J. (2015). Gap safe screening rules for sparse multi-task and multi-class models. In *Advances in Neural Information Processing Systems*, pages 811–819.

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

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

Neyshabur, B. and Srebro, N. (2015). On Symmetric and Asymmetric LSHs for Inner Product Search. In

ICML 2015 - Proceedings of the 32th International Conference on Machine Learning, pages 1926–1934.

Nutini, J., Laradji, I., and Schmidt, M. (2017). Let's make block coordinate descent go fast: Faster greedy rules, message-passing, active-set complexity, and superlinear convergence. *arXiv preprint arXiv:1712.08859*.

Nutini, J., Schmidt, M., Laradji, I. H., Friedlander, M., and Koepke, H. (2015). Coordinate Descent Converges Faster with the Gauss-Southwell Rule Than Random Selection. *arXiv:1506.00552 [cs, math, stat]*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830.

Perekrestenko, D., Cevher, V., and Jaggi, M. (2017). Faster Coordinate Descent via Adaptive Importance Sampling. In Singh, A. and Zhu, J., editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 869–877, Fort Lauderdale, FL, USA. PMLR.

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

Shalev-Shwartz, S. and Zhang, T. (2013a). Accelerated Mini-batch Stochastic Dual Coordinate Ascent. In *Proceedings of the 26th International Conference on Neural Information Processing Systems*, NIPS'13, pages 378–385, USA. Curran Associates Inc.

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

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

Shrivastava, A. and Li, P. (2014). Asymmetric LSH (ALSH) for Sublinear Time Maximum Inner Product Search (MIPS). In *NIPS 2014 - Advances in Neural Information Processing Systems 27*, pages 2321–2329.

Song, C., Cui, S., Jiang, Y., and Xia, S.-T. (2017). Accelerated stochastic greedy coordinate descent by soft thresholding projection onto simplex. In *Advances in Neural Information Processing Systems*, pages 4838–4847.

Stich, S. U., Raj, A., and Jaggi, M. (2017a). Approximate Steepest Coordinate Descent. *ICML - Proceedings of the 34th International Conference on Machine Learning*.

Stich, S. U., Raj, A., and Jaggi, M. (2017b). Safe Adaptive Importance Sampling. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 4384–4394. Curran Associates, Inc.

Warga, J. (1963). Minimizing certain convex functions. *Journal of the Society for Industrial and Applied Mathematics*, 11(3):588–593.

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

You, Y., Lian, X., Liu, J., Yu, H.-F., Dhillon, I. S., Demmel, J., and Hsieh, C.-J. (2016). Asynchronous parallel greedy coordinate descent. In *Advances in Neural Information Processing Systems*, pages 4682–4690.

Zhang, Y. and Xiao, L. (2017). Stochastic primal-dual coordinate method for regularized empirical risk minimization. *The Journal of Machine Learning Research*, 18(1):2939–2980.