Randomized Coordinate Descent Methods for Big Data Optimization

THE UNIVERSITY
of EDINBURGH

Doctoral thesis

Martin Takáč

Supervisor:
Dr. Peter Richtárik

Doctor of Philosophy
The University of Edinburgh
2014

Randomized Coordinate Descent Methods for Big Data Optimization

Doctoral thesis

Martin Takáč

School of Mathematics
The University of Edinburgh

Supervisor:
Dr. Peter Richtárik

Edinburgh 2014

Randomized Coordinate Descent Methods
for Big Data Optimization

Martin Takáč
E-mail: *Takac.MT@gmail.com*
Website: `http://mtakac.com`

Dr. Peter Richtárik
E-mail: *Peter.Richtarik@ed.ac.uk*
Website: `http://www.maths.ed.ac.uk/~prichtar`

School of Mathematics
The University of Edinburgh
James Clerk Maxwell Building
The King's Buildings
Mayfield Road
Edinburgh
Scotland EH9 3JZ

Doctoral thesis
Compilation date: May 11, 2014
Typeset in LaTeX

To my wife Mária,

son Martin

and everybody I love.

Declaration

The candidate confirms that the work submitted is his own, except where work which has formed part of jointly-authored publications has been included. The contribution of the candidate and the others authors to this work has been explicitly indicated below. The candidate confirms that appropriate credit has been given within the thesis where reference has been made to the work of others.

The table below indicates which chapters are based on a work from jointly-authored publications. The candidate contributed to every result of all papers this thesis is based on.

Chapter	1	2	3	4	5
Ref.	[53, 54, 55]	[53]	[54, 56]	[55]	[66]

(Martin Takáč, May 11, 2014)

Acknowledgements

> God gave you a gift of 86,400
> seconds today. Have you used
> one to say 'thank you?'
>
> — William Arthur Ward
> (1921–1994)

I would like to express my special appreciation and thanks to my advisor Dr. Peter Richtárik. Peter, you have been a tremendous mentor for me! You have shown the attitude and the substance of a genius: you continually and persuasively conveyed a spirit of adventure in regard to the research. You were constantly encouraging my research and also allowed me to grow as a research scientist. You taught me how to deliver the best talk, you advised me how to prepare the perfect slides, you have spent so much time listening to my practice talks, suggesting improvements and offering career advice. All you did for me, it has all been priceless. With you research was, indeed, fun and a joy and hence it has become almost a leisure activity!

I would like to thank my second supervisor Professor Jacek Gondzio, for his helpfulness and willingness to chat about research. Your advice was very helpful in my formation as a scientist.

Now, I would like to thank several people:

- I also want to say big "Thank you!" to doc. Dr. Milan Hamala, Professor Daniel Ševčovič, Professor Pavel Brunovský and doc. Dr. Margaréta Halická, who initiated and helped to grow my desire in optimization and research in general. I am sure that without them, I wouldn't have started on the academic career-path. Your support and constant interest in my progress and achievements during my PhD study were very encouraging!

- I am also very grateful to many researchers I had the chance to collaborate with. Especially to Professor Yurii Nesterov, Professor François Glineur, Professor Nathan Srebro, Dr. Selin Damla Ahipaşaoğlu, Dr. Ngai-Man Cheung, Dr. Olivier Devolder, Dr. Jakub Mareček and Avleen Bijral.

- I feel indebted to Dr. Rachael Tappenden for many interesting chats, research collaborations, minisymposia co-organization and all the time she spent with proofreading my slides and papers! Thank you Rachael, your help was extremely useful and constructive. You were always willing to help with everything.

- I am filled with gratitude for many interesting discussions and chats with researchers including Professor Stephen Wright, Professor Michael J. D. Powell, Professor Roger Fletcher, Dr. Michel Baes and to all of my colleagues including Professor Jared Tanner, Dr. Coralia Cartis, Dr. Andreas Grothey, Dr. Julian Hall, Professor Ken McKinnon, Dr. Martin Lotz, Dr. Pavel G. Zhlobich and Kimonas Fountoulakis for many interesting talks and ideas.

I am grateful for all the financial support I got, especially from the Centre for Numerical Algorithms and Intelligent Software (funded by EPSRC grant EP/G036136/1) and Scottish Funding Council. I have been also supported by following grants: EPSRC grants EP/J020567/1 (Algorithms for Data Simplicity), EP/I017127/1 (Mathematics for Vast Digital Resources) and EPSRC First Grant EP/K02325X/1 (Accelerated Coordinate Descent Methods for Big Data Problems).

Furthermore, I would also like to thank my parents for their endless love and support.

Finally, and most importantly to me, I would like to express appreciation to my beloved wife PharmDr. Mária Takáčová who spent many sleepless nights (especially when I was travelling far from home) and was always my support in the moments when there was no one to answer my queries. My love, I am sure, I would not have made it thus far! Your help, support, and your faith in me was the fuel which makes me go. I will be grateful forever for your love. Thank you!

Lay Summary of Thesis

In today's digital world there is an ever increasing demand for solving "big data" problems, each described by gigabytes or terabytes of data from sources such as twitter feeds, online image databases, text corpora, videos, government records, scientific experiments or click behaviour of online users. In many applications the problem at hand is formulated as an optimization problem: we seek to determine a number of variables minimizing a certain loss function (or maximizing a profit function), subject to constraints.

Classical optimization algorithms tend to require only a small number of iterations and output high precision result, but this comes at the cost of data-heavy iterations. For example, interior point methods require the solution of an $n \times n$ system of equations, which in general takes $\mathcal{O}(n^3)$ floating point operations (additions or multiplications). If $n = 10^9$ for instance, Gaussian elimination would require 471 years of computation on the currently ranked #1 supercomputer in the world (Tianhe-2) using all of its 3,120,000 cores.

In the big data setting it is often prohibitive or downright impossible to perform even a single iteration of the classical methods, and hence focus is shifting to algorithms which are able to use less information to produce an iteration, thus making the iterative process feasible. This comes at the cost of increasing the number of iterations. In Truss Topology Design , for instance, each iteration of a coordinate descent method (CDM) consists of as little as 8 multiplications and 8 additions, and does not depend on the size of the problem. On the other hand, the number of iterations will depend on the problem size.

In this thesis we analyse iteration complexity (how many iterations of the algorithm is sufficient to to obtain sufficient solution) of coordinate descent methods for various loss functions. Moreover, in order to make use of the modern high-performance computers, the parallel version of CDM is proposed and analysed.

Abstract

> The universe is governed by science. But science tells us that we can't solve the equations, directly in the abstract.
>
> — Stephen Hawking (1942)

This thesis consists of 5 chapters. We develop new serial (Chapter 2), parallel (Chapter 3), distributed (Chapter 4) and primal-dual (Chapter 5) stochastic (randomized) coordinate descent methods, analyze their complexity and conduct numerical experiments on synthetic and real data of huge sizes (GBs/TBs of data, millions/billions of variables).

In Chapter 2 we develop a randomized coordinate descent method for minimizing the sum of a smooth and a simple nonsmooth separable convex function and prove that it obtains an ϵ-accurate solution with probability at least $1-\rho$ in at most $O((n/\epsilon)\log(1/\rho))$ iterations, where n is the number of blocks. This extends recent results of Nesterov [43], which cover the smooth case, to composite minimization, while at the same time improving the complexity by the factor of 4 and removing ϵ from the logarithmic term. More importantly, in contrast with the aforementioned work in which the author achieves the results by applying the method to a regularized version of the objective function with an unknown scaling factor, we show that this is not necessary, thus achieving first true iteration complexity bounds. For strongly convex functions the method converges linearly. In the smooth case we also allow for arbitrary probability vectors and non-Euclidean norms. Our analysis is also much simpler.

In Chapter 3 we show that the randomized coordinate descent method developed in Chapter 2 can be accelerated by parallelization. The speedup, as compared to the serial method, and referring to the number of iterations needed to approximately solve the problem with high probability, is equal to the product of the number of processors and a natural and easily

computable measure of separability of the smooth component of the objective function. In the worst case, when no degree of separability is present, there is no speedup; in the best case, when the problem is separable, the speedup is equal to the number of processors. Our analysis also works in the mode when the number of coordinates being updated at each iteration is random, which allows for modeling situations with variable (busy or unreliable) number of processors. We demonstrate numerically that the algorithm is able to solve huge-scale ℓ_1-regularized least squares problems with a billion variables.

In Chapter 4 we extended coordinate descent into a distributed environment. We initially partition the coordinates (features or examples, based on the problem formulation) and assign each partition to a different node of a cluster. At every iteration, each node picks a random subset of the coordinates from those it owns, independently from the other computers, and in parallel computes and applies updates to the selected coordinates based on a simple closed-form formula. We give bounds on the number of iterations sufficient to approximately solve the problem with high probability, and show how it depends on the data and on the partitioning. We perform numerical experiments with a LASSO instance described by a 3TB matrix.

Finally, in Chapter 5, we address the issue of using mini-batches in stochastic optimization of Support Vector Machines (SVMs). We show that the same quantity, the *spectral norm of the data*, controls the parallelization speedup obtained for both primal stochastic subgradient descent (SGD) and stochastic dual coordinate ascent (SCDA) methods and use it to derive novel variants of mini-batched (parallel) SDCA. Our guarantees for both methods are expressed in terms of the original nonsmooth primal problem based on the hinge-loss.

Our results in Chapters 2 and 3 are cast for blocks (groups of coordinates) instead of coordinates, and hence the methods are better described as *block* coordinate descent methods. While the results in Chapters 4 and 5 are not formulated for blocks, they can be extended to this setting.

Contents

List of Tables

List of Figures

List of Algorithms

1

Introduction

> I have not failed. I've just found
> 10,000 ways that won't work.
>
> — Thomas Alva Edison
> (1847-1931)

1.1 Optimization

In mathematics, optimization is the process of selecting the best decision x^* from a set of available (feasible) decisions. We can associate with every admissible (feasible) decision $x \in X$ a real-valued function $F(x)$ which measures the quality of x (lower values correspond to better decisions). Optimization thus seeks to find a minimizer of F on X. The problem of finding the exact minimizer can be very hard (maybe due to the *size* or *structure* of the function F, or the exact solution may not be obtainable in a closed form). In practice, an approximate solution x^ϵ (where $F(x^\epsilon) \leq F(x^*) + \epsilon$) is sufficient, where ϵ is our target approximation level. The choice of ϵ is problem specific. In many machine learning problems, $\epsilon = 10^{-2}$ can already be sufficient. The approximate solution can be obtained by an iterative process (optimization algorithm), where we start with some starting solution and then in each iteration find a new solution with better quality (lower function value).

Big data optimization. Recently there has been a surge in interest in the design of algorithms suitable for solving convex optimization problems with a huge number of variables [53, 44]. Indeed, the size of problems arising in fields such as machine learning [5], network analysis [85], PDEs [79], truss topology design [52] and compressed sensing [26] usually grows with our capacity to solve them, and is projected to grow dramatically in the next decade. In fact, much of computational science is currently facing the “big data” challenge, and this work is aimed at developing optimization algorithms suitable for the task.

Our goals. The goal of this thesis, in the broadest sense, is to develop efficient methods for solving structured convex optimization problems with some or all of these (not necessarily distinct) properties:

1. **Size of Data.** The size of the problem, measured as the dimension of the variable of interest, is so large that the computation of a single function value or gradient is prohibitive. There are several situations in which this is the case, let us mention two of them.
    - **Memory.** If the dimension of the space of variables is larger than the available memory, the task of forming a gradient or even of evaluating the function value may be impossible to execute and hence the usual gradient methods will not work.
    - **Patience.** Even if the memory does not preclude the possibility of taking a gradient step, for large enough problems this step will take considerable time and, in some applications such as image processing, users might prefer to see/have some intermediary results before a single iteration is over.
2. **Nature of Data.** The nature and structure of data describing the problem may be an obstacle in using current methods for various reasons, including the following.
    - **Completeness.** If the data describing the problem is not immediately available in its entirety, but instead arrives incomplete in pieces and blocks over time, with each block “corresponding to” one variable, it may not be realistic (for various reasons such as “memory” and “patience” described above) to wait for the entire data set to arrive before the optimization process is started.
    - **Source.** If the data is distributed on a network not all nodes of which are equally responsive or functioning, it may be necessary to work with whatever data is available at a given time.

It appears that a very reasonable approach to solving *some* problems characterized above is to use *(block) coordinate descent methods* (CD). In this thesis we design new serial, parallel, distributed and primal-dual stochastic (block) coordinate descent methods.

1.2 Stochastic Block Coordinate Descent Methods

The basic algorithmic strategy of CD methods is known in the literature under various names such as alternating minimization, coordinate relaxation, linear and non-linear Gauss-Seidel methods, subspace correction and domain decomposition. As working with all the variables of an optimization problem at each iteration may be inconvenient, difficult or impossible for any or all of the reasons mentioned above, the variables are partitioned into manageable blocks, with each iteration focused on updating a single block only, the remaining blocks being fixed. Both for their conceptual and algorithmic simplicity, CD methods were among the first optimization approaches proposed and studied in the literature (see [2] and the references therein; for a survey of block CD methods in semidefinite programming we refer the reader to [75]). While they seem to have never belonged to the mainstream focus of the optimization community, a renewed interest in CD methods was sparked recently by their successful application in several areas—training support vector machines in machine learning [7, 4], [61], [82], [81], optimization [31, 35, 34, 45, 36, 37, 46, 25, 22, 28, 74, 72, 84, 58, 43, 76], compressed sensing [26], regression [78], protein loop closure [6] and truss topology design [52]—partly due to a change in the *size* and *nature of data* described above.

Order of coordinates. Efficiency of a CD method will necessarily depend on the balance between time spent on choosing the block to be updated in the current iteration and the quality of this choice in terms of function value decrease. One extreme possibility is a *greedy* strategy in which the block with the largest descent or guaranteed descent is chosen. In our setup such a strategy is prohibitive as i) it would require all data to be available and ii) the work involved would be excessive due to the size of the problem. Even if one is able to compute all partial derivatives, it seems better to then take a full gradient step instead of a coordinate one, and avoid throwing almost all of the computed information away. On the other end of the spectrum are two very cheap strategies for choosing the incumbent coordinate: *cyclic* and *random*. Surprisingly, it appears that complexity analysis of a cyclic CD method in satisfying generality has not yet been done. The only attempt known to us is the work of Saha and Tewari [58]; the authors consider the case of minimizing a smooth convex function and proceed by establishing a sequence of comparison theorems between the iterates of their method and the iterates of a simple gradient method. Their result requires an isotonicity assumption[1]. Note that a cyclic strategy assumes that the data describing the next block is available when needed which may not always be realistic. The situation with a random strategy seems better; here

---

[1] A function $F : \mathbb{R}^N \to \mathbb{R}$ is *isotone* if $x \geq y$ implies $F(x) \geq F(y)$.

are some of the reasons:

(i) Recent efforts suggest that complexity results are perhaps more readily obtained for randomized methods and that randomization can actually improve the convergence rate [65], [23], [61].

(ii) Choosing all blocks with equal probabilities should, intuitively, lead to similar results as is the case with a cyclic strategy. In fact, a randomized strategy is able to avoid worst-case order of coordinates, and hence might be preferable.

(iii) Randomized choice seems more suitable in cases when not all data is available at all times.[2]

(iv) One may study the possibility of choosing blocks with different probabilities (we do this in Section 2.5). The goal of such a strategy may be either to improve the speed of the method (in Section 2.7.1 we introduce a speedup heuristic based on adaptively changing the probabilities), or a more realistic modeling of the availability frequencies of the data defining each block.

Step size. Once a coordinate (or a block of coordinates) is chosen to be updated in the current iteration, partial derivative can be used to drive the steplength in the same way as it is done in the usual gradient methods. As it is sometimes the case that the computation of a partial derivative is *much cheaper and less memory demanding* than the computation of the entire gradient, CD methods seem to be promising candidates for problems described above. It is important that line search, if any is implemented, is very efficient. The entire data set is either huge or not available and hence it is not reasonable to use function values at any point in the algorithm, including the line search. Instead, cheap partial derivative and other information derived from the problem structure should be used to drive such a method.

Coordinate descent methods. Coordinate descent methods (CDM) are one of the most successful classes of algorithms in the big data optimization domain. Broadly speaking, CDMs are based on the strategy of updating a single coordinate (or a single block of coordinates) of the vector of variables at each iteration. This often drastically reduces memory requirements as well as the arithmetic complexity of a single iteration, making the methods easily implementable and

---

[2] If data were not available in memory and one would like to follow, e.g., cyclic order or greedy selection of coordinates, then one would have to wait until the required data are fetched from a source. In contrast, in many applications with streaming data, e.g., involving on-line support vector machine (SVM), one may plausibly assume that the stream of in-coming data is sufficiently random to see a sequential choice of from the stream as a random selection.

scalable. In certain applications, a single iteration can amount to as few as 4 multiplications and additions only [52]! On the other hand, many more iterations are necessary for convergence than it is usual for classical gradient methods. Indeed, the number of iterations a CDM requires to solve a smooth convex optimization problem is $O(\frac{n\tilde{L}R^2}{\epsilon})$, where ϵ is the error tolerance, n is the number variables (or blocks of variables), $\tilde{L}$ is the average of the Lipschitz constants of the gradient of the objective function associated with the variables (blocks of variables) and R is the distance from the starting iterate to the set of optimal solutions. On balance, as observed by numerous authors, serial CDMs are much more efficient for big data optimization problems than most other competing approaches, such as gradient methods [43, 52].

Parallelization. We wish to point out that for truly huge-scale problems it is absolutely necessary to *parallelize*. This is in line with the rise and ever increasing availability of high performance computing systems built around multi-core processors, GPU-accelerators and computer clusters, the success of which is rooted in massive parallelization. This simple observation, combined with the remarkable scalability of serial CDMs, leads to our belief that the study of parallel coordinate descent methods (PCDMs) is a very timely topic.

1.3 A Brief Overview of the Thesis

Section 1.4 introduces the main problem we deal with in the whole thesis. Then the basic notation which is shared in all other chapters is introduced. Afterwards, we state one very important technical result (Theorem 1) which will be used to derive all high probability convergence results in this thesis. Chapter 2 introduces the generic serial coordinate descent algorithm and analyzes its convergence for both its uniform and nonuniform variant. In Chapter 3 we develop parallel variants of the methods, the analysis of which depends on the notion of ESO, which is a novel technical tool developed in [54]. This ESO concept enables us to analyze parallel coordinate descent algorithms (PCDM) and derive iteration complexity results. It turns out that ESO is so powerful that also allows us to analyze a distributed modification of PCDM, which we call Hydra, in Chapter 4. In Chapter 5 we focus on the problem of optimization of Support Vector Machines (SVMs). We apply the ESO concept and show that the parallelization speedup obtained for both primal stochastic subgradient descent (SGD) and stochastic dual coordinate ascent (SCDA) is driven by the same quantity, which depends on the spectral norm of the data. Our mini-batch SDCA method is the first parallel primal-dual coordinate descent method in the literature.

1.4 The Optimization Problem

In this thesis we study the *iteration complexity* of randomized block coordinate descent methods applied to the problem of minimizing a *composite objective function*, i.e., a function formed as the sum of a smooth convex and a simple nonsmooth convex term:

$$\min_{x \in \mathbb{R}^N} F(x) \overset{\text{def}}{=} f(x) + \Psi(x). \tag{1.1}$$

We assume that this problem has a minimum ($F^* > -\infty$), f has (block) coordinate Lipschitz continuous gradient, and Ψ is a (block) separable proper closed convex extended real valued function (block separability will be defined precisely in Section 1.5). Possible choices of Ψ include:

(i) $\Psi \equiv 0$. This covers the case of *smooth minimization* and was considered in [43].

(ii) Ψ is the indicator function of a block-separable convex set (a box), i.e.,

$$\Psi(x) = I_{S_1 \times \cdots \times S_n}(x) \overset{\text{def}}{=} \begin{cases} 0 & \text{if} \quad x^{(i)} \in S_i \quad \forall i, \\ +\infty & \text{otherwise}, \end{cases}$$

where $x^{(i)}$ is block i of $x \in \mathbb{R}^N$ (to be defined precisely in Section 1.5) and $S_1, \dots, S_n$ are closed convex sets. This choice of Ψ models problems with *smooth objective and convex constraints on blocks of variables.* Indeed, (1.1) takes on the form

$$\min \ f(x) \quad \text{subject to} \quad x^{(i)} \in S_i, \quad i = 1, \dots, n.$$

Iteration complexity results in this case were given in [43].

(iii) $\Psi(x) \equiv \lambda \|x\|_1$ for $\lambda > 0$. In this case we decompose $\mathbb{R}^N$ into N blocks, each corresponding to one coordinate of x. Increasing λ encourages the solution of (1.1) to be sparser [77]. Applications abound in, for instance, machine learning [7], statistics [71] and signal processing [26]. The first iteration complexity results for the case with a single block were given in [42].

(iv) There are many more choices such as the elastic net [87], group lasso [83], [32], [48] and sparse group lasso [18]. One may combine indicator functions with other block separable functions such as $\Psi(x) = \lambda \|x\|_1 + I_{S_1 \times \cdots \times S_n}(x)$, $S_i = [l_i, u_i]$, where the sets introduce lower and upper bounds on the coordinates of x.

1.5 Notation and Assumptions

Some elements of the setup described in this section was initially used in the analysis of block coordinate descent methods by Nesterov [43] (e.g., block structure, weighted norms and block Lipschitz constants).

1.5.1 Block Structure

The block structure of (1.1) is given by a decomposition of $\mathbb{R}^N$ into n subspaces as follows. Let $U \in \mathbb{R}^{N\times N}$ be a column permutation[3] of the $N \times N$ identity matrix and further let $U = [U_1, U_2, \dots, U_n]$ be a decomposition of U into n submatrices, with U_i being of size $N \times N_i$, where $\sum_i N_i = N$.

Proposition 1 (Block decomposition[4])**.** *Any vector $x \in \mathbb{R}^N$ can be written uniquely as*

$$x = \sum_{i=1}^{n} U_i x^{(i)}, \tag{1.2}$$

where $x^{(i)} \in \mathbb{R}^{N_i}$. Moreover, $x^{(i)} = U_i^T x$.

Proof. Noting that $UU^T = \sum_i U_i U_i^T$ is the $N \times N$ identity matrix, we have $x = \sum_i U_i U_i^T x$. Let us now show uniqueness. Assume that $x = \sum_i U_i x_1^{(i)} = \sum_i U_i x_2^{(i)}$, where $x_1^{(i)}, x_2^{(i)} \in \mathbb{R}^{N_i}$. Since

$$U_j^T U_i = \begin{cases} N_j \times N_j \quad \text{identity matrix}, & \text{if } i = j, \\ N_j \times N_i \quad \text{zero matrix}, & \text{otherwise}, \end{cases} \tag{1.3}$$

for every j we get $0 = U_j^T(x - x) = U_j^T \sum_i U_i(x_1^{(i)} - x_2^{(i)}) = x_1^{(j)} - x_2^{(j)}$. □

In view of the above proposition, from now on we write $x^{(i)} \overset{\text{def}}{=} U_i^T x \in \mathbb{R}^{N_i}$, and refer to $x^{(i)}$ as the *i-th block* of x. The definition of partial separability in the introduction is with respect to these blocks. For simplicity, we will sometimes write $x = (x^{(1)}, \dots, x^{(n)})$.

Example 1. *Let $n = N$, $N_i = 1$ for all i and $U = [e_1, e_2, \dots, e_n]$ be the $n \times n$ identity matrix. Then $U_i = e_i$ is the i-th unit vector and $x^{(i)} = e_i^T x \in \mathbb{R}_i = \mathbb{R}$ is the i-th coordinate of x. Also, $x = \sum_i e_i x^{(i)}$.*

---

[3] The reason why we work with a *permutation* of the identity matrix, rather than with the identity itself, as in [43], is to enable the blocks being formed by *nonconsecutive* coordinates of x. This way we establish notation which makes it possible to work with (i.e., analyze the properties of) multiple block decompositions, for the sake of picking the best one, subject to some criteria. Moreover, in some applications the coordinates of x have a natural ordering to which the natural or efficient block structure does not correspond.

[4] This is a straightforeard result; we do not claim any novelty and include it solely for the benefit of the reader.

1.5.2 Global Structure

Inner products. The standard Euclidean inner product in spaces $\mathbb{R}^N$ and $\mathbb{R}^{N_i}$, $i \in [n]$, will be denoted by $\langle \cdot, \cdot \rangle$ and $[n]$ will denote a set $\{1, 2, \ldots, n\}$. Letting $x, y \in \mathbb{R}^N$, the relationship between these inner products is given by

$$\langle x, y\rangle \overset{(1.2)}{=} \langle \sum_{j=1}^{n} U_j x^{(j)}, \sum_{i=1}^{n} U_i y^{(i)} \rangle = \sum_{j=1}^{n} \sum_{i=1}^{n} \langle U_i^T U_j x^{(j)}, y^{(i)} \rangle \overset{(1.3)}{=} \sum_{i=1}^{n} \langle x^{(i)}, y^{(i)} \rangle.$$

For any $w \in \mathbb{R}^n$ and $x, y \in \mathbb{R}^N$ we further define

$$\langle x, y\rangle_w \overset{\text{def}}{=} \sum_{i=1}^{n} w_i \langle x^{(i)}, y^{(i)} \rangle. \tag{1.4}$$

For vectors $z = (z_1, \ldots, z_n)^T \in \mathbb{R}^n$ and $w = (w_1, \ldots, w_n)^T \in \mathbb{R}^n$ we write $w \odot z \overset{\text{def}}{=} (w_1 z_1, \ldots, w_n z_n)^T$ and for $u \in \mathbb{R}^n_{++}$ we define $u^{-1} \overset{\text{def}}{=} (1/u_1, \ldots, 1/u_n)^T$.

Norms. Spaces $\mathbb{R}^{N_i}$, $i \in [n]$, are equipped with a pair of conjugate norms: $\|t\|_{(i)} \overset{\text{def}}{=} \langle B_i t, t\rangle^{1/2}$, where B_i is an $N_i \times N_i$ positive definite matrix and $\|t\|^*_{(i)} \overset{\text{def}}{=} \max_{\|s\|_{(i)} \leq 1} \langle s, t\rangle = \langle B_i^{-1} t, t\rangle^{1/2}$, $t \in \mathbb{R}^{N_i}$. For $w \in \mathbb{R}^n_{++}$, define a pair of conjugate norms in $\mathbb{R}^N$ by

$$\|x\|_w = \left[\sum_{i=1}^{n} w_i \|x^{(i)}\|^2_{(i)} \right]^{1/2}, \quad \|y\|^*_w \overset{\text{def}}{=} \max_{\|x\|_w \leq 1} \langle y, x\rangle = \left[\sum_{i=1}^{n} w_i^{-1} (\|y^{(i)}\|^*_{(i)})^2 \right]^{1/2}. \tag{1.5}$$

Note that these norms are induced by the inner product (1.4) and the matrices $B_1, \ldots, B_n$. For a vector w we will define a diagonal matrix $W = \operatorname{diag}(w_1, \ldots, w_n)$ and we write $\operatorname{tr}(W) = \sum_i w_i$.

Often we will use $w = L \overset{\text{def}}{=} (L_1, L_2, \ldots, L_n)^T \in \mathbb{R}^n$, where the constants L_i are defined below.

1.5.3 Smoothness of f

We assume throughout the thesis that the gradient of f is block coordinate-wise Lipschitz continuous, uniformly in x, with positive constants $L_1, \ldots, L_n$, i.e., that for all $x \in \mathbb{R}^N$, $i = 1, 2, \ldots, n$ and $t \in \mathbb{R}_i$ we have

$$\|\nabla_i f(x + U_i t) - \nabla_i f(x)\|^*_{(i)} \leq L_i \|t\|_{(i)}, \tag{1.6}$$

where

$$\nabla_i f(x) \overset{\text{def}}{=} (\nabla f(x))^{(i)} = U_i^T \nabla f(x) \in \mathbb{R}_i. \tag{1.7}$$

An important consequence of (1.6) is the following standard inequality [41]:

$$f(x + U_i t) \leq f(x) + \langle \nabla_i f(x), t \rangle + \tfrac{L_i}{2}\|t\|_{(i)}^2. \tag{1.8}$$

1.5.4 Separability of Ψ

We assume that $\Psi : \mathbb{R}^N \to \mathbb{R} \cup \{+\infty\}$ is block separable, i.e., that it can be decomposed as follows:

$$\Psi(x) = \sum_{i=1}^{n} \Psi_i(x^{(i)}), \tag{1.9}$$

where the functions $\Psi_i : \mathbb{R}_i \to \mathbb{R} \cup \{+\infty\}$ are convex and closed.

1.5.5 Strong Convexity of F

In some of our results we assume, and we always explicitly mention this if we do, that F is strongly convex with respect to the norm $\|\cdot\|_w$ for some W, with (strong) convexity parameter $\mu_F(W) > 0$. A function $\phi : \mathbb{R}^N \to \mathbb{R} \cup \{+\infty\}$ is strongly convex with respect to the norm $\|\cdot\|_w$ with convexity parameter $\mu_\phi(W) \geq 0$ if for all $x, y \in \operatorname{dom}\phi$,

$$\phi(y) \geq \phi(x) + \langle \nabla\phi(x), y - x \rangle + \tfrac{\mu_\phi(W)}{2}\|y - x\|_w^2, \tag{1.10}$$

where $\nabla\phi(x)$ is any subgradient of ϕ at x. The case with $\mu_\phi(W) = 0$ reduces to convexity.

Strong convexity of F may come from f or Ψ or both; we will write $\mu_f(W)$ (resp. $\mu_\Psi(W)$) for the (strong) convexity parameter of f (resp. Ψ). It follows from (1.10) that

$$\mu_F(W) \geq \mu_f(W) + \mu_\Psi(W). \tag{1.11}$$

The following characterization of strong convexity will also be useful. For all $x, y \in \operatorname{dom}\phi$ and $\alpha \in [0, 1]$,

$$\phi(\alpha x + (1-\alpha)y) \leq \alpha\phi(x) + (1-\alpha)\phi(y) - \tfrac{\mu_\phi(W)\alpha(1-\alpha)}{2}\|x - y\|_w^2. \tag{1.12}$$

From the first order optimality conditions for (1.1) we obtain $\langle F'(x^*), x - x^* \rangle \geq 0$ for all $x \in \operatorname{dom} F$ which, combined with (1.10) used with $y = x$ and $x = x^*$, yields

$$F(x) - F^* \geq \tfrac{\mu_F(W)}{2}\|x - x^*\|_w^2, \quad x \in \operatorname{dom} F. \tag{1.13}$$

Also, it can be shown using (1.8) and (1.10) that

$$\mu_f(W) \leq \tfrac{L_i}{w_i}, \qquad \text{and hence} \qquad \mu_f(L) \leq 1. \tag{1.14}$$

Norm scaling. Note that since

$$\mu_\phi(tW) = \tfrac{1}{t}\mu_\phi(W), \qquad t > 0, \tag{1.15}$$

the size of the (strong) convexity parameter depends inversely on the size of W. Hence, if we want to compare convexity parameters for different choices of W, we need to normalize W first. A natural way of normalizing W is to require $\mathrm{tr}(W) = \mathrm{tr}(I) = n$. If we now define

$$\widetilde{W} \overset{\text{def}}{=} \tfrac{n}{\mathrm{tr}(W)} W, \tag{1.16}$$

we have $\mathrm{tr}(\tilde{W}) = n$ and

$$\mu_\phi(\widetilde{W}) \overset{(1.15),(1.16)}{=} \tfrac{\mathrm{tr}(W)}{n}\mu_\phi(W). \tag{1.17}$$

1.5.6 Level Set Radius

The set of optimal solutions of (1.1) is denoted by X^* and x^* is any element of that set. Define

$$\mathcal{R}_W(x) \overset{\text{def}}{=} \max_{y} \max_{x^* \in X^*} \{\|y - x^*\|_w \ : \ F(y) \leq F(x)\}, \tag{1.18}$$

which is a measure of the size of the level set of F given by x. In some of the results in this thesis we will need to assume that $\mathcal{R}_W(x_0)$ is finite for the initial iterate x_0 and some vector $w \in \mathbb{R}^n_{++}$.

1.5.7 Projection Onto a Set of Blocks

For $S \subset [n]$ and $x \in \mathbb{R}^N$ we write

$$x_{[S]} \overset{\text{def}}{=} \sum_{i \in S} U_i x^{(i)}. \tag{1.19}$$

That is, given $x \in \mathbb{R}^N$, $x_{[S]}$ is the vector in $\mathbb{R}^N$ whose blocks $i \in S$ are identical to those of x, but whose other blocks are zeroed out. In view of Proposition 1, we can equivalently define

$x_{[S]}$ block-by-block as follows

$$(x_{[S]})^{(i)} = \begin{cases} x^{(i)}, & i \in S, \\ 0 \ (\in \mathbb{R}^{N_i}), & \text{otherwise.} \end{cases} \tag{1.20}$$

1.6 Key Technical Tool For High Probability Results

Here we present the main technical tool which is used in our iteration complexity proofs.

Theorem 1. *Fix* $x_0 \in \mathbb{R}^N$ *and let* $\{x_k\}_{k\geq 0}, x_k \in \mathbb{R}^N$ *be a Markov process. Let* $\phi : \mathbb{R}^N \to \mathbb{R}$ *be a nonnegative function and define* $\xi_k = \phi(x_k)$*. Lastly, choose accuracy level* $0 < \epsilon < \xi_0$*, confidence level* $\rho \in (0,1)$*, and assume that the sequence of random variables* $\{\xi_k\}_{k\geq 0}$ *is nonincreasing and has one of the following properties:*

(i) $\mathbf{E}[\xi_{k+1} \mid x_k] \leq \xi_k - \frac{\xi_k^2}{c_1}$*, for all* k*, where* $c_1 > 0$ *is a constant,*

(ii) $\mathbf{E}[\xi_{k+1} \mid x_k] \leq (1 - \frac{1}{c_2})\xi_k$*, for all* k *such that* $\xi_k \geq \epsilon$*, where* $c_2 > 1$ *is a constant.*

If property (i) holds and we choose $\epsilon < c_1$ *and*

$$K \geq \tfrac{c_1}{\epsilon}(1 + \log \tfrac{1}{\rho}) + 2 - \tfrac{c_1}{\xi_0}, \tag{1.21}$$

or if property (ii) holds, and we choose

$$K \geq c_2 \log \tfrac{\xi_0}{\epsilon\rho}, \tag{1.22}$$

then

$$\mathbf{Prob}(\xi_K \leq \epsilon) \geq 1 - \rho. \tag{1.23}$$

Proof. First, notice that the sequence $\{\xi_k^\epsilon\}_{k\geq 0}$ defined by

$$\xi_k^\epsilon = \begin{cases} \xi_k & \text{if } \xi_k \geq \epsilon, \\ 0 & \text{otherwise,} \end{cases} \tag{1.24}$$

satisfies $\xi_k^\epsilon > \epsilon \Leftrightarrow \xi_k > \epsilon$. Therefore, by Markov inequality, $\mathbf{Prob}(\xi_k > \epsilon) = \mathbf{Prob}(\xi_k^\epsilon > \epsilon) \leq \frac{\mathbf{E}[\xi_k^\epsilon]}{\epsilon}$, and hence it suffices to show that

$$\theta_K \leq \epsilon\rho, \tag{1.25}$$

where $\theta_k \overset{\text{def}}{=} \mathbf{E}[\xi_k^\epsilon]$. Assume now that property (i) holds. We first claim that then

$$\mathbf{E}[\xi_{k+1}^\epsilon \mid x_k] \leq \xi_k^\epsilon - \tfrac{(\xi_k^\epsilon)^2}{c_1}, \qquad \mathbf{E}[\xi_{k+1}^\epsilon \mid x_k] \leq (1 - \tfrac{\epsilon}{c_1})\xi_k^\epsilon, \qquad k \geq 0. \tag{1.26}$$

Consider two cases. Assuming that $\xi_k \geq \epsilon$, from (1.24) we see that $\xi_k^\epsilon = \xi_k$. This, combined with the simple fact that $\xi_{k+1}^\epsilon \leq \xi_{k+1}$ and property (i), gives

$$\mathbf{E}[\xi_{k+1}^\epsilon \mid x_k] \leq \mathbf{E}[\xi_{k+1} \mid x_k] \leq \xi_k - \tfrac{\xi_k^2}{c_1} = \xi_k^\epsilon - \tfrac{(\xi_k^\epsilon)^2}{c_1}.$$

Assuming that $\xi_k < \epsilon$, we get $\xi_k^\epsilon = 0$ and, from monotonicity assumption, $\xi_{k+1} \leq \xi_k < \epsilon$. Hence, $\xi_{k+1}^\epsilon = 0$. Putting these together, we get $\mathbf{E}[\xi_{k+1}^\epsilon \mid x_k] = 0 = \xi_k^\epsilon - (\xi_k^\epsilon)^2/c_1$, which establishes the first inequality in (1.26). The second inequality in (1.26) follows from the first by again analyzing the two cases: $\xi_k \geq \epsilon$ and $\xi_k < \epsilon$. Now, by taking expectations in (1.26) (and using convexity of $t \mapsto t^2$ in the first case) we obtain, respectively,

$$\theta_{k+1} \leq \theta_k - \tfrac{\theta_k^2}{c_1}, \qquad k \geq 0, \tag{1.27}$$

$$\theta_{k+1} \leq (1 - \tfrac{\epsilon}{c_1})\theta_k, \qquad k \geq 0. \tag{1.28}$$

Notice that (1.27) is better than (1.28) precisely when $\theta_k > \epsilon$. Since

$$\tfrac{1}{\theta_{k+1}} - \tfrac{1}{\theta_k} = \tfrac{\theta_k - \theta_{k+1}}{\theta_{k+1}\theta_k} \geq \tfrac{\theta_k - \theta_{k+1}}{\theta_k^2} \overset{(1.27)}{\geq} \tfrac{1}{c_1},$$

we have $\frac{1}{\theta_k} \geq \frac{1}{\theta_0} + \frac{k}{c_1} = \frac{1}{\xi_0} + \frac{k}{c_1}$. Therefore, if we let $k_1 \geq \frac{c_1}{\epsilon} - \frac{c_1}{\xi_0}$, we obtain $\theta_{k_1} \leq \epsilon$. Finally, letting $k_2 \geq \frac{c_1}{\epsilon} \log \frac{1}{\rho}$, (1.25) follows from

$$\theta_K \overset{(1.21)}{\leq} \theta_{k_1+k_2} \overset{(1.28)}{\leq} (1 - \tfrac{\epsilon}{c_1})^{k_2}\theta_{k_1} \leq ((1 - \tfrac{\epsilon}{c_1})^{\frac{1}{\epsilon}})^{c_1 \log \frac{1}{\rho}} \epsilon \leq (e^{-\frac{1}{c_1}})^{c_1 \log \frac{1}{\rho}} \epsilon = \epsilon\rho.$$

Now assume that property (ii) holds. Using similar arguments as those leading to (1.26), we get $\mathbf{E}[\xi_{k+1}^\epsilon \mid \xi_k^\epsilon] \leq (1 - \frac{1}{c_2})\xi_k^\epsilon$ for all k , which implies

$$\theta_K \leq (1 - \tfrac{1}{c_2})^K \theta_0 = (1 - \tfrac{1}{c_2})^K \xi_0 \overset{(1.22)}{\leq} ((1 - \tfrac{1}{c_2})^{c_2})^{\log \frac{\xi_0}{\epsilon\rho}} \xi_0 \leq (e^{-1})^{\log \frac{\xi_0}{\epsilon\rho}} \xi_0 = \epsilon\rho,$$

again establishing (1.25). □

The above theorem will be used with $\{x_k\}_{k\geq 0}$ corresponding to the iterates of CD algorithm and $\phi(x) = F(x) - F^*$.

Restarting. Note that similar, albeit *slightly weaker,* high probability results can be achieved by *restarting* as follows. We run the random process $\{\xi_k\}$ repeatedly $r = \lceil \log \frac{1}{\rho} \rceil$ times, always starting from ξ_0, each time for the same number of iterations k_1 for which $\mathbf{Prob}(\xi_{k_1} > \epsilon) \leq \frac{1}{e}$. It then follows that the probability that all r values ξ_{k_1} will be larger than ϵ is at most $(\frac{1}{e})^r \leq \rho$. Note that the restarting technique demands that we perform r evaluations of the objective function (in order to find out which random process led to a solution with the smallest value); this is not needed in the one-shot approach covered by the theorem.

It remains to estimate k_1 in the two cases of Theorem 1. We argue that in case (i) we can choose $k_1 = \lceil \frac{c_1}{\epsilon/e} - \frac{c_1}{\xi_0} \rceil$. Indeed, using similar arguments as in Theorem 1 this leads to $\mathbf{E}[\xi_{k_1}] \leq \frac{\epsilon}{e}$, which by Markov inequality implies that in a single run of the process we have

$$\mathbf{Prob}(\xi_{k_1} > \epsilon) \leq \frac{\mathbf{E}[\xi_{k_1}]}{\epsilon} \leq \frac{\epsilon/e}{\epsilon} = \frac{1}{e}.$$

Therefore,

$$K = \lceil \frac{ec_1}{\epsilon} - \frac{c_1}{\xi_0} \rceil \lceil \log \frac{1}{\rho} \rceil$$

iterations suffice in case (i). A similar restarting technique can be applied in case (ii).

Tightness. It can be shown on simple examples that the bounds in the above result are *tight.*

2

Serial Coordinate Descent Method

> Finally, we make some remarks on why linear systems are so important. The answer is simple: because we can solve them!
>
> — Bernard Riemann (1826-1866)

This chapter focuses on the convergence results of the serial coordinate descent method (CDM). We start with a brief literature review relevant to serial CDM in Section 2.1 and then we state the contributions achieved in this chapter in Section 2.3. In Section 2.4 we start with a description of a generic randomized block-coordinate descent algorithm (RCDC) and continue with a study of the performance of a uniform variant (UCDC) of RCDC as applied to a composite objective function. In Section 2.5 we analyze a smooth variant (RCDS) of RCDC; that is, we study the performance of RCDC on a smooth objective function. In Section 2.6 we compare known complexity results for CD methods with the ones established in this chapter. Finally, in Section 2.7 we demonstrate the efficiency of the method on ℓ_1-regularized least squares and linear support vector machine problems.

2.1 Literature Review

Strohmer and Vershynin [65] have recently proposed a randomized Karczmarz method for solving overdetermined consistent systems of linear equations and proved that the method enjoys global linear convergence whose rate can be expressed in terms of the condition number of the underlying matrix. The authors claim that for certain problems their approach can be more efficient than the conjugate gradient method. Motivated by these results, Leventhal and Lewis [23] studied the problem of solving a system of linear equations and inequalities and in the process gave iteration complexity bounds for a randomized CD method applied to the problem of minimizing a convex quadratic function. In their method the probability of choice of each coordinate is proportional to the corresponding diagonal element of the underlying positive semidefinite matrix defining the objective function. These diagonal elements can be interpreted as Lipschitz constants of the derivative of a restriction of the quadratic objective onto one-dimensional lines parallel to the coordinate axes. In the general (as opposed to quadratic) case considered in this chapter (1.1), these Lipschitz constants will play an important role as well. Lin et al. [7] derived iteration complexity results for several smooth objective functions appearing in machine learning. Shalev-Schwarz and Tewari [61] proposed a randomized coordinate descent method with uniform probabilities for minimizing ℓ_1-regularized smooth convex problems. They first transform the problem into a box constrained smooth problem by doubling the dimension and then apply a coordinate gradient descent method in which each coordinate is chosen with equal probability. Nesterov [43] has recently analyzed randomized coordinate descent methods in the smooth unconstrained and box-constrained setting, in effect extending and improving upon some of the results in [23], [7], [61] in several ways.

While the *asymptotic convergence rates* of some variants of CD methods are well understood [31], [74], [72], [84], *iteration complexity* results are very rare. To the best of our knowledge, randomized CD algorithms for minimizing a composite function have been proposed and analyzed (in the iteration complexity sense) in a few special cases only: a) the unconstrained convex quadratic case [23], b) the smooth unconstrained ($\Psi \equiv 0$) and the smooth block-constrained case (Ψ is the indicator function of a direct sum of boxes) [43] and c) the ℓ_1-regularized case [61]. As the approach in [61] is to rewrite the problem into a smooth box-constrained format first, the results of [43] can be viewed as a (major) generalization and improvement of those in [61] (the results were obtained independently).

2.2 The Algorithm

In this section we are going to describe a generic coordinate descent algorithm. But before we do so, let us describe how the step-size should be computed. As already suggested in Section 1.2, the computation of step-size should not require evaluation of function value. Imagine, for a while, that we would like to move in direction of i-th block. Notice that an upper bound on $F(x + U_i t)$, viewed as a function of $t \in \mathbb{R}_i$, is readily available:

$$F(x + U_i t) \overset{(1.1)}{=} f(x + U_i t) + \Psi(x + U_i t) \overset{(1.8)}{\leq} f(x) + V_i(x, t) + C_i(x), \tag{2.1}$$

where

$$V_i(x,t) \stackrel{\text{def}}{=} \langle \nabla_i f(x), t\rangle + \tfrac{L_i}{2}\|t\|_{(i)}^2 + \Psi_i(x^{(i)} + t) \tag{2.2}$$

and

$$C_i(x) \stackrel{\text{def}}{=} \sum_{j\neq i} \Psi_j(x^{(j)}) \tag{2.3}$$

and solving an low-dimensional optimization problem $\min_{t\in\mathbb{R}_i}\{f(x) + V_i(x,t) + C_i(x)\}$ does not require computation of objective function.

We are now ready to describe the generic randomized (block) coordinate descent method for solving (1.1). Given iterate x_k, Algorithm 2.1 picks block $i_k = i \in \{1,2,\dots,n\}$ with probability $p_i > 0$ and then updates the i-th block of x_k so as to minimize (exactly) in t the upper bound (2.1) on $F(x_k + U_i t)$. Note that in certain cases it is possible to minimize $F(x_k + U_i t)$ directly; perhaps in a closed form. This is the case, for example, when f is a convex quadratic and Ψ_i is simple.

Algorithm 2.1 RCDC(p, x_0) (**R**andomized **C**oordinate **D**escent for **C**omposite Functions)

```
for $k = 0, 1, 2, \dots$ do
    Choose $i_k = i \in \{1,2,\dots,n\}$ with probability $p_i$
    $T^{(i)}(x_k) \stackrel{\text{def}}{=} \arg\min\{V_i(x_k,t) \ : \ t \in \mathbb{R}_i\}$
    $x_{k+1} = x_k + U_i T^{(i)}(x_k)$
end for
```

The iterates $\{x_k\}$ are random vectors and the values $\{F(x_k)\}$ are random variables. Clearly, x_{k+1} depends on x_k only. As our analysis will be based on the (expected) per-iteration decrease of the objective function, our results hold if we replace $V_i(x_k,t)$ by $F(x_k + U_i t)$ in Algorithm 2.1.

2.3 Summary of Contributions

In this chapter we further improve upon and extend and simplify the iteration complexity results of Nesterov [43], treating the problem of minimizing the sum of a smooth convex and a simple nonsmooth convex block separable function (1.1). We focus exclusively on simple (as opposed to accelerated) methods. The reason for this is that the per-iteration work of the accelerated algorithm in [43] on huge scale instances of problems with *sparse* data (such as the Google problem where sparsity corresponds to each website linking only to a few other websites or the sparse problems we consider in Section 2.7) is excessive. In fact, even the author of [43] does not recommend using the accelerated method for solving such problems; the simple methods seem to be more efficient.

Each algorithm of this chapter is supported by a high probability iteration complexity result.

That is, for any given *confidence level* $0 < \rho < 1$ and *error tolerance* $\epsilon > 0$, we give an explicit expression for the number of iterations k which guarantee that the method produces a random iterate x_k for which

$$\mathbf{Prob}(F(x_k) - F^* \leq \epsilon) \geq 1 - \rho.$$

Table 2.1 summarizes the main complexity results of this chapter. Algorithm 2.2—Uniform (block) Coordinate Descent for Composite functions (UCDC)—is a method where at each iteration the block of coordinates to be updated (out of a total of $n \leq N$ blocks) is chosen uniformly at random. Algorithm 2.3—Randomized (block) Coordinate Descent for Smooth functions (RCDS)—is a method where at each iteration block $i \in \{1, \dots, n\}$ is chosen with probability $p_i > 0$. Both of these methods are special cases of the generic Algorithm 2.1; Randomized (block) Coordinate Descent for Composite functions (RCDC).

Algorithm	Objective	Complexity
Algorithm 2.2 (UCDC) (Theorem 2)	convex composite	$\frac{2n\max\{\mathcal{R}^2_L(x_0),F(x_0)-F^*\}}{\epsilon}(1+\log\frac{1}{\rho})$
Algorithm 2.2 (UCDC) (Theorem 2)	convex composite	$\frac{2n\mathcal{R}^2_L(x_0)}{\epsilon}\log\left(\frac{F(x_0)-F^*}{\epsilon\rho}\right)$
Algorithm 2.2 (UCDC) (Theorem 4)	strongly convex composite	$n\frac{1+\mu_\Psi(L)}{\mu_f(L)+\mu_\Psi(L)}\log\left(\frac{F(x_0)-F^*}{\epsilon\rho}\right)$
Algorithm 2.3 (RCDS) (Theorem 6)	convex smooth	$\frac{2\mathcal{R}^2_{LP^{-1}}(x_0)}{\epsilon}(1+\log\frac{1}{\rho})-2$
Algorithm 2.3 (RCDS) (Theorem 7)	strongly convex smooth	$\frac{1}{\mu_f(LP^{-1})}\log\left(\frac{f(x_0)-f^*}{\epsilon\rho}\right)$

Table 2.1: Summary of complexity results obtained in this chapter.

The symbols $L, \mathcal{R}^2_W(x_0)$ and $\mu_\phi(W)$ appearing in Table 2.1 are precisely defined in Section 1.5 and P is a diagonal matrix encoding the probabilities $\{p_i\}$.

Let us now briefly outline the main similarities and differences between our results and those in [43]. A more detailed and expanded discussion can be found in Section 2.6.

1. **Composite setting.** We consider the composite setting[1] (1.1), whereas [43] covers the unconstrained and constrained smooth setting only.

---

[1] Note that in [42] Nesterov considered the composite setting and developed standard and accelerated gradient methods with iteration complexity guarantees for minimizing composite objective functions. These can be viewed as block coordinate descent methods with a *single* block.

2. **No need for regularization.** Nesterov's high probability results in the case of minimizing a function which is not strongly convex are based on regularizing the objective to make it strongly convex and then running the method on the regularized function. The regularizing term depends on the distance of the initial iterate to the optimal point, and hence is *unknown*, which means that the analysis in [43] does not lead to true iteration complexity results. Our contribution here is that we show that no regularization is needed by doing a more detailed analysis using a thresholding argument (Theorem 1).

3. **Better complexity.** Our complexity results are better by the constant factor of 4. Also, we have removed ϵ from the logarithmic term (see (2.9) and (2.31)).

4. **General probabilities.** Nesterov considers probabilities p_i proportional to L_i^α, where $\alpha \geq 0$ is a parameter. High probability results are proved in [43] for $\alpha \in \{0, 1\}$ only. Our results in the smooth case hold for an arbitrary probability vector p.

5. **General norms.** Nesterov's expectation results (Theorems 1 and 2) are proved for general norms. However, his high probability results are proved for Euclidean norms only. In our approach all results in the smooth case hold for general norms.

6. **Simplification.** Our analysis is shorter.

In the numerical experiments section we focus on sparse regression. For these problems we introduce a powerful *speedup heuristic* based on adaptively changing the probability vector throughout the iterations.

2.4 Iteration Complexity for Composite Functions

In this section we study the performance of Algorithm 2.1 in the special case when all probabilities are chosen to be the same, i.e., $p_i = \frac{1}{n}$ for all i. For easier future reference we set this method apart and give it a name (Algorithm 2.2).

Algorithm 2.2 UCDC(x_0) (**U**niform **C**oordinate **D**escent for **C**omposite Functions)

```
for $k = 0, 1, 2, \dots$ do
    Choose $i_k = i \in \{1, 2, \dots, n\}$ with probability $\frac{1}{n}$
    $T^{(i)}(x_k) = \arg\min\{V_i(x_k, t) \ : \ t \in \mathbb{R}_i\}$
    $x_{k+1} = x_k + U_i T^{(i)}(x_k)$
end for
```

The following function plays a central role in our analysis:

$$H(x, T) \overset{\text{def}}{=} f(x) + \langle \nabla f(x), T \rangle + \tfrac{1}{2}\|T\|_L^2 + \Psi(x + T). \tag{2.4}$$

Comparing (2.4) with (2.2) using (1.3), (1.7), (1.9) and (1.5) we get

$$H(x,T) = f(x) + \sum_{i=1}^{n} V_i(x, T^{(i)}). \tag{2.5}$$

Therefore, the vector $T(x) = (T^{(1)}(x), \ldots, T^{(n)}(x))$, with the components $T^{(i)}(x)$ defined in Algorithm 2.1, is the minimizer of $H(x,\cdot)$:

$$T(x) = \arg\min_{T \in \mathbb{R}^N} H(x,T). \tag{2.6}$$

Let us start by establishing two auxiliary results which will be used repeatedly.

Lemma 1. *Let $\{x_k\}$, $k \geq 0$, be the random iterates generated by UCDC(x_0). Then*

$$\mathbf{E}[F(x_{k+1}) - F^* \mid x_k] \leq \tfrac{1}{n}\,(H(x_k, T(x_k)) - F^*) + \tfrac{n-1}{n}\,(F(x_k) - F^*). \tag{2.7}$$

Proof.

$$\begin{aligned}
\mathbf{E}[F(x_{k+1}) \mid x_k] \quad &= \quad \sum_{i=1}^{n} \tfrac{1}{n} F(x_k + U_i T^{(i)}(x_k)) \\
&\overset{(2.1)}{\leq} \quad \tfrac{1}{n} \sum_{i=1}^{n} [f(x_k) + V_i(x_k, T^{(i)}(x_k)) + C_i(x_k)] \\
&\overset{(2.5)}{=} \quad \tfrac{1}{n} H(x_k, T(x_k)) + \tfrac{n-1}{n} f(x_k) + \tfrac{1}{n} \sum_{i=1}^{n} C_i(x_k) \\
&\overset{(2.3)}{=} \quad \tfrac{1}{n} H(x_k, T(x_k)) + \tfrac{n-1}{n} f(x_k) + \tfrac{1}{n} \sum_{i=1}^{n} \sum_{j \neq i} \Psi_j(x_k^{(j)}) \\
&= \quad \tfrac{1}{n} H(x_k, T(x_k)) + \tfrac{n-1}{n} F(x_k).
\end{aligned}$$

□

Lemma 2. *For all $x \in \operatorname{dom} F$ we have $H(x, T(x)) \leq \min_{y \in \mathbb{R}^N} \{F(y) + \frac{1-\mu_f(L)}{2} \|y - x\|_L^2\}$.*

Proof.

$$\begin{aligned}
H(x, T(x)) \overset{(2.6)}{=} \min_{T \in \mathbb{R}^N} H(x,T) \quad &= \quad \min_{y \in \mathbb{R}^N} H(x, y - x) \\
&\overset{(2.4)}{=} \quad \min_{y \in \mathbb{R}^N} f(x) + \langle \nabla f(x), y - x \rangle + \Psi(y) + \tfrac{1}{2} \|y - x\|_L^2 \\
&\overset{(1.10)}{\leq} \quad \min_{y \in \mathbb{R}^N} f(y) - \tfrac{\mu_f(L)}{2} \|y - x\|_L^2 + \Psi(y) + \tfrac{1}{2} \|y - x\|_L^2.
\end{aligned}$$

□

2.4.1 Convex Objective

In order for Lemma 1 to be useful, we need to estimate $H(x_k, T(x_k)) - F^*$ from above in terms of $F(x_k) - F^*$.

Lemma 3. *Fix $x^* \in X^*$, $x \in \operatorname{dom}\Psi$ and let $R = \|x - x^*\|_L$. Then*

$$H(x, T(x)) - F^* \leq \begin{cases} \left(1 - \frac{F(x)-F^*}{2R^2}\right)(F(x) - F^*), & \text{if } F(x) - F^* \leq R^2, \\ \frac{1}{2}R^2 < \frac{1}{2}(F(x) - F^*), & \text{otherwise.} \end{cases} \tag{2.8}$$

Proof. Since we do not assume strong convexity, $\mu_f(W) = 0$, and hence

$$\begin{aligned} H(x, T(x)) \overset{\text{Lemma 2}}{\leq} \min_{y \in \mathbb{R}^N} F(y) + \tfrac{1}{2}\|y - x\|_L^2 &\leq \min_{\alpha \in [0,1]} F(\alpha x^* + (1-\alpha)x) + \tfrac{\alpha^2}{2}\|x - x^*\|_L^2 \\ &\leq \min_{\alpha \in [0,1]} F(x) - \alpha(F(x) - F^*) + \tfrac{\alpha^2}{2}R^2. \end{aligned}$$

Minimizing the last expression gives $\alpha^* = \min\left\{1, \frac{1}{R^2}(F(x) - F^*)\right\}$; the result follows. $\square$

We are now ready to estimate the number of iterations needed to push the objective value within ϵ of the optimal value[2] with high probability. Note that since ρ appears in the logarithm, it is easy to attain high confidence.[3]

Theorem 2. *Choose initial point x_0 and target confidence $0 < \rho < 1$. Further, let the target accuracy $\epsilon > 0$ and iteration counter k be chosen in any of the following two ways:*

(i) $\epsilon < F(x_0) - F^*$ and

$$k \geq \frac{2n \max\{\mathcal{R}_L^2(x_0), F(x_0) - F^*\}}{\epsilon}\left(1 + \log\frac{1}{\rho}\right) + 2 - \frac{2n \max\{\mathcal{R}_L^2(x_0), F(x_0) - F^*\}}{F(x_0) - F^*}, \tag{2.9}$$

(ii) $\epsilon < \min\{\mathcal{R}_L^2(x_0), F(x_0) - F^*\}$ and

$$k \geq \frac{2n\mathcal{R}_L^2(x_0)}{\epsilon} \log\frac{F(x_0) - F^*}{\epsilon\rho}. \tag{2.10}$$

If x_k is the random point generated by UCDC(x_0) as applied to the convex function F, then

$$\mathbf{Prob}(F(x_k) - F^* \leq \epsilon) \geq 1 - \rho.$$

---

[2] Note that the $\frac{1}{\epsilon}$ term is the lower-bound for a general convex composite objective under certain conditions, see e.g. [42].

[3] Note that the Theorem 2 requires $\mathcal{R}_L^2(x_0)$ to be bounded. However, recent papers [29, 70] improved the analysis, so the iteration complexity can be obtained even if the levelset is not bounded.

Proof. Since $F(x_k) \le F(x_0)$ for all k, we have $\|x_k - x^*\|_L \le \mathcal{R}_L(x_0)$ for all $x^* \in X^*$. Plugging the inequality (2.8) (Lemma 1) into (2.7) (Lemma 3) then gives that the following holds for all k:

$$\begin{aligned}
\mathbf{E}[F(x_{k+1}) - F^* \mid x_k] &\le \tfrac{1}{n}\max\left\{1 - \tfrac{F(x_k)-F^*}{2\|x_k-x^*\|_L^2}, \tfrac{1}{2}\right\}(F(x_k) - F^*) + \tfrac{n-1}{n}(F(x_k) - F^*) \\
&= \max\left\{1 - \tfrac{F(x_k)-F^*}{2n\|x_k-x^*\|_L^2}, 1 - \tfrac{1}{2n}\right\}(F(x_k) - F^*) \\
&\le \max\left\{1 - \tfrac{F(x_k)-F^*}{2n\mathcal{R}_L^2(x_0)}, 1 - \tfrac{1}{2n}\right\}(F(x_k) - F^*). \qquad (2.11)
\end{aligned}$$

Let $\xi_k = F(x_k) - F^*$ and consider case (i). If we let $c_1 = 2n\max\{\mathcal{R}_L^2(x_0), F(x_0) - F^*\}$, then from (2.11) we obtain

$$\mathbf{E}[\xi_{k+1} \mid x_k] \le (1 - \tfrac{\xi_k}{c_1})\xi_k = \xi_k - \tfrac{\xi_k^2}{c_1}, \qquad k \ge 0.$$

Moreover, $\epsilon < \xi_0 < c_1$. The result then follows by applying Theorem 1. Consider now case (ii). Letting $c_2 = \frac{2n\mathcal{R}_L^2(x_0)}{\epsilon} > 1$, notice that if $\xi_k \ge \epsilon$, inequality (2.11) implies that

$$\mathbf{E}[\xi_{k+1} \mid x_k] \le \max\left\{1 - \tfrac{\epsilon}{2n\mathcal{R}_L^2(x_0)}, 1 - \tfrac{1}{2n}\right\}\xi_k = (1 - \tfrac{1}{c_2})\xi_k.$$

Again, the result follows from Theorem 1. □

2.4.2 Strongly Convex Objective

The following lemma will be useful in proving linear convergence of the expected value of the objective function to the minimum.

Lemma 4. *If* $\mu_f(L) + \mu_\Psi(L) > 0$, *then for all* $x \in \operatorname{dom} F$ *we have*

$$H(x, T(x)) - F^* \le \frac{1 - \mu_f(L)}{1 + \mu_\Psi(L)}(F(x) - F^*). \qquad (2.12)$$

Proof. Letting $\mu_f = \mu_f(L)$, $\mu_\Psi = \mu_\Psi(L)$ and $\alpha^* = (\mu_f + \mu_\Psi)/(1 + \mu_\Psi) \overset{(1.14)}{\le} 1$, we have

$$\begin{aligned}
H(x, T(x)) &\overset{\text{Lem 2}}{\le} \min_{y \in \mathbb{R}^N} F(y) + \tfrac{1-\mu_f}{2}\|y - x\|_L^2 \\
&\le \min_{\alpha \in [0,1]} F(\alpha x^* + (1-\alpha)x) + \tfrac{(1-\mu_f)\alpha^2}{2}\|x - x^*\|_L^2 \\
&\overset{(1.12)+(1.11)}{\le} \min_{\alpha \in [0,1]} \alpha F^* + (1-\alpha)F(x) - \tfrac{(\mu_f+\mu_\Psi)\alpha(1-\alpha) - (1-\mu_f)\alpha^2}{2}\|x - x^*\|_L^2 \\
&\le F(x) - \alpha^*(F(x) - F^*).
\end{aligned}$$

The last inequality follows from the identity $(\mu_f + \mu_\Psi)(1-\alpha^*) - (1-\mu_f)\alpha^* = 0$. $\square$

A modification of the above lemma (and of the subsequent results using it) is possible where the assumption $\mu_f(L) + \mu_\Psi(L) > 0$ replaced by the slightly weaker assumption $\mu_F(L) > 0$. Indeed, in the third inequality in the proof one can replace $\mu_f + \mu_\Psi$ by μ_F; the estimate (2.12) gets improved a bit. However, we prefer the current version for reasons of simplicity of exposition.

We now show that the expected value of $F(x_k)$ converges to F^* linearly.

Theorem 3. *Assume $\mu_f(L) + \mu_\Psi(L) > 0$ and choose initial point x_0. If x_k is the random point generated UCDC(x_0), then*

$$\mathbf{E}[F(x_k) - F^*] \leq \left(1 - \frac{1}{n}\frac{\mu_f(L) + \mu_\Psi(L)}{1 + \mu_\Psi(L)}\right)^k (F(x_0) - F^*). \tag{2.13}$$

Proof. Follows from Lemma 1 and Lemma 4. $\square$

The following is an analogue of Theorem 2 in the case of a strongly convex objective. Note that both the accuracy and confidence parameters appear in the logarithm.

Theorem 4. *Assume $\mu_f(L) + \mu_\Psi(L) > 0$. Choose initial point x_0, target accuracy level $0 < \epsilon < F(x_0) - F^*$, target confidence level $0 < \rho < 1$, and*

$$k \geq n \frac{1 + \mu_\Psi(L)}{\mu_f(L) + \mu_\Psi(L)} \log\left(\frac{F(x_0) - F^*}{\epsilon\rho}\right). \tag{2.14}$$

If x_k is the random point generated by UCDC(x_0), then

$$\mathbf{Prob}(F(x_k) - F^* \leq \epsilon) \geq 1 - \rho.$$

Proof. Using Markov inequality and Theorem 3, we obtain

$$\mathbf{Prob}[F(x_k) - F^* \geq \epsilon] \leq \tfrac{1}{\epsilon}\mathbf{E}[F(x_k) - F^*] \overset{(2.13)}{\leq} \tfrac{1}{\epsilon}\left(1 - \tfrac{1}{n}\tfrac{\mu_f(L)+\mu_\Psi(L)}{1+\mu_\Psi(L)}\right)^k (F(x_0) - F^*) \overset{(2.14)}{\leq} \rho.$$

$\square$

Let us rewrite the *condition number* appearing in the complexity bound (2.14) in a more natural form:

$$\frac{1 + \mu_\Psi(L)}{\mu_f(L) + \mu_\Psi(L)} \overset{(1.17)}{=} \frac{\operatorname{tr}(L)/n + \mu_\Psi\left(\widetilde{L}\right)}{\mu_f\left(\widetilde{L}\right) + \mu_\Psi\left(\widetilde{L}\right)} \leq 1 + \frac{\operatorname{tr}(L)/n}{\mu_f\left(\widetilde{L}\right) + \mu_\Psi\left(\widetilde{L}\right)}. \tag{2.15}$$

Hence, it is (up to the constant 1) equal to the ratio of the *average* of the Lipschitz constants L_i and the sum of (strong) convexity parameters of the functions f and Ψ with respect to the (normalized) norm $\|\cdot\|_{\widetilde{L}}$.

2.4.3 A Regularization Technique

In this section we investigate an alternative approach to establishing an iteration complexity result in the case of an objective function that is not strongly convex. The strategy is very simple. We first regularize the objective function by adding a small quadratic term to it, thus making it strongly convex, and then argue that when Algorithm 2.2 is applied to the regularized objective, we can recover an approximate solution of the original non-regularized problem. This approach was used in [43] to obtain iteration complexity results for a randomized block coordinate descent method applied to a smooth function. Here we use the same idea outlined above with the following differences: i) our proof is different, ii) we get a better complexity result, and iii) our approach works also in the composite setting.

Fix x_0 and $\epsilon > 0$ and consider a regularized version of the objective function defined by

$$F_\mu(x) \overset{\text{def}}{=} F(x) + \tfrac{\mu}{2}\|x - x_0\|_L^2, \qquad \mu = \tfrac{\epsilon}{\|x_0 - x^*\|_L^2}. \tag{2.16}$$

Clearly, F_μ is strongly convex with respect to the norm $\|\cdot\|_L$ with convexity parameter $\mu_{F_\mu}(L) = \mu$. In the rest of this subsection we show that if we apply UCDC(x_0) to F_μ with target accuracy $\frac{\epsilon}{2}$, then with high probability we recover an ϵ-approximate solution of (1.1). Note that μ is *not known* in advance since x^* is not known. This means that any iteration complexity result obtained by applying our algorithm to the objective F_μ will *not* lead to a *true/valid* iteration complexity bound unless a bound on $\|x_0 - x^*\|_L$ is available.

We first need to establish that an approximate minimizer of F_μ must be an approximate minimizer of F.

Lemma 5. *If x' satisfies $F_\mu(x') \le \min_{x\in\mathbb{R}^N} F_\mu(x) + \frac{\epsilon}{2}$, then $F(x') \le F^* + \epsilon$.*

Proof. Clearly,

$$F(x) \le F_\mu(x), \qquad x \in \mathbb{R}^N. \tag{2.17}$$

If we let $x_\mu^* \overset{\text{def}}{=} \arg\min_{x\in\mathbb{R}^N} F_\mu(x)$ then, by assumption,

$$F_\mu(x') - F_\mu(x_\mu^*) \le \tfrac{\epsilon}{2}, \tag{2.18}$$

$$F_\mu(x_\mu^*) = \min_{x\in\mathbb{R}^N} F(x) + \tfrac{\mu}{2}\|x - x_0\|_L^2 \le F(x^*) + \tfrac{\mu}{2}\|x^* - x_0\|_L^2 \overset{(2.16)}{\le} F(x^*) + \tfrac{\epsilon}{2}. \tag{2.19}$$

Putting all these observations together, we get

$$0 \le F(x') - F(x^*) \overset{(2.17)}{\le} F_\mu(x') - F(x^*) \overset{(2.18)}{\le} F_\mu(x_\mu^*) + \tfrac{\epsilon}{2} - F(x^*) \overset{(2.19)}{\le} \epsilon.$$

□

The following theorem is an analogue of Theorem 2. The result we obtain in this way is slightly different to the one given in Theorem 2 in that $2n\mathcal{R}_L^2(x_0)/\epsilon$ is replaced by $n(1 + \|x_0 - x^*\|_L^2/\epsilon)$. In some situations, $\|x_0 - x^*\|_L^2$ can be significantly smaller than $\mathcal{R}_L^2(x_0)$.

Theorem 5. *Choose initial point* x_0*, target accuracy level*

$$0 < \epsilon \le 2(F(x_0) - F^*), \tag{2.20}$$

target confidence level $0 < \rho < 1$*, and*

$$k \ge n\left(1 + \tfrac{\|x_0 - x^*\|_L^2}{\epsilon}\right) \log\left(\tfrac{2(F(x_0) - F^*)}{\epsilon\rho}\right). \tag{2.21}$$

If x_k *is the random point generated by UCDC*(x_0) *as applied to* F_μ*, then*

$$\mathbf{Prob}(F(x_k) - F^* \le \epsilon) \ge 1 - \rho.$$

Proof. Let us apply Theorem 4 to the problem of minimizing F_μ, composed as $f + \Psi_\mu$, with $\Psi_\mu(x) = \Psi(x) + \frac{\mu}{2}\|x - x_0\|_L^2$, with accuracy level $\frac{\epsilon}{2}$. Note that $\mu_{\Psi_\mu}(L) = \mu$,

$$F_\mu(x_0) - F_\mu(x_\mu^*) \overset{(2.16)}{=} F(x_0) - F_\mu(x_\mu^*) \overset{(2.17)}{\le} F(x_0) - F(x_\mu^*) \le F(x_0) - F^*, \tag{2.22}$$

$$n(1 + \tfrac{1}{\mu}) \overset{(2.16)}{=} n\left(1 + \tfrac{\|x_0 - x^*\|_L^2}{\epsilon}\right). \tag{2.23}$$

Comparing (2.14) and (2.21) in view of (2.22) and (2.23), Theorem 4 implies that

$$\mathbf{Prob}(F_\mu(x_k) - F_\mu(x_\mu^*) \le \tfrac{\epsilon}{2}) \ge 1 - \rho.$$

It now suffices to apply Lemma 5. □

2.5 Iteration Complexity for Smooth Functions

In this section we give a much simplified and improved treatment of the smooth case ($\Psi \equiv 0$) as compared to the analysis in Sections 2 and 3 of [43].

As alluded to in the above, we will develop the analysis in the smooth case for arbitrary, possibly non-Euclidean, norms $\|\cdot\|_{(i)}$, $i = 1, 2, \dots, n$. Let $\|\cdot\|$ be an arbitrary norm in $\mathbb{R}^l$. Then its dual is defined in the usual way:

$$\|s\|^* = \max_{\|t\|=1} \langle s, t\rangle. \tag{2.24}$$

The following (Lemma 6) is a simple result which is used in [43] without being fully articulated nor proved as it constitutes a straightforward extension of a fact that is trivial in the Euclidean setting to the case of general norms. Since we will also need to use it, and because we think it is perhaps not standard, we believe it deserves to be spelled out explicitly. Note that the main problem which needs to be solved at each iteration of Algorithm 2.1 in the smooth case is of the form (2.25), with $s = -\frac{1}{L_i}\nabla_i f(x_k)$ and $\|\cdot\| = \|\cdot\|_{(i)}$.

Lemma 6. *If by $s^\#$ we denote an optimal solution of the problem*

$$\min_t \left\{ u(t) \overset{def}{=} -\langle s, t\rangle + \tfrac{1}{2}\|t\|^2 \right\}, \tag{2.25}$$

then

$$u(s^\#) = -\tfrac{1}{2}\left(\|s\|^*\right)^2, \qquad \|s^\#\| = \|s\|^*, \qquad (\alpha s)^\# = \alpha(s^\#),\ \alpha \in \mathbb{R}. \tag{2.26}$$

Proof. For $\alpha = 0$ the last statement is trivial. If we fix $\alpha \neq 0$, then clearly

$$u((\alpha s)^\#) = \min_{\|t\|=1} \min_\beta \{-\langle \alpha s, \beta t\rangle + \tfrac{1}{2}\|\beta t\|^2\}. \tag{2.27}$$

For fixed t (with $\|t\| = 1$) the solution of the inner problem is $\beta = \langle \alpha s, t\rangle$, whence

$$u((\alpha s)^\#) = \min_{\|t\|=1} -\tfrac{1}{2}\langle \alpha s, t\rangle^2 = -\tfrac{1}{2}\alpha^2 \left(\max_{\|t\|=1} \langle s, t\rangle\right)^2 = -\tfrac{1}{2}(\|\alpha s\|^*)^2, \tag{2.28}$$

proving the first claim. Next, note that optimal $t = t^*$ in (2.28) maximizes $\langle s, t\rangle$ over $\|t\| = 1$. Therefore, $\langle s, t^*\rangle \overset{(2.24)}{=} \|s\|^*$. Since t^* depends on s only, we have

$$(\alpha s)^\# \overset{(2.27),(2.28)}{=} \beta^* \left(\arg\min_{\|t\|=1} -\tfrac{1}{2}\langle \alpha s, t\rangle^2\right) = \beta^* t^* = \langle \alpha s, t^*\rangle t^* \tag{2.29}$$

and, in particular, $s^\# = \langle s, t^* \rangle t^*$. Therefore, $(\alpha s)^\# = \alpha(s^\#)$. Finally,

$$\|(\alpha s)^\#\| \overset{(2.29)}{=} \|\beta t^*\| = |\beta^*|\|t^*\| = |\beta^*| = |\langle \alpha s, t^* \rangle| = |\alpha||\langle s, t^* \rangle| = |\alpha|\|s\|^* = \|\alpha s\|^*$$

giving the second claim. □

We can use Lemma 6 to rewrite the main step of Algorithm 2.1 in the smooth case into the more explicit form,

$$\begin{aligned} T^{(i)}(x) = \arg\min_{t \in \mathbb{R}_i} V_i(x,t) &\overset{(2.2)}{=} \arg\min_{t \in \mathbb{R}_i} \langle \nabla_i f(x), t \rangle + \tfrac{L_i}{2} \|t\|_{(i)}^2 \\ &\overset{(2.25)}{=} \left(-\tfrac{\nabla_i f(x)}{L_i}\right)^\# \overset{(2.26)}{=} -\tfrac{1}{L_i} (\nabla_i f(x))^\#, \end{aligned}$$

leading to Algorithm 2.3.

Algorithm 2.3 RCDS(p, x_0) (**R**andomized **C**oordinate **D**escent for **S**mooth Functions)

```
for $k = 0, 1, 2, \dots$ do
    Choose $i_k = i \in \{1, 2, \dots, n\}$ with probability $p_i$
    $x_{k+1} = x_k - \frac{1}{L_i} U_i (\nabla_i f(x_k))^\#$
end for
```

The main utility of Lemma 6 for the purpose of the subsequent complexity analysis comes from the fact that it enables us to give an *explicit* bound on the decrease in the objective function during one iteration of the method in the same form as in the Euclidean case:

$$\begin{aligned} f(x) - f(x + U_i T^{(i)}(x)) &\overset{(1.8)}{\geq} -[\langle \nabla_i f(x), T^{(i)}(x) \rangle + \tfrac{L_i}{2} \|T^{(i)}(x)\|_{(i)}^2] \\ &= -L_i u((-\tfrac{\nabla_i f(x)}{L_i})^\#) \qquad (2.30) \\ &\overset{(2.26)}{=} \tfrac{L_i}{2} (\| - \tfrac{\nabla_i f(x)}{L_i} \|_{(i)}^*)^2 = \tfrac{1}{2L_i} (\|\nabla_i f(x)\|_{(i)}^*)^2. \end{aligned}$$

2.5.1 Convex Objective

We are now ready to state the main result of this section.

Theorem 6. *Choose initial point* x_0*, target accuracy* $0 < \epsilon < \min\{f(x_0) - f^*, 2\mathcal{R}^2_{LP^{-1}}(x_0)\}$*, target confidence* $0 < \rho < 1$ *and*

$$k \geq \tfrac{2\mathcal{R}^2_{LP^{-1}}(x_0)}{\epsilon} \left(1 + \log \tfrac{1}{\rho}\right) + 2 - \tfrac{2\mathcal{R}^2_{LP^{-1}}(x_0)}{f(x_0) - f^*}, \qquad (2.31)$$

or

$$k \geq \tfrac{2\mathcal{R}^2_{LP^{-1}}(x_0)}{\epsilon}\left(1+\log\tfrac{1}{\rho}\right)-2. \tag{2.32}$$

If x_k is the random point generated by RCDS(p, x_0) as applied to convex f, then

$$\mathbf{Prob}(f(x_k)-f^*\leq\epsilon)\geq 1-\rho.$$

Proof. Let us first estimate the expected decrease of the objective function during one iteration of the method:

$$\begin{aligned} f(x_k)-\mathbf{E}[f(x_{k+1})\mid x_k] &= \sum_{i=1}^n p_i[f(x_k)-f(x_k+U_iT^{(i)}(x_k))] \\ &\overset{(2.30)}{\geq} \tfrac{1}{2}\sum_{i=1}^n p_i\tfrac{1}{L_i}(\|\nabla_i f(x_k)\|^*_{(i)})^2 = \tfrac{1}{2}(\|\nabla f(x_k)\|^*_w)^2, \end{aligned}$$

where $W=LP^{-1}$. Since $f(x_k)\leq f(x_0)$ for all k and because f is convex, we get $f(x_k)-f^*\leq \max_{x^*\in X^*}\langle\nabla f(x_k), x_k-x^*\rangle\leq\|\nabla f(x_k)\|^*_w\mathcal{R}_W(x_0)$, whence

$$f(x_k)-\mathbf{E}[f(x_{k+1})\mid x_k]\geq\tfrac{1}{2}\left(\tfrac{f(x_k)-f^*}{\mathcal{R}_W(x_0)}\right)^2.$$

By rearranging the terms we obtain

$$\mathbf{E}[f(x_{k+1})-f^*\mid x_k]\leq f(x_k)-f^*-\tfrac{(f(x_k)-f^*)^2}{2\mathcal{R}^2_W(x_0)}.$$

If we now use Theorem 1 with $\xi_k=f(x_k)-f^*$ and $c_1=2\mathcal{R}^2_W(x_0)$, we obtain the result for k given by (2.31). We now claim that $2-\frac{c_1}{\xi_0}\leq -2$, from which it follows that the result holds for k given by (2.32). Indeed, first notice that this inequality is equivalent to

$$f(x_0)-f^*\leq\tfrac{1}{2}\mathcal{R}^2_W(x_0). \tag{2.33}$$

Now, a straightforward extension of Lemma 2 in [43] to general weights states that ∇f is Lipschitz continuous with respect to the norm $\|\cdot\|_V$ with the constant $\mathrm{tr}(LV^{-1})$. This, in turn, implies the inequality

$$f(x)-f^*\leq\tfrac{1}{2}\,\mathrm{tr}(LV^{-1})\|x-x^*\|^2_V,$$

from which (2.33) follows by setting $V=W$ and $x=x_0$. □

2.5.2 Strongly Convex Objective

Assume now that f is strongly convex with respect to the norm $\|\cdot\|_{LP^{-1}}$ (see definition (1.10)) with convexity parameter $\mu_f(LP^{-1}) > 0$. Using (1.10) with $x = x^*$ and $y = x_k$, we obtain

$$\begin{aligned} f^* - f(x_k) &\geq \langle \nabla f(x_k), h\rangle + \tfrac{\mu_f(LP^{-1})}{2}\|h\|^2_{LP^{-1}} \\ &= \mu_f(LP^{-1})\left(\langle \tfrac{1}{\mu_f(LP^{-1})}\nabla f(x_k), h\rangle + \tfrac{1}{2}\|h\|^2_{LP^{-1}}\right), \end{aligned}$$

where $h = x^* - x_k$. Applying Lemma 6 to estimate the right hand side of the above inequality from below, we obtain

$$f^* - f(x_k) \geq -\tfrac{1}{2\mu_f(LP^{-1})}(\|\nabla f(x_k)\|^*_{LP^{-1}})^2. \tag{2.34}$$

Let us now write down an efficiency estimate for the case of a strongly convex objective.

Theorem 7. *Let F be strongly convex with respect to $\|\cdot\|_{LP^{-1}}$ with convexity parameter $\mu_f(LP^{-1}) > 0$. Choose initial point x_0, target accuracy $0 < \epsilon < f(x_0) - f^*$, target confidence $0 < \rho < 1$ and*

$$k \geq \tfrac{1}{\mu_f(LP^{-1})}\log\tfrac{f(x_0)-f^*}{\epsilon\rho}. \tag{2.35}$$

If x_k is the random point generated by RCDS(p, x_0) as applied to f, then

$$\mathbf{Prob}(f(x_k) - f^* \leq \epsilon) \geq 1 - \rho.$$

Proof. The expected decrease of the objective function during one iteration of the method can be estimated as follows:

$$\begin{aligned} f(x_k) - \mathbf{E}[f(x_{k+1}) \mid x_k] &= \sum_{i=1}^n p_i[f(x_k) - f(x_k + U_i T^{(i)}(x_k))] \\ &\overset{(2.30)}{\geq} \tfrac{1}{2}\sum_{i=1}^n p_i \tfrac{1}{L_i}(\|\nabla_i f(x_k)\|^*_{(i)})^2 \\ &= \tfrac{1}{2}(\|\nabla f(x_k)\|^*_{LP^{-1}})^2 \\ &\overset{(2.34)}{\geq} \mu_f(LP^{-1})(f(x_k) - f^*). \end{aligned}$$

After rearranging the terms we obtain $\mathbf{E}[f(x_{k+1}) - f^* \mid x_k] \leq (1 - \mu_f(LP^{-1}))\mathbf{E}[f(x_k) - f^* \mid x_k]$. It now remains to use part (ii) of Theorem 1 with $\xi_k = f(x_k) - f^*$ and $c_2 = \frac{1}{\mu_f(LP^{-1})}$. ☐

The leading factor $\frac{1}{\mu_f(LP^{-1})}$ in the complexity bound (2.35) can in special cases be written in a more natural form; we now give two examples.

1. *Uniform probabilities.* If $p_i = \frac{1}{n}$ for all i, then

$$\frac{1}{\mu_f(LP^{-1})} = \frac{1}{\mu_f(nL)} \overset{(1.15)}{=} \frac{n}{\mu_f(L)} \overset{(2.15)}{=} \frac{\operatorname{tr}(L)}{\mu_f(\tilde{L})} = n\frac{\operatorname{tr}(L)/n}{\mu_f(\tilde{L})}.$$

2. *Probabilities proportional to the Lipschitz constants.* If $p_i = \frac{L_i}{\operatorname{tr}(L)}$ for all i, then

$$\frac{1}{\mu_f(LP^{-1})} = \frac{1}{\mu_f(\operatorname{tr}(L)I)} \overset{(1.15)}{=} \frac{\operatorname{tr}(L)}{\mu_f(I)} = n\frac{\operatorname{tr}(L)/n}{\mu_f(I)}.$$

In both cases, $\frac{1}{\mu_f(LP^{-1})}$ is equal to n multiplied by a *condition number* of the form $\frac{\operatorname{tr}(L)/n}{\mu_f(W)}$, where the numerator is the average of the Lipschitz constants $L_1, \dots, L_n$, W is a diagonal matrix of weights summing up to n and $\mu_f(W)$ is the (strong) convexity parameter of f with respect to $\|\cdot\|_w$.

2.6 Comparison of CD Methods with Complexity Guarantees

In this section we compare the results obtained in this chapter with existing CD methods endowed with iteration complexity bounds.

2.6.1 Smooth Case ($\Psi = 0$)

In Table 2.2 we look at the results for unconstrained smooth minimization of Nesterov [43] and contrast these with our approach. For brevity we only include results for the non-strongly convex case. We will now comment on the contents of Table 2.2 in detail.

1. *Uniform probabilities.* Note that in the uniform case ($p_i = \frac{1}{n}$ for all i) we have

$$\mathcal{R}^2_{LP^{-1}}(x_0) = n\mathcal{R}^2_L(x_0),$$

   and hence the leading term (ignoring the logarithmic factor) in the complexity estimate of Theorem 6 (line 3 of Table 2.2) coincides with the leading term in the complexity estimate of Theorem 2 (line 4 of Table 2.2; the second result): in both cases it is

$$\frac{2n\mathcal{R}^2_L(x_0)}{\epsilon}.$$

   Note that the leading term of the complexity estimate given in Theorem 3 of [43] (line 2 of Table 2.2), which covers the uniform case, is worse by a factor of 4.

Algorithm	Ψ	p_i	Norms	Complexity	Objective
Nesterov [43] (Theorem 4)	0	$\frac{L_i}{\sum_i L_i}$	Eucl.	$(2n + \frac{8(\sum_i L_i)\mathcal{R}_I^2(x_0)}{\epsilon}) \log \frac{4(f(x_0)-f^*)}{\epsilon\rho}$	$f(x) + \frac{\epsilon\\|x-x_0\\|_I^2}{8\mathcal{R}_I^2(x_0)}$
Nesterov [43] (Theorem 3)	0	$\frac{1}{n}$	Eucl.	$\frac{8n\mathcal{R}_L^2(x_0)}{\epsilon} \log \frac{4(f(x_0)-f^*)}{\epsilon\rho}$	$f(x) + \frac{\epsilon\\|x-x_0\\|_L^2}{8\mathcal{R}_L^2(x_0)}$
Algorithm 2.3 (Theorem 6)	0	> 0	general	$\frac{2\mathcal{R}_{LP^{-1}}^2(x_0)}{\epsilon}(1 + \log \frac{1}{\rho}) - 2$	$f(x)$
Algorithm 2.2 (Theorem 2)	separable	$\frac{1}{n}$	Eucl.	$\frac{2n \max\{\mathcal{R}_L^2(x_0), F(x_0)-F^*\}}{\epsilon}(1 + \log \frac{1}{\rho})$ $\frac{2n\mathcal{R}_L^2(x_0)}{\epsilon} \log \frac{F(x_0)-F^*}{\epsilon\rho}$	$F(x)$

Table 2.2: Comparison of our results to the results in [43] in the non-strongly convex case. The complexity is for achieving $\mathbf{Prob}(F(x_k) - F^* \leq \epsilon) \geq 1 - \rho$.

2. *Probabilities proportional to Lipschitz constants.* If we set $p_i = \frac{L_i}{\mathrm{tr}(L)}$ for all i, then

$$\mathcal{R}_{LP^{-1}}^2(x_0) = \mathrm{tr}(L)\mathcal{R}_I^2(x_0).$$

   In this case Theorem 4 in [43] (line 1 of Table 2.2) gives the complexity bound $2[n + \frac{4\,\mathrm{tr}(L)\mathcal{R}_I^2(x_0)}{\epsilon}]$ (ignoring the logarithmic factor), whereas we obtain the bound $\frac{2\,\mathrm{tr}(L)\mathcal{R}_I^2(x_0)}{\epsilon}$ (line 3 of Table 2.2), an improvement by a factor of 4. Note that there is a further additive decrease by the constant $2n$ (*and* the additional constant $\frac{2\mathcal{R}_{LP^{-1}}^2(x_0)}{f(x_0)-f^*} - 2$ if we look at the sharper bound (2.31)).

3. *General probabilities.* Note that unlike the results in [43], which cover the choice of two probability vectors only (lines 1 and 2 of Table 2.2)—uniform and proportional to L_i—our result (line 3 of Table 2.2) covers the case of arbitrary probability vector p. This opens the possibility for fine-tuning the choice of p, in certain situations, so as to minimize $\mathcal{R}_{LP^{-1}}^2(x_0)$.

4. *Logarithmic factor.* Note that in our results we have managed to push ϵ out of the logarithm.

5. *Norms.* Our results hold for general norms.

6. *No need for regularization.* Our results hold for applying the algorithms to F directly; i.e., there is no need to first regularize the function by adding a small quadratic term to it (in a similar fashion as we have done it in Section 2.4.3). This is an essential feature

as the regularization constants are not known and hence the complexity results obtained that way are not true/valid complexity results.

2.6.2 Nonsmooth Case ($\Psi \neq 0$)

In Table 2.3 we summarize the main characteristics of known complexity results for coordinate (or block coordinate) descent methods for minimizing composite functions. Note that the methods of Saha & Tewari and Shalev-Shwartz & Tewari cover the ℓ_1 regularized case only, whereas the other methods cover the general block-separable case. However, while the greedy approach of Yun & Tseng requires per-iteration work which grows with increasing problem dimension, our randomized strategy can be implemented cheaply. This gives an important advantage to randomized methods for problems of large enough size.

Algorithm	Lipschitz constant(s)	Ψ	block	Choice of coordinate	Work per 1 iteration
Yun & Tseng [73]	$L(\nabla f)$	separable	Yes	greedy	expensive
Saha & Tewari [58]	$L(\nabla f)$	$\|\cdot\|_1$	No	cyclic	cheap
Shalev-Shwartz & Tewari [61]	$\|L\|_\infty = \max_i L_i$	$\|\cdot\|_1$	No	$\frac{1}{n}$	cheap
This chapter (Algorithm 2.2)	L_i	separable	Yes	$\frac{1}{n}$	cheap

Table 2.3: Comparison of CD approaches for minimizing composite functions (for which iteration complexity results are provided).

The methods of Yun & Tseng and Saha & Tewari use one Lipschitz constant only, the Lipschitz constant $L(\nabla f)$ of the gradient of f with respect to the standard Euclidean norm. Note that $\max_i L_i \leq L(\nabla f) \leq \sum_i L_i$. If n is large, this constant is typically much larger than the (block) coordinate constants L_i. Shalev-Shwartz & Tewari use coordinate Lipschitz constants, but assume that all of them are the same. This is suboptimal as in many applications the constants $\{L_i\}$ will have a large variation and hence if one chooses $\max_i L_i$ for the common Lipschitz constant, steplengths will necessarily be small (see Figure 2.2 in Section 2.7).[4]

Let us now compare the impact of the Lipschitz constants on the complexity estimates. For simplicity assume $N = n$ and let $u = x^* - x_0$. The estimates are listed in Table 2.4. It is clear

---

[4] Note that in all algorithms it is sufficient to use some upper-bounds on corresponding Lipschitz constants. One can even adopt modified strategy for learning the Lipschitz constant, see e.g. [42].

Algorithm	complexity	complexity (expanded)
Yun & Tseng [73]	$O(\frac{nL(\nabla f)\|x^*-x_0\|_2^2}{\epsilon})$	$O(\frac{n}{\epsilon}\sum_i L(\nabla f)(u^{(i)})^2)$
Saha & Tewari [58]	$O(\frac{nL(\nabla f)\|x^*-x_0\|_2^2}{\epsilon})$	$O(\frac{n}{\epsilon}\sum_i L(\nabla f)(u^{(i)})^2)$
Shwartz & Tewari [61]	$O(\frac{n\|L\|_\infty\|x^*-x_0\|_2^2}{\epsilon})$	$O(\frac{n}{\epsilon}\sum_i(\max_i L_i)(u^{(i)})^2)$
This chapter (Algorithm 2.2)	$O(\frac{n\|x^*-x_0\|_L^2}{\epsilon})$	$O(\frac{n}{\epsilon}\sum_i L_i(u^{(i)})^2)$

Table 2.4: Comparison of iteration complexities of the methods listed in Table 2.3. The complexity in the case of the randomized methods gives iteration counter k for which $\mathbf{E}(F(x_k)-F^*)\leq\epsilon$.

from the last column that the the approach with individual constants L_i for each coordinate gives the best complexity.

2.7 Numerical Experiments

In this section we study the numerical behavior of RCDC on synthetic and real problem instances of two problem classes: Sparse Regression / Lasso [71] (Section 2.7.1) and Linear Support Vector Machines (Section 2.7.2). As an important concern in Section 2.7.1 is to demonstrate that our methods scale well with size, our algorithms were written in C and all experiments were run on a PC with 480GB RAM.

2.7.1 Sparse Regression / Lasso

Consider the problem

$$\min_{x\in\mathbb{R}^n} \tfrac{1}{2}\|Ax-b\|_2^2+\lambda\|x\|_1, \tag{2.36}$$

where $A=[a_1,\dots,a_n]\in\mathbb{R}^{m\times n}$, $b\in\mathbb{R}^m$, and $\lambda\geq 0$. The parameter λ is used to induce sparsity in the resulting solution. Note that (2.36) is of the form (1.1), with $f(x)=\frac{1}{2}\|Ax-b\|_2^2$ and $\Psi(x)=\lambda\|x\|_1$. Moreover, if we let $N=n$ and $U_i=e_i$ for all i, then the Lipschitz constants L_i can be computed explicitly:

$$L_i=\|a_i\|_2^2.$$

Computation of $t=T^{(i)}(x)$ reduces to the "soft-thresholding" operator [82]. In some of the experiments in this section we will allow the probability vector p to change throughout the iterations even though we do not give a theoretical justification for this. With this modification, a direct specialization of RCDC to (2.36) takes the form of Algorithm 2.4. If uniform probabilities are used throughout, we refer to the method as UCDC.

Algorithm 2.4 RCDC for Sparse Regression

```
Choose $x_0 \in \mathbb{R}^n$ and set $g_0 = Ax_0 - b$
for $k = 0, 1, 2, \dots$ do
    Choose $i_k = i \in \{1, 2, \dots, n\}$ with probability $p_k^{(i)}$
    $\alpha = a_i^T g_k$
    $t = \begin{cases} -\frac{\alpha+\lambda}{\|a_i\|_2^2}, & \text{if } x_k^{(i)} - \frac{\alpha+\lambda}{\|a_i\|_2^2} > 0 \\ -\frac{\alpha-\lambda}{\|a_i\|_2^2}, & \text{if } x_k^{(i)} - \frac{\alpha-\lambda}{\|a_i\|_2^2} < 0 \\ -x_k^{(i)}, & \text{otherwise} \end{cases}$
    $x_{k+1} = x_k + te_i, \quad g_{k+1} = g_k + ta_i$
end for
```

Instance generator In order to be able to test Algorithm 2.4 under controlled conditions we use a (variant of the) instance generator proposed in Section 6 of [42] (the generator was presented for $\lambda = 1$ but can be easily extended to any $\lambda > 0$). In it, one chooses the sparsity level of A and the optimal solution x^*; after that A, b, x^* and $F^* = F(x^*)$ are generated. For details we refer the reader to the aforementioned paper.

In what follows we use the notation $\|A\|_0$ and $\|x\|_0$ to denote the number of nonzero elements of matrix A and of vector x, respectively.

Speed versus sparsity In the first experiment we investigate, on problems of size $m = 10^7$ and $n = 10^6$, the dependence of the time it takes for UCDC to complete a block of n iterations (the measurements were done by running the method for $10 \times n$ iterations and then dividing by 10) on the sparsity levels of A and x^*. Looking at Table 2.5, we see that the speed of UCDC depends roughly linearly on the sparsity level of A (and does not depend on $\|x^*\|_0$ at all). Indeed, as $\|A\|_0$ increases from 10^7 through 10^8 to 10^9, the time it takes for the method to complete n iterations increases from about 0.9s through 4–6s to about 46 seconds. This is to be expected since the amount of work per iteration of the method in which coordinate i is chosen is proportional to $\|a_i\|_0$ (computation of α, $\|a_i\|_2^2$ and g_{k+1}).

$\|x^*\|_0$	$\|A\|_0 = 10^7$	$\|A\|_0 = 10^8$	$\|A\|_0 = 10^9$
16×10^2	0.89	5.89	46.23
16×10^3	0.85	5.83	46.07
16×10^4	0.86	4.28	46.93

Table 2.5: The time it takes for UCDC to complete a block of n iterations increases linearly with $\|A\|_0$ and does not depend on $\|x^*\|_0$.

Efficiency on huge-scale problems Tables 2.6 and 2.7 present typical results of the performance of UCDC, started from $x_0 = 0$, on synthetic sparse regression instances of big/huge size. The instance in the first table is of size $m = 2 \times 10^7$ and $n = 10^6$, with A having 5×10^7 nonzeros and the support of x^* being of size $160,000$.

$A \in \mathbb{R}^{2\cdot 10^7 \times 10^6}$, $\|A\|_0 = 5 \cdot 10^7$

k/n	$\frac{F(x_k)-F^*}{F(x_0)-F^*}$	$\|x_k\|_0$	time [sec]
0.00	10^{0}	0	0.0
2.12	10^{-1}	880,056	5.6
4.64	10^{-2}	990,166	12.3
5.63	10^{-3}	996,121	15.1
7.93	10^{-4}	998,981	20.7
10.39	10^{-5}	997,394	27.4
12.11	10^{-6}	993,569	32.3
14.46	10^{-7}	977,260	38.3
18.07	10^{-8}	847,156	48.1
19.52	10^{-9}	701,449	51.7
21.47	10^{-10}	413,163	56.4
23.92	10^{-11}	210,624	63.1
25.18	10^{-12}	179,355	66.6
27.38	10^{-13}	163,048	72.4
29.96	10^{-14}	160,311	79.3
30.94	10^{-15}	160,139	82.0
32.75	10^{-16}	160,021	86.6
34.17	10^{-17}	160,003	90.1
35.26	10^{-18}	160,000	93.0
36.55	10^{-19}	160,000	96.6
38.52	10^{-20}	160,000	101.4
39.99	10^{-21}	160,000	105.3
40.98	10^{-22}	160,000	108.1
43.14	10^{-23}	160,000	113.7
47.28	10^{-24}	160,000	124.8
47.28	10^{-25}	160,000	124.8
47.96	10^{-26}	160,000	126.4
49.58	10^{-27}	160,000	130.3
52.31	10^{-28}	160,000	136.8
53.43	10^{-29}	160,000	139.4

Table 2.6: Performance of UCDC on a sparse regression instance with a million variables.

In both tables the first column corresponds to the "full-pass" iteration counter k/n. That is, after $k = n$ coordinate iterations the value of this counter is 1, reflecting a single "pass" through the coordinates. The remaining columns correspond to, respectively, the size of the current residual $F(x_k) - F^*$ relative to the initial residual $F(x_0) - F^*$, size $\|x_k\|_0$ of the support of the current iterate x_k, and time (in seconds). A row is added whenever the residual initial residual is decreased by an additional factor of 10.

Let us first look at the smaller of the two problems (Table 2.6). After $35 \times n$ coordinate iterations, UCDC decreases the initial residual by a factor of 10^{18}, and this takes about a minute and a half. Note that the number of nonzeros of x_k has stabilized at this point at $160,000$, the support size of the optima solution. The method has managed to identify the support. After 139.4 seconds the residual is decreased by a factor of 10^{29}. This surprising convergence speed and ability to find solutions of high accuracy can in part be explained by the fact that for random instances with $m > n$, f will typically be strongly convex, in which case UCDC converges linearly (Theorem 4). It should also be noted that decrease factors this high ($10^{18} - 10^{29}$) would rarely be needed in practice. However, it is nevertheless interesting to see that a simple algorithm can achieve such high levels of accuracy on certain problem instances

$A \in \mathbb{R}^{10^{10}\times 10^9}$, $\|A\|_0 = 2 \times 10^{10}$

k/n	$\frac{F(x_k)-F^*}{F(x_0)-F^*}$	$\|x_k\|_0$	time [hours]
0	10^0	0	0.00
1	10^{-1}	14,923,993	1.43
3	10^{-2}	22,688,665	4.25
16	10^{-3}	24,090,068	22.65

Table 2.7: Performance of UCDC on a sparse regression instance with a billion variables and 20 billion nonzeros in matrix A.

in huge dimensions.

UCDC has a very similar behavior on the larger problem as well (Table 2.7). Note that A has 20 billion nonzeros. In $1 \times n$ iterations the initial residual is decreased by a factor of 10, and this takes less than an hour and a half. After less than a day, the residual is decreased by a factor of 1000. Note that it is very unusual for convex optimization methods equipped with iteration complexity guarantees to be able to solve problems of these sizes.

Performance on fat matrices ($m < n$) When $m < n$, then f is not strongly convex and UCDC has the complexity $O(\frac{n}{\epsilon}\log\frac{1}{\rho})$ (Theorem 2). In Table 2.8 we illustrate the behavior of the method on such an instance; we have chosen $m = 10^4$, $n = 10^5$, $\|A\|_0 = 10^7$ and $\|x^*\|_0 = 1,600$. Note that after the first $5,010 \times n$ iterations UCDC decreases the residual by a factor of 10+ only; this takes less than 19 minutes. However, the decrease from 10^2 to 10^{-3} is done in $15 \times n$ iterations and takes 3 seconds only, suggesting very fast local convergence.

k/n	$F(x_k) - F^*$	$\|x_k\|_0$	time [s]
1	$> 10^7$	63,106	0.21
$5,010$	$< 10^6$	33,182	1,092.59
$18,286$	$< 10^5$	17,073	3,811.67
$21,092$	$< 10^4$	15,077	4,341.52
$21,416$	$< 10^3$	11,469	4,402.77
$21,454$	$< 10^2$	5,316	4,410.09
$21,459$	$< 10^1$	1,856	4,411.04
$21,462$	$< 10^0$	1,609	4,411.63
$21,465$	$< 10^{-1}$	1,600	4,412.21
$21,468$	$< 10^{-2}$	1,600	4,412.79
$21,471$	$< 10^{-3}$	1,600	4,413.38

Table 2.8: UCDC needs many more iterations when $m < n$, but local convergence is still fast.

Comparing different probability vectors Nesterov [43] considers only probabilities proportional to a power of the Lipschitz constants:

$$p_i = \frac{L_i^\alpha}{\sum_{i=1}^n L_i^\alpha}, \qquad 0 \le \alpha \le 1. \tag{2.37}$$

In Figure 2.1 we compare the behavior of RCDC, with the probability vector chosen according to the power law (2.37), for three different values of α (0, 0.5 and 1). All variants of RCDC were compared on a single instance with $m = 1,000$, $n = 2,000$ and $\|x^*\|_0 = 300$ (different instances produced by the generator yield similar results) and with $\lambda \in \{0, 1\}$. The plot on the left corresponds to $\lambda = 0$, the plot on the right to $\lambda = 1$.

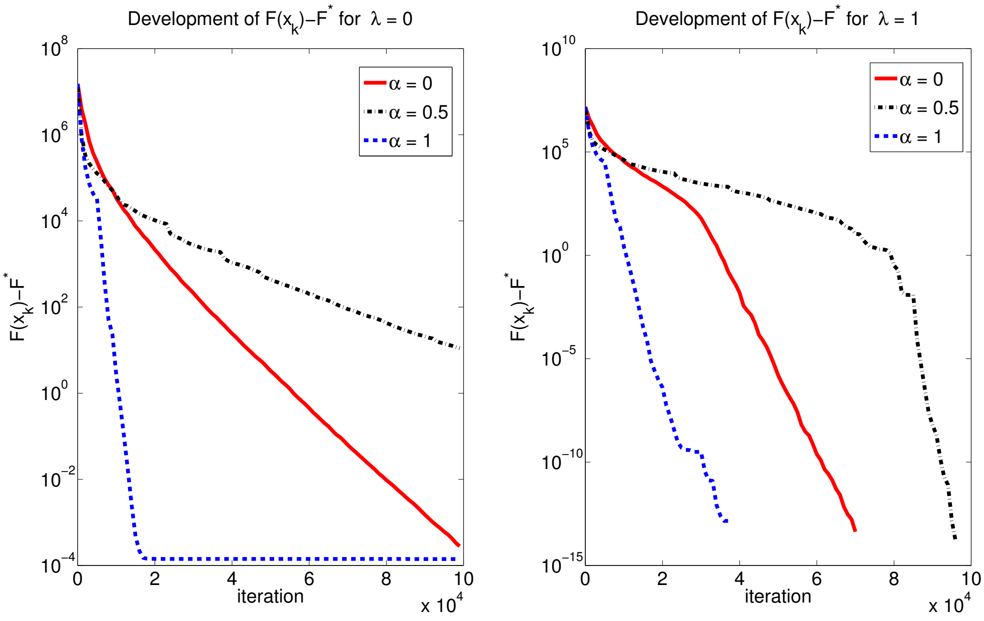


Figure 2.1: Development of $F(x_k) - F^*$ for sparse regression problem with $\lambda = 0$ (left) and $\lambda = 1$ (right).

Note that in both cases the choice $\alpha = 1$ is the best. In other words, coordinates with large L_i have a tendency to decrease the objective function the most. However, looking at the $\lambda = 0$ case, we see that the method with $\alpha = 1$ stalls after about 20,000 iterations. The reason for this is that now the coordinates with small L_i should be chosen to further decrease the objective value. However, they are chosen with very small probability and hence the slowdown. A solution to this could be to start the method with $\alpha = 1$ and then switch to $\alpha = 0$ later on. On the problem with $\lambda = 1$ this effect is less pronounced. This is to be expected as now the objective function is a combination of f and Ψ, with Ψ exerting its influence and mitigating the effect of the Lipschitz constants.

Coordinate descent vs. a full-gradient method In Figure 2.2 we compare the performance of RCDC with the full gradient (FG) algorithm [42] (with the Lipschitz constant $L_{FG} = \lambda_{\max}(A^T A)$ for four different distributions of the Lipschitz constants L_i. Note that $\max_i L_i \leq L_{FG} \leq \sum_i L_i$. Since the work performed during one iteration of FG is comparable with the work performed by UCDC during n coordinate iterations[5], for FG we multiply the iteration count by n. In all four tests we solve instances with $A \in \mathbb{R}^{2{,}000\times 1{,}000}$.

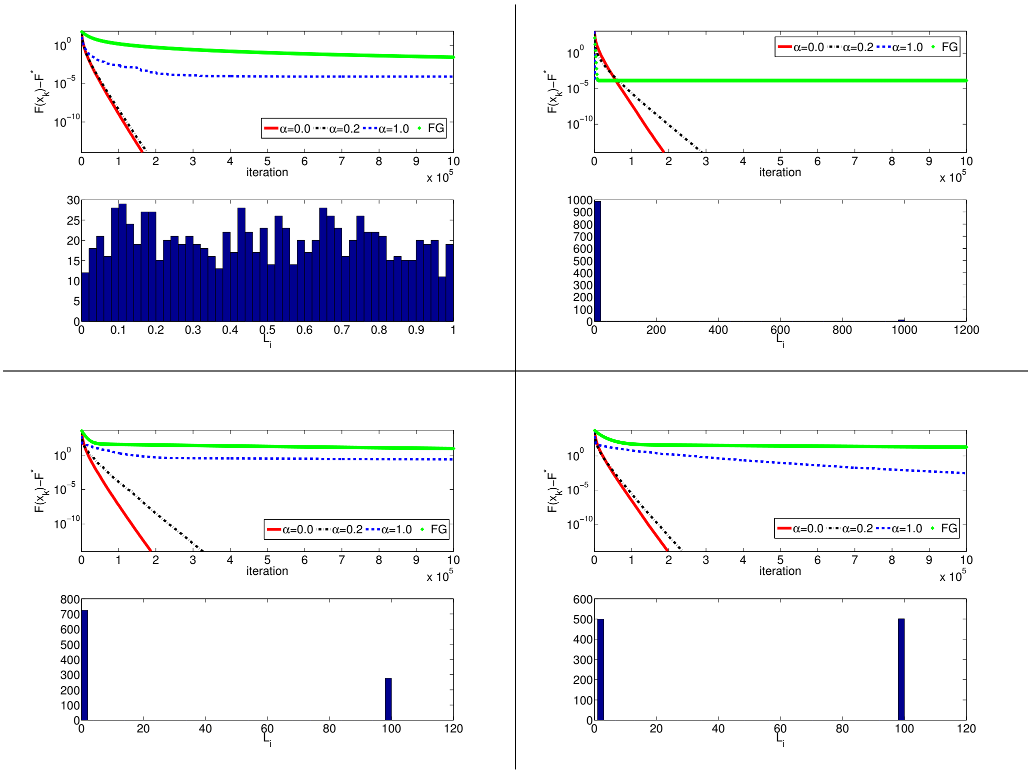


Figure 2.2: Comparison of RCDC with different choices of α with a full-gradient method (essentially UCDC with one block: $n = 1$) for four different distributions of the Lipschitz constants L_i.

In the 1-1 plot of Figure 2.2 (plot in the 1-1 position, i.e., in the upper-left corner), the Lipschitz constants L_i were generated uniformly at random in the interval $(0, 1)$. We see that the RCDC variants with $\alpha = 0$ and $\alpha = 0.2$ exhibit virtually the same behavior, whereas $\alpha = 1$ and FG struggle finding a solution with error tolerance below 10^{-5} and 10^{-2}, respectively. The $\alpha = 1$ method does start off a bit faster, but then stalls due to the fact that the coordinates with small Lipschitz constants are chosen with extremely small probabilities. For a more accurate solution one needs to be updating these coordinates as well.

In order to zoom in on this phenomenon, in the 1-2 plot we construct an instance with an extreme distribution of Lipschitz constants: 98% of the constants have the value 10^{-6}, whereas the remaining 2% have the value 10^3. Note that while the FG and $\alpha = 1$ methods are able to quickly decrease the objective function within 10^{-4} of the optimum, they get stuck afterwards since they effectively never update the coordinates with $L_i = 10^{-6}$. On the other hand, the $\alpha = 0$ method starts off slowly, but does not stop and manages to solve the problem eventually,

---

[5] This will not be the case for certain types of matrices, such as those arising from wavelet bases or FFT.

in about 2×10^5 iterations.

In the 2-1 (resp. 2-2) plot we choose 70% (resp. 50%) of the Lipschitz constants L_i to be 1, and the remaining 30% (resp. 50%) equal to 100. Again, the $\alpha = 0$ and $\alpha = 0.2$ methods give the best long-term performance.

In summary, if fast convergence to a solution with a moderate accuracy us needed, then $\alpha = 1$ is the best choice (and is always better than FG). If one desires a solution of higher accuracy, it is recommended to switch to $\alpha = 0$. In fact, it turns out that we can do much better than this using a "shrinking" heuristic.

Speedup by adaptive change of probability vectors It is well-known that increasing values of λ encourage increased sparsity in the solution of (2.36). In the experimental setup of this section we observe that from certain iteration onwards, the sparsity pattern of the iterates of RCDC is a very good predictor of the sparsity pattern of the optimal solution x^* the iterates converge to. More specifically, we often observe in numerical experiments that for large enough k the following holds:

$$(x_k^{(i)} = 0) \quad \Rightarrow \quad (\forall l \geq k \quad x_l^{(i)} = (x^*)^{(i)} = 0). \tag{2.38}$$

In words, for large enough k, zeros in x_k typically stay zeros in all subsequent iterates[6] and correspond to zeros in x^*. Note that RCDC is *not* able to take advantage of this. Indeed, RCDC, as presented in the theoretical sections of this chapter, uses the fixed probability vector p to randomly pick a single coordinate i to be updated in each iteration. Hence, eventually, $\sum_{i:x_k^{(i)}=0} p_i$ proportion of time will be spent on vacuous updates.

Looking at the data in Table 2.6 one can see that after approximately $35 \times n$ iterations, x_k has the same number of non-zeros as x^* (160,000). What is not visible in the table is that, in fact, the relation (2.38) holds for this instance much sooner. In Figure 2.3 we illustrate this phenomenon in more detail on an instance with $m = 500$, $n = 1,000$ and $\|x^*\|_0 = 100$.

First, note that the *number of nonzeros* (solid blue line) in the current iterate, $\#\{i : x_k^{(i)} \neq 0\}$, is first growing from zero (since we start with $x_0 = 0$) to just below n in about 0.6×10^4 iterations. This value then starts to decrease starting from about $k \approx 15n$ and reaches the optimal number of nonzeros at iteration $k \approx 30n$ and stays there afterwards. Note that the

---

[6] There are various theoretical results on the *identification of active manifolds* explaining numerical observations of this type; see [24] and the references therein. See also [82].

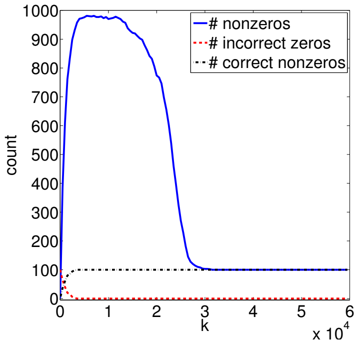


Figure 2.3: Development of non-zero elements in x_k through iterations.

number of *correct nonzeros*,

$$cn_k = \#\{i : x_k^{(i)} \neq 0 \ \&\ (x^*)^{(i)} \neq 0\},$$

is increasing (for this particular instance) and reaches the optimal level $\|x^*\|_0$ very quickly (at around $k \approx 3n$). An alternative, and perhaps a more natural, way to look at the same thing is via the *number of incorrect zeros*,

$$iz_k = \#\{i : x_k^{(i)} = 0 \ \&\ (x^*)^{(i)} \neq 0\}.$$

Indeed, we have $cn_k + iz_k = \|x^*\|_0$. Note that for our problem $iz_k \approx 0$ for $k \geq k_0 \approx 3n$.

The above discussion suggests that an *iterate-dependent* policy for updating of the probability vectors p_k in Algorithm 2.4 might help to accelerate the method. Let us now introduce a simple *q-shrinking* strategy for adaptively changing the probabilities as follows: at iteration $k \geq k_0$, where k_0 is large enough, set

$$p_k^{(i)} = \hat{p}_k^{(i)}(q) \overset{\text{def}}{=} \begin{cases} \frac{1-q}{n}, & \text{if } x_k^{(i)} = 0, \\ \frac{1-q}{n} + \frac{q}{\|x_k\|_0}, & \text{otherwise.} \end{cases}$$

This is equivalent to choosing i_k uniformly from the set $\{1, 2, \dots, n\}$ with probability $1-q$ and uniformly from the support set of x_k with probability q. Clearly, different variants of this can be implemented, such as fixing a new probability vector for $k \geq k_0$ (as opposed to changing it for every k) ; and some may be more effective and/or efficient than others in a particular context. In Figure 2.4 we illustrate the effectiveness of q-shrinking on an instance of size $m = 500$, $n = 1,000$ with $\|x^*\|_0 = 50$. We apply to this problem a modified version of RCDC started from the origin ($x_0 = 0$) in which uniform probabilities are used in iterations $0, \dots, k_0 - 1$, and

q-shrinking is introduced as of iteration k_0:

$$p_k^{(i)} = \begin{cases} \frac{1}{n}, & \text{for} \quad k = 0, 1, \dots, k_0 - 1, \\ \hat{p}_k^{(i)}(q), & \text{for} \quad k \geq k_0. \end{cases}$$

We have used $k_0 = 5 \times n$.

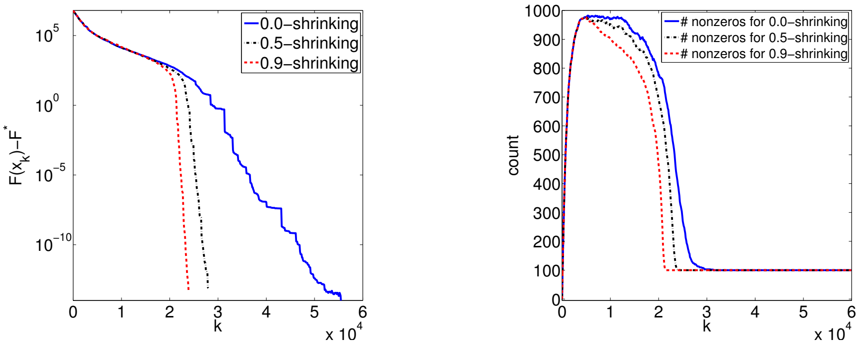


Figure 2.4: Comparison of different shrinking strategies.

Notice that as the number of nonzero elements of x_k decreases, the time savings from q-shrinking grow. Indeed, 0.9-shrinking introduces a saving of nearly 70% when compared to 0-shrinking to obtain x_k satisfying $F(x_k) - F^* \leq 10^{-14}$. We have repeated this experiment with two modifications: a) a random point was used as the initial iterate (scaled so that $\|x_0\|_0 = n$) and b) $k_0 = 0$. The corresponding plots are very similar to Figure 2.4 with the exception that the lines in the second plot start from $\|x_0\|_0 = n$.

2.7.2 Linear Support Vector Machines

Consider the problem of training a linear classifier with training examples $\{(x_1, y_1), \dots, (x_m, y_m)\}$, where x_i are the feature vectors and $y_i \in \{-1, +1\}$ the corresponding labels (classes). This problem is usually cast as an optimization problem of the form (1.1),

$$\min_{w \in \mathbb{R}^n} F(w) = f(w) + \Psi(w), \tag{2.39}$$

where

$$f(w) = \gamma \sum_{i=1}^{m} \mathcal{L}(w; x_i, y_i),$$

$\mathcal{L}$ is a nonnegative convex loss function and $\Psi(\cdot) = \|\cdot\|_1$ for L1-regularized and $\Psi(\cdot) = \|\cdot\|_2$ for L2-regularized linear classifier. Some popular loss functions are listed in Table 2.9. For more details we refer the reader to [82] and the references therein; for a survey of recent advances in large-scale linear classification see [81].

$\mathcal{L}(w; x_i, y_i)$	name	property
$\max\{0, 1 - y_j w^T x_j\}$	L1-SVM loss (L1-SVM)	C^0 continuous
$\max\{0, 1 - y_j w^T x_j\}^2$	L2-SVM loss (L2-SVM)	C^1 continuous
$\log(1 + e^{-y_j w^T x_j})$	logistic loss (LG)	C^2 continuous

Table 2.9: A list of a few popular loss functions.

Because our setup requires f to be at least C^1 continuous, we will consider the L2-SVM and LG loss functions only. In the experiments below we consider the L1 regularized setup.

A few implementation remarks The Lipschitz constants and coordinate derivatives of f for the L2-SVM and LG loss functions are listed in Table 2.10.

Loss function	L_i	$\nabla_i f(w)$
L2-SVM	$2\gamma \sum_{j=1}^{m} (y_j x_j^{(i)})^2$	$-2\gamma \cdot \sum_{j \,:\, -y_j w^T x_j > -1} y_j x_j^{(i)} (1 - y_j w^T x_j)$
LG	$\frac{\gamma}{4} \sum_{j=1}^{m} (y_j x_j^{(i)})^2$	$-\gamma \cdot \sum_{j=1}^{m} y_j x_j^{(i)} \frac{e^{-y_j w^T x_j}}{1 + e^{-y_j w^T x_j}}$

Table 2.10: Lipschitz constants and coordinate derivatives for SVM.

For an efficient implementation of UCDC we need to be able to cheaply update the partial derivatives after each step of the method. If at step k coordinate i gets updated, via $w_{k+1} = w_k + te_i$, and we let $r_k^{(j)} \overset{\text{def}}{=} -y_j w^T x_j$ for $j = 1, \dots, m$, then

$$r_{k+1}^{(j)} = r_k^{(j)} - t y_j x_j^{(i)}, \quad j = 1, \dots, m. \tag{2.40}$$

Let o_i be the number of observations feature i appears in, i.e., $o_i = \#\{j : x_j^{(i)} \neq 0\}$. Then the update (2.40), and consequently the update of the partial derivative (see Table 2.10), requires $O(o_i)$ operations. In particular, in *feature-sparse problems* where $\frac{1}{n} \sum_{i=1}^{n} o_i \ll m$, an average iteration of UCDC will be very cheap.

Small scale test

We perform only preliminary results on the dataset `rcv1.binary`[7]. This dataset has 47,236 features and 20,242 training and 677,399 testing instances. We train the classifier on 90% of training instances (18,217); the rest we used for cross-validation for the selection of the

---

[7] `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html`

parameter γ. In Table 2.11 we list cross-validation accuracy (CV-A) for various choices of γ and testing accuracy (TA) on 677,399 instances. The best constant γ is 1 for both loss functions in cross-validation.

Loss function	γ	CV-A	TA	γ	CV-A	TA
L2-SVM	0.0625	94.1%	93.2%	2	97.0%	95.6%
	0.1250	95.5%	94.5%	4	97.0%	95.4%
	0.2500	96.5%	95.4%	8	96.9%	95.1%
	0.5000	97.0%	95.8%	16	96.7%	95.0%
	1.0000	**97.0%**	**95.8%**	32	96.4%	94.9%
LG	0.5000	0.0%	0.0%	8	40.7%	37.0%
	1.0000	**96.4%**	**95.2%**	16	37.7%	36.0%
	2.0000	43.2%	39.4%	32	37.6%	33.4%
	4.0000	39.3%	36.5%	64	36.9%	34.1%

Table 2.11: Cross validation accuracy (CV-A) and testing accuracy (TA) for various choices of γ.

In Figure 2.5 we present dependence of TA on the number of iterations we run UCDC for (we measure this number in multiples of n). As you can observe, UCDC finds good solution after $10 \times n$ iterations, which for this data means less then half a second. Let us remark that we did not include bias term or any scaling of the data.

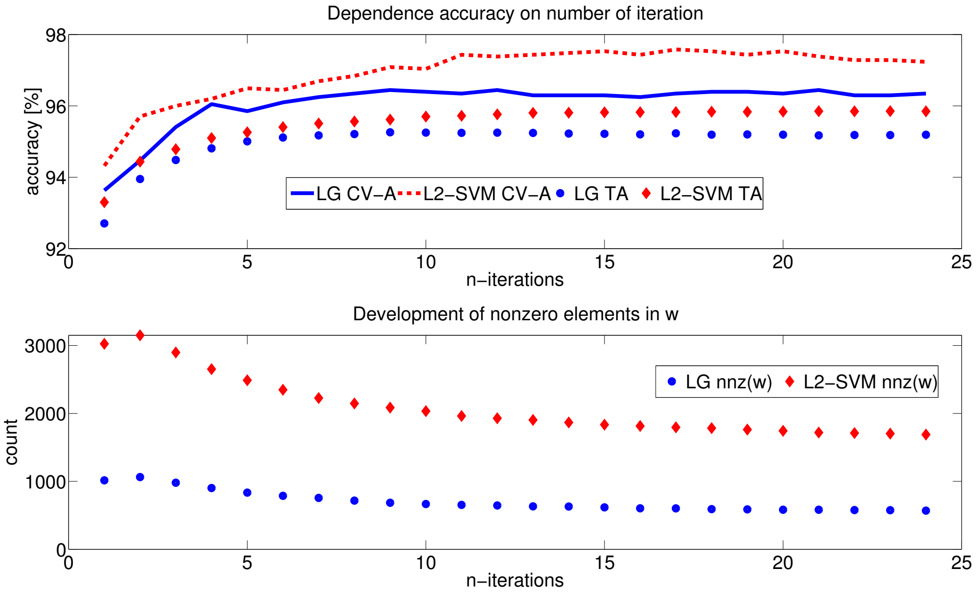


Figure 2.5: Dependence of tested accuracy (TA) on the number of full passes through the coordinates.

Large scale test We have used the dataset `kdd2010` (bridge to algebra)[8], which has 29,890,095 features and 19,264,097 training and 748,401 testing instances. Training the classifier on the entire training set required approximately 70 seconds in the case of L2-SVM loss and 112 seconds in the case of LG loss. We have run UCDC for $n \approx 30 \cdot 10^6$ iterations. This again demonstrates the efficiency of coordinate descent algorithms analyzed in this chapter.

---

[8] `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html`

3

Parallel Coordinate Descent Method

If only I had the theorems! Then
I should find the proofs easily
enough.

— Richard P. Feynman
(1918-1988)

The work presented in this chapter was motivated by the desire to answer the following question:

> *Under what natural and easily verifiable structural assumptions on the objective function does parallelization of a coordinate descent method lead to acceleration?*

Our starting point was the following simple observation. Assume that we wish to minimize a separable function F of n variables (i.e., a function that can be written as a sum of n functions each of which depends on a single variable only). For simplicity, in this thought experiment, assume that there are no constraints. Clearly, the problem of minimizing F can be trivially decomposed into n independent univariate problems. Now, if we have n processors/threads/cores, each assigned with the task of solving one of these problems, the number of parallel iterations should not depend on the dimension of the problem[1]. In other words, we get an n-times speedup compared to the situation with a single processor only. Note that any parallel algorithm of this type can be viewed as a parallel coordinate descent method. Hence, a PCDM with n processors should be n-times faster than a serial one. If τ processors are used instead, where $1 \leq \tau \leq n$, one would expect a τ-times speedup.

By extension, one would perhaps expect that optimization problems with objective functions which are "close to being separable" would also be amenable to acceleration by parallelization,

---

[1] For simplicity, assume the distance from the starting point to the set of optimal solutions does not depend on the dimension.

where the acceleration factor τ would be reduced with the reduction of the "degree of separability". One of the main messages of this chapter is an affirmative answer to this. Moreover, we give explicit and simple formulae for the speedup factors.

As it turns out, and as we discuss later in this section, many real-world big data optimization problems are, quite naturally, "close to being separable". We believe that this means that PCDMs is a very promising class of algorithms when it comes to solving structured big data optimization problems.

Minimizing a partially separable composite objective. In this chapter we study the problem (1.1) with one modification, and it is the assumption that f is also a (block) *partially separable* smooth convex function. Let us now describe the key concept of partial separability. In the light of block structure introduced in Section 1.5.1 we know that $x \in \mathbb{R}^N$ can be decomposed into n non-overlapping blocks of variables $x^{(1)}, \dots, x^{(n)}$. We assume throughout this chapter that $f : \mathbb{R}^N \to \mathbb{R}$ is *partially separable of degree* ω, i.e., that it can be written in the form

$$f(x) = \sum_{J \in \mathcal{J}} f_J(x), \tag{3.1}$$

where $\mathcal{J}$ is a finite collection of nonempty subsets of $[n] \overset{\text{def}}{=} \{1, 2, \dots, n\}$ (possibly containing identical sets multiple times), f_J are differentiable convex functions such that f_J depends on blocks $x^{(i)}$ for $i \in J$ only, and

$$|J| \leq \omega \quad \text{for all} \quad J \in \mathcal{J}. \tag{3.2}$$

Clearly, $1 \leq \omega \leq n$. The PCDM algorithms we develop and analyze in this chapter only need to know ω, they do not need to know the decomposition of f giving rise to this ω.

Examples of partially separable functions. Many objective functions naturally encountered in the big data setting are partially separable. Here we give examples of three loss/objective functions frequently used in the machine learning literature and also elsewhere. For simplicity, we assume all blocks are of size 1 (i.e., $N = n$). Let

$$f(x) = \sum_{j=1}^{m} \mathcal{L}(x, A_j, y_j), \tag{3.3}$$

where m is the number of examples, $x \in \mathbb{R}^n$ is the vector of features, $(A_j, y_j) \in \mathbb{R}^n \times \mathbb{R}$ are labeled examples and $\mathcal{L}$ is one of the three loss functions listed in Table 3.1. Let $A \in \mathbb{R}^{m \times n}$ with row j equal to A_j^T.

Square Loss (SL)	$\frac{1}{2}(A_j^T x - y_j)^2$
Logistic Loss (LL)	$\log(1 + e^{-y_j A_j^T x})$
Hinge Square Loss (HL)	$\frac{1}{2}\max\{0, 1 - y_j A_j^T x\}^2$

Table 3.1: Three examples of loss of functions.

Often, each example depends on a few features only; the maximum over all features is the degree of partial separability ω. More formally, note that the j-th function in the sum (3.3) in all cases depends on $\|A_j\|_0$ coordinates of x (the number of nonzeros in the j-th row of A) and hence f is partially separable of degree

$$\omega = \max_j \|A_j\|_0.$$

All three functions of Table 3.1 are smooth (based on the definition of smoothness in Section 1.5.3). We refer the reader to [50] for more examples of interesting (but nonsmooth) partially separable functions arising in graph cuts and matrix completion.

3.1 Literature review

Several papers were written recently studying the iteration complexity of *serial* CDMs of various flavours and in various settings. We will only provide a brief summary here, for a more detailed account we refer the reader to [53].

Classical CDMs update the coordinates in a cyclic order; the first attempt at analyzing the complexity of such a method is due to [58]. Stochastic/randomized CDMs, that is, methods where the coordinate to be updated is chosen randomly, were first analyzed for quadratic objectives [65, 23], later independently generalized to L_1-regularized problems [61] and smooth block-structured problems [43], and finally unified and refined in [51, 53]. The problems considered in the above papers are either unconstrained or have (block) separable constraints. Recently, randomized CDMs were developed for problems with linearly coupled constraints [39, 40].

A greedy CDM for L_1-regularized problems was first analyzed in [52]; more work on this topic include [26, 12]. A CDM with inexact updates was first proposed and analyzed in [69]. Partially separable problems were independently studied in [50], where an asynchronous parallel stochastic gradient algorithm was developed to solve them.

When writing this chapter, the authors were aware only of the parallel CDM proposed and analyzed in [5]. Several papers on the topic appeared around the time this chapter was finalized or after [38, 80, 59, 59, 47, 21, 29, 30, 64, 63, 9]. Further papers on various aspects of the topic of parallel CDMs, building on the work in this chapter, include [66, 55, 17, 56, 70].

3.2 Contents

We start in Section 3.3, where we propose and comment in detail on two parallel coordinate descent methods. In Section 3.4 we summarize the main contributions of this chapter. In Section 3.5 we deal with issues related to the selection of the blocks to be updated in each iteration. It will involve the development of some elementary random set theory. Sections 3.6-3.7 deal with issues related to the computation of the update to the selected blocks and develop a theory of Expected Separable Overapproximation (ESO), which is a novel tool we propose for the analysis of our algorithms. In Section 3.8 we analyze the iteration complexity of our methods and finally, Section 3.10 reports on promising computational results. For instance, we conduct an experiment with a big data (cca 350GB) LASSO problem with a billion variables. We are able to solve the problem using one of our methods on a large memory machine with 24 cores in 2 hours, pushing the difference between the objective value at the starting iterate and the optimal point from 10^{22} down to 10^{-14}. We also conduct experiments on real data problems coming from machine learning.

3.3 The Algorithm

In this chapter we develop and study two generic parallel coordinate descent methods. The main method is PCDM1; PCDM2 is its “regularized” version which explicitly enforces monotonicity. As we will see, both of these methods come in many variations, depending on how Step 3 is performed.

Algorithm 3.1 PCDM1 (**P**arallel **C**oordinate **D**escent **M**ethod 1)

```
1: Choose initial point $x_0 \in \mathbb{R}^N$
2: for $k = 0, 1, 2, \dots$ do
3:     Randomly generate a set of blocks $S_k \subseteq \{1, 2, \dots, n\}$
4:     $x_{k+1} \leftarrow x_k + (h(x_k))_{[S_k]}$          (in parallel)
5: end for
```

Let us comment on the individual steps of the two methods.

Step 3. At the beginning of iteration k we pick a random set (S_k) of blocks to be updated (in parallel) during that iteration. The set S_k is a realization of a random set-valued mapping

Algorithm 3.2 PCDM2 (**P**arallel **C**oordinate **D**escent **M**ethod 2)

```
Choose initial point $x_0 \in \mathbb{R}^N$
for $k = 0, 1, 2, \dots$ do
    Randomly generate a set of blocks $S_k \subseteq \{1, 2, \dots, n\}$
    $x_{k+1} \leftarrow x_k + (h(x_k))_{[S_k]}$        (in parallel)
    If $F(x_{k+1}) > F(x_k)$, then $x_{k+1} \leftarrow x_k$
end for
```

$\hat{S}$ with values in $2^{[n]}$ or, more precisely, the sets S_k are iid random sets with the distribution of $\hat{S}$. For brevity, in this chapter we refer to such a mapping by the name *sampling*. We limit our attention to *uniform* samplings, i.e., random sets having the following property: $\mathbf{Prob}(i \in \hat{S})$ is independent of i. That is, the probability that a block gets selected is the same for all blocks. Although we give an iteration complexity result covering all such samplings (provided that each block has a chance to be updated, i.e., $\mathbf{Prob}(i \in \hat{S}) > 0$), there are interesting subclasses of uniform samplings (such as doubly uniform and nonoverlapping uniform samplings; see Section 3.5) for which we give better results.

Step 4. For $x \in \mathbb{R}^N$ we define[2].

$$h(x) \overset{\text{def}}{=} \arg\min_{h \in \mathbb{R}^N} H_{\beta,w}(x,h), \tag{3.4}$$

where

$$H_{\beta,w}(x,h) \overset{\text{def}}{=} f(x) + \langle \nabla f(x), h \rangle + \tfrac{\beta}{2}\|h\|_w^2 + \Psi(x+h), \tag{3.5}$$

and $\beta > 0$, $w = (w_1, \dots, w_n)^T \in \mathbb{R}^n_{++}$ are parameters of the method that we will comment on later. Note that in view of (1.2), (1.5) and (1.9), $H_{\beta,w}(x,\cdot)$ is block separable;

$$H_{\beta,w}(x,h) = f(x) + \sum_{i=1}^n \left\{ \langle \nabla_i f(x), h^{(i)} \rangle + \tfrac{\beta w_i}{2}\|h^{(i)}\|_{(i)}^2 + \Psi_i(x^{(i)} + h^{(i)}) \right\}.$$

Consequently, we have $h(x) = (h^{(1)}(x), \cdots, h^{(n)}(x)) \in \mathbb{R}^N$, where

$$h^{(i)}(x) = \arg\min_{t \in \mathbb{R}^{N_i}} \{ \langle \nabla_i f(x), t \rangle + \tfrac{\beta w_i}{2}\|t\|_{(i)}^2 + \Psi_i(x^{(i)} + t) \}.$$

We mentioned in the introduction that besides (block) separability, we require Ψ to be "simple". By this we mean that the above optimization problem leading to $h^{(i)}(x)$ is "simple" (e.g., it has a closed-form solution). Recall from (1.20) that $(h(x_k))_{[S_k]}$ is the vector in $\mathbb{R}^N$ identical to

---

[2] A similar map was used in [43] (with $\Omega \equiv 0$ and $\beta = 1$) and [53] (with $\beta = 1$) in the analysis of *serial* coordinate descent methods in the smooth and composite case, respectively. In loose terms, the novelty here is the introduction of the parameter β and in developing theory which describes what value β should have. Maps of this type are known as composite gradient mapping in the literature, and were introduced in [42]

$h(x_k)$ except for blocks $i \notin S_k$, which are zeroed out. Hence, Step 4 of both methods can be written as follows:

$$\text{In parallel for } i \in S_k \text{ do:} \quad x_{k+1}^{(i)} \leftarrow x_k^{(i)} + h^{(i)}(x_k).$$

Parameters β and w depend on f and $\hat{S}$ and stay constant throughout the algorithm. We are not ready yet to explain *why* the update is computed via (3.4) and (3.5) because we need technical tools, which will be developed in Section 3.5, to do so. Here it suffices to say that the parameters β and w come from a separable quadratic overapproximation of $\mathbf{E}[f(x + h_{[\hat{S}]})]$, viewed as a function of $h \in \mathbb{R}^N$. Since expectation is involved, we refer to this by the name Expected Separable Overapproximation (ESO). This novel concept, developed in this chapter, is one of the main tools of our complexity analysis. Section 3.6 motivates and formalizes the concept, answers the *why* question, and develops some basic ESO theory.

Section 3.7 is devoted to the computation of β and w for partially separable f and various special classes of uniform samplings $\hat{S}$. Typically we will have $w_i = L_i$, while β will depend on *easily computable* properties of f and $\hat{S}$. For example, if $\hat{S}$ is chosen as a subset of $[n]$ of cardinality τ, with each subset chosen with the same probability (we say that $\hat{S}$ is τ-nice) then, assuming $n > 1$, we may choose $w = L$ and $\beta = 1 + \frac{(\omega-1)(\tau-1)}{n-1}$, where ω is the degree of partial separability of f. More generally, if $\hat{S}$ is any uniform sampling with the property $|\hat{S}| = \tau$ with probability 1, then we may choose $w = L$ and $\beta = \min\{\omega, \tau\}$. Note that in both cases $w = L$ and that the latter β is always larger than (or equal to) the former one. This means, as we will see in Section 3.8, that we can give better complexity results for the former, more specialized, sampling. We analyze several more options for $\hat{S}$ than the two just described, and compute parameters β and w that should be used with them (for a summary, see Table 3.4).

Step 5. The reason why, besides PCDM1, we also consider PCDM2, is the following: in some situations we are not able to analyze the iteration complexity of PCDM1 (non-strongly-convex F where monotonicity of the method is not guaranteed by other means than by directly enforcing it by inclusion of Step 5). Let us remark that this issue arises for general Ψ only. It does not exist for $\Psi = 0$, $\Psi(\cdot) = \lambda\|\cdot\|_1$ and for Ψ encoding simple constraints on individual blocks; in these cases one does not need to consider PCDM2. Even in the case of general Ψ we sometimes get monotonicity for free, in which case there is no need to enforce it. Let us stress, however, that we do not recommend implementing PCDM2 as this would introduce too much overhead; in our experience PCDM1 works well even in cases when we can only analyze PCDM2.

3.4 Summary of Contributions

In this section we summarize the main contributions of this chapter (not in order of significance).

1. **Problem generality.** We give the first complexity analysis for *parallel* coordinate descent methods for problem (1.1) in its full generality.

2. **Complexity.** We show theoretically (Section 3.8) and numerically (Section 3.10) that PCDM accelerates on its serial counterpart for partially separable problems. In particular, we establish two complexity theorems giving lower bounds on the number of iterations k sufficient for one or both of the PCDM variants (for details, see the precise statements in Section 3.8) to produce a random iterate x_k for which the problem is approximately solved with high probability, i.e., $\mathbf{Prob}(F(x_k) - F^* \leq \epsilon) \geq 1 - \rho$. The results, summarized in Table 3.2, hold under the standard assumptions listed in Section 1.5.1 *and* the additional assumption that $f, \hat{S}, \beta$ and w satisfy the following inequality for all $x, h \in \mathbb{R}^N$:

$$\mathbf{E}[f(x + h_{[\hat{S}]})] \leq f(x) + \tfrac{\mathbf{E}[|\hat{S}|]}{n} \left(\langle \nabla f(x), h \rangle + \tfrac{\beta}{2} \|h\|_w^2 \right). \tag{3.6}$$

   This inequality, which we call Expected Separable Overapproximation (ESO), is the main new theoretical tool that we develop in this chapter for the analysis of our methods (Sections 3.5-3.7 are devoted to the development of this theory).

Setting	Complexity	Theorem
Convex f	$\mathcal{O}\left(\frac{\beta n}{\mathbf{E}[\lvert\hat{S}\rvert]} \frac{1}{\epsilon} \log\left(\frac{1}{\rho}\right)\right)$	19
Strongly convex f $\mu_f(W) + \mu_\Psi(W) > 0$	$\frac{n}{\mathbf{E}[\lvert\hat{S}\rvert]} \frac{\beta + \mu_\Psi(W)}{\mu_f(W) + \mu_\Psi(W)} \log\left(\frac{F(x_0) - F^*}{\epsilon\rho}\right)$	20

Table 3.2: Summary of the main complexity results for PCDM established in this chapter.

   The main observation here is that as the average number of block updates per iteration increases (say, $\hat{\tau} = \mathbf{E}[|\hat{S}|]$), enabled by the utilization of more processors, the leading term in the complexity estimate, $n/\hat{\tau}$, decreases in proportion. However, β will generally grow with $\hat{\tau}$, which has an adverse effect on the speedup. Much of the theory in this chapter goes towards producing formulas for β (and w), for partially separable f and various classes of *uniform* samplings $\hat{S}$. Naturally, the ideal situation is when β does not grow with $\hat{\tau}$ at all, or if it only grows very slowly. We show that this is the case for partially separable functions f with small ω. For instance, in the extreme case when f is separable ($\omega = 1$), we have $\beta = 1$ and we obtain linear speedup in $\hat{\tau}$. As ω increases, so does β, depending on the law governing $\hat{S}$. Formulas for β and ω for various samplings $\hat{S}$ are summarized in Table 3.4.

3. **Algorithm unification.** Depending on the choice of the block structure (as implied by the choice of n and the matrices $U_1, \dots, U_n$) and the way blocks are selected at every iteration (as given by the choice of $\hat{S}$), our framework encodes a *family* of known and new algorithms[3] (see Table 3.3).

Method	Parameters	Comment
Gradient descent	$n = 1$	[42]
Serial random CDM	$N_i = 1$ for all i and $\mathbf{Prob}(\lvert\hat{S}\rvert = 1) = 1$	[53]
Serial block random CDM	$N_i \geq 1$ for all i and $\mathbf{Prob}(\lvert\hat{S}\rvert = 1) = 1$	[53]
Parallel random CDM	$\mathbf{Prob}(\lvert\hat{S}\rvert > 1) > 0$	NEW
Distributed random CDM	$\hat{S}$ is a distributed sampling	[55]

Table 3.3: New and known gradient methods obtained as special cases of our general framework.

In particular, PCDM is the first method which "continuously" interpolates between serial coordinate descent and gradient (by manipulating n and/or $\mathbf{E}[|\hat{S}|]$).

4. **Partial separability.** We give the first analysis of a coordinate descent type method dealing with a partially separable loss / objective. In order to run the method, we need to know the Lipschitz constants L_i and the degree of partial separability ω. It is crucial that these quantities are often easily computable/predictable in the huge-scale setting. For example, if $f(x) = \frac{1}{2}\|Ax - b\|^2$ and we choose all blocks to be of size 1, then L_i is equal to the squared Euclidean norm of the i-th column of A and ω is equal to the maximum number of nonzeros in a row of A. Many problems in the big data setting have small ω compared to n.

5. **Choice of blocks.** To the best of our knowledge, existing randomized strategies for paralleling gradient-type methods (e.g., [5]) assume that $\hat{S}$ (or an equivalent thereof, based on the method) is chosen as a subset of $[n]$ of a fixed cardinality, uniformly at random. We refer to such $\hat{S}$ by the name *nice sampling* in this chapter. We relax this assumption and our treatment is hence much more general. In fact, we allow for $\hat{S}$ to be any *uniform* sampling. It is possible to further consider *nonuniform samplings*[4], but this is beyond the scope of this chapter.

---

[3] All the methods are in their proximal variants due to the inclusion of the term Ψ in the objective.

[4] see e.g. our work in [56].

In particular, as a special case, our method allows for a variable number of blocks to be updated throughout the iterations (this is achieved by the introduction of *doubly uniform samplings*). This may be useful in some settings such as when the problem is being solved in parallel by τ unreliable processors each of which computes its update $h^{(i)}(x_k)$ with probability p_b and is busy/down with probability $1 - p_b$ (*binomial sampling*).

Uniform, doubly uniform, nice, binomial and other samplings are defined, and their properties studied, in Section 3.5.

6. **ESO and formulas for β and w.** In Table 3.4 we list parameters β and w for which ESO inequality (3.6) holds. Each row corresponds to a specific sampling $\hat{S}$ (see Section 3.5 for the definitions). The last 5 samplings are special cases of one or more of the first three samplings. Details such as what is ν, γ and "monotonic" ESO are explained in appropriate sections later in the text. When a specific sampling $\hat{S}$ is used in the algorithm to select blocks in each iteration, the corresponding parameters β and w are to be used in the method for the computation of the update (see (3.4) and (3.5)).

sampling $\hat{S}$	$\mathbf{E}[\|\hat{S}\|]$	β	w	ESO monotonic?	Follows from
uniform	$\mathbf{E}[\|\hat{S}\|]$	1	$\nu \odot L$	No	Thm 14
nonoverlapping uniform	$\frac{n}{l}$	1	$\gamma \odot L$	Yes	Thm 15
doubly uniform	$\mathbf{E}[\|\hat{S}\|]$	$1 + \frac{(\omega-1)\left(\frac{\mathbf{E}[\|\hat{S}\|^2]}{\mathbf{E}[\|\hat{S}\|]}-1\right)}{\max(1,n-1)}$	L	No	Thm 17
τ-uniform	τ	$\min\{\omega, \tau\}$	L	Yes	Thm 14
τ-nice	τ	$1 + \frac{(\omega-1)(\tau-1)}{\max(1,n-1)}$	L	No	Thm 16/17
(τ, p_b)-binomial	τp_b	$1 + \frac{p_b(\omega-1)(\tau-1)}{\max(1,n-1)}$	L	No	Thm 17
serial	1	1	L	Yes	Thm 15/16/17
fully parallel	n	ω	L	Yes	Thm 15/16/17

Table 3.4: Values of parameters β and w for various samplings $\hat{S}$.

En route to proving the iteration complexity results for our algorithms, we develop a theory of deterministic and expected separable overapproximation (Sections 3.6 and 3.7) which we believe is of independent interest, too. For instance, methods based on ESO can be compared favorably to the Diagonal Quadratic Approximation (DQA) approach used in the decomposition of stochastic optimization programs [57].

7. **Parallelization speedup.** Our complexity results can be used to derive theoretical *parallelization speedup factors*. For several variants of our method, in case of a non-strongly convex objective, these are given in Section 3.8.1 (Table 3.5). For instance, in the case when all block are updated at each iteration (we later refer to $\hat{S}$ having this

property by the name *fully parallel* sampling), the speedup factor is equal to $\frac{n}{\omega}$. If the problem is separable ($\omega = 1$), the speedup is equal to n; if the problem is not separable ($\omega = n$), there may be no speedup. For strongly convex F the situation is even better; the details are given in Section 2.4.2.

8. **Relationship to existing results.** To the best of our knowledge, there are just two papers analyzing a parallel coordinate descent algorithm for convex optimization problems[5, 38]. In the first paper all blocks are of size 1, $\hat{S}$ corresponds to what we call in this chapter a τ-*nice* sampling (i.e., all sets of τ coordinates are updated at each iteration with equal probability) and hence their algorithm is somewhat comparable to one of the many variants of our general method. While the analysis in [5] works for a restricted range of values of τ, our results hold for all $\tau \in [n]$. Moreover, the authors consider a more restricted class of functions f and the special case $\Psi = \lambda\|x\|_1$, which is simpler to analyze. Lastly, the theoretical speedups obtained in [5], when compared to the serial CDM method, depend on a quantity σ that is hard to compute in big data settings (it involves the computation of an eigenvalue of a huge-scale matrix). Our speedups are expressed in terms of natural and easily computable quantity: the degree ω of partial separability of f. In the setting considered by [5], in which more structure is available, it turns out that ω is an upper bound on σ. Hence, we show that one can develop the theory in a more general setting, and that it is not necessary to compute σ (which may be complicated in the big data setting). The parallel CDM method of the second paper [38] only allows all blocks to be updated at each iteration. Unfortunately, the analysis (and the method) is too coarse as it does not offer any theoretical speedup when compared to its serial counterpart. In the special case when only a single block is updated in each iteration, uniformly at random, our theoretical results specialize to those established in [53].

9. **Computations.** We demonstrate that our method is able to solve a LASSO problem involving a matrix with a billion columns and 2 billion rows on a large memory node with 24 cores in 2 hours (Section 3.10), achieving a 20× speedup compared to the serial variant and pushing the residual by more than 30 degrees of magnitude. While this is done on an artificial problem under ideal conditions (controlling for small ω), large speedups are possible in real data with ω small relative to n. We also perform additional experiments on real data sets from machine learning (e.g., training linear SVMs) to illustrate that the predictions of our theory match reality.

10. **Code.** The open source code with an efficient implementation of the algorithm(s) developed in this chapter is published here: http://code.google.com/p/ac-dc/.

3.5 Block Samplings

In Step 3 of both PCDM1 and PCDM2 we choose a random set of blocks S_k to be updated at the current iteration. Formally, S_k is a realization of a *random set-valued mapping* $\hat{S}$ with values in $2^{[n]}$, the collection of subsets of $[n]$. For brevity, in this chapter we refer to $\hat{S}$ by the name *sampling*. A sampling $\hat{S}$ is uniquely characterized by the probability mass function

$$\mathbf{Prob}(S) \overset{\text{def}}{=} \mathbf{Prob}(\hat{S} = S), \quad S \subseteq [n]; \tag{3.7}$$

that is, by assigning probabilities to all subsets of $[n]$. Further, we let $p = (p_1, \dots, p_n)^T$, where

$$p_i \overset{\text{def}}{=} \mathbf{Prob}(i \in \hat{S}). \tag{3.8}$$

In Section 3.5.1 we describe those samplings for which we analyze our methods and in Section 3.5.2 we prove several technical results, which will be useful in the rest of the chapter.

3.5.1 Uniform, Doubly Uniform and Nonoverlapping Uniform Samplings

A sampling is *proper* if $p_i > 0$ for all blocks i. That is, from the perspective of PCDM, under a proper sampling each block gets updated with a positive probability at each iteration. Clearly, PCDM can not converge for a sampling that is not proper.

A sampling $\hat{S}$ is *uniform* if all blocks get updated with the same probability, i.e., if $p_i = p_j$ for all i, j. We show in (3.20) that, necessarily, $p_i = \frac{\mathbf{E}[|\hat{S}|]}{n}$. Further, we say $\hat{S}$ is *nil* if $\mathbf{Prob}(\emptyset) = 1$. Note that a uniform sampling is proper if and only if it is not nil.

All our iteration complexity results in this chapter are for PCDM used with a proper uniform sampling (see Theorems 18 and 19) for which we can compute β and w giving rise to an inequality (we call "expected separable overapproximation") of the form (3.30). We derive such inequalities for all proper uniform samplings (Theorem 14) as well as refined results for two special subclasses thereof: doubly uniform samplings (Theorem 17) and nonoverlapping uniform samplings (Theorem 15). We will now give the definitions:

1. **Doubly Uniform (DU) samplings.** A DU sampling is one which generates all sets of equal cardinality with equal probability. That is, $\mathbf{Prob}(S') = \mathbf{Prob}(S'')$ whenever

$|S'| = |S''|$. The name comes from the fact that this definition postulates a different uniformity property, "standard" uniformity is a consequence. Indeed, let us show that a DU sampling is necessarily uniform. Let $q_j = \mathbf{Prob}(|\hat{S}| = j)$ for $j = 0, 1, \dots, n$ and note that from the definition we know that whenever S is of cardinality j, we have $\mathbf{Prob}(S) = q_j / \binom{n}{j}$. Finally, using this we obtain

$$p_i = \sum_{S: i \in S} \mathbf{Prob}(S) = \sum_{j=1}^{n} \sum_{\substack{S: i \in S \\ |S| = j}} \mathbf{Prob}(S) = \sum_{j=1}^{n} \sum_{\substack{S: i \in S \\ |S| = j}} \frac{q_j}{\binom{n}{j}} = \sum_{j=1}^{n} \frac{\binom{n-1}{j-1}}{\binom{n}{j}} q_j$$

$$= \tfrac{1}{n} \sum_{j=1}^{n} q_j j = \tfrac{\mathbf{E}[|\hat{S}|]}{n}.$$

It is clear that each DU sampling is uniquely characterized by the vector of probabilities q; its density function is given by

$$\mathbf{Prob}(S) = \frac{q_{|S|}}{\binom{n}{|S|}}, \quad S \subseteq [n]. \tag{3.9}$$

2. **Nonoverlapping Uniform (NU) samplings.** A NU sampling is one which is uniform and which assigns positive probabilities only to sets forming a partition of $[n]$. Let $S^1, S^2, \dots, S^l$ be a partition of $[n]$, with $|S^j| > 0$ for all j. The density function of a NU sampling corresponding to this partition is given by

$$\mathbf{Prob}(S) = \begin{cases} \frac{1}{l}, & \text{if } S \in \{S^1, S^2, \dots, S^l\}, \\ 0, & \text{otherwise.} \end{cases} \tag{3.10}$$

   Note that $\mathbf{E}[|\hat{S}|] = \frac{n}{l}$.

Let us now describe several interesting special cases of DU and NU samplings:

3. **Nice sampling.** Fix $1 \le \tau \le n$. A τ-nice sampling is a DU sampling with $q_\tau = 1$.

   Interpretation: There are τ processors/threads/cores available. At the beginning of each iteration we choose a set of blocks using a τ-nice sampling (i.e., each subset of τ blocks is chosen with the same probability), and assign each block to a dedicated processor/thread-/core. Processor assigned with block i would compute and apply the update $h^{(i)}(x_k)$. This is the sampling we use in our computational experiments.

4. **Independent sampling.** Fix $1 \le \tau \le n$. A τ-independent sampling is a DU sampling

with

$$q_k = \begin{cases} \binom{n}{k} c_k, & k = 1, 2, \dots, \tau, \\ 0, & k = \tau + 1, \dots, n, \end{cases}$$

where $c_1 = \left(\frac{1}{n}\right)^\tau$ and $c_k = \left(\frac{k}{n}\right)^\tau - \sum_{i=1}^{k-1} \binom{k}{i} c_i$ for $k \geq 2$.

Interpretation: There are τ processors/threads/cores available. Each processor chooses one of the n blocks, uniformly at random and independently of the other processors. It turns out that the set $\hat{S}$ of blocks selected this way is DU with q as given above. Since in one parallel iteration of our methods each block in $\hat{S}$ is updated exactly once, this means that if two or more processors pick the same block, all but one will be idle. On the other hand, this sampling can be generated extremely easily and in parallel! For $\tau \ll n$ this sampling is a good (and fast) approximation of the τ-nice sampling. For instance, for $n = 10^3$ and $\tau = 8$ we have $q_8 = 0.9723$, $q_7 = 0.0274$, $q_6 = 0.0003$ and $q_k \approx 0$ for $k = 1, \dots, 5$.

5. **Binomial sampling.** Fix $1 \leq \tau \leq n$ and $0 < p_b \leq 1$. A (τ, p_b)-binomial sampling is defined as a DU sampling with

$$q_k = \binom{\tau}{k} p_b^k (1 - p_b)^{\tau - k}, \quad k = 0, 1, \dots, \tau. \tag{3.11}$$

Notice that $\mathbf{E}[|\hat{S}|] = \tau p_b$ and $\mathbf{E}[|\hat{S}|^2] = \tau p_b(1 + \tau p_b - p_b)$.

Interpretation: Consider the following situation with *independent equally unreliable processors.* We have τ processors, each of which is at any given moment available with probability p_b and busy with probability $1 - p_b$, independently of the availability of the other processors. Hence, the number of available processors (and hence blocks that can be updated in parallel) at each iteration is a binomial random variable with parameters τ and p_b. That is, the number of available processors is equal to k with probability q_k.

- *Case 1 (explicit selection of blocks):* We learn that k processors are available *at the beginning* of each iteration. Subsequently, we choose k blocks using a k-nice sampling and "assign one block" to each of the k available processors.
- *Case 2 (implicit selection of blocks):* We choose τ blocks using a τ-nice sampling and assign one to each of the τ processors (we do not know which will be available at the beginning of the iteration). With probability q_k, k of these will send their updates. It is easy to check that the resulting effective sampling of blocks is (τ, p_b)-binomial.

6. **Serial sampling.** This is a DU sampling with $q_1 = 1$. Also, this is a NU sampling with $l = n$ and $S^j = \{j\}$ for $j = 1, 2, \dots, l$. That is, at each iteration we update a single block, uniformly at random. This was studied in [53].

7. **Fully parallel sampling.** This is a DU sampling with $q_n = 1$. Also, this is a NU sampling with $l = 1$ and $S^1 = [n]$. That is, at each iteration we update all blocks.

The following simple result says that the intersection between the class of DU and NU samplings is very thin. A sampling is called *vacuous* if $\mathbf{Prob}(\emptyset) > 0$.

Proposition 2. *There are precisely two nonvacuous samplings which are both DU and NU: i) the serial sampling and ii) the fully parallel sampling.*

Proof. Assume $\hat{S}$ is nonvacuous, NU and DU. Since $\hat{S}$ is nonvacuous, $\mathbf{Prob}(\hat{S} = \emptyset) = 0$. Let $S \subset [n]$ be any set for which $\mathbf{Prob}(\hat{S} = S) > 0$. If $1 < |S| < n$, then there exists $S' \neq S$ of the same cardinality as S having a nonempty intersection with S. Since $\hat{S}$ is doubly uniform, we must have $\mathbf{Prob}(\hat{S} = S') = \mathbf{Prob}(\hat{S} = S) > 0$. However, this contradicts the fact that $\hat{S}$ is non-overlapping. Hence, $\hat{S}$ can only generate sets of cardinalities 1 or n with positive probability, but not both. One option leads to the fully parallel sampling, the other one leads to the serial sampling. □

3.5.2 Technical Results

For a given sampling $\hat{S}$ and $i, j \in [n]$ we let

$$p_{ij} \overset{\text{def}}{=} \mathbf{Prob}(i \in \hat{S}, j \in \hat{S}) = \sum_{S : \{i,j\} \subset S} \mathbf{Prob}(S). \tag{3.12}$$

The following simple result has several consequences which will be used throughout the chapter.

Lemma 7 (Sum over a random index set)**.** *Let $\emptyset \neq J \subset [n]$ and $\hat{S}$ be any sampling. If θ_i, $i \in [n]$, and θ_{ij}, for $(i, j) \in [n] \times [n]$ are real constants, then*[5]

$$\mathbf{E}\left[\sum_{i \in J \cap \hat{S}} \theta_i\right] = \sum_{i \in J} p_i \theta_i,$$

$$\mathbf{E}\left[\sum_{i \in J \cap \hat{S}} \theta_i \mid |J \cap \hat{S}| = k\right] = \sum_{i \in J} \mathbf{Prob}(i \in \hat{S} \mid |J \cap \hat{S}| = k)\theta_i, \tag{3.13}$$

---

[5] Sum over an empty index set will, for convenience, be defined to be zero.

$$\mathbf{E}\left[\sum_{i\in J\cap\hat{S}}\sum_{j\in J\cap\hat{S}}\theta_{ij}\right] = \sum_{i\in J}\sum_{j\in J} p_{ij}\theta_{ij}. \tag{3.14}$$

Proof. We prove the first statement, proof of the remaining statements is essentially identical:

$$\begin{aligned}\mathbf{E}\left[\sum_{i\in J\cap\hat{S}}\theta_i\right] &\overset{(3.7)}{=} \sum_{S\subset[n]}\left(\sum_{i\in J\cap S}\theta_i\right)\mathbf{Prob}(S) = \sum_{i\in J}\sum_{S:i\in S}\theta_i\mathbf{Prob}(S)\\ &= \sum_{i\in J}\theta_i\sum_{S:i\in S}\mathbf{Prob}(S) = \sum_{i\in J}p_i\theta_i.\end{aligned}$$

□

The consequences are summarized in the next theorem and the discussion that follows.

Theorem 8. *Let $\emptyset \neq J \subset [n]$ and $\hat{S}$ be an arbitrary sampling. Further, let $a, h \in \mathbb{R}^N$, $w \in \mathbb{R}^n_+$ and let g be a block separable function, i.e., $g(x) = \sum_i g_i(x^{(i)})$. Then*

$$\begin{aligned}\mathbf{E}\left[|J\cap\hat{S}|\right] &= \sum_{i\in J} p_i, &(3.15)\\ \mathbf{E}\left[|J\cap\hat{S}|^2\right] &= \sum_{i\in J}\sum_{j\in J} p_{ij}, &(3.16)\\ \mathbf{E}\left[\langle a, h_{[\hat{S}]}\rangle_w\right] &= \langle a, h\rangle_{p\odot w}, &(3.17)\\ \mathbf{E}\left[\|h_{[\hat{S}]}\|_w^2\right] &= \|h\|^2_{p\odot w}, &(3.18)\\ \mathbf{E}\left[g(x+h_{[\hat{S}]})\right] &= \sum_{i=1}^n\left[p_i g_i(x^{(i)}+h^{(i)}) + (1-p_i)g_i(x^{(i)})\right]. &(3.19)\end{aligned}$$

Moreover, the matrix $P \overset{def}{=} (p_{ij})$ is positive semidefinite.

Proof. Noting that $|J\cap\hat{S}| = \sum_{i\in J\cap\hat{S}} 1$, $|J\cap\hat{S}|^2 = (\sum_{i\in J\cap\hat{S}} 1)^2 = \sum_{i\in J\cap\hat{S}}\sum_{j\in J\cap\hat{S}} 1$, $\langle a, h_{[\hat{S}]}\rangle_w = \sum_{i\in\hat{S}} w_i\langle a^{(i)}, h^{(i)}\rangle$, $\|h_{[\hat{S}]}\|^2_w = \sum_{i\in\hat{S}} w_i\|h^{(i)}\|^2_{(i)}$ and

$$g(x+h_{[\hat{S}]}) = \sum_{i\in\hat{S}} g_i(x^{(i)}+h^{(i)}) + \sum_{i\notin\hat{S}} g_i(x^{(i)}) = \sum_{i\in\hat{S}} g_i(x^{(i)}+h^{(i)}) + \sum_{i=1}^n g_i(x^{(i)}) - \sum_{i\in\hat{S}} g_i(x^{(i)}),$$

all five identities follow directly by applying Lemma 7. Finally, for any $\theta = (\theta_1, \dots, \theta_n)^T \in \mathbb{R}^n$,

$$\theta^T P\theta = \sum_{i=1}^n\sum_{j=1}^n p_{ij}\theta_i\theta_j \overset{(3.14)}{=} \mathbf{E}[(\sum_{i\in\hat{S}}\theta_i)^2] \geq 0. \qquad \square$$

The above results hold for arbitrary samplings. Let us specialize them, in order of decreasing generality, to uniform, doubly uniform and nice samplings.

- **Uniform samplings.** If $\hat{S}$ is uniform, then from (3.15) using $J = [n]$ we get

$$p_i = \tfrac{\mathbf{E}[|\hat{S}|]}{n}, \qquad i \in [n]. \tag{3.20}$$

Plugging (3.20) into (3.15), (3.17), (3.18) and (3.19) yields

$$\begin{aligned} \mathbf{E}\left[|J \cap \hat{S}|\right] &= \tfrac{|J|}{n}\mathbf{E}[|\hat{S}|], & (3.21)\\ \mathbf{E}\left[\langle a, h_{[\hat{S}]}\rangle_w\right] &= \tfrac{\mathbf{E}[|\hat{S}|]}{n}\langle a, h\rangle_w, & (3.22)\\ \mathbf{E}\left[\|h_{[\hat{S}]}\|_w^2\right] &= \tfrac{\mathbf{E}[|\hat{S}|]}{n}\|h\|_w^2, & (3.23)\\ \mathbf{E}\left[g(x + h_{[\hat{S}]})\right] &= \tfrac{\mathbf{E}[|\hat{S}|]}{n} g(x+h) + \left(1 - \tfrac{\mathbf{E}[|\hat{S}|]}{n}\right) g(x). & (3.24) \end{aligned}$$

- **Doubly uniform samplings.** Consider the case $n > 1$; the case $n = 1$ is trivial. For doubly uniform $\hat{S}$, p_{ij} is constant for $i \neq j$:

$$p_{ij} = \tfrac{\mathbf{E}[|\hat{S}|^2 - |\hat{S}|]}{n(n-1)}. \tag{3.25}$$

Indeed, this follows from

$$p_{ij} = \sum_{k=1}^{n} \mathbf{Prob}(\{i,j\} \subseteq \hat{S} \mid |\hat{S}| = k)\mathbf{Prob}(|\hat{S}| = k) = \sum_{k=1}^{n} \tfrac{k(k-1)}{n(n-1)}\mathbf{Prob}(|\hat{S}| = k).$$

Substituting (3.25) and (3.20) into (3.16) then gives[6]

$$\mathbf{E}[|J \cap \hat{S}|^2] = (|J|^2 - |J|)\tfrac{\mathbf{E}[|\hat{S}|^2 - |\hat{S}|]}{n\max\{1,n-1\}} + |J|\tfrac{\mathbf{E}[|\hat{S}|]}{n}. \tag{3.26}$$

- **Nice samplings.** Finally, if $\hat{S}$ is τ-nice (and $\tau \neq 0$), then $\mathbf{E}[|\hat{S}|] = \tau$ and $\mathbf{E}[|\hat{S}|^2] = \tau^2$, which used in (3.26) gives

$$\mathbf{E}[|J \cap \hat{S}|^2] = \tfrac{|J|\tau}{n}\left(1 + \tfrac{(|J|-1)(\tau-1)}{\max\{1,n-1\}}\right). \tag{3.27}$$

Moreover, assume that $\mathbf{Prob}(|J \cap \hat{S}| = k) \neq 0$ (this happens precisely when $0 \leq k \leq |J|$ and $k \leq \tau \leq n - |J| + k$). Then for all $i \in J$,

$$\mathbf{Prob}(i \in \hat{S} \mid |J \cap \hat{S}| = k) = \frac{\binom{|J|-1}{k-1}\binom{n-|J|}{\tau-k}}{\binom{|J|}{k}\binom{n-|J|}{\tau-k}} = \frac{k}{|J|}.$$

---

[6] The following formula holds also for the trivial case $n = 1$.

Substituting this into (3.13) yields

$$\mathbf{E}\left[\sum_{i\in J\cap\hat{S}} \theta_i \mid |J\cap\hat{S}| = k\right] = \tfrac{k}{|J|}\sum_{i\in J}\theta_i. \tag{3.28}$$

3.6 Expected Separable Overapproximation

Recall that given x_k, in PCDM1 the next iterate is the random vector $x_{k+1} = x_k + h_{[\hat{S}]}$ for a particular choice of $h \in \mathbb{R}^N$. Further recall that in PCDM2,

$$x_{k+1} = \begin{cases} x_k + h_{[\hat{S}]}, & \text{if } F(x_k + h_{[\hat{S}]}) \le F(x_k), \\ x_k, & \text{otherwise,} \end{cases}$$

again for a particular choice of h. While in Section 3.3 we mentioned how h is computed, i.e., that h is the minimizer of $H_{\beta,w}(x,\cdot)$ (see (3.4) and (3.5)), we did not explain *why* is h computed this way. The reason for this is that the tools needed for this were not yet developed at that point (as we will see, some results from Section 3.5 are needed). In this section we give an answer to this *why* question.

Given $x_k \in \mathbb{R}^N$, after one step of PCDM1 performed with update h we get $\mathbf{E}[F(x_{k+1}) \mid x_k] = \mathbf{E}[F(x_k + h_{[\hat{S}]}) \mid x_k]$. On the the other hand, after one step of PCDM2 we have

$$\mathbf{E}[F(x_{k+1}) \mid x_k] = \mathbf{E}[\min\{F(x_k + h_{[\hat{S}]}), F(x_k)\} \mid x_k] \le \min\{\mathbf{E}[F(x_k + h_{[\hat{S}]}) \mid x_k], F(x_k)\}.$$

So, for both PCDM1 and PCDM2 the following estimate holds,

$$\mathbf{E}[F(x_{k+1}) \mid x_k] \le \mathbf{E}[F(x_k + h_{[\hat{S}]}) \mid x_k]. \tag{3.29}$$

A good choice for h to be used in the algorithms would be one minimizing the right hand side of inequality (3.29). At the same time, we would like the minimization process to be decomposable so that the updates $h^{(i)}$, $i \in \hat{S}$, could be computed in parallel. However, the problem of finding such h is intractable in general even if we do not require parallelizability. Instead, we propose to construct/compute a "simple" separable overapproximation of the right-hand side of (3.29). Since the overapproximation will be separable, parallelizability is guaranteed; "simplicity" means that the updates $h^{(i)}$ can be computed easily (e.g., in closed form).

From now on we replace, for simplicity and w.l.o.g., the random vector x_k by a fixed deterministic vector $x \in \mathbb{R}^N$. We can thus remove conditioning in (3.29) and instead study the quantity $\mathbf{E}[F(x+h_{[\hat{S}]})]$. Further, fix $h \in \mathbb{R}^N$. Note that if we can find $\beta > 0$ and $w \in \mathbb{R}^n_{++}$ such that

$$\mathbf{E}\left[f\left(x+h_{[\hat{S}]}\right)\right] \quad \leq \quad f(x) + \tfrac{\mathbf{E}[|\hat{S}|]}{n}\left(\langle \nabla f(x), h\rangle + \tfrac{\beta}{2}\|h\|_w^2\right), \tag{3.30}$$

we indeed find a simple separable overapproximation of $\mathbf{E}[F(x+h_{[\hat{S}]})]$:

$$\begin{aligned} \mathbf{E}[F(x+h_{[\hat{S}]})] \quad &\overset{(1.1)}{=} \quad \mathbf{E}[f(x+h_{[\hat{S}]}) + \Psi(x+h_{[\hat{S}]})] \\ &\overset{(3.30),(3.24)}{\leq} \quad f(x) + \tfrac{\mathbf{E}[|\hat{S}|]}{n}\left(\langle \nabla f(x), h\rangle + \tfrac{\beta}{2}\|h\|_w^2\right) \\ &\qquad + \left(1 - \tfrac{\mathbf{E}[|\hat{S}|]}{n}\right)\Psi(x) + \tfrac{\mathbf{E}[|\hat{S}|]}{n}\Psi(x+h) \\ &= \quad \left(1 - \tfrac{\mathbf{E}[|\hat{S}|]}{n}\right)F(x) + \tfrac{\mathbf{E}[|\hat{S}|]}{n}H_{\beta,w}(x,h), \end{aligned} \tag{3.31}$$

where we recall from (3.5) that $H_{\beta,w}(x,h) = f(x) + \langle \nabla f(x), h\rangle + \frac{\beta}{2}\|h\|_w^2 + \Psi(x+h)$.

That is, (3.31) says that the expected objective value after one parallel step of our methods, if block $i \in \hat{S}$ is updated by $h^{(i)}$, is bounded above by a convex combination of $F(x)$ and $H_{\beta,w}(x,h)$. The natural choice of h is to set

$$h(x) = \arg\min_{h \in \mathbb{R}^N} H_{\beta,w}(x,h). \tag{3.32}$$

Note that this is precisely the choice we make in our methods. Since $H_{\beta,w}(x,0) = F(x)$, both PCDM1 and PCDM2 are monotonic in expectation.

The above discussion leads to the following definition.

Definition 9 (Expected Separable Overapproximation (ESO))**.** Let $\beta > 0$, $w \in \mathbb{R}^n_{++}$ and let $\hat{S}$ be a proper uniform sampling. We say that $f : \mathbb{R}^N \to \mathbb{R}$ admits a (β, w)-ESO with respect to $\hat{S}$ if inequality (3.30) holds for all $x, h \in \mathbb{R}^N$. For simplicity, we write $(f, \hat{S}) \sim ESO(\beta, w)$.

A few remarks:

1. **Inflation.** If $(f, \hat{S}) \sim ESO(\beta, w)$, then for $\beta' \geq \beta$ and $w' \geq w$, $(f, \hat{S}) \sim ESO(\beta', w')$.
2. **Reshuffling.** Since for any $c > 0$ we have $\|h\|_{cw}^2 = c\|h\|_w^2$, one can "shuffle" constants between β and w as follows:

$$(f, \hat{S}) \sim ESO(c\beta, w) \quad \Leftrightarrow \quad (f, \hat{S}) \sim ESO(\beta, cw), \qquad c > 0. \tag{3.33}$$

3. **Strong convexity.** If $(f,\hat{S}) \sim ESO(\beta, w)$, then

$$\beta \geq \mu_f(W). \tag{3.34}$$

Indeed, it suffices to take expectation in (1.10) with y replaced by $x + h_{[\hat{S}]}$ and compare the resulting inequality with (3.30) and using (3.22) and (3.23) (this gives $\beta\|h\|_w^2 \geq \mu_f(W)\|h\|_w^2$, which must hold for all h).

Recall that Step 5 of PCDM2 was introduced so as to explicitly enforce monotonicity into the method as in some situations, as we will see in Section 3.8, we can only analyze a monotonic algorithm. However, sometimes even PCDM1 behaves monotonically (without enforcing this behavior externally as in PCDM2). The following definition captures this.

Definition 10 (Monotonic ESO)**.** Assume $(f,\hat{S}) \sim ESO(\beta, w)$ and let $h(x)$ be as in (3.32). We say that the ESO is *monotonic* if $F(x+(h(x))_{[\hat{S}]}) \leq F(x)$, with probability 1, for all $x \in \operatorname{dom} F$.

3.6.1 Deterministic Separable Overapproximation (DSO) of Partially Separable Functions

The following theorem will be useful in deriving ESO for uniform samplings (Section 3.7.1) and nonoverlapping uniform samplings (Section 3.7.2). It will also be useful in establishing monotonicity of some ESOs (Theorems 14 and 15).

Theorem 11 (DSO)**.** *Assume f is partially separable (i.e., it can be written in the form* (3.1)*). Letting $support(h) \overset{def}{=} \{i \in [n] \ : \ h^{(i)} \neq 0\}$, for all $x, h \in \mathbb{R}^N$ we have*

$$f(x+h) \leq f(x) + \langle \nabla f(x), h\rangle + \frac{\max_{J\in\mathcal{J}} |J \cap support(h)|}{2}\|h\|_L^2. \tag{3.35}$$

Proof. Let us fix x and define $\phi(h) \overset{\text{def}}{=} f(x+h) - f(x) - \langle \nabla f(x), h\rangle$. Fixing h, we need to show that $\phi(h) \leq \frac{\theta}{2}\|h\|_L^2$ for $\theta = \max_{J\in\mathcal{J}} \theta^J$, where $\theta^J \overset{\text{def}}{=} |J \cap support(h)|$. One can define functions ϕ^J in an analogous fashion from the constituent functions f_J, which satisfy

$$\phi(h) = \sum_{J\in\mathcal{J}} \phi^J(h), \tag{3.36}$$

$$\phi^J(0) = 0, \qquad J \in \mathcal{J}. \tag{3.37}$$

Note that (1.8) can be written as

$$\phi(U_i h^{(i)}) \leq \tfrac{L_i}{2}\|h^{(i)}\|_{(i)}^2, \qquad i = 1, 2, \dots, n. \tag{3.38}$$

Now, since ϕ^J depends on the intersection of J and the support of its argument only, we have

$$\phi(h) \overset{(3.36)}{=} \sum_{J\in\mathcal{J}} \phi^J(h) = \sum_{J\in\mathcal{J}} \phi^J\left(\sum_{i=1}^n U_i h^{(i)}\right) = \sum_{J\in\mathcal{J}} \phi^J\left(\sum_{i\in J\cap support(h)} U_i h^{(i)}\right). \tag{3.39}$$

The argument in the last expression can be written as a convex combination of $1+\theta^J$ vectors: the zero vector (with weight $\frac{\theta-\theta^J}{\theta}$) and the θ^J vectors $\{\theta U_i h^{(i)} : i \in J \cap support(h)\}$ (with weights $\frac{1}{\theta}$):

$$\sum_{i\in J\cap support(h)} U_i h^{(i)} = \left(\tfrac{\theta-\theta^J}{\theta}\times 0\right) + \left(\tfrac{1}{\theta}\times \sum_{i\in J\cap support(h)} \theta U_i h^{(i)}\right). \tag{3.40}$$

Finally, we plug (3.40) into (3.39) and use convexity and some simple algebra:

$$\begin{aligned}
\phi(h) &\leq \sum_{J\in\mathcal{J}}\left[\tfrac{\theta-\theta^J}{\theta}\phi^J(0) + \tfrac{1}{\theta}\sum_{i\in J\cap support(h)} \phi^J(\theta U_i h^{(i)})\right]\\
&\overset{(3.37)}{=} \tfrac{1}{\theta}\sum_{J\in\mathcal{J}}\sum_{i\in J\cap support(h)} \phi^J(\theta U_i h^{(i)})\\
&= \tfrac{1}{\theta}\sum_{J\in\mathcal{J}}\sum_{i=1}^n \phi^J(\theta U_i h^{(i)}) = \tfrac{1}{\theta}\sum_{i=1}^n\sum_{J\in\mathcal{J}} \phi^J(\theta U_i h^{(i)}) \overset{(3.36)}{=} \tfrac{1}{\theta}\sum_{i=1}^n \phi(\theta U_i h^{(i)})\\
&\overset{(3.38)}{\leq} \tfrac{1}{\theta}\sum_{i=1}^n \tfrac{L_i}{2}\|\theta h^{(i)}\|_{(i)}^2 = \tfrac{\theta}{2}\|h\|_L^2. \qquad \square
\end{aligned}$$

Besides the usefulness of the above result in deriving ESO inequalities, it is interesting on its own for the following reasons.

1. **Block Lipschitz continuity of ∇f.** The DSO inequality (3.35) is a generalization of (1.8) since (1.8) can be recovered from (3.35) by choosing h with $support(h) = \{i\}$ for $i \in [n]$.

2. **Global Lipschitz continuity of ∇f.** The DSO inequality also says that the gradient of f is Lipschitz continuous with Lipschitz constant ω with respect to the norm $\|\cdot\|_L$:

$$f(x+h) \leq f(x) + \langle \nabla f(x), h\rangle + \tfrac{\omega}{2}\|h\|_L^2. \tag{3.41}$$

Indeed, this follows from (3.35) via $\max_{J\in\mathcal{J}} |J \cap support(h)| \leq \max_{J\in\mathcal{J}} |J| = \omega$. For

$\omega = n$ this has been shown in [43]; our result for partially separable functions appears to be new.

3. **Tightness of the global Lipschitz constant.** The Lipschitz constant ω is "tight" in the following sense: there are functions for which ω cannot be replaced in (3.41) by any smaller number. We will show this on a simple example. Let $f(x) = \frac{1}{2}\|Ax\|^2$ with $A \in \mathbb{R}^{m\times n}$ (blocks are of size 1). Note that we can write $f(x+h) = f(x) + \langle \nabla f(x), h\rangle + \frac{1}{2}h^T A^T Ah$, and that $L = (L_1, \dots, L_n) = \operatorname{diag}(A^T A)$. Let $D = \operatorname{Diag}(L)$. We need to argue that there exists A for which $\sigma \overset{\text{def}}{=} \max_{h\neq 0} \frac{h^T A^T Ah}{\|h\|_L^2} = \omega$. Since we know that $\sigma \leq \omega$ (otherwise (3.41) would not hold), all we need to show is that there is A and h for which

$$h^T A^T Ah = \omega h^T Dh. \tag{3.42}$$

Since $f(x) = \sum_{j=1}^m (A_j^T x)^2$, where A_j is the j-th row of A, we assume that each row of A has at most ω nonzeros (i.e., f is partially separable of degree ω). Let us pick A with the following further properties: a) A is a 0-1 matrix, b) all rows of A have exactly ω ones, c) all columns of A have exactly the same number (k) of ones. Immediate consequences: $L_i = k$ for all i, $D = kI_n$ and $\omega m = kn$. If we let e_m be the $m\times 1$ vector of all ones and e_n be the $n \times 1$ vector of all ones, and set $h = k^{-1/2}e_n$, then

$$h^T A^T Ah = \tfrac{1}{k}e_n^T A^T Ae_n = \tfrac{1}{k}(\omega e_m)^T(\omega e_m) = \tfrac{\omega^2 m}{k} = \omega n = \omega\tfrac{1}{k}e_n^T kI_n e_n = \omega h^T Dh,$$

establishing (3.42). Using similar techniques one can easily prove the following more general result: Tightness also occurs for matrices A which in each row contain ω identical nonzero elements (but which can vary from row to row).

3.6.2 ESO for a Convex Combination of Samplings

Let $\hat{S}_1, \hat{S}_2, \dots, \hat{S}_m$ be a collection of samplings and let $q \in \mathbb{R}^m$ be a probability vector. By $\sum_j q_j \hat{S}_j$ we denote the sampling $\hat{S}$ given by

$$\mathbf{Prob}\left(\hat{S} = S\right) = \sum_{j=1}^m q_j \mathbf{Prob}(\hat{S}_j = S). \tag{3.43}$$

This procedure allows us to build new samplings from existing ones. A natural interpretation of $\hat{S}$ is that it arises from a two stage process as follows. Generating a set via $\hat{S}$ is equivalent to first choosing j with probability q_j, and then generating a set via $\hat{S}_j$.

Lemma 8. *Let $\hat{S}_1, \hat{S}_2, \dots, \hat{S}_m$ be arbitrary samplings, $q \in \mathbb{R}^m$ a probability vector and $\kappa : 2^{[n]} \to \mathbb{R}$ any function mapping subsets of $[n]$ to reals. If we let $\hat{S} = \sum_j q_j \hat{S}_j$, then*

(i) $\mathbf{E}[\kappa(\hat{S})] = \sum_{j=1}^m q_j \mathbf{E}[\kappa(\hat{S}_j)]$,

(ii) $\mathbf{E}[|\hat{S}|] = \sum_{j=1}^m q_j \mathbf{E}[|\hat{S}_j|]$,

(iii) $\mathbf{Prob}(i \in \hat{S}) = \sum_{j=1}^m q_j \mathbf{Prob}(i \in \hat{S}_j)$, *for any* $i = 1, 2, \dots, n$,

(iv) If $\hat{S}_1, \dots, \hat{S}_m$ *are uniform (resp. doubly uniform), so is* $\hat{S}$.

Proof. Statement (i) follows by writing $\mathbf{E}[\kappa(\hat{S})]$ as

$$\begin{aligned}\sum_S \mathbf{Prob}(\hat{S} = S)\kappa(S) &\overset{(3.43)}{=} \sum_S \sum_{j=1}^m q_j \mathbf{Prob}(\hat{S}_j = S)\kappa(S) \\ &= \sum_{j=1}^m q_j \sum_S \mathbf{Prob}(\hat{S}_j = S)\kappa(S) = \sum_{j=1}^m q_j \mathbf{E}[\kappa(\hat{S}_j)].\end{aligned}$$

Statement (ii) follows from (i) by choosing $\kappa(S) = |S|$, and (iii) follows from (i) by choosing κ as follows: $\kappa(S) = 1$ if $i \in S$ and $\kappa(S) = 0$ otherwise. Finally, if the samplings $\hat{S}_j$ are uniform, from (3.20) we know that $\mathbf{Prob}(i \in \hat{S}_j) = \mathbf{E}[|\hat{S}_j|]/n$ for all i and j. Plugging this into identity (iii) shows that $\mathbf{Prob}(i \in \hat{S})$ is independent of i, which shows that $\hat{S}$ is uniform. Now assume that $\hat{S}_j$ are doubly uniform. Fixing arbitrary $\tau \in \{0\} \cup [n]$, for every $S \subset [n]$ such that $|S| = \tau$ we have

$$\mathbf{Prob}(\hat{S} = S) \overset{(3.43)}{=} \sum_{j=1}^m q_j \mathbf{Prob}(\hat{S}_j = S) = \sum_{j=1}^m q_j \frac{\mathbf{Prob}(|\hat{S}_j| = \tau)}{\binom{n}{\tau}}.$$

As the last expression depends on S via $|S|$ only, $\hat{S}$ is doubly uniform. ☐

Remarks:

1. If we fix $S \subset [n]$ and define $k(S') = 1$ if $S' = S$ and $k(S') = 0$ otherwise, then statement (i) of Lemma 8 reduces to (3.43).
2. All samplings arise as a combination of *elementary* samplings, i.e., samplings whose all weight is on one set only. Indeed, let $\hat{S}$ be an arbitrary sampling. For all subsets S_j of $[n]$ define $\hat{S}_j$ by $\mathbf{Prob}(\hat{S}_j = S_j) = 1$ and let $q_j = \mathbf{Prob}(\hat{S} = S_j)$. Then clearly, $\hat{S} = \sum_j q_j \hat{S}_j$.
3. All doubly uniform samplings arise as convex combinations of nice samplings.

Often it is easier to establish ESO for a simple class of samplings (e.g., nice samplings) and then use it to obtain an ESO for a more complicated class (e.g., doubly uniform samplings as they arise as convex combinations of nice samplings). The following result is helpful in this regard.

Theorem 12 (Convex Combination of Uniform Samplings)**.** *Let $\hat{S}_1, \dots, \hat{S}_m$ be uniform samplings satisfying $(f, \hat{S}_j) \sim ESO(\beta_j, w_j)$ and let $q \in \mathbb{R}^m$ be a probability vector. If $\sum_j q_j \hat{S}_j$ is not nil, then*

$$\left(f, \sum_{j=1}^m q_j \hat{S}_j\right) \sim ESO\left(\frac{1}{\sum_{j=1}^m q_j \mathbf{E}[|\hat{S}_j|]}, \sum_{j=1}^m q_j \mathbf{E}[|\hat{S}_j|]\beta_j w_j\right).$$

Proof. First note that from part (iv) of Lemma 8 we know that $\hat{S} \overset{\text{def}}{=} \sum_j q_j \hat{S}_j$ is uniform and hence it makes sense to speak about ESO involving this sampling. Next, we can write

$$\begin{aligned}\mathbf{E}\left[f(x+h_{[\hat{S}]})\right] &= \sum_S \mathbf{Prob}(\hat{S}=S) f(x+h_{[S]}) \overset{(3.43)}{=} \sum_S \sum_j q_j \mathbf{Prob}(\hat{S}_j = S) f(x+h_{[S]})\\ &= \sum_j q_j \sum_S \mathbf{Prob}(\hat{S}_j = S) f(x+h_{[S]}) = \sum_j q_j \mathbf{E}\left[f(x+h_{[\hat{S}_j]})\right].\end{aligned}$$

It now remains to use (3.30) and part (ii) of Lemma 8:

$$\begin{aligned}\sum_{j=1}^m q_j \mathbf{E}\left[f(x+h_{[\hat{S}_j]})\right] &\overset{(3.30)}{\leq} \sum_{j=1}^m q_j \left(f(x) + \tfrac{\mathbf{E}[|\hat{S}_j|]}{n}\left(\langle \nabla f(x), h\rangle + \tfrac{\beta_j}{2}\|h\|_{w_j}^2\right)\right)\\ &= f(x) + \tfrac{\sum_j q_j \mathbf{E}[|\hat{S}_j|]}{n} \langle \nabla f(x), h\rangle + \tfrac{1}{2n}\sum_j q_j \mathbf{E}[|\hat{S}_j|]\beta_j \|h\|_{w_j}^2\\ &\overset{\text{(Lemma 8 (ii))}}{=} f(x) + \tfrac{\mathbf{E}[|\hat{S}|]}{n}\left(\langle \nabla f(x), h\rangle + \tfrac{\sum_j q_j \mathbf{E}[|\hat{S}_j|]\beta_j \|h\|_{w_j}^2}{2\sum_j q_j \mathbf{E}[|\hat{S}_j|]}\right)\\ &= f(x) + \tfrac{\mathbf{E}[|\hat{S}|]}{n}\left(\langle \nabla f(x), h\rangle + \tfrac{1}{2\sum_j q_j \mathbf{E}[|\hat{S}_j|]}\|h\|_w^2\right),\end{aligned}$$

where $w = \sum_j q_j \mathbf{E}[|\hat{S}_j|]\beta_j w_j$. In the third step we have also used the fact that $\mathbf{E}[|\hat{S}|] > 0$ which follows from the assumption that $\hat{S}$ is not nil. □

3.6.3 ESO for a Conic Combination of Functions

We now establish an ESO for a conic combination of functions each of which is already equipped with an ESO. It offers a complementary result to Theorem 12.

Theorem 13 (Conic Combination of Functions)**.** *If $(f_j, \hat{S}) \sim ESO(\beta_j, w_j)$ for $j = 1, \dots, m$, then for any $c_1, \dots, c_m \geq 0$ we have*

$$\left(\sum_{j=1}^m c_j f_j, \hat{S}\right) \sim ESO\left(1, \sum_{j=1}^m c_j \beta_j w_j\right).$$

Proof. Letting $f = \sum_j c_j f_j$, we get

$$\begin{aligned}
\mathbf{E}\left[\sum_j c_j f_j\left(x + h_{[\hat{S}]}\right)\right] &= \sum_j c_j\, \mathbf{E}\left[f_j\left(x + h_{[\hat{S}]}\right)\right] \\
&\leq \sum_j c_j \left(f_j(x) + \tfrac{\mathbf{E}[|\hat{S}|]}{n}\left(\langle \nabla f_j(x), h\rangle + \tfrac{\beta_j}{2}\|h\|^2_{w_j}\right)\right) \\
&= \sum_j c_j f_j(x) + \tfrac{\mathbf{E}[|\hat{S}|]}{n}\left(\sum_j c_j \langle \nabla f_j(x), h\rangle + \sum_j \tfrac{c_j\beta_j}{2}\|h\|^2_{w_j}\right) \\
&= f(x) + \tfrac{\mathbf{E}[|\hat{S}|]}{n}\left(\langle \nabla f(x), h\rangle + \tfrac{1}{2}\|h\|^2_{\sum_j c_j\beta_j w_j}\right). \qquad \square
\end{aligned}$$

3.7 Expected Separable Overapproximation (ESO) of Partially Separable Functions

Here we derive ESO inequalities for partially separable smooth functions f and (proper) uniform (Section 3.7.1), nonoverlapping uniform (Section 3.7.2), nice (Section 3.7.3) and doubly uniform (Section 3.7.4) samplings.

3.7.1 Uniform Samplings

Consider an arbitrary proper sampling $\hat{S}$ and let $\nu = (\nu_1, \dots, \nu_n)^T$ be defined by

$$\nu_i \overset{\text{def}}{=} \mathbf{E}\left[\min\{\omega, |\hat{S}|\} \mid i \in \hat{S}\right] = \tfrac{1}{p_i} \sum_{S: i\in S} \mathbf{Prob}(S) \min\{\omega, |S|\}, \qquad i \in [n].$$

Lemma 9. *If $\hat{S}$ is proper, then*

$$\mathbf{E}\left[f(x + h_{[\hat{S}]})\right] \leq f(x) + \langle \nabla f(x), h\rangle_p + \tfrac{1}{2}\|h\|^2_{p\odot\nu\odot L}. \tag{3.44}$$

Proof. Let us use Theorem 11 with h replaced by $h_{[\hat{S}]}$. Note that $\max_{J\in\mathcal{J}} |J \cap \mathit{support}(h_{[\hat{S}]})| \leq \max_{J\in\mathcal{J}} |J \cap \hat{S}| \leq \min\{\omega, |\hat{S}|\}$. Taking expectations of both sides of (3.35) we therefore get

$$\begin{aligned}
\mathbf{E}\left[f(x + h_{[\hat{S}]})\right] &\overset{(3.35)}{\leq} f(x) + \mathbf{E}\left[\langle \nabla f(x), h_{[\hat{S}]}\rangle\right] + \tfrac{1}{2}\,\mathbf{E}\left[\min\{\omega, |\hat{S}|\}\|h_{[\hat{S}]}\|^2_L\right] \\
&\overset{(3.17)}{=} f(x) + \langle \nabla f(x), h\rangle_p + \tfrac{1}{2}\,\mathbf{E}\left[\min\{\omega, |\hat{S}|\}\|h_{[\hat{S}]}\|^2_L\right].
\end{aligned} \tag{3.45}$$

It remains to bound the last term in the expression above. Letting $\theta_i = L_i\|h^{(i)}\|^2_{(i)}$, we have

$$\begin{aligned}
\mathbf{E}\left[\min\{\omega,|\hat{S}|\}\|h_{[\hat{S}]}\|_L^2\right] &= \mathbf{E}\left[\sum_{i\in\hat{S}}\min\{\omega,|\hat{S}|\}L_i\|h^{(i)}\|^2_{(i)}\right] = \sum_{S\subset[n]}\mathbf{Prob}(S)\sum_{i\in S}\min\{\omega,|S|\}\theta_i \\
&= \sum_{i=1}^n \theta_i \sum_{S:i\in S}\min\{\omega,|S|\}\mathbf{Prob}(S) = \sum_{i=1}^n \theta_i p_i\,\mathbf{E}\left[\min\{\omega,|\hat{S}|\}\mid i\in\hat{S}\right] = \sum_{i=1}^n \theta_i p_i \nu_i \\
&= \|h\|^2_{p\odot\nu\odot L}. \qquad (3.46)
\end{aligned}$$

□

The above lemma will now be used to establish ESO for arbitrary (proper) uniform samplings.

Theorem 14. *If $\hat{S}$ is proper and uniform, then*

$$(f,\hat{S}) \sim ESO(1,\nu\odot L). \qquad (3.47)$$

If, in addition, $\mathbf{Prob}(|\hat{S}|=\tau)=1$ *(we say that $\hat{S}$ is τ-uniform), then*

$$(f,\hat{S}) \sim ESO(\min\{\omega,\tau\},L). \qquad (3.48)$$

Moreover, ESO (3.48) *is monotonic.*

Proof. First, (3.47) follows from (3.44) since for a uniform sampling one has $p_i = \mathbf{E}[|\hat{S}|]/n$ for all i. If $\mathbf{Prob}(|\hat{S}|=\tau)=1$, we get $\nu_i = \min\{\omega,\tau\}$ for all i; (3.48) therefore follows from (3.47) and (3.33). Let us now establish monotonicity. Using the deterministic separable overapproximation (3.35) with $h = h_{[\hat{S}]}$,

$$\begin{aligned}
F(x+h_{[\hat{S}]}) &\leq f(x) + \langle\nabla f(x), h_{[\hat{S}]}\rangle + \max_{J\in\mathcal{J}}\tfrac{|J\cap\hat{S}|}{2}\|h_{[\hat{S}]}\|_L^2 + \Psi(x+h_{[\hat{S}]}) \\
&\leq f(x) + \langle\nabla f(x), h_{[\hat{S}]}\rangle + \tfrac{\beta}{2}\|h_{[\hat{S}]}\|_w^2 + \Psi(x+h_{[\hat{S}]}) \qquad (3.49) \\
&= f(x) + \sum_{i\in\hat{S}}\underbrace{\left(\langle\nabla f(x), U_i h^{(i)}\rangle + \tfrac{\beta w_i}{2}\|h^{(i)}\|^2_{(i)} + \Psi_i(x^{(i)}+h^{(i)})\right)}_{\overset{\text{def}}{=}\kappa_i(h^{(i)})} + \sum_{i\notin\hat{S}}\Psi_i(x^{(i)}).
\end{aligned}$$

$$(3.50)$$

Now let $h(x) = \arg\min_h H_{\beta,w}(x,h)$ and recall that

$$\begin{aligned}H_{\beta,w}(x,h) &= f(x) + \langle \nabla f(x), h\rangle + \tfrac{\beta}{2}\|h\|_w^2 + \Psi(x+h)\\ &= f(x) + \sum_{i=1}^n \left(\langle \nabla f(x), U_i h^{(i)}\rangle + \tfrac{\beta w_i}{2}\|h^{(i)}\|_{(i)}^2 + \Psi_i(x^{(i)}+h^{(i)})\right) = f(x) + \sum_{i=1}^n \kappa_i(h^{(i)}).\end{aligned}$$

So, by definition, $(h(x))^{(i)}$ minimizes $\kappa_i(t)$ and hence, $(h(x))_{[\hat{S}]}$ (recall (1.19)) minimizes the upper bound (3.50). In particular, $(h(x))_{[\hat{S}]}$ is better than a nil update, which immediately gives $F(x + (h(x))_{[\hat{S}]}) \leq f(x) + \sum_{i\in\hat{S}} \kappa_i(0) + \sum_{i\notin\hat{S}} \Psi_i(x^{(i)}) = F(x)$. □

Besides establishing an ESO result, we have just shown that, in the case of τ-uniform samplings with a conservative estimate for β, PCDM1 is monotonic, i.e., $F(x_{k+1}) \leq F(x_k)$. In particular, PCDM1 and PCDM2 coincide. We call the estimate $\beta = \min\{\omega, \tau\}$ "conservative" because it can be improved (made smaller) in special cases; e.g., for the τ-nice sampling. Indeed, Theorem 16 establishes an ESO for the τ-nice sampling with the same w ($w = L$), but with $\beta = 1 + \frac{(\omega-1)(\tau-1)}{n-1}$, which is better (and can be much better than) $\min\{\omega,\tau\}$. Other things equal, smaller β directly translates into better complexity. The price for the small β in the case of the τ-nice sampling is the loss of monotonicity. This is not a problem for strongly convex objective, but for merely convex objective this is an issue as the analysis techniques we developed are only applicable to the monotonic method PCDM2 (see Theorem 18).

3.7.2 Nonoverlapping Uniform Samplings

Let $\hat{S}$ be a (proper) nonoverlapping uniform sampling as defined in (3.10). If $i \in S^j$, for some $j \in \{1,2,\dots,l\}$, define

$$\gamma_i \overset{\text{def}}{=} \max_{J\in\mathcal{J}} |J \cap S^j|, \tag{3.51}$$

and let $\gamma = (\gamma_1,\dots,\gamma_n)^T$.

Note that, for example, if $\hat{S}$ is the serial uniform sampling, then $l = n$ and $S^j = \{j\}$ for $j = 1,2,\dots,l$, whence $\gamma_i = 1$ for all $i \in [n]$. For the fully parallel sampling we have $l = 1$ and $S^1 = \{1,2,\dots,n\}$, whence $\gamma_i = \omega$ for all $i \in [n]$.

Theorem 15. *If $\hat{S}$ is a nonoverlapping uniform sampling, then*

$$(f,\hat{S}) \sim ESO(1, \gamma \odot L). \tag{3.52}$$

Moreover, this ESO is monotonic.

Proof. By Theorem 11, used with h replaced by $h_{[S^j]}$ for $j = 1, 2, \dots, l$, we get

$$f(x + h_{[S^j]}) \leq f(x) + \langle \nabla f(x), h_{[S^j]} \rangle + \max_{J \in \mathcal{J}} \tfrac{|J \cap S^j|}{2} \|h_{[S^j]}\|_L^2. \tag{3.53}$$

Since $\hat{S} = S^j$ with probability $\frac{1}{l}$,

$$\begin{aligned}
\mathbf{E}\left[f(x + h_{[\hat{S}]})\right] &\overset{(3.53)}{\leq} \tfrac{1}{l} \sum_{j=1}^{l} \left(f(x) + \langle \nabla f(x), h_{[S^j]} \rangle + \max_{J \in \mathcal{J}} \tfrac{|J \cap S^j|}{2} \|h_{[S^j]}\|_L^2 \right) \\
&\overset{(3.51)}{=} f(x) + \tfrac{1}{l} \left(\langle \nabla f(x), h \rangle + \tfrac{1}{2} \sum_{j=1}^{l} \sum_{i \in S^j} \gamma_i L_i \|h^{(i)}\|_{(i)}^2 \right) \\
&= f(x) + \tfrac{1}{l} \left(\langle \nabla f(x), h \rangle + \tfrac{1}{2} \|h\|_{\gamma \odot L}^2 \right),
\end{aligned}$$

which establishes (3.52). It now only remains to establish monotonicity. Adding $\Psi(x + h_{[\hat{S}]})$ to (3.53) with S^j replaced by $\hat{S}$, we get $F(x+h_{[\hat{S}]}) \leq f(x) + \langle \nabla f(x), h_{[\hat{S}]} \rangle + \frac{\beta}{2} \|h_{[\hat{S}]}\|_w^2 + \Psi(x+h_{[\hat{S}]})$. From this point on the proof is identical to that in Theorem 14, following equation (3.49). □

3.7.3 Nice Samplings

In this section we establish an ESO for nice samplings.

Theorem 16. *If $\hat{S}$ is the τ-nice sampling and $\tau \neq 0$, then*

$$(f, \hat{S}) \sim ESO\left(1 + \frac{(\omega - 1)(\tau - 1)}{\max(1, n - 1)}, L\right). \tag{3.54}$$

Proof. Let us fix x and define ϕ and ϕ^J as in the proof of Theorem 11. Since

$$\mathbf{E}\left[\phi(h_{[\hat{S}]})\right] = \mathbf{E}\left[f(x + h_{[\hat{S}]}) - f(x) - \langle \nabla f(x), h_{[\hat{S}]} \rangle\right] \overset{(3.22)}{=} \mathbf{E}\left[f(x + h_{[\hat{S}]})\right] - f(x) - \tfrac{\tau}{n} \langle \nabla f(x), h \rangle,$$

it now only remains to show that

$$\mathbf{E}\left[\phi(h_{[\hat{S}]})\right] \leq \tfrac{\tau}{2n} \left(1 + \tfrac{(\omega-1)(\tau-1)}{\max(1,n-1)}\right) \|h\|_L^2. \tag{3.55}$$

Let us now adopt the convention that expectation conditional on an event which happens with probability 0 is equal to 0. Letting $\eta_J \overset{\text{def}}{=} |J \cap \hat{S}|$, and using this convention, we can write

$$\begin{aligned}
\mathbf{E}\left[\phi(h_{[\hat{S}]})\right] = \sum_{J \in \mathcal{J}} \mathbf{E}\left[\phi^J(h_{[\hat{S}]})\right] &= \sum_{k=0}^{n} \sum_{J \in \mathcal{J}} \mathbf{E}\left[\phi^J(h_{[\hat{S}]}) \mid \eta_J = k\right] \mathbf{Prob}(\eta_J = k) \\
&= \sum_{k=0}^{n} \mathbf{Prob}(\eta_J = k) \sum_{J \in \mathcal{J}} \mathbf{E}\left[\phi^J(h_{[\hat{S}]}) \mid \eta_J = k\right]. \qquad (3.56)
\end{aligned}$$

Note that the last identity follows if we assume, without loss of generality, that all sets J have the same cardinality ω (this can be achieved by introducing "dummy" dependencies). Indeed, in such a case $\mathbf{Prob}(\eta_J = k)$ does not depend on J. Now, for any $k \geq 1$ for which $\mathbf{Prob}(\eta_J = k) > 0$ (for some J and hence for all), using convexity of ϕ^J, we can now estimate

$$\begin{aligned}
\mathbf{E}\left[\phi^J(h_{[\hat{S}]}) \mid \eta_J = k\right] &= \mathbf{E}\left[\phi^J\left(\tfrac{1}{k}\sum_{i\in J\cap\hat{S}} kU_i h^{(i)}\right) \mid \eta_J = k\right] \\
&\leq \mathbf{E}\left[\tfrac{1}{k}\sum_{i\in J\cap\hat{S}} \phi^J\left(kU_i h^{(i)}\right) \mid \eta_J = k\right] \\
&\overset{(3.28)}{=} \tfrac{1}{\omega}\sum_{i\in J}\phi^J\left(kU_i h^{(i)}\right). \qquad (3.57)
\end{aligned}$$

If we now sum the inequalities (3.57) for all $J \in \mathcal{J}$, we get

$$\begin{aligned}
\sum_{J\in\mathcal{J}} \mathbf{E}\left[\phi^J(h_{[\hat{S}]}) \mid \eta_J = k\right] &\overset{(3.57)}{\leq} \tfrac{1}{\omega}\sum_{J\in\mathcal{J}}\sum_{i\in J}\phi^J\left(kU_i h^{(i)}\right) = \tfrac{1}{\omega}\sum_{J\in\mathcal{J}}\sum_{i=1}^{n}\phi^J\left(kU_i h^{(i)}\right) \\
&= \tfrac{1}{\omega}\sum_{i=1}^{n}\sum_{J\in\mathcal{J}}\phi^J\left(kU_i h^{(i)}\right) = \tfrac{1}{\omega}\sum_{i=1}^{n}\phi\left(kU_i h^{(i)}\right) \\
&\overset{(3.38)}{\leq} \tfrac{1}{\omega}\sum_{i=1}^{n}\tfrac{L_i}{2}\|kh^{(i)}\|_{(i)}^2 = \tfrac{k^2}{2\omega}\|h\|_L^2. \qquad (3.58)
\end{aligned}$$

Finally, (3.55) follows after plugging (3.58) into (3.56):

$$\mathbf{E}\left[\phi(h_{[\hat{S}]})\right] \leq \sum_k \mathbf{Prob}(\eta_J = k)\tfrac{k^2}{2\omega}\|h\|_L^2 = \tfrac{1}{2\omega}\|h\|_L^2\mathbf{E}[|J\cap\hat{S}|^2] \overset{(3.27)}{=} \tfrac{\tau}{2n}\left(1 + \tfrac{(\omega-1)(\tau-1)}{\max(1,n-1)}\right)\|h\|_L^2. \qquad \square$$

3.7.4 Doubly Uniform Samplings

We are now ready, using a bootstrapping argument, to formulate and prove a result covering all doubly uniform samplings.

Theorem 17. *If $\hat{S}$ is a (proper) doubly uniform sampling, then*

$$(f, \hat{S}) \sim ESO\left(1 + \frac{(\omega-1)\left(\frac{\mathbf{E}[|\hat{S}|^2]}{\mathbf{E}[|\hat{S}|]} - 1\right)}{\max(1, n-1)}, L\right). \qquad (3.59)$$

Proof. Letting $q_k = \mathbf{Prob}(|\hat{S}| = k)$ and $d = \max\{1, n-1\}$, we have

$$\begin{aligned}
\mathbf{E}\left[f(x+h_{[\hat{S}]})\right] &= \mathbf{E}\left[\mathbf{E}\left[f(x+h_{[\hat{S}]}) \mid |\hat{S}|\right]\right] = \sum_{k=0}^{n} q_k \, \mathbf{E}\left[f(x+h_{[\hat{S}]}) \mid |\hat{S}| = k\right] \\
&\overset{(3.54)}{\leq} \sum_{k=0}^{n} q_k \left[f(x) + \tfrac{k}{n}\left(\langle \nabla f(x), h\rangle + \tfrac{1}{2}\left(1 + \tfrac{(\omega-1)(k-1)}{N}\right)\|h\|_L^2\right)\right] \\
&= f(x) + \tfrac{1}{n}\sum_{k=0}^{n} q_k k \langle \nabla f(x), h\rangle + \tfrac{1}{2n}\sum_{k=1}^{n} q_k \left[k\left(1 - \tfrac{\omega-1}{N}\right) + k^2 \tfrac{\omega-1}{N}\right]\|h\|_L^2 \\
&= f(x) + \tfrac{\mathbf{E}[|\hat{S}|]}{n}\langle \nabla f(x), h\rangle + \tfrac{1}{2n}\left(\mathbf{E}[|\hat{S}|]\left(1 - \tfrac{\omega-1}{N}\right) + \mathbf{E}[|\hat{S}|^2]\tfrac{\omega-1}{N}\right)\|h\|_L^2.
\end{aligned}$$

This theorem could have alternatively been proved by writing $\hat{S}$ as a convex combination of nice samplings and applying Theorem 12. □

Note that Theorem 17 reduces to that of Theorem 16 in the special case of a nice sampling, and gives the same result as Theorem 15 in the case of the serial and fully parallel samplings.

3.8 Iteration Complexity

In this section we prove two iteration complexity theorems[7]. The first result (Theorem 18) is for non-strongly-convex F and covers PCDM2 with no restrictions and PCDM1 only in the case when a monotonic ESO is used. The second result (Theorem 19) is for strongly convex F and covers PCDM1 without any monotonicity restrictions.

Let us first establish two auxiliary results.

Lemma 10. *For all $x \in \operatorname{dom} F$, $H_{\beta,w}(x, h(x)) \leq \min_{y\in\mathbb{R}^N}\{F(y) + \frac{\beta - \mu_f(W)}{2}\|y-x\|_w^2\}$.*

Proof.

$$\begin{aligned}
H_{\beta,w}(x,h(x)) \overset{(3.4)}{=} \min_{y\in\mathbb{R}^N} H_{\beta,w}(x, y-x) &= \min_{y\in\mathbb{R}^N} f(x) + \langle \nabla f(x), y-x\rangle + \Psi(y) + \tfrac{\beta}{2}\|y-x\|_w^2 \\
&\overset{(1.10)}{\leq} \min_{y\in\mathbb{R}^N} f(y) - \tfrac{\mu_f(W)}{2}\|y-x\|_w^2 + \Psi(y) + \tfrac{\beta}{2}\|y-x\|_w^2.
\end{aligned}$$ □

Lemma 11. *(i) Let x^* be an optimal solution of* (1.1)*, $x \in \operatorname{dom} F$ and let $R = \|x - x^*\|_w$. Then*

$$H_{\beta,w}(x,h(x)) - F^* \leq \begin{cases} \left(1 - \frac{F(x)-F^*}{2\beta R^2}\right)(F(x) - F^*), & \text{if } F(x) - F^* \leq \beta R^2, \\ \frac{1}{2}\beta R^2 < \frac{1}{2}(F(x) - F^*), & \text{otherwise.} \end{cases} \tag{3.60}$$

---

[7] The development is similar to that in Chapter 2 for the *serial* block coordinate descent method, in the composite case. However, the results are vastly different.

(ii) If $\mu_f(W) + \mu_\Psi(W) > 0$ *and* $\beta \geq \mu_f(W)$*, then for all* $x \in \operatorname{dom} F$*,*

$$H_{\beta,w}(x, h(x)) - F^* \leq \frac{\beta - \mu_f(W)}{\beta + \mu_\Psi(W)}(F(x) - F^*). \tag{3.61}$$

Proof. Part (i): Since we do not assume strong convexity, we have $\mu_f(W) = 0$, and hence

$$\begin{aligned} H_{\beta,w}(x, h(x)) &\overset{\text{(Lemma 10)}}{\leq} \min_{y \in \mathbb{R}^N} \{F(y) + \tfrac{\beta}{2}\|y - x\|_w^2\} \\ &\leq \min_{\lambda \in [0,1]} \{F(\lambda x^* + (1-\lambda)x) + \tfrac{\beta\lambda^2}{2}\|x - x^*\|_w^2\} \\ &\leq \min_{\lambda \in [0,1]} \{F(x) - \lambda(F(x) - F^*) + \tfrac{\beta\lambda^2}{2}R^2\}. \end{aligned}$$

Minimizing the last expression in λ gives $\lambda^* = \min\left\{1, (F(x) - F^*)/(\beta R^2)\right\}$; the result follows. Part (ii): Letting $\mu_f = \mu_f(W)$, $\mu_\Psi = \mu_\Psi(W)$ and $\lambda^* = (\mu_f + \mu_\Psi)/(\beta + \mu_\Psi) \leq 1$, we have

$$\begin{aligned} H_{\beta,w}(x, h(x)) &\overset{\text{(Lemma 10)}}{\leq} \min_{y \in \mathbb{R}^N} \{F(y) + \tfrac{\beta - \mu_f}{2}\|y - x\|_w^2\} \\ &\leq \min_{\lambda \in [0,1]} \{F(\lambda x^* + (1-\lambda)x) + \tfrac{(\beta - \mu_f)\lambda^2}{2}\|x - x^*\|_w^2\} \\ &\overset{(1.12)+(1.11)}{\leq} \min_{\lambda \in [0,1]} \{\lambda F^* + (1-\lambda)F(x) - \tfrac{(\mu_f + \mu_\Psi)\lambda(1-\lambda) - (\beta - \mu_f)\lambda^2}{2}\|x - x^*\|_w^2\} \\ &\leq F(x) - \lambda^*(F(x) - F^*). \end{aligned}$$

The last inequality follows from the identity $(\mu_f + \mu_\Psi)(1 - \lambda^*) - (\beta - \mu_f)\lambda^* = 0$. $\square$

We could have formulated part (ii) of the above result using the weaker assumption $\mu_F(w) > 0$, leading to a slightly stronger result. However, we prefer the above treatment as it gives more insight.

3.8.1 Convex Case

Theorem 1 will be used to finish off the proof of the complexity result of this section. This theorem was recently extended in [69] so as to aid the analysis of a *serial* coordinate descent method with *inexact* updates, i.e., with $h(x)$ chosen as an *approximate* rather than exact minimizer of $H_{1,L}(x, \cdot)$ (see (3.4)). While in this chapter we deal with exact updates only, the results can be extended to the inexact case.

Theorem 18. *Assume that* $(f, \hat{S}) \sim ESO(\beta, w)$*, where* $\hat{S}$ *is a proper uniform sampling, and let* $\alpha = \frac{\mathbf{E}[|\hat{S}|]}{n}$*. Choose* $x_0 \in \operatorname{dom} F$ *satisfying*

$$\mathcal{R}_W(x_0, x^*) \overset{def}{=} \max_x\{\|x - x^*\|_w \; : \; F(x) \leq F(x_0)\} < +\infty, \tag{3.62}$$

where x^* *is an optimal point of* (1.1)*. Further, choose target confidence level* $0 < \rho < 1$*, target accuracy level* $\epsilon > 0$ *and iteration counter* K *in any of the following two ways:*

(i) $\epsilon < F(x_0) - F^*$ *and*

$$K \geq 2 + \frac{2\left(\frac{\beta}{\alpha}\right)\max\left\{\mathcal{R}_W^2(x_0,x^*), \frac{F(x_0)-F^*}{\beta}\right\}}{\epsilon}\left(1 - \frac{\epsilon}{F(x_0)-F^*} + \log\left(\frac{1}{\rho}\right)\right), \quad (3.63)$$

(ii) $\epsilon < \min\{\beta\mathcal{R}_W^2(x_0,x^*), F(x_0) - F^*\}$ *and*

$$K \geq \frac{2\left(\frac{\beta}{\alpha}\right)\mathcal{R}_W^2(x_0,x^*)}{\epsilon}\log\left(\frac{F(x_0)-F^*}{\epsilon\rho}\right). \quad (3.64)$$

If $\{x_k\}$*,* $k \geq 0$*, are the random iterates of PCDM (use PCDM1 if the ESO is monotonic, otherwise use PCDM2), then* $\mathbf{Prob}(F(x_K) - F^* \leq \epsilon) \geq 1 - \rho$*.*

Proof. Since either PCDM2 is used (which is monotonic) or otherwise the ESO is monotonic, we must have $F(x_k) \leq F(x_0)$ for all k. In particular, in view of (3.62) this implies that $\|x_k - x^*\|_w \leq \mathcal{R}_w(x_0,x^*)$. Letting $\xi_k = F(x_k) - F^*$, we have

$$\begin{aligned}
\mathbf{E}[\xi_{k+1} \mid x_k] &\overset{(3.31)}{\leq} (1-\alpha)\xi_k + \alpha(H_{\beta,w}(x_k,h(x_k)) - F^*) \\
&\overset{(3.60)}{\leq} (1-\alpha)\xi_k + \alpha\max\left\{1 - \frac{\xi_k}{2\beta\|x_k - x^*\|_w^2}, \frac{1}{2}\right\}\xi_k \\
&= \max\left\{1 - \frac{\alpha\xi_k}{2\beta\|x_k - x^*\|_w^2}, 1 - \frac{\alpha}{2}\right\}\xi_k \\
&\leq \max\left\{1 - \frac{\alpha\xi_k}{2\beta\mathcal{R}_w^2(x_0,x^*)}, 1 - \frac{\alpha}{2}\right\}\xi_k. \qquad (3.65)
\end{aligned}$$

Consider case (i) and let $c_1 = 2\frac{\beta}{\alpha}\max\{\mathcal{R}_w^2(x_0,x^*), \frac{\xi_0}{\beta}\}$. Continuing with (3.65), we then get

$$\mathbf{E}[\xi_{k+1} \mid x_k] \leq (1 - \tfrac{\xi_k}{c_1})\xi_k$$

for all $k \geq 0$. Since $\epsilon < \xi_0 < c_1$, it suffices to apply Theorem 1(i). Consider now case (ii) and let $c_2 = 2\frac{\beta}{\alpha}\frac{\mathcal{R}_w^2(x_0,x^*)}{\epsilon}$. Observe now that whenever $\xi_k \geq \epsilon$, from (3.65) we get $\mathbf{E}[\xi_{k+1} \mid x_k] \leq (1 - \frac{1}{c_2})\xi_k$. By assumption, $c_2 > 1$, and hence it remains to apply Theorem 1(ii). ☐

The important message of the above theorem is that the iteration complexity of our methods in the convex case is $O(\frac{\beta}{\alpha}\frac{1}{\epsilon})$. Note that for the serial method (PCDM1 used with $\hat{S}$ being the serial sampling) we have $\alpha = \frac{1}{n}$ and $\beta = 1$ (see Table 3.4), and hence $\frac{\beta}{\alpha} = n$. It will be interesting to study the *parallelization speedup factor* defined by

$$\text{parallelization speedup factor} = \frac{\frac{\beta}{\alpha} \text{ of the serial method}}{\frac{\beta}{\alpha} \text{ of a parallel method}} = \frac{n}{\frac{\beta}{\alpha} \text{ of a parallel method}}. \tag{3.66}$$

Table 3.5, computed from the data in Table 3.4, gives expressions for the parallelization speedup factors for PCDM based on a DU sampling (expressions for 4 special cases are given as well).

$\hat{S}$	Parallelization speedup factor
doubly uniform	$\frac{\mathbf{E}[\lvert\hat{S}\rvert]}{1+\frac{(\omega-1)\left((\mathbf{E}[\lvert\hat{S}\rvert^2]/\mathbf{E}[\lvert\hat{S}\rvert])-1\right)}{\max(1,n-1)}}$
(τ, p_b)-binomial	$\frac{\tau}{\frac{1}{p_b}+\frac{(\omega-1)(\tau-1)}{\max(1,n-1)}}$
τ-nice	$\frac{\tau}{1+\frac{(\omega-1)(\tau-1)}{\max(1,n-1)}}$
fully parallel	$\frac{n}{\omega}$
serial	1

Table 3.5: Convex F: Parallelization speedup factors for DU samplings. The factors below the line are special cases of the general expression. Maximum speedup is naturally obtained by the fully parallel sampling: $\frac{n}{\omega}$.

The speedup of the serial sampling (i.e., of the algorithm based on it) is 1 as we are comparing it to itself. On the other end of the spectrum is the fully parallel sampling with a speedup of $\frac{n}{\omega}$. If the degree of partial separability is small, then this factor will be high — especially so if n is huge, which is the domain we are interested in. This provides an affirmative answer to the research question stated in italics in the introduction.

Let us now look at the speedup factor in the case of a τ-nice sampling. Letting $r = \frac{\omega-1}{\max(1,n-1)} \in [0,1]$ (degree of partial separability normalized), the speedup factor can be written as

$$s(r) = \frac{\tau}{1 + r(\tau-1)}.$$

Note that as long as $r \leq \frac{k-1}{\tau-1} \approx \frac{k}{\tau}$, the speedup factor will be at least $\frac{\tau}{k}$. Also note that $\max\{1, \frac{\tau}{\omega}\} \leq s(r) \leq \min\{\tau, \frac{n}{\omega}\}$. Finally, if a speedup of at least s is desired, where $s \in [0, \frac{n}{\omega}]$, one needs to use at least $\frac{1-r}{1/s-r}$ processors. For illustration, in Figure 3.1 we plotted $s(r)$ for a few values of τ. Note that for small values of τ, the speedup is significant and can be as large as the number of processors (in the separable case). We wish to stress that in many applications ω will be a constant independent of n, which means that r will indeed be very small in the huge-scale optimization setting.

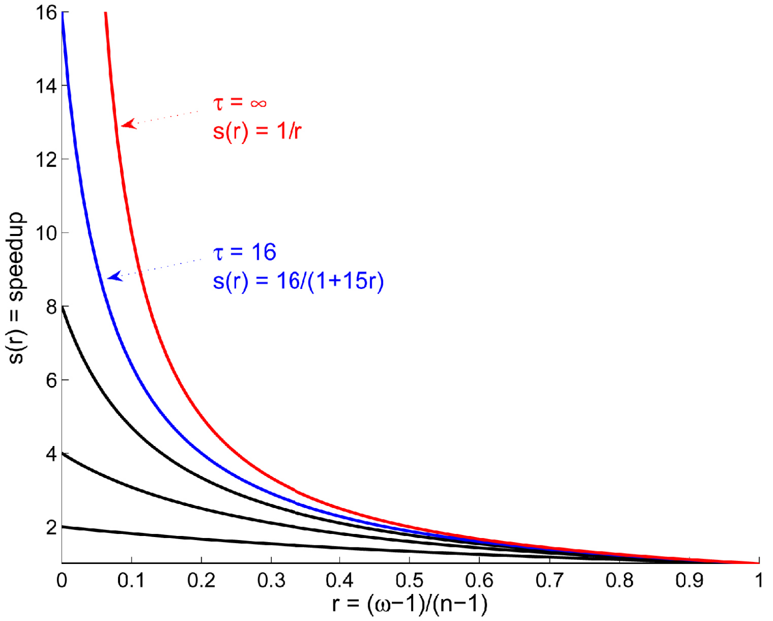


Figure 3.1: Parallelization speedup factor of PCDM1/PCDM2 used with τ-nice sampling as a function of the normalized/relative degree of partial separability r.

3.8.2 Strongly Convex Case

In this section we assume that F is strongly convex with respect to the norm $\|\cdot\|_w$ and show that $F(x_k)$ converges to F^* linearly, with high probability.

Theorem 19. *Assume F is strongly convex with $\mu_f(W)+\mu_\Psi(W) > 0$. Further, assume $(f,\hat{S}) \sim ESO(\beta,w)$, where $\hat{S}$ is a proper uniform sampling and let $\alpha = \frac{\mathbf{E}[|\hat{S}|]}{n}$. Choose initial point $x_0 \in \operatorname{dom} F$, target confidence level $0<\rho<1$, target accuracy level $0<\epsilon<F(x_0)-F^*$ and*

$$K \geq \frac{1}{\alpha}\frac{\beta+\mu_\Psi(W)}{\mu_f(W)+\mu_\Psi(W)}\log\left(\frac{F(x_0)-F^*}{\epsilon\rho}\right). \tag{3.67}$$

If $\{x_k\}$ are the random points generated by PCDM1 or PCDM2, then $\mathbf{Prob}(F(x_K)-F^* \leq \epsilon) \geq 1-\rho$.

Proof. Letting $\xi_k = F(x_k)-F^*$, we have

$$\mathbf{E}[\xi_{k+1} \mid x_k] \overset{(3.31)}{\leq} (1-\alpha)\xi_k+\alpha(H_{\beta,w}(x_k,h(x_k))-F^*) \overset{(3.61)}{\leq} \left(1-\alpha\frac{\mu_f(W)+\mu_\Psi(W)}{\beta+\mu_\Psi(W)}\right)\xi_k \overset{\text{def}}{=} (1-\gamma)\xi_k.$$

Note that $0 < \gamma \leq 1$ since $0 < \alpha \leq 1$ and $\beta \geq \mu_f(W)$ by (3.34). By taking expectation in x_k, we obtain $\mathbf{E}[\xi_k] \leq (1-\gamma)^k \xi_0$. Finally, it remains to use Markov inequality:

$$\mathbf{Prob}(\xi_K > \epsilon) \leq \frac{\mathbf{E}[\xi_K]}{\epsilon} \leq \frac{(1-\gamma)^K \xi_0}{\epsilon} \overset{(3.67)}{\leq} \rho. \qquad \square$$

Instead of doing a direct calculation, we could have finished the proof of Theorem 19 by applying Lemma 1(ii) to the inequality $\mathbf{E}[\xi_{k+1} \mid x_k] \leq (1-\gamma)\xi_k$. However, in order to be able to use Lemma 1, we would have to first establish monotonicity of the sequence $\{\xi_k\}$, $k \geq 0$. This is not necessary using the direct approach of Theorem 19. Hence, in the strongly convex case we can analyze PCDM1 and are not forced to resort to PCDM2. Consider now the following situations:

1. $\mu_f(W) = 0$. Then the leading term in (3.67) is $\frac{1+\beta/\mu_\Psi(W)}{\alpha}$.
2. $\mu_\Psi(W) = 0$. Then the leading term in (3.67) is $\frac{\beta/\mu_f(W)}{\alpha}$.
3. $\mu_\Psi(W)$ is "large enough". Then $\frac{\beta+\mu_\Psi(w)}{\mu_f(W)+\mu_\Psi(w)} \approx 1$ and the leading term in (3.67) is $\frac{1}{\alpha}$.

In a similar way as in the non-strongly convex case, define the parallelization speedup factor as the ratio of the leading term in (3.67) for the serial method (which has $\alpha = \frac{1}{n}$ and $\beta = 1$) and the leading term for a parallel method:

$$\text{parallelization speedup factor} = \frac{n\frac{1+\mu_\Psi(W)}{\mu_f(W)+\mu_\Psi(W)}}{\frac{1}{\alpha}\frac{\beta+\mu_\Psi(W)}{\mu_f(W)+\mu_\Psi(W)}} = \frac{n}{\frac{\beta+\mu_\Psi(W)}{\alpha(1+\mu_\Psi(W))}}. \tag{3.68}$$

First, note that the speedup factor is independent of μ_f. Further, note that as $\mu_\Psi(W) \to 0$, the speedup factor approaches the factor we obtained in the non-strongly convex case (see (3.66) and also Table 3.5). That is, for large values of $\mu_\Psi(W)$, the speedup factor is approximately equal $\alpha n = \mathbf{E}[|\hat{S}|]$, which is the average number of blocks updated in a single parallel iteration. Note that this quantity *does not* depend on the degree of partial separability of f.

3.9 Optimal Probabilities in the Smooth Case

In this section we consider smooth version of (1.1), i.e. with $\Psi \equiv 0$ and $f(x)$ is convex and differentiable. We propose a new algorithm, and call it 'NSync (Nonuniform SYNchronous Coordinate descent).

In 'NSync, we first assign a probability $p_S \geq 0$ to every subset S of $[n]$, with $\sum_S p_S = 1$, and pick stepsize parameters $w_i > 0$, $i = 1, 2, \ldots, n$. At every iteration, a random set $\hat{S}$ is

Algorithm 3.3 'NSync: Nonuniform SYNchronous Coordinate descent

```
**Input:** Initial point $x_0 \in \mathbb{R}^n$, subset probabilities $\{p_S\}$ and stepsize parameters $w_1, \dots, w_n > 0$
for $k = 0, 1, 2, \dots$ do
    Select a random set of coordinates $\hat{S} \subseteq \{1, \dots, n\}$ such that $\mathbf{Prob}(\hat{S} = S) = p_S$
    Update selected coordinates: $x_{k+1} = x_k - \sum_{i \in \hat{S}} \frac{1}{w_i} \nabla_i f(x_k) e_i$
end for
```

generated, independently from previous iterations, following the law $\mathbf{Prob}(\hat{S} = S) = p_S$, and then coordinates $i \in \hat{S}$ are updated in parallel by moving in the direction of the negative partial derivative with stepsize $1/w_i$. The updates are synchronized: no processor/thread is allowed to proceed before all updates are applied, generating the new iterate x_{k+1}. We specifically study samplings $\hat{S}$ which are *non-uniform* in the sense that $p_i \overset{\text{def}}{=} \mathbf{Prob}(i \in \hat{S}) = \sum_{S: i \in S} p_S$ is allowed to vary with i. By $\nabla_i f(x)$ we mean $\langle \nabla f(x), e_i \rangle$, where $e_i \in \mathbb{R}^n$ is the i-th unit coordinate vector. Let us remark that to the best of our knowledge, 'NSync is the first *nonuniform parallel* coordinate descent method.

3.9.1 Analysis

Our analysis of 'NSync is based on two assumptions. The first assumption generalizes the ESO concept introduced in Section 3.6 and later used in [66, 68, 17, 15, 55] to *nonuniform* samplings. The second assumption requires that f be strongly convex.

Assumption 1 (Nonuniform ESO: Expected Separable Overapproximation)**.** *Assume* $p = (p_1, \dots, p_n)^T > 0$ *and that for some positive vector* $w \in \mathbb{R}^n$ *and all* $x, h \in \mathbb{R}^n$*,*

$$\mathbf{E}[f(x + h_{[\hat{S}]})] \leq f(x) + \langle \nabla f(x), h \rangle_p + \tfrac{1}{2} \|h\|^2_{p \odot w}. \tag{3.69}$$

Inequalities of the type (3.69)*, in the uniform case (*$p_i = p_j$ *for all* i, j*), were studied in Chapter 3 and in [54, 66, 17, 55].*

Assumption 2 (Strong convexity)**.** *We assume that* f *is* γ*-strongly convex with respect to the norm* $\| \cdot \|_v$*, where* $v = (v_1, \dots, v_n)^T > 0$ *and* $\gamma > 0$*. That is, we require that for all* $x, h \in \mathbb{R}^n$*,*

$$f(x + h) \geq f(x) + \langle \nabla f(x), h \rangle + \tfrac{\gamma}{2} \|h\|^2_v. \tag{3.70}$$

We can now establish a bound on the number of iterations sufficient for 'NSync to approximately solve (1.1) with high probability.

Theorem 20. *Let Assumptions 1 and 2 be satisfied. Choose* $x_0 \in \mathbb{R}^n$, $0 < \epsilon < f(x_0) - f^*$ *and* $0 < \rho < 1$, *where* $f^* \overset{def}{=} \min_x f(x)$. *Let*

$$\Lambda \overset{def}{=} \max_i \tfrac{w_i}{p_i v_i}. \tag{3.71}$$

If $\{x_k\}$ *are the random iterates generated by 'NSync, then*

$$K \geq \tfrac{\Lambda}{\gamma} \log\left(\tfrac{f(x_0)-f^*}{\epsilon\rho}\right) \qquad \Rightarrow \qquad \mathbf{Prob}(f(x_K) - f^* \leq \epsilon) \geq 1 - \rho. \tag{3.72}$$

Moreover, we have the lower bound $\Lambda \geq (\sum_i \frac{w_i}{v_i})/\mathbf{E}[|\hat{S}|]$.

Proof. We first claim that f is μ-strongly convex with respect to the norm $\|\cdot\|_{w\odot p^{-1}}$, i.e.,

$$f(x+h) \geq f(x) + \langle \nabla f(x), h\rangle + \tfrac{\mu}{2}\|h\|^2_{w\odot p^{-1}}, \tag{3.73}$$

where $\mu \overset{\text{def}}{=} \gamma/\Lambda$. Indeed, this follows by comparing (3.70) and (3.73) in the light of (3.71). Let x^* be such that $f(x^*) = f^*$. Using (3.73) with $h = x^* - x$,

$$f^* - f(x) \overset{(3.73)}{\geq} \min_{h'\in\mathbb{R}^n} \langle \nabla f(x), h'\rangle + \tfrac{\mu}{2}\|h'\|^2_{w\odot p^{-1}} = -\tfrac{1}{2\mu}\|\nabla f(x)\|^2_{p\odot w^{-1}}. \tag{3.74}$$

Let $h_k \overset{\text{def}}{=} -(\mathrm{Diag}(w))^{-1}\nabla f(x_k)$. Then $x_{k+1} = x_k + (h_k)_{[\hat{S}]}$, and utilizing Assumption 1, we get

$$\mathbf{E}[f(x_{k+1}) \mid x_k] = \mathbf{E}[f(x_k + (h_k)_{[\hat{S}]})] \overset{(3.69)}{\leq} f(x_k) + \langle \nabla f(x_k), h_k\rangle_p + \tfrac{1}{2}\|h_k\|^2_{p\odot w} \tag{3.75}$$

$$= f(x_k) - \tfrac{1}{2}\|\nabla f(x_k)\|^2_{p\odot w^{-1}} \overset{(3.74)}{\leq} f(x_k) - \mu(f(x_k) - f^*). \tag{3.76}$$

Taking expectations in the last inequality and rearranging the terms, we obtain $\mathbf{E}[f(x_{k+1}) - f^*] \leq (1-\mu)\mathbf{E}[f(x_k) - f^*] \leq (1-\mu)^{k+1}(f(x_0) - f^*)$. Using this, Markov inequality, and the definition of K, we finally get $\mathbf{Prob}(f(x_K) - f^* \geq \epsilon) \leq \mathbf{E}[f(x_K) - f^*]/\epsilon \leq (1-\mu)^K(f(x_0) - f^*)/\epsilon \leq \rho$. Let us now establish the last claim. First, note that (see [54, Sec 3.2] for more results of this type),

$$\textstyle\sum_i p_i = \sum_i \sum_{S: i\in S} p_S = \sum_S \sum_{i: i\in S} p_S = \sum_S p_S |S| = \mathbf{E}[|\hat{S}|]. \tag{3.77}$$

Letting $\Delta \overset{\text{def}}{=} \{p' \in \mathbb{R}^n : p' \geq 0, \sum_i p'_i = \mathbf{E}[|\hat{S}|]\}$, we have

$$\Lambda \overset{(3.71)+(3.77)}{\geq} \min_{p'\in\Delta} \max_i \tfrac{w_i}{p'_i v_i} = \tfrac{1}{\mathbf{E}[|\hat{S}|]} \sum_i \tfrac{w_i}{v_i},$$

where the last equality follows since optimal p'_i is proportional to v_i/w_i. $\square$

Theorem 20 is generic in the sense that we do not say when Assumptions 1 and 2 are satisfied, how should one go about to choose the stepsizes w and probabilities $\{p_S\}$. In the next section we address these issues. On the other hand, this abstract setting allowed us to write a brief complexity proof.

Change of variables. Consider the change of variables $y = \mathrm{Diag}(d)x$, where $d > 0$. Defining $f^d(y) \overset{\text{def}}{=} f(x)$, we get $\nabla f^d(y) = (\mathrm{Diag}(d))^{-1}\nabla f(x)$. It can be seen that (3.69), (3.70) can equivalently be written in terms of f^d, with w replaced by $w^d \overset{\text{def}}{=} w \odot d^{-2}$ and v replaced by $v^d \overset{\text{def}}{=} v \odot d^{-2}$. By choosing $d_i = \sqrt{v_i}$, we obtain $v_i^d = 1$ for all i, recovering standard strong convexity.

3.9.2 Nonuniform Samplings and ESO

Consider now problem (1.1) with f of the form

$$f(x) \overset{\text{def}}{=} \phi(x) + \tfrac{\gamma}{2}\|x\|_v^2, \tag{3.78}$$

where $v > 0$. Note that Assumption 2 is satisfied. We further make the following two assumptions.

Assumption 3 (Smoothness). *ϕ has Lipschitz continuous gradient with respect to the coordinates, with positive constants $L_1, \ldots, L_n$. That is, $|\nabla_i \phi(x) - \nabla_i \phi(x + te_i)| \le L_i|t|$ for all $x \in \mathbb{R}^n$ and $t \in \mathbb{R}$.*

Assumption 4 (Partial separability). *$\phi(x)$ is partially separable of degree ω.*

Nonuniform sampling. Instead of considering the general case of arbitrary p_S assigned to all subsets of $[n]$, here we consider a special kind of sampling having two advantages:

1. sets can be generated easily,
2. it leads to larger stepsizes $1/w_i$ and hence improved convergence rate (see Theorem 20 and Theorem 21).

Fix $\tau \in [n]$ and $c \ge 1$ and let $S_1, \ldots, S_c$ be a collection of (possibly overlapping) subsets of $[n]$ such that $|S_j| \ge \tau$ for all j and $\cup_{j=1}^c S_j = [n]$. Moreover, let $q = (q_1, \ldots, q_c) > 0$ be a probability vector. Let $\hat{S}_j$ be τ-nice sampling from S_j; that is, $\hat{S}_j$ picks subsets of S_j having cardinality τ, uniformly at random. We assume these samplings are independent. Now, $\hat{S}$ is generated as follows. We first pick $j \in \{1, \dots, c\}$ with probability q_j, and then draw $\hat{S}_j$. Note that we do not need to compute the quantities p_S, $S \subseteq [n]$, to execute 'NSync. In fact, it is much easier to implement the sampling via the two-tier procedure explained above. Sampling $\hat{S}$ is a nonuniform variant of the τ-nice sampling studied in Section 3.7.3, which here arises as a special case for $c = 1$.[8]

Note that

$$p_i = \textstyle\sum_{j=1}^{c} q_j \frac{\tau}{|S_j|} \delta_{ij} > 0, \quad i \in [n], \tag{3.79}$$

where $\delta_{ij} = 1$ if $i \in S_j$, and 0 otherwise.

Theorem 21. *Let Assumptions 3 and 4 be satisfied, and let $\hat{S}$ be the sampling described above. Then Assumption 1 is satisfied with p given by* (3.79) *and any $w = (w_1, \dots, w_n)^T$ for which*

$$w_i \geq w_i^* \overset{def}{=} \tfrac{L_i + v_i}{p_i} \textstyle\sum_{j=1}^{c} q_j \frac{\tau}{|S_j|} \delta_{ij} \left(1 + \frac{(\tau-1)(\omega_j - 1)}{\max\{1, |S_j| - 1\}}\right), \qquad i \in [n], \tag{3.80}$$

where $\omega_j \overset{def}{=} \max_{J \in \mathcal{J}} |J \cap S_j| \leq \omega$.

Proof. Since f is separable of degree ω, so is ϕ (because $\frac{1}{2}\|x\|_v^2$ is separable). Now,

$$\mathbf{E}[f(x + h_{[\hat{S}]})] = \mathbf{E}[\mathbf{E}[f(x + h_{[\hat{S}_j]}) \mid j]] = \textstyle\sum_{j=1}^{c} q_j \mathbf{E}[f(x + h_{[\hat{S}_j]})] \tag{3.81}$$

$$\leq \textstyle\sum_{j=1}^{c} q_j \left\{ f(x) + \frac{\tau}{|S_j|} \left(\langle \nabla f(x), h_{[S_j]} \rangle + \frac{1}{2} \left(1 + \frac{(\tau-1)(\omega_j-1)}{\max\{1, |S_j|-1\}}\right) \|h_{[S_j]}\|_{L+v}^2 \right) \right\}, \tag{3.82}$$

where the last inequality follows from the ESO for τ-nice samplings established in Theorem 16. The claim now follows by comparing the above expression and (3.69). □

3.9.3 Optimal Probabilities

Observe that the formula (3.80) can be used to *design* a sampling (characterized by the sets S_j and probabilities q_j) that *maximizes* μ, which in view of Theorem 20 *optimizes the convergence rate* of the method.

Serial setting. Consider the serial version of 'NSync ($\mathbf{Prob}(|\hat{S}| = 1) = 1$). We can model this via $c = n$, with $S_i = \{i\}$ and $p_i = q_i$ for all $i \in [n]$. In this case, using (3.79) and (3.80), we get $w_i = w_i^* = L_i + v_i$. Minimizing Λ in (3.71) over the probability vector p gives the *optimal probabilities* (we refer to this as the *optimal serial* method) and *optimal complexity*

$$p_i^* = \tfrac{(L_i + v_i)/v_i}{\sum_j (L_j + v_j)/v_j}, \quad i \in [n], \qquad \Lambda_{OS} = \textstyle\sum_i \frac{L_i + v_i}{v_i} = n + \sum_i \frac{L_i}{v_i}, \tag{3.83}$$

---

[8] Note, that in general non-uniform sampling one can sample sets with different cardinality, whereas in our special nonuniform sampling, every sampled set S has cardinality τ.

respectively. Note that the *uniform sampling*, $p_i = 1/n$ for all i, leads to $\Lambda_{US} \overset{\text{def}}{=} n + n \max_j L_j/v_j$ (we call this the *uniform serial* method), which can be much larger than Λ_{OS}. Moreover, under the change of variables $y = \mathrm{Diag}(d)x$, the gradient of $f^d(y) \overset{\text{def}}{=} f(\mathrm{Diag}(d^{-1})y)$ has coordinate Lipschitz constants $L_i^d = L_i/d_i^2$, while the weights in (3.78) change to $v_i^d = v_i/d_i^2$. Hence, the condition numbers L_i/v_i can not be improved via such a change of variables.

Optimal serial method can be faster than the fully parallel method. To model the fully parallel setting (i.e., the variant of 'NSync updating *all* coordinates at every iteration), we can set $c = 1$ and $\tau = n$, which yields $\Lambda_{FP} = \omega + \omega \max_j L_j/v_j$. Since $\omega \leq n$, it is clear that $\Lambda_{US} \geq \Lambda_{FP}$. However, for large enough ω it will be the case that $\Lambda_{FP} \geq \Lambda_{OS}$, implying, surprisingly, that the optimal serial method can be faster than the fully parallel method.

Parallel setting. Fix τ and sets S_j, $j = 1, 2, \dots, c$, and define $\theta \overset{\text{def}}{=} \max_j \left(1 + \frac{(\tau-1)(\omega_j-1)}{\max\{1,|S_j|-1\}}\right)$. Consider running 'NSync with stepsizes $w_i = \theta(L_i + v_i)$ (note that $w_i \geq w_i^*$, so we are fine). From (3.71), (3.79) and (3.80) we see that the complexity of 'NSync is determined by

$$\Lambda = \max_i \tfrac{w_i}{p_i v_i} = \tfrac{\theta}{\tau} \max_i \left(1 + \tfrac{L_i}{v_i}\right) \left(\textstyle\sum_{j=1}^c q_j \tfrac{\delta_{ij}}{|S_j|}\right)^{-1}.$$

The probability vector q minimizing this quantity can be computed by solving a linear program with $c+1$ variables $(q_1, \dots, q_c, \alpha)$, $2n$ linear inequality constraints and a single linear equality constraint:

$$\max_{\alpha,q} \left\{\alpha \text{ subject to } \alpha \leq (b^i)^T q \text{ for all } i,\ q \geq 0,\ \textstyle\sum_j q_j = 1\right\},$$

where $b^i \in \mathbb{R}^c$, $i \in [n]$, are given by $b_j^i = \frac{v_i}{(L_i+v_i)} \frac{\delta_{ij}}{|S_j|}$.

3.10 Numerical Experiments

In Section 3.10.1 we present preliminary but very encouraging results showing that PCDM1 run on a system with 24 cores can solve huge-scale partially-separable LASSO problems with a billion variables in 2 hours, compared with 41 hours on a single core. In Section 3.10.2 we demonstrate that our analysis is in some sense tight. In particular, we show that the speedup predicted by the theory can be matched almost exactly by actual wall time speedup for a particular problem. Finally, in Section 3.10.5 we show a benefit of optimal probabilities.

3.10.1 A LASSO Problem with 1 Billion Variables

In this experiment we solve a single randomly generated huge-scale LASSO instance, i.e., (1.1) with

$$f(x) = \tfrac{1}{2}\|Ax - b\|_2^2, \qquad \Psi(x) = \|x\|_1,$$

where $A = [a_1, \dots, a_n]$ has 2×10^9 rows and $N = n = 10^9$ columns. We generated the problem using a modified primal-dual generator [53] enabling us to choose the optimal solution x^* (and hence, indirectly, F^*) and thus to control its cardinality $\|x^*\|_0$, as well as the sparsity level of A. In particular, we made the following choices: $\|x^*\|_0 = 10^5$, each column of A has exactly 20 nonzeros and the maximum cardinality of a row of A is $\omega = 35$ (the degree of partial separability of f). The histogram of cardinalities is displayed in Figure 3.2.

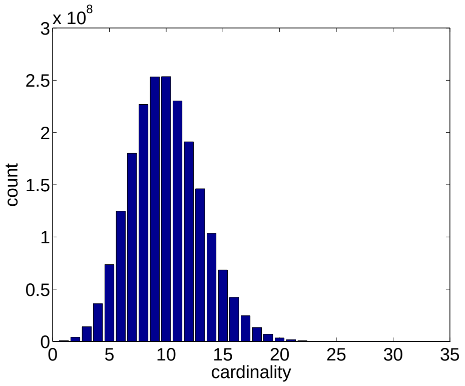


Figure 3.2: Histogram of the cardinalities of the rows of A.

We solved the problem using PCDM1 with τ-nice sampling $\hat{S}$, $\beta = 1 + \frac{(\omega-1)(\tau-1)}{n-1}$ and $w = L = (\|a_1\|_2^2, \cdots, \|a_n\|_2^2)$, for $\tau = 1, 2, 4, 8, 16, 24$, on a single large-memory computer utilizing τ of its 24 cores. The problem description took around 350GB of memory space. In fact, in our implementation we departed from the just described setup in two ways. First, we implemented an *asynchronous*[9] version of the method; i.e., one in which cores do not wait for others to update the current iterate within an iteration before reading x_{k+1} and proceeding to another update step. Instead, each core reads the current iterate whenever it is ready with the previous update step and applies the new update as soon as it is computed. Second, as mentioned in Section 3.5, the τ-independent sampling is for $\tau \ll n$ a very good approximation of the τ-nice sampling. We therefore allowed each processor to pick a block uniformly at random, independently from the other processors.

---

[9] We have used shared memory parallelization using OpenMP. The asynchronicity is assumed in such a sense that no thread synchronization is performed at any point. We have used atomic operations to avoid race conditions.

	$F(x_k) - F^*$						Elapsed Time					
$\frac{\tau k}{n}$	$\tau = 1$	$\tau = 2$	$\tau = 4$	$\tau = 8$	$\tau = 16$	$\tau = 24$	$\tau = 1$	$\tau = 2$	$\tau = 4$	$\tau = 8$	$\tau = 16$	$\tau = 24$
0	6.27e+22	6.27e+22	6.27e+22	6.27e+22	6.27e+22	6.27e+22	0.00	0.00	0.00	0.00	0.00	0.00
1	2.24e+22	2.24e+22	2.24e+22	2.24e+22	2.24e+22	2.24e+22	0.89	0.43	0.22	0.11	0.06	0.05
2	2.24e+22	2.24e+22	2.24e+22	3.64e+19	2.24e+22	8.13e+18	1.97	1.06	0.52	0.27	0.14	0.10
3	1.15e+20	2.72e+19	8.37e+19	1.94e+19	1.37e+20	5.74e+18	3.20	1.68	0.82	0.43	0.21	0.16
4	5.25e+19	1.45e+19	2.22e+19	1.42e+18	8.19e+19	5.06e+18	4.28	2.28	1.13	0.58	0.29	0.22
5	1.59e+19	2.26e+18	1.13e+19	1.05e+17	3.37e+19	3.14e+18	5.37	2.91	1.44	0.73	0.37	0.28
6	1.97e+18	4.33e+16	1.11e+19	1.17e+16	1.33e+19	3.06e+18	6.64	3.53	1.75	0.89	0.45	0.34
7	2.40e+16	2.94e+16	7.81e+18	3.18e+15	8.39e+17	3.05e+18	7.87	4.15	2.06	1.04	0.53	0.39
8	5.13e+15	8.18e+15	6.06e+18	2.19e+14	5.81e+16	9.22e+15	9.15	4.78	2.37	1.20	0.61	0.45
9	8.90e+14	7.87e+15	2.09e+16	2.08e+13	2.24e+16	5.63e+15	10.43	5.39	2.67	1.35	0.69	0.51
10	5.81e+14	6.52e+14	7.75e+15	3.42e+12	2.89e+15	2.20e+13	11.73	6.02	2.98	1.51	0.77	0.57
11	5.13e+14	1.97e+13	2.55e+15	1.54e+12	2.55e+15	7.30e+12	12.81	6.64	3.29	1.66	0.84	0.63
12	5.04e+14	1.32e+13	1.84e+13	2.18e+11	2.12e+14	1.44e+12	14.08	7.26	3.60	1.83	0.92	0.68
13	2.18e+12	7.06e+11	6.31e+12	1.33e+10	1.98e+14	6.37e+11	**15.35**	**7.88**	**3.91**	**1.99**	**1.00**	0.74
14	7.77e+11	7.74e+10	3.10e+12	3.43e+09	1.89e+12	1.20e+10	16.65	8.50	4.21	2.14	1.08	0.80
15	1.80e+10	6.23e+10	1.63e+11	1.60e+09	5.29e+11	4.34e+09	17.94	9.12	4.52	2.30	1.16	0.86
16	1.38e+09	2.27e+09	7.86e+09	1.15e+09	1.46e+11	1.38e+09	19.23	9.74	4.83	2.45	1.24	0.91
17	3.63e+08	3.99e+08	3.07e+09	6.47e+08	2.92e+09	7.06e+08	20.49	10.36	5.14	2.61	1.32	0.97
18	2.10e+08	1.39e+08	2.76e+08	1.88e+08	1.17e+09	5.93e+08	21.76	10.98	5.44	2.76	1.39	1.03
19	3.81e+07	1.92e+07	7.47e+07	1.55e+06	6.51e+08	5.38e+08	23.06	11.60	5.75	2.91	1.47	1.09
20	1.27e+07	1.59e+07	2.93e+07	6.78e+05	5.49e+07	8.44e+06	24.34	12.22	6.06	3.07	1.55	1.15
21	4.69e+05	2.65e+05	8.87e+05	1.26e+05	3.84e+07	6.32e+06	25.42	12.84	6.36	3.22	1.63	1.21
22	1.47e+05	1.16e+05	1.83e+05	2.62e+04	3.09e+06	1.41e+05	26.64	13.46	6.67	3.38	1.71	1.26
23	5.98e+04	7.24e+03	7.94e+04	1.95e+04	5.19e+05	6.09e+04	27.92	14.08	6.98	3.53	1.79	1.32
24	3.34e+04	3.26e+03	5.61e+04	1.75e+04	3.03e+04	5.52e+04	29.21	14.70	7.28	3.68	1.86	1.38
25	3.19e+04	2.54e+03	2.17e+03	5.00e+03	6.43e+03	4.94e+04	30.43	15.32	7.58	3.84	1.94	1.44
26	3.49e+02	9.62e+01	1.57e+03	4.11e+01	3.68e+03	4.91e+04	**31.71**	**15.94**	**7.89**	**3.99**	**2.02**	1.49
27	1.92e+02	8.38e+01	6.23e+01	5.70e+00	7.77e+02	4.90e+04	33.00	16.56	8.20	4.14	2.10	1.55
28	1.07e+02	2.37e+01	2.38e+01	2.14e+00	6.69e+02	4.89e+04	34.23	17.18	8.49	4.30	2.17	1.61
29	6.18e+00	1.35e+00	1.52e+01	2.35e-01	3.64e+01	4.89e+04	35.31	17.80	8.79	4.45	2.25	1.67
30	4.31e+00	3.93e-01	6.25e-01	4.03e-02	2.74e+00	3.15e+01	36.60	18.43	9.09	4.60	2.33	1.73
31	6.17e-01	3.19e-01	1.24e-01	3.50e-02	6.20e-01	9.29e+00	37.90	19.05	9.39	4.75	2.41	1.78
32	1.83e-02	3.06e-01	3.25e-02	2.41e-03	2.34e-01	3.10e-01	39.17	19.67	9.69	4.91	2.48	1.84
33	3.80e-03	1.75e-03	1.55e-02	1.63e-03	1.57e-02	2.06e-02	40.39	20.27	9.99	5.06	2.56	1.90
34	7.28e-14	7.28e-14	1.52e-02	7.46e-14	1.20e-02	1.58e-02	41.47	20.89	10.28	5.21	2.64	1.96
35	-	-	1.24e-02	-	1.23e-03	8.70e-14	-	-	10.58	-	2.72	2.02
36	-	-	2.70e-03	-	3.99e-04	-	-	-	10.88	-	2.80	-
37	-	-	7.28e-14	-	7.46e-14	-	-	-	11.19	-	2.87	-

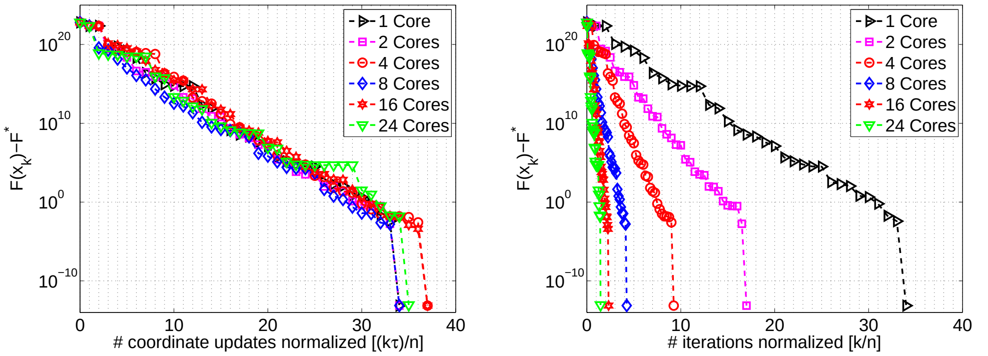
Table 3.6: A LASSO problem with 10^9 variables solved by PCDM1 with $\tau = 1, 2, 4, 8, 16$ and 24.

Choice of the first column of Table 3.6. In Table 3.6 we show the development of the gap $F(x_k) - F^*$ as well as the elapsed time. The choice and meaning of the first column of the table, $\frac{\tau k}{n}$, needs some commentary. Note that exactly τk coordinate updates are performed after k iterations. Hence, the first column denotes the total number of coordinate updates normalized by the number of coordinates n. As an example, let $\tau_1 = 1$ and $\tau_2 = 24$. Then if the serial method is run for $k_1 = 24$ iterations and the parallel one for $k_2 = 1$ iteration, both methods would have updated the same number ($\tau_1 k_1 = \tau_2 k_2 = 24$) of coordinates; that is, they would "be" in the same row of Table 3.6. In summary, each row of the table represents, in the sense described above, the "same amount of work done" for each choice of τ.

Progress to solving the problem. One can conjecture that the above meaning of the phrase "same amount of work done" would perhaps be roughly equivalent to a different one: "same progress to solving the problem". Indeed, it turns out, as can be seen from the table and also from Figure 3.3(a), that in each row for all algorithms the value of $F(x_k) - F^*$ is roughly of the same order of magnitude. This is not a trivial finding since, with increasing τ, older information is used to update the coordinates, and hence one would expect that convergence would be slower. It does seem to be slower—the gap $F(x_k) - F^*$ is generally higher if more processors are used—but the slowdown is limited. Looking at Table 3.6 and/or Figure 3.3(a), we see that for all choices of τ, PCDM1 managed to push the gap below 10^{-13} after $34n$ to $37n$ coordinate updates.

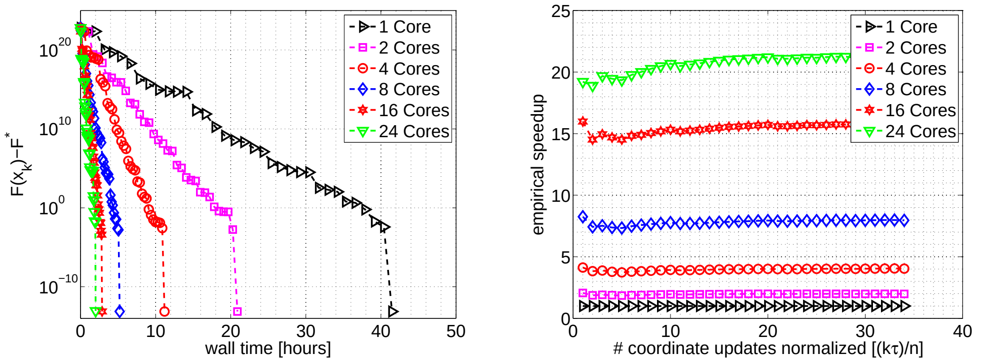
The progress to solving the problem during the final 1 billion coordinate updates (i.e., when moving from the last-but-one to the last nonempty line in each of the columns of Table 3.6 showing $F(x_k) - F^*$) is remarkable. The method managed to push the optimality gap by 9-12 degrees of magnitude. We do not have an explanation for this phenomenon; we do not give local convergence estimates in this chapter. It is certainly the case though that once the method managed to find the nonzero places of x^*, fast local convergence comes in.

Parallelization speedup. Since a parallel method utilizing τ cores manages to do the same number of coordinate updates as the serial one τ times faster, a direct consequence of the above observation is that doubling the number of cores corresponds to roughly halving the number of iterations (see Figure 3.3(b). This is due to the fact that $\omega \ll n$ and $\tau \ll n$. It turns out that the number of iterations is an excellent predictor of wall time; this can be seen by comparing Figures 3.3(b) and 3.3(c). Finally, it follows from the above, and can be seen in Figure 3.3(d), that the speedup of PCDM1 utilizing τ cores is roughly equal to τ. Note that this is caused by the fact that the problem is, relative to its dimension, partially separable to a very high degree.



(a) For each τ, PCDM1 needs roughly the same number of coordinate updates to solve the problem.

(b) Doubling the number of cores corresponds to roughly halving the number of iterations.



(c) Doubling the number of cores corresponds to roughly halving the wall time.

(d) Parallelization speedup is essentially equal to the number of cores.

Figure 3.3: Four computational insights into the workings of PCDM1.

3.10.2 Theory Versus Reality

In our second experiment we demonstrate numerically that our parallelization speedup estimates are in some sense tight. For this purpose it is not necessary to reach for complicated problems and high dimensions; we hence minimize the function $\frac{1}{2}\|Ax-b\|_2^2$ with $A \in \mathbb{R}^{3000\times 1000}$. Matrix A was generated so that its every row contains exactly ω non-zero values all of which are equal (recall the construction in point 3 at the end of Section 3.6.1).

We generated 4 matrices with $\omega = 5, 10, 50$ and 100 and measured the number of iterations needed for PCDM1 used with τ-nice sampling to get within $\epsilon = 10^{-6}$ of the optimal value. The experiment was done for a range of values of τ (between 1 core and 1000 cores).

The solid lines in Figure 3.4 present the *theoretical speedup factor* for the τ-nice sampling, as presented in Table 3.5. The markers in each case correspond to *empirical speedup factor* defined as

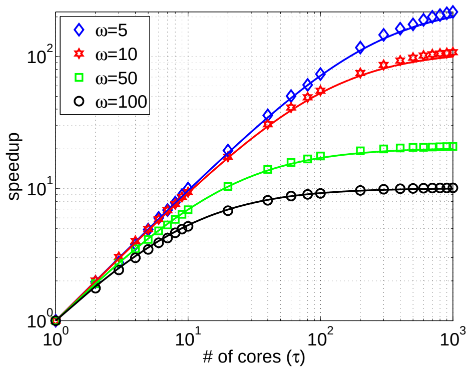


Figure 3.4: Theoretical speedup factor predicts the actual speedup almost exactly for a carefully constructed problem.

$$\frac{\#\text{ of iterations till }\epsilon\text{-solution is found by PCDM1 used with serial sampling}}{\#\text{ of iterations till }\epsilon\text{-solution is found by PCDM1 used with }\tau\text{-nice sampling}}.$$

As can be seen in Figure 3.4, the match between theoretical prediction and reality is remarkable! A partial explanation of this phenomenon lies in the fact that we have carefully designed the problem so as to ensure that the degree of partial separability is equal to the Lipschitz constant σ of ∇f (i.e., that it is not a gross overestimation of it; see Section 3.6.1). This fact is useful since it is possible to prove complexity results with ω replaced by σ. However, this answer is far from satisfying, and a deeper understanding of the phenomenon remains an open problem.

3.10.3 Training Linear SVMs with Bad Data for PCDM

In this experiment we test PCDM on the problem of training a linear Support Vector Machine (SVM) based on n labeled training examples: $(y_i, A_i) \in \{+1, -1\} \times \mathbb{R}^N$, $i = 1, 2, \dots, n$. In particular, we consider the primal problem of minimizing L2-regularized average hinge-loss,

$$\min_{w\in\mathbb{R}^N} \left\{ g(w) \stackrel{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{n}[1 - y_i\langle w, a_i\rangle]_+ + \frac{\lambda}{2}\|w\|_2^2 \right\},$$

and the dual problem of maximizing a concave quadratic subject to zero-one box constraints,

$$\max_{x\in\mathbb{R}^n,\ 0\le x^{(i)}\le 1} \left\{ -f(x) \stackrel{\text{def}}{=} -\frac{1}{2\lambda n^2}x^T Z x + \frac{1}{n}\sum_{i=1}^{n} x^{(i)} \right\},$$

where $Z \in \mathbb{R}^{n\times n}$ with $Z_{ii} = y_i y_j \langle A_i, A_j\rangle$. It is a standard practice to apply *serial* coordinate descent to the dual. Here we apply *parallel* coordinate descent (PCDM; with τ-nice sampling of coordinates) to the dual; i.e., minimize the convex function f subject to box constraints. In this setting all blocks are of size $N_i = 1$. The dual can be written in the form (1.1), i.e.,

$$\min_{x\in\mathbb{R}^n} \{F(x) = f(x) + \Psi(x)\},$$

where $\Psi(x) = 0$ whenever $x^{(i)} \in [0, 1]$ for all $i = 1, 2, \dots, n$, and $\Psi(x) = +\infty$ otherwise.

We consider the `rcv1.binary` dataset[10]. The training data has $n = 677,399$ examples, $d = 47,236$ features, $49,556,258$ nonzero elements and requires cca 1GB of RAM for storage. Hence, this is a small-scale problem. The degree of partial separability of f is $\omega = 291,516$ (i.e., the maximum number of examples sharing a given feature). This is a very large number relative to n, and hence our theory would predict rather bad behavior for PCDM. We use PCDM1 with τ-nice sampling (approximating it by τ-independent sampling for added efficiency) with β following Theorem 16: $\beta = 1 + \frac{(\tau-1)(\omega-1)}{n-1}$.

The results of our experiments are summarized in Figure 3.5. Each column corresponds to a different level of regularization: $\lambda \in \{1, 10^{-3}, 10^{-5}\}$. The rows show the 1) duality gap, 2) dual suboptimality, 3) train error and 4) test error; each for 1,4 and 16 processors ($\tau = 1, 4, 16$). Observe that the plots in the first two rows are nearly identical; which means that the method is able to solve the primal problem at about the same speed as it can solve the dual problem.

Observe also that in all cases, duality gap of around 0.01 is sufficient for training as training error (classification performance of the SVM on the train data) does not decrease further after this point. Also observe the effect of λ on training accuracy: accuracy increases from about 92% for $\lambda = 1$, through 95.3% for $\lambda = 10^{-3}$ to above 97.8% with $\lambda = 10^{-5}$. In our case, choosing smaller λ does not lead to overfitting; the test error on test dataset (# features =677,399, # examples = 20,242) increases as λ decreases, quickly reaching about 95% (after 2 seconds of training) for $\lambda = 0.001$ and for the smallest λ going beyond 97%.

Note that PCDM with $\tau = 16$ is about 2.5× faster than PCDM with $\tau = 1$. This is much less than linear speedup, but is fully in line with our theoretical predictions. Indeed, for $\tau = 16$ we get $\beta = 7.46$. Consulting Table 3.5, we see that the theory says that with $\tau = 16$ processors we should expect the parallelization speedup to be $PSF = \tau/\beta = 2.15$.

---

[10] `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#rcv1.binary`

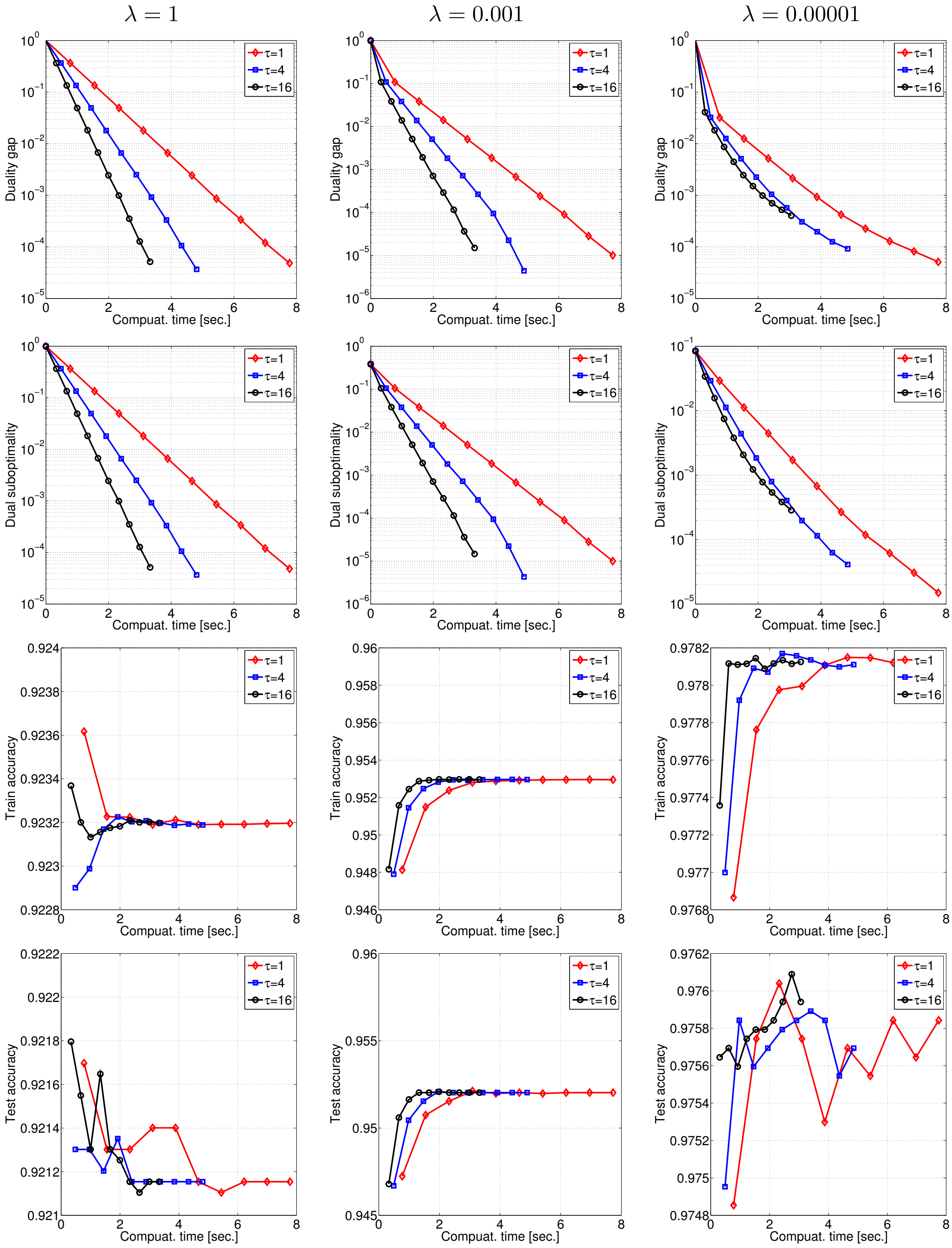


Figure 3.5: The performance of PCDM on the rcv1 dataset (this dataset is *not* good for the method).

3.10.4 $L2$-regularized Logistic Regression with Good Data for PCDM

In our last experiment we solve a problem of the form (1.1) with f being a sum of logistic losses and Ψ being an L2 regularizer,

$$\min_{x\in\mathbb{R}^n}\left\{\sum_{j=1}^{d}\log(1+e^{-y_jA_j^Tx})+\lambda\|x\|_2^2\right\},$$

where $(y_j, A_j) \in \{+1,-1\}\times\mathbb{R}^n$, $j = 1,2,\dots,d$, are labeled examples.

We have used the the `KDDB` dataset from the same source as the `rcv1.binary` dataset considered in the previous experiment. The data contains $n = 29,890,095$ features and is divided into two parts: a training set with $d = 19,264,097$ examples (and $566,345,888$ nonzeros; cca 8.5 GB) and a testing with $d = 748,401$ examples (and $21,965,075$ nonzeros; cca 0.32 GB).

This training dataset is good for PCDM as each example depends on at most 75 features. That is, $\omega = 75$, which is much smaller than n. As before, we will use PCDM1 with τ-nice sampling (approximated by τ-independent sampling) for $\tau = 1,2,4,8$ and set $\lambda = 1$.

Figure 3.6 depicts the evolution of the regularized loss $F(x_k)$ throughout the run of the 4 versions of PCDM (starting with x_0 for which $F(x_0) = 13,352,855$). Each marker corresponds to approximately $n/3$ coordinate updates (n coordinate updates will be referred to as an "epoch"). Observe that as more processors are used, it takes less time to achieve any given level of loss; nearly in exact proportion to the increase in the number of processors.

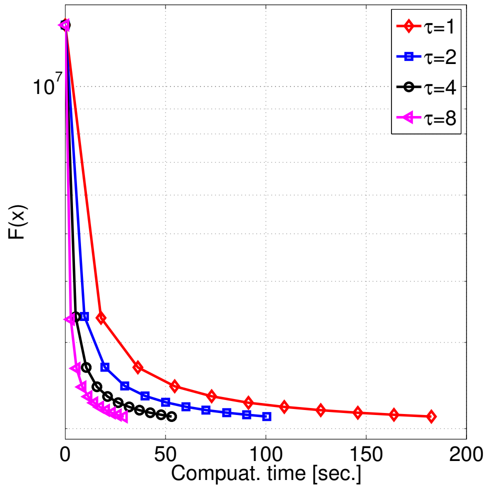


Figure 3.6: PCDM accelerates well with more processors on a dataset with small ω.

Table 3.7 offers an alternative view of the same experiment. In the first 4 columns ($F(x_0)/F(x_k)$) we can see that no matter how many processors are used, the methods produce similar loss values after working through the same number of coordinates. However, since the method uti-

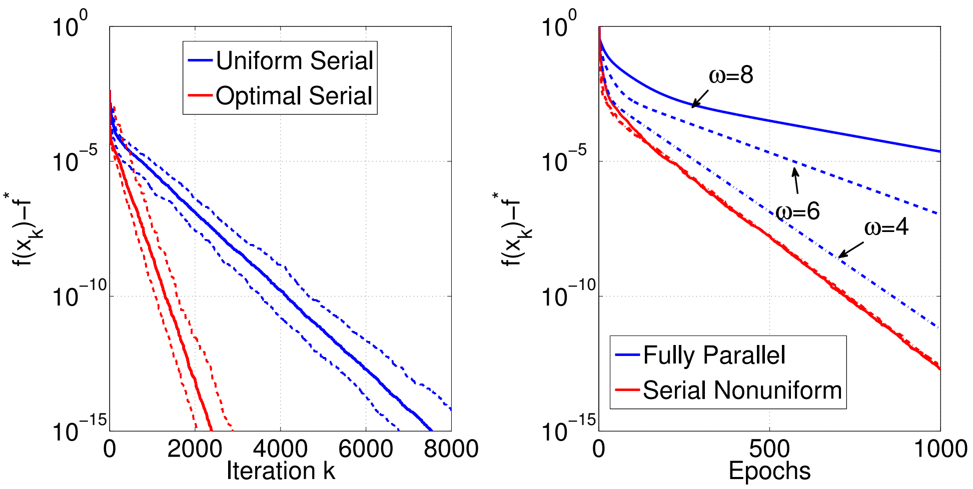


Figure 3.7: Comparison of uniform vs. optimal serial (left) and optimal serial vs. fully parallel (right).

lizing $\tau = 8$ processors updates 8 coordinates in parallel, it does the job approximately 8 times faster. Indeed, we can see this speedup in the table.

Let us remark that the training and testing accuracy stopped increasing after having trained the classifier for 1 epoch; they were 86.07% and 88.77%, respectively. This is in agreement with the common wisdom in machine learning that training beyond a single pass through the data rarely improves testing accuracy (as it may lead to overfitting). This is also the reason behind the success of light-touch methods, such as coordinate descent and stochastic gradient descent, in machine learning applications.

	$F(x_0)/F(x_k)$				time			
Epoch	$\tau = 1$	$\tau = 2$	$\tau = 4$	$\tau = 8$	$\tau = 1$	$\tau = 2$	$\tau = 4$	$\tau = 8$
1	3.96490	3.93909	3.94578	3.99407	17.83	9.57	5.20	2.78
2	5.73498	5.72452	5.74053	5.74427	73.00	39.77	21.11	11.54
3	6.12115	6.11850	6.12106	6.12488	127.35	70.13	37.03	20.29

Table 3.7: PCDM accelerates linearly in τ on a good dataset.

3.10.5 Optimal Probabilities

We now conduct 2 preliminary small scale experiments to illustrate the theory (Theorem 20); the results are depicted below. All experiments are with problems of the form (3.78) with $\phi = \frac{1}{2}\|Ax - b\|^2$.

In Figure 3.7 **left plot** we chose $A \in \mathbb{R}^{2\times 30}$, $\gamma = 1$, $v_1 = 0.05$, $v_i = 1$ for $i \neq 1$ and $L_i = 1$ for all i. We compare the US method ($p_i = 1/n$, blue) with the OS method (p_i given by (3.83), red). The dashed lines show 95% confidence intervals (we run the methods 100 times, the line in the middle is the average behavior). While OS can be faster, it is sensitive to over/under-estimation of the constants L_i, v_i. In the **right plot** we show that a nonuniform serial (NS) method can be faster than the fully parallel (FP) variant (we have chosen $m = 8$, $n = 10$ and 3 values of ω). On the horizontal axis we display the number of epochs, where 1 epoch corresponds to updating n coordinates (for FP this is a single iteration, whereas for NS it corresponds to n iterations).

4

Distributed Coordinate Descent Method

> There is a computer disease that anybody who works with computers knows about. It's a very serious disease and it interferes completely with the work. The trouble with computers is that you 'play' with them!
>
> — Richard P. Feynman
> (1918-1988)

It is clear that in order to utilize modern shared-memory parallel computers, more coordinates should be updated at each iteration. One way to approach this is via partitioning the coordinates into blocks, and operating on a single randomly chosen block at a time, utilizing parallel linear algebra libraries. This approach was pioneered by [43] for smooth losses, and was extended to regularized problems in [53]. Another popular approach involves working with a random subset of coordinates [5]. These approaches can be combined, and theory was developed for methods that update a random subset of blocks of coordinates at a time [54, 17]. Further recent works on parallel coordinate descent include [52, 33, 15, 68, 62, 38, 3].

However, none of these methods are directly scalable to problems of sizes so large that a single computer is unable to store the data describing the instance, or is unable to do so efficiently (e.g., in memory). In a big data scenario of this type, it is imperative to split the data across several nodes (computers) of a cluster, and design efficient methods for this memory-distributed setting.

Hydra. In this work we design and analyze the first distributed coordinate descent method: *Hydra: HYbriD cooRdinAte descent.* The method is "hybrid" in the sense that it uses parallelism at two levels: i) across a number of nodes in a cluster and ii) utilizing the parallel processing power of individual nodes[1].

Assume we have c nodes (computers) available, each with parallel processing power. In Hydra, we initially partition the coordinates $\{1, 2, \dots, N\}$ into c sets, $\mathcal{P}_1, \dots, \mathcal{P}_c$, and assign each set to a single computer. For simplicity, we assume that the partition is balanced: $|\mathcal{P}_k| = |\mathcal{P}_l|$ for all k, l. Each computer *owns* the coordinates belonging to its partition for the duration of the iterative process. Also, these coordinates are stored locally. The data matrix describing the problem is partitioned in such a way that all data describing features belonging to $\mathcal{P}_l$ is stored at computer l. Now, at each iteration, each computer, independently from the others, chooses a random subset of τ coordinates from those they own, and computes and applies updates to these coordinates. Hence, once all computers are done, $c\tau$ coordinates will have been updated. The resulting vector, stored as c vectors of size $s = N/c$ each, in a distributed way, is the new iterate. This process is repeated until convergence. It is important that the computations are done locally on each node, with minimum communication overhead. We comment on this and further details in the text.

The main insight. We show that the parallelization potential of Hydra, that is, its ability to accelerate as τ is increased, depends on two data-dependent quantities: i) the *spectral norm of the data (σ)* and ii) a *partition-induced norm of the data (σ').* The first quantity completely describes the behavior of the method in the $c = 1$ case. If σ is small, then utilization of more processors (i.e., increasing τ) leads to nearly linear speedup. If σ is large, speedup may be negligible, or there may be no speedup whatsoever. Hence, the size of σ suggests whether it is worth to use more processors or not. The second quantity, σ', characterizes the effect of the initial partition on the algorithm, and as such is relevant in the $c > 1$ case. Partitions with small σ' are preferable. We show that, surprisingly, that as long as $\tau \geq 2$, the effect of a bad partitioning is that it most doubles the number of iterations of Hydra. Hence, data partitioning can be used to optimize for different aspects of the method, such as reducing communication complexity, if needed.

For all of these quantities we derive easily computable and interpretable estimates (ω for σ and ω' for σ'), which may be used by practitioners to gauge, a-priori, whether their problem of interest is likely to be a good fit for Hydra or not. We show that for strongly convex losses, Hydra outputs an ϵ-accurate solution with probability at least $1 - \rho$ after $\frac{N\beta}{c\tau\mu} \log(\frac{1}{\epsilon\rho})$ iterations (we ignore some small details here), where a single iteration corresponds to changing of τ coordinates by each of the c nodes; β is a stepsize parameter and μ is a strong convexity constant.

---

[1] We like to think of each node of the cluster as one of the many heads of the mythological Hydra.

Outline. In Section 4.1 we describe the structure of the optimization problem we consider in this chapter and state assumptions. We then proceed to Section 4.1, in which we describe the method. In Section 4.2 we prove bounds on the number of iterations sufficient for Hydra to find an approximate solution with arbitrarily high probability. A discussion of various aspects of our results, as well as a comparison with existing work, can be found in Section 4.3. Implementation details of our distributed communication protocol are laid out in Section 4.4. Finally, we comment on our computational experiments with a big data (3TB matrix) L1 regularized least-squares instance in Section 4.5.

4.1 The Algorithm

We study the problem (1.1), with a modified assumption on f defined below.

Function f. We assume that there exists a positive definite matrix $\mathbf{M} \in \mathbb{R}^{N\times N}$ such that for all $x, h \in \mathbb{R}^N$,

$$f(x+h) \leq f(x) + (\nabla f(x))^T h + \tfrac{1}{2} h^T \mathbf{M} h, \tag{4.1}$$

and write $\mathbf{M} = \mathbf{A}^T\mathbf{A}$, where $\mathbf{A}$ is some m-by-N matrix and $m > N$.

Example. These assumptions are natural satisfied in many popular problems. A typical loss function has the form

$$f(x) = \sum_{j=1}^{n} \mathcal{L}(x, \mathbf{A}_j, y_j), \tag{4.2}$$

where $\mathbf{A} \in \mathbb{R}^{m\times N}$ is a matrix encoding m examples with N features, $\mathbf{A}_j$ denotes j-th row of $\mathbf{A}$, $\mathcal{L}$ is some loss function acting on a single example and $y \in \mathbb{R}^m$ is a vector of labels. For instance, in the case of the three losses ℓ in Table 3.1, assumption (4.1) holds with $\mathbf{M} = \mathbf{A}^T\mathbf{A}$ for SL and HL, and $\mathbf{M} = \frac{1}{4}\mathbf{A}^T\mathbf{A}$ for LL [5].

We consider a setup with c computers (nodes) and first partition the N coordinates (features) into c sets $\mathcal{P}_1, \dots, \mathcal{P}_c$ of equal cardinality, $s \overset{\text{def}}{=} N/c$, and assign set $\mathcal{P}_l$ to node l. Hydra is described in Algorithm 4.1. Hydra's convergence rate depends on the partition; we comment on this later in Sections 4.2 and 4.3. Here we simply assume that we work with a fixed partition. We now comment on the steps.

Step 4. Here we are just establishing a way of labeling the iterates. Starting with x_k, all c computers modify $c\tau$ entries of x_k in total, in a distributed way, and the result is called x_{k+1}. No computer is allowed to proceed before all computers are done with computing their updates. The resulting vector, x_{k+1}, is the new iterate. Note that, due to this, our method is inherently

Algorithm 4.1 Hydra (HYbriD cooRdinAte descent)

```
 1: Input: $x_0 \in \mathbb{R}^N$, $\{\mathcal{P}_1, \dots, \mathcal{P}_c\}$, $\beta > 0$, $\tau$
 2: set $k \leftarrow 0$
 3: for $k = 0, 1, 2, \dots$ do
 4:     $x_{k+1} \leftarrow x_k$
 5:     for computer $l \in \{1, \dots, c\}$ in parallel do
 6:         pick a random set of coordinates $\hat{S}_l \subseteq \mathcal{P}_l$ , $|\hat{S}_l| = \tau$
 7:         for $i \in \hat{S}_l$ in parallel do
 8:             compute update: $h_k^{(i)} \leftarrow \arg\min_t \nabla f_i(x_k)t + \frac{M_{ii}\beta}{2}t^2 + \Psi_i(x_k^{(i)} + t)$
 9:             apply the update: $x_{k+1}^{(i)} \leftarrow x_{k+1}^{(i)} + h_k^{(i)}$
10:         end for
11:     end for
12: end for
```

synchronous. In practice, a carefully designed asynchronous implementation will be faster, and our experiments in Section 4.5 are done with such an implementation.

Steps 5–6. At every iteration, each of the c computers picks a random subset of τ features from those that it owns, uniformly at random, independently of the choice of the other computers. Let $\hat{S}_l$ denote the set picked by node l . More formally, we require that i) $\hat{S}_l \subseteq \mathcal{P}_l$, ii) $\mathbf{Prob}(|\hat{S}_l| = \tau) = 1$, where $1 \leq \tau \leq s$, and that iii) all subsets of $\mathcal{P}_l$ of cardinality τ are chosen equally likely. In summary, at every iteration of the method, features belonging to the random set $\hat{S} \overset{\text{def}}{=} \cup_{l=1}^{c} \hat{S}_l$ are updated. Note that $\hat{S}$ has size $c\tau$, but that, as a sampling from the set $\{1, 2, \dots, N\}$, it does not choose all cardinality $c\tau$ subsets of $\{1, 2, \dots, N\}$ with equal probability. Hence, the analysis of parallel coordinate descent methods of [54] does not apply. We will say that $\hat{S}$ is a *τ-distributed sampling* with respect to the partition $\{\mathcal{P}_1, \dots, \mathcal{P}_c\}$.

Step 7. Once computer l has chosen its set of τ coordinates to work on in Step 5, it will *in parallel* compute (Step 8) and apply (Step 9) updates to them.

Step 8. This is a critical step where updates to coordinates $i \in \hat{S}_l$ are computed. Notice that the formula is very simple as it involves one dimensional optimization.

Closed-form formulas. Often, $h_k^{(i)}$ can be computed in closed form. For $\Psi_i(t) = \lambda_i |t|$ (weighted L1 regularizer), h_k^i is the point in the interval $[\frac{-\lambda_i - \nabla f_i(x_k)}{\mathbf{M}_{ii}\beta}, \frac{\lambda_i - \nabla f_i(x_k)}{\mathbf{M}_{ii}\beta}]$ which is closest to $-x_k^{(i)}$. If $\Psi_i(t) = \frac{\lambda_i}{2}t^2$ (weighted L2 regularizer), then $h_k^{(i)} = -\frac{\nabla f_i(x_k) + \lambda_i x_k^{(i)}}{\lambda_i + \mathbf{M}_{ii}\beta}$.

Choice of β. The choice of the step-size parameter β is of paramount significance for the performance of the algorithm, as argued for different but related algorithms by [54, 66, 17]. We will discuss this issue at length in Sections 4.2 and 4.3.

Implementation issues: Note that computer l needs to know the partial derivatives of f at x_k for coordinates $i \in \hat{S}_l \subseteq \mathcal{P}_l$. However, x_k, as well as the data describing f, is distributed among the c computers. One thus needs to devise a fast and communication efficient way of computing these derivatives. This issue will be dealt with in Section 4.4.

Step 9. Here all the τ updates computed in Step 8 are applied to the iterate. Note that the updates are *local*: computer l only updates coordinates it owns, which are stored locally. Hence, this step is communication-free.[2]

4.2 Iteration Complexity

Notation: For any $\mathbf{G} \in \mathbb{R}^{N\times N}$, let $D^{\mathbf{G}} = \mathrm{Diag}(\mathbf{G})$. That is, $D^{\mathbf{G}}_{ii} = \mathbf{G}_{ii}$ for all i and $D^{\mathbf{G}}_{ij} = 0$ for $i \neq j$. Further, let $B^{\mathbf{G}} \in \mathbb{R}^{N\times N}$ be the block diagonal of $\mathbf{G}$ associated with the partition $\{\mathcal{P}_1, \dots, \mathcal{P}_c\}$. That is, $B^{\mathbf{G}}_{ij} = \mathbf{G}_{ij}$ whenever $i, j \in \mathcal{P}_l$ for some l, and $B^{\mathbf{G}}_{ij} = 0$ otherwise.

4.2.1 Four Important Quantities: $\sigma', \omega', \sigma, \omega$

Here we define two quantities, σ' and σ, which, as we shall see, play an important role in the computation of the stepsize parameter β of Algorithm 4.1, and through it, in understanding its rate of convergence and potential for speedup by parallelization and distribution. As we shall see, these quantities might not be easily computable. We therefore also provide each with an easily computable and interpretable upper bound, ω' for σ' and ω for σ.

Let

$$\mathbf{Q} \overset{\text{def}}{=} (D^{\mathbf{M}})^{-1/2}\mathbf{M}(D^{\mathbf{M}})^{-1/2}, \tag{4.3}$$

and notice that, by construction, $\mathbf{Q}$ has ones on the diagonal. Since M is positive definite, $\mathbf{Q}$ is as well. For each $l \in \{1, \dots, c\}$, let $\mathbf{A}_l \in \mathbb{R}^{m\times s}$ be the column submatrix of $\mathbf{A}$ corresponding to coordinates $i \in \mathcal{P}_l$. The diagonal blocks of $B^{\mathbf{Q}}$ are the matrices $\mathbf{Q}^{ll}$, $l = 1, 2, \dots, c$, where

$$\mathbf{Q}^{kl} \overset{\text{def}}{=} (D^{\mathbf{A}_k^T\mathbf{A}_k})^{-1/2}\mathbf{A}_k^T\mathbf{A}_l(D^{\mathbf{A}_l^T\mathbf{A}_l})^{-1/2} \in \mathbb{R}^{s\times s} \tag{4.4}$$

for each $k, l \in \{1, 2, \dots, c\}$. We now define

$$\sigma' \overset{\text{def}}{=} \max\{x^T\mathbf{Q}x \ : \ x \in \mathbb{R}^N, \ x^T B^{\mathbf{Q}} x \leq 1\}, \tag{4.5}$$

$$\sigma \overset{\text{def}}{=} \max\{x^T\mathbf{Q}x \ : \ x \in \mathbb{R}^N, \ x^T x \leq 1\}. \tag{4.6}$$

A useful consequence of (4.5) is the inequality

$$x^T(\mathbf{Q} - B^{\mathbf{Q}})x \leq (\sigma' - 1)x^T B^{\mathbf{Q}} x. \tag{4.7}$$

---

[2] Note, that some communication will be still required in order to process to next iteration to allows computation of ∇f_i, see Section 4.4.

Sparsity. Let a_{rl} be the r-th row of $\mathbf{A}_l$, and define

$$\omega' \overset{\text{def}}{=} \max_{1\le r\le m} \left\{\omega'(r) \overset{\text{def}}{=} |\{l \,:\, l \in \{1,\dots,c\},\ a_{rl} \neq 0\}|\right\},$$

where $\omega'(r)$ is the number of matrices $\mathbf{A}_l$ with a nonzero in row r. Likewise, define

$$\omega \overset{\text{def}}{=} \max_{1\le r\le m} \left\{\omega(r) \overset{\text{def}}{=} |\{j \,:\, j \in \{1,\dots,N\},\ \mathbf{A}_{rj} \neq 0\}|\right\},$$

where $\omega(r)$ is the number of nonzeros in the r-th row of $\mathbf{A}$.

Lemma 12. *The following relations hold:*

$$\max\{1, \tfrac{\sigma}{s}\} \le \sigma' \le \omega' \le c, \qquad 1 \le \sigma \le \omega \le N. \tag{4.8}$$

Proof.

1. The inequality $\omega' \le c$ is obviously true. By considering x with zeroes in all coordinates except those that belong to $\mathcal{P}_k$ (where k is an arbitrary but fixed index), we see that $x^T\mathbf{Q}x = x^T B^{\mathbf{Q}} x$, and hence $\sigma' \ge 1$.

2. We now establish that $\sigma' \le \omega'$. Let $\phi(x) = \frac{1}{2}x^T\mathbf{Q}x$, $x \in \mathbb{R}^N$; its gradient is

   $$\nabla\phi(x) = \mathbf{Q}x. \tag{4.9}$$

   For each $k = 1, 2, \dots, c$, define a pair of conjugate norms on $\mathbb{R}^s$ as follows:

   $$\|v\|^2_{(k)} \overset{\text{def}}{=} \langle \mathbf{Q}^{kk}v, v\rangle, \qquad (\|v\|^*_{(k)})^2 \overset{\text{def}}{=} \max_{\|v'\|_{(k)}\le 1} \langle v', v\rangle = \langle (\mathbf{Q}^{kk})^{-1}v, v\rangle. \tag{4.10}$$

   Let $\mathbf{U}_k$ be a column submatrix of the N-by-N identity matrix corresponding to columns $i \in \mathcal{P}_k$. Clearly, $\mathbf{A}_k = \mathbf{A}\mathbf{U}_k$ and $\mathbf{U}_k^T\mathbf{Q}\mathbf{U}_k$ is the k-th diagonal block of $\mathbf{Q}$, i.e.,

   $$\mathbf{U}_k^T\mathbf{Q}\mathbf{U}_k \overset{(4.3)}{=} \mathbf{Q}^{kk}. \tag{4.11}$$

   Moreover, for $x \in \mathbb{R}^N$ and $k \in \{1, 2, \dots, c\}$, let $x^{(k)} = \mathbf{U}_k^T x$ and, fixing positive scalars

$w_1, \dots, w_c$, define a norm on $\mathbb{R}^N$ as follows:

$$\|x\|_w \overset{\text{def}}{=} \left(\sum_{k=1}^{c} w_k \|x^{(k)}\|_{(k)}^2 \right)^{1/2}. \tag{4.12}$$

Now, we claim that for each k,

$$\|\mathbf{U}_k^T \nabla \phi(x + \mathbf{U}_k h^{(k)}) - \mathbf{U}_k^T \nabla \phi(x)\|_{(k)}^* \leq \|h^{(k)}\|_{(k)}.$$

This means that $\nabla\phi$ is block Lipschitz continuous (with blocks corresponding to variables in $\mathcal{P}_k$), with respect to the norm $\|\cdot\|_{(k)}$, with Lipschitz constant 1. Indeed, this is, in fact, satisfied with equality:

$$\begin{aligned}
\|\mathbf{U}_k^T \nabla \phi(x + \mathbf{U}_k h^{(k)}) - \mathbf{U}_k^T \nabla \phi(x)\|_{(k)}^* &\overset{(4.9)}{=} \|\mathbf{U}_k^T \mathbf{Q}(x + \mathbf{U}_k h^{(k)}) - \mathbf{U}_k \mathbf{Q} x\|_{(k)}^* \\
&= \|\mathbf{U}_k^T \mathbf{Q} \mathbf{U}_k h^{(k)}\|_{(k)}^* \\
&\overset{(4.11)}{=} \|\mathbf{Q}^{kk} h^{(k)}\|_{(k)}^* \\
&\overset{(4.10)}{=} \langle (\mathbf{Q}^{kk})^{-1} \mathbf{Q}^{kk} h^{(k)}, \mathbf{Q}^{kk} h^{(k)} \rangle \overset{(4.10)}{=} \|h^{(k)}\|_{(k)}.
\end{aligned}$$

This is relevant because then, by [54, Theorem 7; see Comment 2 following the theorem], it follows that $\nabla\phi$ is Lipschitz continuous with respect to $\|\cdot\|_w$, where $w_k = 1$ for all $k = 1, \dots, c$, with Lipschitz constant ω' (ω' is the degree of partial block separability of ϕ with respect to the blocks $\mathcal{P}_k$). Hence,

$$\tfrac{1}{2} x^T \mathbf{Q} x = \phi(x) \leq \phi(0) + (\nabla \phi(0))^T x + \frac{\omega'}{2} \|x\|_w^2 \overset{(4.10)+(4.12)}{=} \frac{\omega'}{2} \sum_{k=1}^{c} \langle \mathbf{Q}^{kk} x^{(k)}, x^{(k)} \rangle = \frac{\omega'}{2} (x^T B^{\mathbf{Q}} x),$$

which establishes the inequality $\sigma' \leq \omega'$.

3. We now show that $\frac{\sigma}{s} \leq \sigma'$. If we let $\theta \overset{\text{def}}{=} \max\{x^T B^{\mathbf{Q}} x : x^T x \leq 1\}$, then $x^T B^{\mathbf{Q}} x \leq \theta x^T x$ and hence $\{x \,:\, x^T x \leq 1\} \subseteq \{x \,:\, x^T B^{\mathbf{Q}} x \leq \theta\}$. This implies that

$$\sigma = \max_x \{x^T \mathbf{Q} x \;:\; x^T x \leq 1\} \leq \max_x \{x^T \mathbf{Q} x \;:\; x^T B^{\mathbf{Q}} x \leq \theta\} = \theta \sigma'.$$

It now only remains to argue that $\theta \leq s$. For $x \in \mathbb{R}^N$, let $x^{(k)}$ denote its subvector in $\mathbb{R}^s$ corresponding to coordinates $i \in \mathcal{P}_k$ and $\Delta = \{p \in \mathbb{R}^c : p \geq 0,\ \sum_{k=1}^c p_k = 1\}$. We can now write

$$\begin{aligned}\theta &= \max_x \left\{ \sum_{k=1}^{c} (x^{(k)})^T \mathbf{Q}^{kk} x^{(k)} \; : \; \sum_{k=1}^{c} (x^{(k)})^T x^{(k)} \leq 1 \right\} \\ &= \max_{p \in \Delta} \sum_{k=1}^{c} \left\{ \max (x^{(k)})^T \mathbf{Q}^{kk} x^{(k)} \; : \; (x^{(k)})^T x^{(k)} = p_k \right\} \\ &= \max_{p \in \Delta} \sum_{k=1}^{c} p_k \max \left\{ (x^{(k)})^T \mathbf{Q}^{kk} x^{(k)} \; : \; (x^{(k)})^T x^{(k)} = 1 \right\} \\ &= \max_{1 \leq k \leq c} \max \left\{ (x^{(k)})^T \mathbf{Q}^{kk} x^{(k)} \; : \; (x^{(k)})^T x^{(k)} = 1 \right\} \quad \leq \quad s.\end{aligned}$$

In the last step we have used the fact that $\sigma(\mathbf{Q}) = \sigma \leq \dim(\mathbf{Q})$, proved in steps 1 and 2, applied to the setting $\mathbf{Q} \leftarrow \mathbf{Q}^{kk}$.[3]

4. The chain of inequalities $1 \leq \sigma \leq \omega \leq c$ is obtained as a special case of the chain $1 \leq \sigma' \leq \omega' \leq N$ (proved above) when $c = d$ (and hence $\mathcal{P}_l = \{l\}$ for $l = 1, \ldots, N$). Indeed, in this case $B^{\mathbf{Q}} = D^{\mathbf{Q}}$, and so $x^T B^{\mathbf{Q}} x = x^T D^{\mathbf{Q}} x = x^T x$, which means that $\sigma' = \sigma$ and $\omega' = \omega$.

□

4.2.2 Choice of the Stepsize Parameter β

We analyze Hydra with stepsize parameter $\beta \geq \beta^*$, where

$$\begin{aligned}\beta^* &\overset{\text{def}}{=} \beta_1^* + \beta_2^*, \\ \beta_1^* &\overset{\text{def}}{=} 1 + \frac{(\tau - 1)(\sigma - 1)}{s_1}, \\ \beta_2^* &\overset{\text{def}}{=} \left(\frac{\tau}{s} - \frac{\tau - 1}{s_1} \right) \frac{\sigma' - 1}{\sigma'} \sigma,\end{aligned} \tag{4.13}$$

and $s_1 = \max(1, s - 1)$.

As we shall see in Theorem 19, fixing c and τ, the number of iterations needed by Hydra to find a solution is proportional to β. Hence, we would wish to use β which is as small as possible, but not smaller than the safe choice $\beta = \beta^*$, for which convergence is guaranteed. In practice, β can often be chosen smaller than the safe but conservative value of β^*, leading to larger steps and faster convergence.

If the quantities σ and σ' are hard to compute, then one can replace them by the easily computable upper bounds ω and ω', respectively. However, there are cases when σ can be efficiently approximated and is much smaller than ω. In some ML datasets with $\mathbf{A} \in \{0, 1\}^{m \times N}$,

---

[3] Or saying differently, $\forall k$ and any $x^{(k)}$, such that $(x^{(k)})^T x^{(k)} = 1$, we have $(x^{(k)})^T \mathbf{Q}^{kk} x^{(k)} \leq s$.

σ is close to the average number of nonzeros in a row of $\mathbf{A}$, which can be significantly smaller than the maximum, ω. On the other hand, if σ is difficult to compute, ω may provide a good proxy. Similar remarks apply to σ'.

More importantly, if $\tau \geq 2$ (which covers all interesting uses of Hydra), we may ignore β_2^* altogether, as implied by the following result.

Lemma 13. *If $\tau \geq 2$, then $\beta^* \leq 2\beta_1^*$.*

Proof. It is enough to argue that $\beta_2^* \leq \beta_1^*$. Notice that β_2^* is increasing in σ'. On the other hand, from Lemma 12 we know that $\sigma' \leq c = \frac{N}{s}$. So, it suffices to show that

$$\left(\frac{\tau}{s} - \frac{\tau-1}{s-1}\right)\left(1 - \frac{s}{N}\right)\sigma \leq 1 + \frac{(\tau-1)(\sigma-1)}{s-1}.$$

After straightforward simplification we observe that this inequality is equivalent to $(s-\tau) + (\tau-2)\sigma + \frac{\sigma}{N}(s+\tau) \geq 0$, which clearly holds. ☐

Clearly, $\beta^* \geq \beta_1^*$. Hence, if in Hydra we instead of $\beta = \beta^*$ (best/smallest value prescribed by our theory) use $\beta = 2\beta_1^*$ —eliminating the need to compute σ'—the number of iterations will *at most double.* Since σ', present in β_2^*, captures the effect of the initial partition on the iteration complexity of the algorithm, we conclude that this effect is under control.

4.2.3 Expected Separable Approximation

We first establish a useful identity for the expected value of a random quadratic form obtained by sampling the rows and columns of the underlying matrix via the distributed sampling $\hat{S}$. Note that the result is a direct generalization of Lemma 1 in [66] to the $c > 1$ case.

Lemma 14. *Fix arbitrary $\mathbf{G} \in \mathbb{R}^{N\times N}$ and $x \in \mathbb{R}^N$ and let $s_1 = \max(1, s-1)$. Then*

$$\mathbf{E}[(x_{[\hat{S}]})^T \mathbf{G} x_{[\hat{S}]}] = \frac{\tau}{s}\left[\alpha_1 x^T D^{\mathbf{G}} x + \alpha_2 x^T \mathbf{G} x + \alpha_3 x^T(\mathbf{G} - B^{\mathbf{G}})x\right], \tag{4.14}$$

where $\alpha_1 = 1 - \frac{\tau-1}{s_1}$, $\alpha_2 = \frac{\tau-1}{s_1}$, $\alpha_3 = \frac{\tau}{s} - \frac{\tau-1}{s_1}$.

Proof. In the $s = 1$ case the statement is trivially true. Indeed, we must have $\tau = 1$ and thus $\mathbf{Prob}(\hat{S} = \{1, 2, \ldots, N\}) = 1$, $h_{[\hat{S}]} = h$, and hence

$$\mathbf{E}\left[(h_{[\hat{S}]})^T \mathbf{Q} h_{[\hat{S}]}\right] = h^T \mathbf{Q} h.$$

This finishes the proof since $\frac{\tau-1}{s_1} = 0$.

Consider now the $s > 1$ case. From Lemma 3 in [54] we get

$$\mathbf{E}\left[(h_{[\hat{S}]})^T \mathbf{Q} h_{[\hat{S}]}\right] = \mathbf{E}\left[\sum_{i\in\hat{S}}\sum_{j\in\hat{S}} \mathbf{Q}_{ij} h^{(i)} h^{(j)}\right] = \sum_{i=1}^{N}\sum_{j=1}^{N} p_{ij}\mathbf{Q}_{ij} h^{(i)} h^{(j)}, \tag{4.15}$$

where $p_{ij} = \mathbf{Prob}(i \in \hat{S} \ \&\ j \in \hat{S})$. One can easily verify that

$$p_{ij} = \begin{cases} \frac{\tau}{s}, & \text{if } i = j, \\ \frac{\tau(\tau-1)}{s(s-1)}, & \text{if } i \neq j \text{ and } i \in \mathcal{P}_l,\ j \in \mathcal{P}_l \text{ for some } l, \\ \frac{\tau^2}{s^2}, & \text{if } i \neq j \text{ and } i \in \mathcal{P}_k,\ j \in \mathcal{P}_l \text{ for } k \neq l. \end{cases}$$

In particular, the first case follows from (3.20) and the second from (3.25). It only remains to substitute p_{ij} into (4.15) and transform the result into the desired form.

□

We now use the above lemma to compute a separable quadratic upper bound on $\mathbf{E}[(h_{[\hat{S}]})^T \mathbf{M} h_{[\hat{S}]}]$.

Lemma 15. *For all $h \in \mathbb{R}^N$,*

$$\mathbf{E}\left[(h_{[\hat{S}]})^T \mathbf{M} h_{[\hat{S}]}\right] \leq \frac{\tau}{s}\beta^* \left(h^T D^{\mathbf{M}} h\right). \tag{4.16}$$

Proof. For $x \overset{\text{def}}{=} (D^{\mathbf{M}})^{1/2} h$, we have $(h_{[\hat{S}]})^T \mathbf{M} h_{[\hat{S}]} = (x_{[\hat{S}]})^T \mathbf{Q} x_{[\hat{S}]}$. Taking expectations on both sides, and applying Lemma 14, we see that $\mathbf{E}[(h_{[\hat{S}]})^T \mathbf{M} h_{[\hat{S}]}]$ is equal to (4.14) for $\mathbf{G} = \mathbf{Q}$. It remains to bound the three quadratics in (4.14). Since $D^{\mathbf{Q}}$ is the identity matrix, $x^T D^{\mathbf{Q}} x = h^T D^{\mathbf{M}} h$. In view of (4.6), the 2nd term is bounded as $x^T \mathbf{Q} x \leq \sigma x^T x = \sigma h^T D^{\mathbf{M}} h$. Finally,

$$\begin{aligned} x^T(\mathbf{Q} - B^{\mathbf{Q}})x &= \frac{\sigma' - 1}{\sigma'} x^T(\mathbf{Q} - B^{\mathbf{Q}})x + \frac{1}{\sigma'} x^T(\mathbf{Q} - B^{\mathbf{Q}})x \\ &\overset{(4.7)}{\leq} \frac{\sigma' - 1}{\sigma'} x^T(\mathbf{Q} - B^{\mathbf{Q}})x + \frac{\sigma' - 1}{\sigma'} x^T B^{\mathbf{Q}} x \\ &= \frac{\sigma' - 1}{\sigma'} x^T \mathbf{Q} x \overset{(4.6)}{\leq} \frac{\sigma' - 1}{\sigma'} \sigma x^T x = \frac{\sigma' - 1}{\sigma'} \sigma h^T D^{\mathbf{M}} h. \end{aligned}$$

It only remains to plug in these three bounds into (4.14). □

We met with inequalities of type (4.16) in Chapter 3, where we call them Expected Separable Overapproximation (ESO), and we show their importance for the convergence of parallel coordinate descent methods. However, Chapter 3 studied a different class of loss functions f (convex smooth and partially separable) and different types of random samplings $\hat{S}$, which did not allow us to propose an efficient distributed sampling protocol leading to a distributed algorithm. An ESO inequality was recently used by [66] to design a mini-batch stochastic dual coordinate ascent method (parallelizing the original SDCA methods of [19]) and mini-batch stochastic subgradient descent method (Pegasos of [60]), and give bounds on how mini-batching leads to acceleration. While it was long observed that mini-batching often accelerates Pegasos in practice, it was only shown with the help of an ESO inequality that this is so also in theory. Recently, [17] have derived ESO inequalities for smooth approximations of nonsmooth loss functions and hence showed that parallel coordinate descent methods can accelerate on their serial counterparts on a class of structured nonsmooth convex losses. As a special case, they obtain a parallel randomized coordinate descent method for minimizing the logarithm of the exponential loss. Again, the class of losses considered in that paper, and the samplings $\hat{S}$, are different from ours. None of the above methods are distributed.

4.2.4 Fast Rates for Distributed Learning with Hydra

Let x_0 be the starting point of Algorithm 4.1, x_* be an optimal solution of problem (1.1) and let $F^* = F(x_*)$. Further, define $\|x\|_{\mathbf{M}}^2 \overset{\text{def}}{=} \sum_{i=1}^N \mathbf{M}_{ii}(x^{(i)})^2$ (a weighted Euclidean norm on $\mathbb{R}^N$) and assume that f and Ψ are strongly convex with respect to this norm with convexity parameters μ_f and μ_Ψ, respectively.

We now show that Hydra decreases strongly convex F with an exponential rate in ϵ.

Theorem 22. *Assume F is strongly convex with respect to the norm $\|\cdot\|_{\mathbf{M}}$, with $\mu_f + \mu_\Psi > 0$. Choose $x_0 \in \mathbb{R}^N$, $0 < \rho < 1$, $0 < \epsilon < F(x_0) - F^*$ and*

$$K \quad \geq \quad \frac{N}{c\tau} \times \frac{\beta + \mu_\Psi}{\mu_f + \mu_\Psi} \times \log\left(\frac{F(x_0) - F^*}{\epsilon\rho}\right), \tag{4.17}$$

where $\beta \geq \beta^*$ and β^* is given by (4.13). If $\{x_k\}$ are the random points generated by Hydra (Algorithm 4.1), then

$$\mathbf{Prob}(F(x_K) - F^* \leq \epsilon) \geq 1 - \rho.$$

Proof. We first claim that for all $x, h \in \mathbb{R}^N$,

$$\mathbf{E}\left[f(x + h_{[\hat{S}]})\right] \leq f(x) + \frac{\mathbf{E}[|\hat{S}|]}{N}\left((\nabla f(x))^T h + \frac{\beta}{2} h^T D^{\mathbf{M}} h\right). \tag{4.18}$$

To see this, substitute $h \leftarrow h_{[\hat{S}]}$ into (4.1), take expectations on both sides and then use Lemma 15 together with the fact that for any vector a, $\mathbf{E}[a^T h_{[\hat{S}]}] = \frac{\mathbf{E}[|\hat{S}|]}{N} a^T h = \frac{\tau c}{sc} a^T h = \frac{\tau}{s} a^T h$. The rest follows by following the steps in the proof in Theorem 19. $\square$

A similar result, albeit with the weaker rate $O(\frac{s\beta}{\tau\epsilon})$, can be established in the case when neither f nor Ψ are strongly convex. In big data setting, where parallelism and distribution is unavoidable, it is much more relevant to study the dependence of the rate on parameters such as τ and c. We shall do so in the next section.

4.3 Discussion

In this section we comment on several aspects of the rate captured in (4.17) and compare Hydra to selected methods.

4.3.1 Insights Into the Convergence Rate

Here we comment in detail on the influence of the various design parameters ($c = \#$ computers, $s = \#$ coordinates owned by each computer, and $\tau = \#$ coordinates updated by each computer in each iteration), instance-dependent parameters $(\sigma, \omega, \mu_\Psi, \mu_f)$, and parameters depending both on the instance and design (σ', ω'), on the stepsize parameter β, and through it, on the convergence rate described in Theorem 22.

Strong convexity. Notice that the size of $\mu_\Psi > 0$ mitigates the effect of a possibly large β on the bound (4.17). Indeed, for large μ_Ψ, the factor $(\beta + \mu_\Psi)/(\mu_f + \mu_\Psi)$ approaches 1, and the bound (4.17) is dominated by the term $\frac{N}{c\tau}$, which means that Hydra enjoys linear speedup in c and τ. In the following comments we will assume that $\mu_\Psi = 0$, and focus on studying the dependence of the leading term $N\frac{\beta}{c\tau}$ on various quantities, including τ, c, σ and σ'.

Search for small but safe β. As shown by [66, Section 4.1], mini-batch SDCA might *diverge* in the setting with $\mu_f = 0$ and $\Psi(x) \equiv 0$, even for a simple quadratic function with $N = 2$, provided that $\beta = 1$. Hence, small values of β need to be avoided. However, in view of Theorem 22, it is good if β is as small as possible. So, there is a need for a "safe" formula for a small β. Our formula (4.13), $\beta = \beta^*$, is serving that purpose. For a detailed introduction into the issues related to selecting a good β for parallel coordinate descent methods, we refer the reader to the first 5 pages of [17].

The effect of σ'. If $c = 1$, then by Lemma 4.8, $\sigma' = c = 1$, and hence $\beta_2^* = 0$. However, for $c > 1$ we *may* have $\beta_2^* > 0$, which can hence be seen as a price we need to pay for using more nodes. The price depends on the way the data is partitioned to the nodes, as captured by σ'. In favorable circumstances, $\sigma' \approx 1$ even if $c > 1$, leading to $\beta_2^* \approx 0$. However, in general we

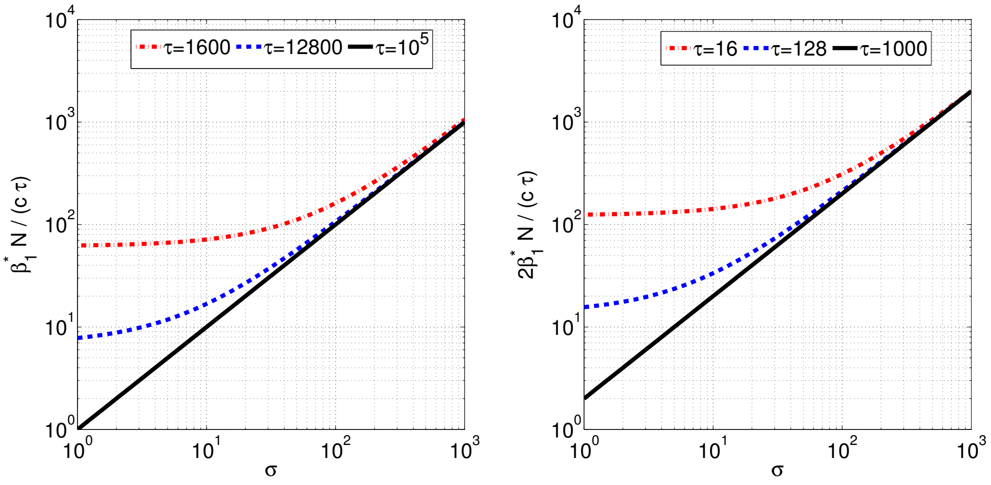
special case	β^*	$\beta^*/(c\tau)$
any c $\tau = 1$	$1 + \frac{\sigma}{s}\left(\frac{\sigma'-1}{\sigma'}\right)$	$s + \sigma\left(\frac{\sigma'-1}{\sigma'}\right)$
$c = 1$ any τ	$1 + \frac{(\tau-1)(\sigma-1)}{N-1}$	$\frac{N}{\tau}\left(1 + \frac{(\tau-1)(\sigma-1)}{N-1}\right)$
$\tau c = N$	σ	σ

Table 4.1: Stepsize parameter $\beta = \beta^*$ and the leading factor in the rate (4.17) (assuming $\mu_\Psi = 0$) for several special cases of Hydra.

have the bound $\sigma' \geq \frac{c\sigma}{N}$, which gets worse as c increases and, in fact, σ' can be as large as c. Note also that $\beta_2^*(\tau, s)$ is decreasing in τ, and that $\beta_2^*(s, s) = 0$. This means that by choosing $\tau = s$ (which effectively removes randomization from Hydra), the effect of β_2^* is eliminated. This may not be always possible as often one needs to solve problems with s vastly larger than the number of updates that can be performed on any given node in parallel. If $\tau \ll s$, the effect of $beta_2^*$ can be controlled, to a certain extent, by choosing a partition with small σ'. Due to the way σ' is defined, this may not be an easy task. However, it may be easier to find partitions that minimize ω', which is often a good proxy for σ'. Alternatively, we may ignore estimating σ' altogether by setting $\beta = 2\beta_1^*$, as mentioned before, at the price of at most doubling the number of iterations.

Speedup by increasing τ. Let us fix c and compare the quantities $\gamma_\tau \overset{\text{def}}{=} \frac{\beta^*}{c\tau}$ for $\tau = 1$ and $\tau = s$. We now show that $\gamma_1 \geq \gamma_s$, which means that if all coordinates are updated at every node, as opposed to one only, then Hydra run with $\beta = \beta^*$ will take fewer iterations. Comparing the 1st and 3rd row of Table 4.1, we see that $\gamma_1 = s + \sigma\frac{\sigma'-1}{\sigma'}$ and $\gamma_s = \sigma$. By Lemma 12, $\gamma_1 - \gamma_s = s - \frac{\sigma}{\sigma'} \geq 0$.

Price of distribution. For illustration purposes, consider a problem with $N = 10^5$ coordinates. In Figure 4.1(a) we depict the size of $\frac{N\beta_1^*}{c\tau}$ for $c = 1$ and several choices of τ, as a function of σ. We see that Hydra works better for small values of σ and that with increasing σ, the benefit of using updating more coordinates diminishes. In Figure 4.1(b) we consider the same scenario, but with $c = 100$ and $s = 1000$, and we plot $\frac{N2\beta_1^*}{c\tau}$ on the y axis. Note that the red dash-dot line in both plots corresponds to a parallel update of 1600 coordinates. In (a) all are updated on a single node, whereas in (b) we have 100 nodes, each updating 16 coordinates at a time. Likewise, the dashed blue and solid black lines are also comparable in both plots. Note that the setup with $c = 100$ has a slightly weaker performance, the lines are a bit higher. This is the price we pay for using c nodes as opposed to a single node (obviously, we are ignoring communication cost here). However, in big data situations one simply has no other choice but



$(c, s) = (1, 10^5)$ $(c, s) = (10^2, 10^3)$

Figure 4.1: In terms of the number of iterations, very little is lost by using $c > 1$ as opposed to $c = 1$.

to utilize more nodes.

4.3.2 Comparison with Other Methods

While we are not aware of any other *distributed* coordinate descent method, Hydra in the $c = 1$ case is closely related to several existing parallel coordinate descent methods.

Hydra vs Shotgun. The Shotgun algorithm (parallel coordinate descent) of [5] is similar to Hydra for $c = 1$. Some of the differences: [5] only consider Ψ equal to the $\|\cdot\|_1$ and their method works in dimension $2N$ instead of the native dimension N. Shotgun was not analyzed for strongly convex f, and convergence in expectation was established. Moreover, [5] analyze the step-size choice $\beta = 1$, fixed independently of the number of parallel updates τ, and give results that hold only in a "small τ" regime. In contrast, our analysis works for any choice of τ.

Hydra vs PCDM. For $c = 1$, Hydra reduces to the parallel coordinate descent method (PCDM) of Section 3.3, but with a *better* stepsize parameter β. We were able to achieve smaller β (and hence better rates) because we analyze a different and more specialized class of loss functions (those satisfying (4.1)). In comparison, Section 3.3 look at a general class of partially separable losses. Indeed, in the $c = 1$ case, our distributed sampling $\hat{S}$ reduces to the sampling considered in Section 3.3 (τ-nice sampling). Moreover, our formula for β (see Table 4.1) is essentially identical to the formula for β provided in Section 3.3, with the exception

ℓ	$\nabla f_i(x)$	$\mathbf{M}_{ii}$
SL	$\sum_{j=1}^m \quad -\mathbf{A}_{ji}(y_j - \mathbf{A}_j^T x)$	$\|\mathbf{A}_{:i}\|_2^2$
LL	$\sum_{j=1}^m \quad -y_j \mathbf{A}_{ji} \frac{\exp(-y_j \mathbf{A}_j^T x)}{1+\exp(-y_j \mathbf{A}_j^T x)}$	$\frac{1}{4}\|\mathbf{A}_{:i}\|_2^2$
HL	$\sum_{j \,:\, y_j \mathbf{A}_j^T x < 1} \left(-y_j \mathbf{A}_{ji}(1 - y_j \mathbf{A}_j^T x)\right)$	$\|\mathbf{A}_{:i}\|_2^2$

Table 4.2: Information needed in Step 6 of Hydra for f given by (4.2) in the case of the three losses ℓ from Table 3.1.

that we have σ where they have ω. By (4.8), we have $\sigma \leq \omega$, and hence our β is smaller.

Hydra vs SPCDM. SPCDM of [17] is PCDM applied to a smooth approximation of a nonsmooth convex loss; with a special choice of β, similar to β_1. As such, it extends the reach of PCDM to a large class of nonsmooth losses, obtaining $O(\frac{1}{\epsilon^2})$ rates. It is possible to develop accelerated Hydra with $O(\frac{1}{\epsilon})$ rates by combining ideas from this paper, [17] with the APPROX method of [16].

Hydra vs mini-batch SDCA. [66] (Section 5.5.2) studied the performance of a mini-batch stochastic dual coordinate ascent for SVM dual ("mini-batch SDCA"). This is a special case of our setup with $c = 1$, convex quadratic f and $\Psi_i(t) = 0$ for $t \in [0, 1]$ and $\Psi_i(t) = +\infty$ otherwise. Our results can thus be seen as a generalization of the results in that paper to a larger class of loss functions f, more general regularizers Ψ, and most importantly, to the distributed setting ($c > 1$). Also, we give $O(\log \frac{1}{\epsilon})$ bounds under strong convexity, whereas [66] give $O(\frac{1}{\epsilon})$ results without assuming strong convexity. However, [66] perform a primal-dual analysis, whereas we do not.

4.4 Distributed Computation of the Gradient

In this section we describe some important elements of our distributed implementation for three loss functions from Table 2.9.

Note that in Hydra, x_k is stored in a distributed way. That is, the values $x_k^{(i)}$ for $i \in \mathcal{P}_l$ are stored on computer l. Moreover, Hydra partitions $\mathbf{A}$ columnwise as $\mathbf{A} = [\mathbf{A}^{(1)}, \dots, \mathbf{A}^{(c)}]$, where $\mathbf{A}^{(l)}$ consists of columns $i \in \mathcal{P}_l$ of $\mathbf{A}$, and stores $\mathbf{A}^{(l)}$ on computer l. So, $\mathbf{A}$ is chopped into smaller pieces which are stored in a distributed way in fast memory (if possible) across the c nodes. Note that this allows the method to work with large matrices.

At Step 5 of Hydra, node l at iteration $k+1$ needs to know the partial derivatives $\nabla f_i(x_{k+1})$ for $i \in \hat{S}_l \subseteq \mathcal{P}_l$. We now describe several efficient distributed protocols for the computation of $\nabla f_i(x_{k+1})$ for functions f of the form (4.2), in the case of the three losses ℓ given in Table 2.9

(SL, LL, HL). The formulas for $\nabla f_i(x)$ are summarized in Table 4.2. Let $D^y \stackrel{\text{def}}{=} \text{Diag}(y)$.

4.4.1 Basic Protocol

If we write $h_k^{(i)} = 0$ if i is not updated in iteration k, then

$$x_{k+1} = x_k + \sum_{l=1}^{c} \sum_{i \in \hat{S}_l} h_k^{(i)} e_i. \tag{4.19}$$

Now, if we let

$$g_k \stackrel{\text{def}}{=} \begin{cases} \mathbf{A}x_k - y, & \text{for SL,} \\ -D^y \mathbf{A}x_k, & \text{for LL and HL,} \end{cases} \tag{4.20}$$

then by combining (4.19) and (4.20), we get

$$g_{k+1} = g_k + \sum_{l=1}^{c} \delta g_{k,l}, \qquad \text{where}$$

$$\delta g_{k,l} = \begin{cases} \sum_{i \in \hat{S}_l} h_k^{(i)} \mathbf{A}_{:i}, & \text{for SL,} \\ \sum_{i \in \hat{S}_l} -h_k^{(i)} D^y \mathbf{A}_{:i}, & \text{for LL and HL.} \end{cases}$$

Note that the value $\delta g_{k,l}$ can be computed on node l as all the required data is stored locally. Hence, we let each node compute $\delta g_{k,l}$, and then use a *reduce all* operation to add up the updates to obtain g_{k+1}, and pass the sum to all nodes. Knowing g_{k+1}, node l is then able to compute $\nabla f_i(x_{k+1})$ for any $i \in \mathcal{P}_l$ as follows:

$$\nabla f_i(x_{k+1}) = \begin{cases} \mathbf{A}_{:i}^T g_{k+1} = \sum_{j=1}^{m} \mathbf{A}_{ji} g_{k+1}^j, & \text{for SL,} \\ \sum_{j=1}^{m} y_j \mathbf{A}_{ji} \frac{\exp(g_{k+1}^j)}{1+\exp(g_{k+1}^j)}, & \text{for LL,} \\ \sum_{j \,:\, g_{k+1}^j > -1} y_j \mathbf{A}_{ji} (1 + g_{k+1}^j), & \text{for HL.} \end{cases}$$

4.4.2 Advanced Protocols

The basic protocol discussed above has obvious drawbacks. Here we identify them and propose modifications leading to better performance.

- *alternating Parallel and Serial regions (PS):* The basic protocol alternates between two procedures: i) a computationally heavy one (done in parallel) with no MPI communication, and ii) MPI communication (serial). An easy fix would be to dedicate 1 thread to deal with communication and the remaining threads within the same computer for computation. We call this protocol *Fully Parallel (FP)*. Figure 4.2 compares the basic (left) and FP (right) approaches.

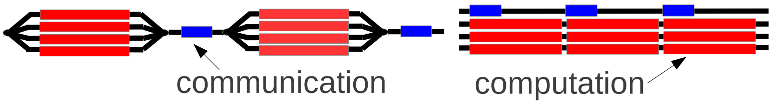


Figure 4.2: Parallel-serial (PS; left) vs Fully Parallel (FP; right) approach.

- *Reduce All (RA):* In general, reduce all operations may significantly degrade the performance of distributed algorithms. Communication taking place only between nodes close to each other in the network, e.g., nodes directly connected by a cable, is more efficient. Here we propose the *Asynchronous StreamLined (ASL)* communication protocol in which each node, in a given iteration, sends only 1 message (asynchronously) to a nearby computer, and also receives only one message (asynchronously) from another nearby computer. Communication hence takes place in an *Asynchronous Ring*. This communication protocol requires significant changes in the algorithm. Figure 4.3 illustrates the flow of messages at the end of the k-th iteration for $c = 4$. In the ideal case the communication between nodes should use the knowledge of the communication topology of the computer facility, i.e. the communication could be just between nodes which are directly connected by network cable.

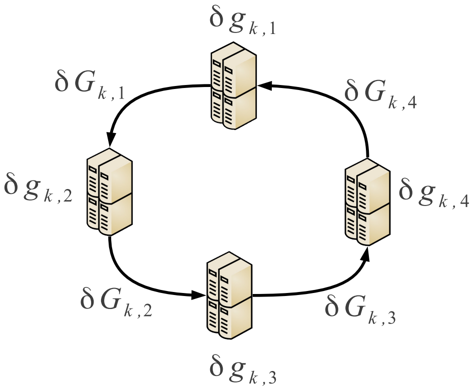


Figure 4.3: ASL protocol with $c = 4$ nodes. In iteration k, node l computes $\delta g_{k,l}$, and sends $\delta G_{k,l}$ to l_+.

We order the nodes into a ring, denoting l_- and l_+ the two nodes neighboring node l. Node l only receives data from l_-, and sends data to l_+. Let us denote by $\delta G_{k,l}$ the data sent by node l to l_+ at the end of iteration k. When l starts iteration k, it already knows

$\delta G_{k-1,l_-}$.[4] Hence, data which will be sent at the end of the k-th iteration by node l is given by

$$\delta G_{k,l} = \delta G_{k-1,l_-} - \delta g_{k-c,l} + \delta g_{k,l}. \tag{4.21}$$

This leads to the update rule

$$g_{k+1,l} = g_{k,l} + \delta g_{k,l} + \delta G_{k,l_-} - \delta g_{k-c+1,l}.$$

ASL needs less communication per iteration. On the other hand, information is propagated more slowly to the nodes through the ring, which may adversely affect the number of iterations till convergence (note that we do not analyze Hydra with this communication protocol). Indeed, it takes $c-1$ iterations to propagate information to all nodes. Also, storage requirements have increased: at iteration k we need to store the vectors $\delta g_{t,l}$ for $k-c \leq t \leq k$ on computer l.

4.5 Experiments

In this section we present numerical evidence that Hydra is capable to efficiently solve big data problems. We have a C++ implementation, using Boost::MPI and OpenMP. Experiments were executed on a Cray XE6 cluster with 128 nodes; with each node equipped with two AMD Opteron Interlagos 16-core processors and 32 GB of RAM.

4.5.1 Performance of Communication Protocols

In this experiment we consider a LASSO problem, i.e., f given by (4.2) with ℓ being the square loss (SL) and $\Psi(x) = \|x\|_1$. In order to to test Hydra under controlled conditions, we adapted the LASSO generator proposed by [42, Section 6]; modifications were necessary as the generator does not work well in the big data setting.

As discussed in Section 4.4, the advantage of the RA protocol is the fact that Theorem 22 was proved in this setting, and hence can be used as a safe benchmark for comparison with the advanced protocols.

Table 4.3 compares the average time per iteration for the 3 approaches and 3 choices of τ. We used 128 nodes, each running 4 MPI processes (hence $c = 512$). Each MPI process runs 8 OpenMP threads, giving 4,096 cores in total. The data matrix $\mathbf{A}$ has $m = 10^9$ rows and $N = 5 \times 10^8$ columns, and has 3 TB, double precision. One can observe that in all cases,

---

[4] Initially, we let $\delta g_{k,l} = \delta G_{k,l} = 0$ for all $k \leq 0$.

$\mathcal{T}$	comm. protocol	organization	avg. time	speedup
10	RA	PS	0.040	—
10	RA	FP	0.035	1.15
10	ASL	FP	0.025	1.62
10^2	RA	PS	0.100	—
10^2	RA	FP	0.077	1.30
10^2	ASL	FP	0.032	3.11
10^3	RA	PS	0.321	—
10^3	RA	FP	0.263	1.22
10^3	ASL	FP	0.249	1.29

Table 4.3: Duration of a single Hydra iteration for 3 communication protocols. The basic RA-PS protocol is always the slowest, but follows the theoretical analysis. ASL-FP can be 3× faster.

ASL-FP yields largest gains compared to the benchmark RA-PS protocol. Note that ASL has some overhead in each iteration, and hence in cases when computation per node is small ($\tau = 10$), the speedup is only 1.62. When $\tau = 10^2$ (in this case the durations of computation and communication were comparable), ASL-FP is 3.11 times faster than RA-PS. But the gain becomes again only moderate for $\tau = 10^3$; this is because computation now takes much longer than communication, and hence the choice of strategy for updating the auxiliary vector g_k is less significant. Let us remark that the use of larger τ requires larger *beta*, and hence possibly more iterations (in the worst case).

We now move on to solving an artificial big data LASSO problem with matrix $\mathbf{A}$ in block angular form, depicted in (4.22).

$$\mathbf{A} = \left(\begin{array}{c:c:c:c} \mathbf{A}_{loc}^{(1)} & 0 & \cdots & 0 \\ 0 & \mathbf{A}_{loc}^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \hdashline \mathbf{A}_{glob}^{(1)} & \mathbf{A}_{glob}^{(2)} & \cdots & \mathbf{A}_{glob}^{(c)} \end{array}\right). \tag{4.22}$$

Such matrices often arise as a dual in a stochastic programming. Each Hydra head (=node) l owns two matrices: $\mathbf{A}_{loc}^{(l)} \in \mathbb{R}^{1,952,148 \times 976,562}$ and $\mathbf{A}_{glob}^{(l)} \in \mathbb{R}^{500,224 \times 976,562}$. The average number of nonzero elements per row in the local part of $\mathbf{A}^{(l)}$ is 175, and 1,000 for the global part. Optimal solution x_* has exactly 160,000 nonzero elements. Figure 4.4 compares the evolution of $F(x_k) - F^*$ for ASL-FP and RA-FP.

Remark: When communicating g_{kl}, only entries corresponding to the global part of $\mathbf{A}^{(l)}$ need to be communicated, and hence in RA, a *reduce all* operation is applied to vectors $\delta g_{glob,l} \in \mathbb{R}^{500,224}$. In ASL, vectors with the same length are sent.

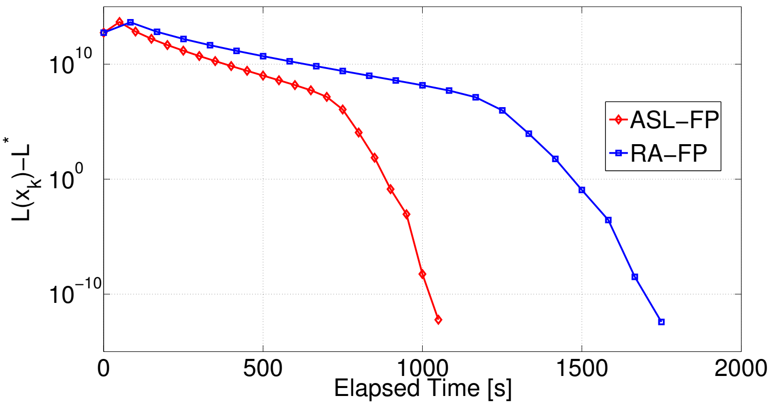


Figure 4.4: Evolution of $F(x_k) - F^*$ in time. ASL-FP significantly outperforms RA-FP. The loss F is pushed down by 25 degrees of magnitude in less than 30 minutes (3TB problem).

5

Primal-Dual Coordinate Descent Method

> We cannot solve our problems with the same thinking we used when we created them.
>
> — Albert Einstein (1879–1955)

5.1 Introduction

Stochastic optimization approaches have been shown to have significant theoretical and empirical advantages in training linear Support Vector Machines (SVMs), as well as in many other learning applications, and are often the methods of choice in practice. Such methods use a single, randomly chosen, training example at each iteration. In the context of SVMs, approaches of this form include primal stochastic gradient descent (SGD) (e.g., Pegasos, [60], NORMA, [86, 20], and dual stochastic coordinate ascent (e.g., SDCA, [19], RCDM, [53]).

However, the inherent sequential nature of such approaches becomes a problematic limitation for parallel and distributed computations as the predictor must be updated after each training point is processed, providing very little opportunity for parallelization. A popular remedy is to use *mini-batches*. That is, to use several training points at each iteration, instead of just one, calculating the update based on each point separately and aggregating the updates. The question is then whether basing each iteration on several points can indeed reduce the number of required iterations, and thus yield parallelization speedups.

In this chapter, we consider using mini-batches with Pegasos (SGD on the primal objective) and with Stochastic Dual Coordinate Ascent (SDCA). We show that for *both methods*, the quantity that controls the speedup obtained using mini-batching/parallelization is the *spectral norm of the data.*

In Section 5.4 we provide the first analysis of mini-batched Pegasos (with the original, non-smooth, SVM objective) that provably leads to parallelization speedups (Theorem 23). The idea of using mini-batches with Pegasos is not new, and is discussed already by [60], albeit without a theoretical justification. The original analysis does not benefit from using mini-batches—the same number of iterations is required even when large mini-batches are used, there is no speedup, and the serial runtime (overall number of operations, in this case data accesses) increases linearly with the mini-batch size. In fact, no parallelization speedup can be guaranteed based only on a bound on the radius of the data, as in the original Pegasos analysis. Instead, we provide a refined analysis based on the spectral norm of the data.

We then move on to SDCA (Section 5.5). We show the situation is more involved, and a modification to the method is necessary. SDCA has been consistently shown to outperform Pegasos in practice [19, 60], and is also popular as it does not rely on setting a step-size as in Pegasos. It is thus interesting and useful to obtain mini-batch variants of SDCA as well. We first show that a naive mini-batching approach for SDCA can fail, in particular when the mini-batch size is large relative to the spectral norm (Section 5.5.1). We then present a "safe" variant, and an analysis that establishes the same spectral-norm-dependent parallelization speedups as for Pegasos (Section 5.5.2). Similar to a recent analysis of non-mini-batched SDCA by [64], we establish a guarantee on the duality gap, and thus also on the sub-optimality of the *primal* SVM objective, when using mini-batched SDCA (Theorem 24). We then go on to describe a more aggressive, adaptive, method for mini-batched SDCA, which is based on the analysis of the "safe" approach, and which we show often outperforms it in practice (Section 5.5.3, with experiments in Section 2.7).

For simplicity of presentation we focus on the hinge loss, as in the SVM objective. However, *all our results for both Pegasos and SDCA are valid for any Lipschitz continuous loss function* and are provided in [67].

5.2 Related Work

Several recent papers consider the use of mini-batches in stochastic gradient descent, as well as stochastic dual averaging and stochastic mirror descent, when minimizing a *smooth* loss function [11, 1, 8]. These papers establish parallelization speedups for *smooth* loss minimization with mini-batches, possibly with the aid of some "acceleration" techniques, and without relying on, or considering, the spectral norm of the data. However, these results do not apply to SVM training, where the objective to be minimized is the non-smooth hinge loss. In fact, the only data assumption in these papers is an assumption on the radius of the data, which is *not* enough for obtaining parallelization guarantees when the loss is non-smooth.

Our contribution is thus orthogonal to prior work, showing that it is possible to obtain parallelization speedups even for non-smooth objectives, but only with a dependence on the spectral norm. We also analyze SDCA, which is a substantially different method from the methods analyzed in these papers. It is interesting to note that a bound of the spectral norm could perhaps indicate that it is easier to "smooth" the objective, and thus allow obtaining results similar to ours (i.e. on the suboptimality of the original non-smooth objective) by smoothing the objective and relying on mini-batched smooth SGD, where the spectral norm might control how well the smoothed loss captures the original loss. But we are not aware of any analysis of this nature, nor whether such an analysis is possible.

There has been some recent work on mini-batched coordinate descent methods for ℓ_1-regularized problems (and, more generally, regularizes by a separable convex function), similar to the SVM dual. [5] presented and analyzed SHOTGUN, a parallel coordinate descent method for ℓ_1-regularized problems, showing linear speedups for mini-batch sizes bounded in terms of the spectral norm of the data. The analysis does not directly apply to the SVM dual because of the box constraints, but is similar in spirit. Furthermore, [5] do not discuss a "safe" variant which is applicable for any mini-batch size, and only study the analogue of what we refer to as "naive" mini-batching (Section 5.5.1). More directly related is recent work of [53, 54, 69, 55] which provided a theoretical framework and analysis for a more general setting than SHOTGUN, that includes also the SVM dual as a special case. However, guarantees in this framework, as well as those of [5], are only on the dual suboptimality (in our terminology), and not on the more relevant primal suboptimality, i.e., the suboptimality of the original SVM problem we are interested in. Our theoretical analysis builds on that of [54], combined with recent ideas of [64] for "standard" (serial) SDCA, to obtain bounds on the duality gap and primal suboptimality.

5.3 Support Vector Machines

We consider the optimization problem of training a linear[1] Support Vector Machine (SVM) based on n labeled training examples $\{(x_i, y_i)\}_{i=1}^n$, where $x_i \in \mathbb{R}^d$ and $y_i \in \pm 1$. We use $X = [x_1, \ldots, x_n] \in \mathbb{R}^{d\times n}$ to denote the matrix of training examples. We assume the data is normalized such that $\max_i \|x_i\| \leq 1$, and thus suppress the dependence on $\max_i \|x_i\|$ in all

---

[1] Since both Pegasos and SDCA can be kernelized, all methods discussed are implementable also with kernels, and all our results hold. However, the main advantage of SGD and SDCA is where the feature map is given explicitly, and so we focus our presentation on this setting.

results. Training a SVM corresponds to finding a linear predictor $w \in \mathbb{R}^d$ with low ℓ_2-norm $\|w\|$ and small (empirical) average hinge loss $\hat{L}(w) \overset{\text{def}}{=} \frac{1}{n}\sum_{i=1}^n \ell(y_i\langle w, x_i\rangle)$, where $\ell(z) \overset{\text{def}}{=} [1-z]_+ = \max\{0, 1-z\}$. This bi-objective problem can be serialized as

$$\min_{w\in\mathbb{R}^d}\left[P(w) \overset{\text{def}}{=} \frac{1}{n}\sum_{i=1}^n \ell(y_i\langle w, x_i\rangle) + \frac{\lambda}{2}\|w\|^2\right], \tag{5.1}$$

where $\lambda > 0$ is a regularization trade-off parameter. It is also useful to consider the dual of (5.1):

$$\max_{\alpha\in\mathbb{R}^n, 0\le\alpha^{(i)}\le 1}\left[D(\alpha) \overset{\text{def}}{=} \frac{-\alpha^T Q\alpha}{2\lambda n^2} + \frac{1}{n}\sum_{i=1}^n \alpha^{(i)}\right], \tag{5.2}$$

$$Q \in \mathbb{R}^{n\times n}, \quad Q_{i,j} = y_i y_j\langle x_i, x_j\rangle; \tag{5.3}$$

Q is the Gram matrix of the (labeled) data. The (primal) optimum of (5.1) is given by

$$w^* = \frac{1}{\lambda n}\sum_{i=1}^n (\alpha^*)^{(i)} y_i x_i,$$

where α^* is the (dual) optimum of (5.2). It is thus natural to associate with each dual solution α a primal solution (i.e., a linear predictor)

$$w(\alpha) \overset{\text{def}}{=} \frac{1}{\lambda n}\sum_{i=1}^n \alpha^{(i)} y_i x_i. \tag{5.4}$$

We will be discussing "mini-batches" of size τ, represented by *random subsets* $S \subseteq [n] \overset{\text{def}}{=} \{1, 2, \dots, n\}$ of examples, drawn uniformly at random from all subsets of $[n]$ of cardinality τ.[2] Whenever we draw such a subset, we for simplicity write $S \in \hat{S}_\tau$. By $Q_S \in \mathbb{R}^{\tau\times\tau}$ we denote the submatrix of Q corresponding to rows and columns indexed by S, $v_S \in \mathbb{R}^\tau$ to denote a similar restriction of a vector $v \in \mathbb{R}^n$, and $v_{[S]} \in \mathbb{R}^n$ for the "censored" vector where entries inside S are as in v and entries outside S are zero. The average hinge loss on examples in S is denoted by

$$\hat{L}_S(w) \overset{\text{def}}{=} \frac{1}{\tau}\sum_{i\in S} \ell(y_i\langle w, x_i\rangle). \tag{5.5}$$

5.4 Mini-Batches in Primal Stochastic Gradient Descent Methods

Pegasos is an SGD approach to solving (5.1), where at each iteration the iterate w_t is updated based on an unbiased estimator of a sub-gradient of the objective $P(w)$. Whereas in a "pure"

---

[2] If we use the terminology from Chapter 3, then "mini-batch" is nothing more than a τ-nice sampling (see Section 3.5.1), but we prefer to use the terminology "mini-batch" which is a convention in machine learning literature.

Algorithm 5.1 Pegasos with Mini-Batches

```
Input: ${(x_i, y_i)}_{i=1}^n$, $\lambda > 0$, $\tau \in [n]$, $T \geq 1$
Initialize: set $w_1 = 0 \in \mathbb{R}^d$
for $t = 1$ to $T$ do
    Choose random mini-batch $S_t \in \hat{S}_\tau$
    $\eta_t = \frac{1}{\lambda t}$, $S_t^+ = \{i \in S_t \ : \ y_i \langle w_t, x_i \rangle < 1\}$
    $w_{t+1} = (1 - \eta_t \lambda) w_t + \frac{\eta_t}{\tau} \sum_{i \in S_t^+} y_i x_i$
end for
Output: $\bar{w}_T = \frac{2}{T} \sum_{t=\lfloor T/2 \rfloor + 1}^T w_t$
```

stochastic setting, the sub-gradient is estimated based on only a *single* training example,[3] in our mini-batched variation (Algorithm 5.1) at each iteration we consider the partial objective:

$$P_t(w) \overset{\text{def}}{=} \hat{L}_{S_t}(w) + \frac{\lambda}{2} \|w\|^2 , \tag{5.6}$$

where $S_t \in \hat{S}_\tau$. We then calculate the subgradient of the partial objective P_t at w_t:

$$\boldsymbol{\nabla}^{(t)} \overset{\text{def}}{=} \nabla P_t(w_t) \overset{(5.6)}{=} \nabla \hat{L}_{S_t}(w_t) + \lambda w_t, \tag{5.7}$$

where

$$\nabla \hat{L}_S(w) \overset{(5.5)}{=} -\frac{1}{\tau} \sum_{i \in S} \chi_i(w) y_i x_i \tag{5.8}$$

and $\chi_i(w) \overset{\text{def}}{=} 1$ if $y_i \langle w, x_i \rangle < 1$ and 0 otherwise (indicator for not classifying example i correctly with a margin). The next iterate is obtained by setting $w_{t+1} = w_t - \eta_t \boldsymbol{\nabla}_t$. We can now write

$$w_{t+1} \overset{(5.7)+(5.8)}{=} (1 - \eta_t \lambda) w_t + \frac{\eta_t}{\tau} \sum_{i \in S_t} \chi_i(w_t) y_i x_i. \tag{5.9}$$

Analysis of mini-batched Pegasos rests on bounding the norm of the subgradient estimates $\boldsymbol{\nabla}^{(t)}$. The standard Pegasos analysis uses the bound $\|\nabla \hat{L}_S(w)\| \leq \frac{1}{\tau} \sum_{i \in S} \|\chi_i(w) y_i x_i\| \leq \frac{1}{\tau} \sum_{i \in S} 1 = 1$. From (5.7) we then get $\|\boldsymbol{\nabla}^{(t)}\| \leq \lambda \|w_t\| + 1$; the standard Pegasos analysis follows. This bound relies only on the assumption $\max_i \|x_i\| \leq 1$, and is the tightest bound without further assumptions on the data.

The core novel observation here is that the expected (square) norm of $\nabla \hat{L}_S$ can be bounded in terms of (an upper bound on) *the spectral norm of the data:*

$$\sigma^2 \geq \frac{1}{n} \|X\|^2 = \frac{1}{n} \left\| \sum_i x_i x_i^T \right\| \overset{(5.3)}{=} \frac{1}{n} \|Q\| , \tag{5.10}$$

---

[3] The example is chosen uniformly at random and hence the random sub-gradient is unbiased estimator of the sub-gradient of the objective $P(w)$.

where $\|\cdot\|$ denotes the spectral norm (largest singular value) of a matrix. In order to bound $\nabla\hat{L}_S$, we first perform the following calculation, introducing the key quantity β_τ, useful also in the analysis of SDCA.

Lemma 16. *For any* $v \in \mathbb{R}^n$, $\tilde{Q} \in \mathbb{R}^{n\times n}$,

$$\mathbf{E}[v_{[\hat{S}_\tau]}^T \tilde{Q} v_{[\hat{S}_\tau]}] = \frac{\tau}{n}\left[\left(1 - \frac{\tau-1}{n-1}\right)\sum_{i=1}^n \tilde{Q}_{ii}(v^{(i)})^2 + \frac{\tau-1}{n-1} v^T\tilde{Q}v\right].$$

Moreover, if $\tilde{Q}_{ii} \le 1$ *for all* i *and* $\frac{1}{n}\|\tilde{Q}\| \le \sigma^2$, *then*

$$\mathbf{E}[v_{[\hat{S}_\tau]}^T \tilde{Q} v_{[\hat{S}_\tau]}] \le \frac{\tau}{n}\beta_\tau \left\|v\right\|^2,$$

where

$$\beta_\tau \overset{def}{=} 1 + \frac{(\tau-1)(n\sigma^2-1)}{n-1}. \tag{5.11}$$

Proof.

$$\begin{aligned}
\mathbf{E}[v_{[\hat{S}_\tau]}^T \tilde{Q} v_{[\hat{S}_\tau]}] &= \mathbf{E}\left[\sum_{i\in\hat{S}_\tau}(v^{(i)})^2\tilde{Q}_{ii} + \sum_{i,j\in\hat{S}_\tau, i\neq j} v^{(i)}v^{(j)}\tilde{Q}_{ij}\right] \\
&= \tau\mathbf{E}_i[(v^{(i)})^2\tilde{Q}_{ii}] + \tau(\tau-1)\mathbf{E}_{i,j}[v^{(i)}v^{(j)}\tilde{Q}_{ij}] \\
&= \frac{\tau}{n}\sum_i \tilde{Q}_{ii}(v^{(i)})^2 + \frac{\tau(\tau-1)}{n(n-1)} v^T(\tilde{Q} - \mathrm{diag}(\tilde{Q}))v \\
&= \frac{\tau}{n}\left[(1 - \frac{\tau-1}{n-1})\sum_i \tilde{Q}_{ii}(v^{(i)})^2 + \frac{\tau-1}{n-1}v^T\tilde{Q}v\right],
\end{aligned}$$

where the expectations are over i, j chosen uniformly at random without replacement. Now using $\tilde{Q}_{ii} \le 1$ and $\|\tilde{Q}\| \le n\sigma^2$, we can further write

$$\le \frac{\tau}{n}\left[(1 - \frac{\tau-1}{n-1})\left\|v\right\|^2 + \frac{\tau-1}{n-1}n\sigma^2\left\|v\right\|^2\right] = \frac{\tau}{n}\beta_\tau\left\|v\right\|^2.$$

□

We can now apply Lemma 16 to $\nabla\hat{L}_{\hat{S}_\tau}$:

Lemma 17. *For any* $w \in \mathbb{R}^d$ *we have* $\mathbf{E}[\|\nabla\hat{L}_{\hat{S}_\tau}(w)\|^2] \le \frac{\beta_\tau}{\tau}$, *where* β_τ *is as in Lemma 16.*

Proof. If $\chi \in \mathbb{R}^n$ is the vector with entries $\chi_i(w)$, then

$$\mathbf{E}[\|\nabla \hat{L}_{\hat{S}_\tau}(w)\|^2] \overset{(5.8)}{=} \mathbf{E}\left[\left\|\frac{1}{\tau}\sum_{i\in\hat{S}_\tau} \boldsymbol{\chi}_i y_i x_i\right\|^2\right] \overset{(5.3)}{=} \frac{1}{\tau^2}\mathbf{E}[\boldsymbol{\chi}^T_{[\hat{S}_\tau]} Q \boldsymbol{\chi}_{[\hat{S}_\tau]}]$$

$$\overset{(\text{Lemma 16})}{\leq} \frac{1}{\tau^2}\frac{\tau}{n}\beta_\tau \|\boldsymbol{\chi}\|^2 \leq \frac{\beta_\tau}{\tau}.$$

□

Using the by-now standard analysis of SGD for strongly convex functions, we obtain the main result of this section:

Theorem 23. *After T iterations of Pegasos with mini-batches (Algorithm 1), we have that for the averaged iterate $\bar{w}_T = \frac{2}{T}\sum_{t=\lfloor T/2\rfloor+1}^{T} w_t$:*

$$\mathbf{E}\left[P(\bar{w}_T)\right] - \inf_{w\in\mathbb{R}^d} P(w) \leq \frac{\beta_\tau}{\tau}\cdot\frac{30}{\lambda T}.$$

Proof. Unrolling (5.9) with $\eta_t = 1/(\lambda t)$ yields

$$w_t = -\frac{1}{\lambda(t-1)}\sum_{i=1}^{t-1} g_i, \tag{5.12}$$

where $g_i \overset{\text{def}}{=} \nabla \hat{L}_{S_i}(w_\tau)$. Using the inequality $\|\sum_{i=1}^{t-1} g_i\|^2 \leq (t-1)\sum_{i=1}^{t-1}\|g_i\|^2$, we now get

$$\mathbf{E}[\|w_t\|^2] \overset{(5.12)}{\leq} \sum_{i=1}^{t-1}\frac{\mathbf{E}[\|g_i\|^2]}{\lambda^2(t-1)} \overset{(\text{Lemma 17})}{\leq} \frac{\beta_\tau}{\lambda^2\tau}, \tag{5.13}$$

$$\mathbf{E}[\|\boldsymbol{\nabla}^{(t)}\|^2] \overset{(5.7)+(\text{Lem2})}{\leq} 2(\lambda^2\mathbf{E}[\|w_t\|^2] + \frac{\beta_\tau}{\tau}) \overset{(5.13)}{\leq} 4\frac{\beta_\tau}{\tau}.$$

The performance guarantee is now given by the analysis of SGD with tail averaging (Theorem 5 of [49], with $\alpha = \frac{1}{2}$ and $G^2 = 4\frac{\beta_\tau}{\tau}$). □

Parallelization speedup. When $\tau = 1$ we have $\beta_\tau = 1$ (see (5.11)) and Theorem 23 agrees with the standard (serial) Pegasos analysis[4] [60]. For larger mini-batches, the guarantee depends on the quantity β_τ, which in turn depends on the spectral norm σ^2. Since $\frac{1}{n} \leq \sigma^2 \leq 1$, we have $1 \leq \beta_\tau \leq \tau$.

The worst-case situation is at a degenerate extreme, when all data points lie on a single line, and so $\sigma^2 = 1$ and $\beta_\tau = \tau$. In this case Lemma 17 degenerates to the worst-case bound

---

[4] Except that we avoid the logarithmic factor by relying on tail averaging and a more modern SGD analysis.

of $\mathbf{E}[\|\nabla \hat{L}_S(\mathbf{w})\|^2] \leq 1$, and in Theorem 23 we have $\frac{\beta_\tau}{\tau} = 1$, indicating that using larger mini-batches does not help at all, and the same number of parallel iterations is required.

However, when $\sigma^2 < 1$, and so $\beta_\tau < 1$, we see a benefit in using mini-batches in Theorem 23, corresponding to a parallelization speedup of $\frac{\tau}{\beta_\tau}$. The best situation is when $\sigma^2 = \frac{1}{n}$, and so $\beta_\tau = 1$, which happens when all training points are orthogonal. In this case there is never any interaction between points in the mini-batch, and using a mini-batch of size τ is just as effective as making τ single-example steps. When $\beta_\tau = 1$ we indeed see that the speedup is equal to the number of mini-batches, and that the behavior in terms of the number of data accesses (equivalently, serial runtime) τT, does not depend on τ; that is, even with larger mini-batches, we require no more data accesses, and we gain linearly from being able to perform the accesses in parallel. The case $\sigma^2 = \frac{1}{n}$ is rather extreme, but even for intermediate values $\frac{1}{n} < \sigma^2 < 1$ we get speedup. In particular, as long as $\tau \leq \frac{1}{\sigma^2}$, we have $\beta_\tau \leq 2$, and an essentially linear speedup. Roughly speaking, $\frac{1}{\sigma^2}$ captures the number of examples in the mini-batch beyond which we start getting significant interactions between points.

5.5 Mini-Batches in Dual Stochastic Coordinate Ascent Methods

An alternative stochastic method to Pegasos is Stochastic Dual Coordinate Ascent (SDCA, [19]), aimed to solve the dual problem (5.2). At each iteration we choose a single training example (x_i, y_i), uniformly at random, corresponding to a single dual variable (coordinate) $\alpha^{(i)} = e_i^T \alpha$. Subsequently, $\alpha^{(i)}$ is updated so as to maximize the (dual) objective, keeping all other coordinates of α unchanged and maintaining the box constraints. At iteration t, the update $\delta_t^{(i)}$ to $\alpha_t^{(i)}$ is computed via

$$\begin{aligned}
\delta_t^{(i)} &\overset{\text{def}}{=} \arg\max_{0 \leq \alpha_t^{(i)} + \delta \leq 1} D(\alpha_t + \delta e_i) \\
&\overset{(5.2)}{=} \arg\max_{0 \leq \alpha_t^{(i)} + \delta \leq 1} (\lambda n - (Q e_i)^T \alpha_t)\delta - \frac{Q_{i,i}}{2}\delta^2 \\
&= \operatorname{clip}_{[-\alpha_t^{(i)}, 1-\alpha_t^{(i)}]} \frac{\lambda n - (Q e_i)^T \alpha_t}{Q_{i,i}} \\
&\overset{(5.3),(5.4)}{=} \operatorname{clip}_{[-\alpha_t^{(i)}, 1-\alpha_t^{(i)}]} \frac{\lambda n (1 - y_i \langle w(\alpha_t), x_i \rangle)}{\|x_i\|^2},
\end{aligned} \tag{5.14}$$

where clip_I is projection onto the interval I. Variables $\alpha_t^{(j)}$ for $j \neq i$ are unchanged. Hence, a single iteration has the form $\alpha_{t+1} = \alpha_t + \delta_t^{(i)} e_i$. Similar to a Pegasos update, at each iteration a single, random, training point is considered, the "response" $y_i \langle w(\alpha_t), x_i \rangle$ is calculated (this operation dominates the computational effort), and based on the response, a multiple of x_i is added to the weight vector w (corresponding to changing $\alpha^{(i)}$). The two methods thus involve fairly similar operations at each iteration, with essentially identical computational costs. They differ in that in Pegasos, $\alpha^{(i)}$ is changed according to some pre-determined step-size, while SDCA changes it optimally so as to maximize the dual objective (and maintain dual feasibility); there is no step-size parameter.

SDCA was suggested and studied empirically by [19], where empirical advantages over Pegasos were often observed. In terms of a theoretical analysis, by considering the dual problem (5.2) as an ℓ_1-regularized, box-constrained quadratic problem, it is possible to obtain guarantees on the *dual* suboptimality, $D(\alpha^*) - D(\alpha_t)$, after a finite number of SDCA iterations [61, 43, 53]. However, such guarantees do *not* directly imply guarantees on the *primal* suboptimality of $w(\alpha_t)$. Recently, [64] bridged this gap, and provided guarantees on $P(w(\alpha_t)) - P(w^*)$ after a finite number of SDCA iterations. These guarantees serve as the starting point for our theoretical study.

5.5.1 Naive Mini-Batching

A naive approach to parallelizing SDCA using mini-batches is to compute $\delta_t^{(i)}$ in parallel, according to (5.14), for all $i \in S_t$, all based on the current iterate α_t, and then update $\alpha_{t+1}^{(i)} = \alpha_t^{(i)} + \delta_t^{(i)}$ for $i \in S_t$, and keep $\alpha_{t+1}^{(j)} = \alpha_t^{(j)}$ for $j \notin S_t$. However, not only might this approach not reduce the number of required iterations, it might actually *increase* the number of required iterations. This is because the dual objective need *not improve monotonically* (as it does for "pure" SDCA), and even not converge.

To see this, consider an extreme situation with only two identical training examples: $Q = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$, $\lambda = \frac{1}{n} = \frac{1}{2}$ and mini-batch size $\tau = 2$ (i.e., in each iteration we use both examples). If we start with $\alpha_0 = \mathbf{0}$ with $D(\alpha_0) = 0$ then $\delta_1^{(0)} = \delta_2^{(0)} = 1$ and following the naive approach we have $\alpha_1 = (1, 1)^T$ with objective value $D(\alpha_1) = 0$. In the next iteration $\delta_1^{(1)} = \delta_2^{(1)} = -1$ which brings us back to $\alpha_2 = \mathbf{0}$. So the algorithm will alternate between those two solutions with objective value $D(\alpha) = 0$, while at the optimum $D(\alpha^*) = D((0.5, 0.5)^T) = 0.25$.

This is of course a simplistic toy example, but the same phenomenon will occur when a large number of training examples are identical or highly correlated. This can also be observed empirically in some of our experiments discussed later, e.g., in Figure 5.2.

The problem here is that since we update each $\alpha^{(i)}$ independently to its optimal value *as if all other coordinates were fixed*, we are ignoring interactions between the updates. As we see in the extreme example above, two different $i, j \in S_t$, might suggest essentially the same change to $w(\alpha_t)$, but we would then perform this update *twice*, overshooting and yielding a new iterate which is actually worse then the previous one.

5.5.2 Safe Mini-Batching

Properly accounting for the interactions between coordinates in the mini-batch would require jointly optimizing over all $\alpha^{(i)}$, $i \in S_t$. This would be a very powerful update and no-doubt reduce the number of required iterations, but would require solving a box-constrained quadratic program, with a quadratic term of the form $\delta_S^T Q_S \delta_S$, $\delta_S \in \mathbb{R}^\tau$, at each iteration. This quadratic program cannot be distributed to different machines, each handling only a single data point.

Instead, we propose a "safe" variant, where the term $\delta_S^T Q_S \delta_S$ is approximately bounded by the separable surrogate $\beta \left\|\delta_S\right\|^2$, for some $\beta > 0$ which we will discuss later. That is, the update is given by:

$$\begin{aligned}\delta_t^{(i)} &\stackrel{\text{def}}{=} \underset{0 \leq \alpha_t^{(i)} + \delta \leq 1}{\arg\max}\ (\lambda n - (Qe_i)^T \alpha_t)\delta - \frac{\beta}{2}\delta^2 \\ &= \operatorname{clip}_{[-\alpha_t^{(i)}, 1-\alpha_t^{(i)}]} \frac{\lambda n(1 - y_i \langle w(\alpha_t), x_i \rangle)}{\beta},\end{aligned} \tag{5.15}$$

with $\alpha_{t+1}^{(i)} = \alpha_t^{(i)} + \delta_t^{(i)}$ for $i \in S_t$, and $\alpha_{t+1}^{(j)} = \alpha_t^{(j)}$ for $j \notin S_t$. In essence, $\frac{1}{\beta}$ serves as a step-size, where we are now careful not to take steps so big that they will accumulate together and overshoot the objective. If handling only a single point at each iteration, such a short-step approach is not necessary, we do not need a step-size, and we can take a "full step", setting $\alpha^{(i)}$ optimally ($\beta = 1$). But with the potential for interaction between coordinates updated in parallel, we must use a smaller step.

We will first rely on the bound (5.10), and establish that the choice $\beta = \beta_\tau$ as in (5.11) provides for a safe step size. To do so, we consider the dual objective at $\alpha + \delta$,

$$D(\alpha + \delta) = -\frac{\alpha^T Q \alpha + 2\alpha^T Q \delta + \delta^T Q \delta}{2\lambda n^2} + \sum_{i=1}^{n} \frac{\alpha^{(i)} + \delta^{(i)}}{n}, \tag{5.16}$$

and the following separable approximation to it:

$$H(\delta, \alpha) \stackrel{\text{def}}{=} -\frac{\alpha^T Q \alpha + 2\alpha^T Q \delta + \beta_\tau \left\|\delta\right\|^2}{2\lambda n^2} + \sum_{i=1}^{n} \frac{\alpha^{(i)} + \delta^{(i)}}{n}, \tag{5.17}$$

in which $\beta_\tau \left\|\delta\right\|^2$ replaces $\delta^T Q \delta$. Our update (5.15) with $\beta = \beta_\tau$ can be written as $\delta = \arg \max_{\delta : 0 \leq \alpha + \delta \leq 1} H(\delta, \alpha)$ (we then use the coordinates $\delta^{(i)}$ for $i \in S$ and ignore the rest). We are essentially performing parallel coordinate ascent on $H(\delta, \alpha)$ instead of on $D(\alpha + \delta)$. To understand this approximation, we note that $H(\mathbf{0}, \alpha) = D(\alpha)$, and show that $H(\delta, \alpha)$ provides an expected lower bound on $D(\alpha + \delta)$:

Lemma 18. *For any $\alpha, \delta \in \mathbb{R}^n$,*

$$\mathbf{E}_{\hat{S}}[D(\alpha + \delta_{[\hat{S}]})] \geq \left(1 - \frac{\tau}{n}\right) D(\alpha) + \frac{\tau}{n} H(\delta, \alpha).$$

Proof. Examining (5.16) and (5.17), the terms that do not depend on δ are equal on both sides. For the linear term in δ, we have that $\mathbf{E}[\delta_{[S]}] = \frac{\tau}{n}\delta$, and again we have equality on both sides. For the quadratic term we use Lemma 16 which yields $\mathbf{E}[\delta_{[S]}^T Q \delta_{[S]}] \leq \frac{\tau}{n}\beta_\tau \|\delta\|^2$, and after negation establishes the desired bound. □

Inequalities of this general type are also studied in [54] (see Sections 3 and 4). Based on the above lemma, we can modify the analysis of [64] to obtain (see complete proof in the supplementary material):

Theorem 24. *Consider the SDCA updates given by* (5.15)*, with $S_t \in \hat{S}_\tau$, starting from $\alpha^{(0)} = \mathbf{0}$ and with $\beta = \beta_\tau$ (given in eq.* (5.11)*). For any $\epsilon > 0$ and*

$$\begin{aligned} t_0 &\geq \max\{0, \lceil \frac{n}{b} \log(\frac{2\lambda n}{\beta_\tau}) \rceil\}, && (5.18)\\ T_0 &\geq t_0 + \frac{\beta_\tau}{b}\left[\frac{4}{\lambda\epsilon} - 2\frac{n}{\beta_\tau}\right]_+, && (5.19)\\ T &\geq T_0 + \max\{\lceil \frac{n}{b} \rceil, \frac{\beta_\tau}{\tau}\frac{1}{\lambda\epsilon}\}, && (5.20)\\ \bar{\alpha} &\stackrel{def}{=} \frac{1}{T - T_0}\sum_{t=T_0}^{T-1} \alpha_t, && (5.21) \end{aligned}$$

we have

$$\mathbf{E}[P(w(\bar{\alpha}))] - P(w^*) \leq \mathbf{E}[P(w(\bar{\alpha})) - D(\bar{\alpha})] \leq \epsilon.$$

By Theorem 24, the # of iterations of mini-batched SDCA, sufficient to reach *primal* sub-optimality ϵ, is

$$\tilde{O}\left(\frac{n}{\tau} + \frac{\beta_\tau}{\tau} \cdot \frac{1}{\lambda\epsilon}\right), \tag{5.22}$$

We observe the *same speedup* as in the case of mini-batched Pegasos: factor of $\frac{\tau}{\beta_\tau}$, with an essentially linear speedup when $\tau \leq \frac{1}{\sigma^2}$. It is interesting to note that the quantity β_τ only affects the second, ϵ-dependent, term in (5.22). The "fixed cost" term, which essentially requires a full pass over the data, is *not* affected by β_τ, and is *always* scaled down by τ.

5.5.3 Aggressive Mini-Batching

Using $\beta = \beta_\sigma$ is safe, but might be too conservative. In particular, we used the spectral norm to bound $\delta^T Q \delta \leq \|Q\| \, \|\delta\|^2$ in Lemma 18 (through Lemma 16), but this is a worst case bound over all possible vectors, and might be loose for the relevant vectors δ. Relying on a worst-case bound might mean we are taking much smaller steps then we could. Furthermore, the approach we presented thus far relies on knowing the spectral norm of the data, or at least a bound on the spectral norm (recall (5.10)), in order to set the step-size. Although it is possible to estimate this quantity by sampling, this can certainly be inconvenient.

Instead, we suggest a more aggressive variant of mini-batched SDCA which gradually adapts β based on the actual values of $\|\delta^{(t)}_{[S_t]}\|^2$ and $\delta^{(t)}_{[S_t]} Q \delta^{(t)}_{[S_t]}$. In Section 2.7 one can observe the advantages of this aggressive strategy. In this variant, at each iteration we calculate the ratio $\rho = \tilde{\delta}^T_{[S]} Q \tilde{\delta}^T_{[S]} / \|\tilde{\delta}_{[S]}\|^2$, and nudge the step size towards it by updating it to a weighted geometric average of the previous step size and "optimal" step size based on the step δ considered. One complication is that due to the box constraints, not only the magnitude but also the direction of the step δ depends on the step-size β, leading to a circular situation. The approach we take is as follows: we maintain a "current step size" β. At each iteration, we first calculate a tentative step $\tilde{\delta}_S$, according to (5.15), with the current β. We then calculate ρ according to this step direction, and update β to $\beta^\gamma \rho^{1-\gamma}$ for some pre-determined parameter $0 < \gamma < 1$ that controls how quickly the step-size adapts. But, instead of using $\tilde{\delta}$ calculated with the previous β, we actually re-compute δ_S using the step-size ρ. We note that this means the ratio ρ does *not* correspond to the step δ_S actually taken, but rather to the tentative step $\tilde{\delta}_S$. We could potentially continue iteratively updating ρ according to δ_S and δ_S according to ρ, but we found that this does not improve performance significantly and is not worth the extra computational effort. This aggressive strategy is summarized in Algorithm 5.2. Note that we initialize $\beta = \beta_\tau$, and also constrain β to remain in the range $[1, \beta_\tau]$, but we can use a very crude upper bound σ^2 for calculating β_τ. Also, in our aggressive strategy, we refuse steps that do not actually increase the dual objective, corresponding to overly aggressive step sizes.

Carrying out the aggressive strategy requires computing $\tilde{\delta}^T_{[S]} Q \tilde{\delta}_{[S]}$ and the dual objective efficiently and in parallel. The main observation here is that:

$$\tilde{\delta}^T_{[S]} Q \tilde{\delta}_{[S]} = \| \sum_{i \in S} \tilde{\delta}^{(i)} y_i \mathbf{x_i} \|^2 \tag{5.23}$$

and so the main operation to be performed is an aggregation of $\sum_{i \in S} \tilde{\delta}^{(i)} y_i x_i$, similar to the operation required in mini-batched Pegasos. As for the dual objective, it can be written as

Algorithm 5.2 SDCA with Mini-Batches (aggressive)

```
Initialize: set $\alpha_0 = \mathbf{0}$, $w_0 = \mathbf{0}$, $\beta_0 = \beta_\tau$
for $t = 0$ to $T$ do
  Choose $S_t \in \hat{S}_\tau$
  For $i \in S_t$, compute $\tilde{\delta}^{(i)}$ from (5.15) using $\beta = \beta_t$
  Sum $\zeta := \sum_{i \in S_t} (\tilde{\delta}^{(i)})^2$ and $\tilde{\Delta} := \sum_{i \in S_t} \tilde{\delta}^{(i)} y_i x_i$
  Compute $\rho = \mathrm{clip}_{[1,\beta_\tau]}(\|\tilde{\Delta}\|^2 / \zeta)$
  For $i \in S_t$, compute $\delta^{(i)}$ from (5.15) using $\beta = \rho$.
  $\beta_{t+1} := (\beta_t)^\gamma \rho^{1-\gamma}$
  if $D(\alpha_t + \delta_{[S_t]}) > D(\alpha_t)$ then
    $\alpha_{t+1} = \alpha_t + \delta_{[S_t]}$,
    $w_{t+1} = w_t + \frac{1}{\lambda n} \sum_{i \in S_t} \delta^{(i)} y_i x_i$
  else
    $\alpha_{t+1} = \alpha_t$, $w_{t+1} = w_t$
  end if
end for
```

$D(\alpha) = -\left\|w(\alpha)\right\|^2 - \frac{1}{n}\left\|\alpha\right\|_1$ and can thus be readily calculated if we maintain $w(\alpha)$, its norm, and $\left\|\alpha\right\|_1$.

5.6 Experiments

Figure 5.1 shows the required number of iterations (corresponding to the parallel runtime) required for achieving a primal suboptimality of 0.001 using Pegasos, naive SDCA, safe SDCA and aggressive SDCA, on four benchmark datasets detailed in Table 5.1, using different mini-batch sizes. Also shown (on an independent scale; right axis) is the leading term $\frac{\beta_\tau}{\tau}$ in our complexity results. The results confirm the advantage of SDCA over Pegasos, at least for $\tau = 1$, and that both Pegasos and SDCA enjoy nearly-linear speedups, at least for small batch sizes. Once the mini-batch size is such that $\frac{\beta_\tau}{\tau}$ starts flattening out (corresponding to $\tau \approx \frac{1}{\sigma^2}$, and so significant correlations inside each mini-batch), the safe variant of SDCA follows a similar behavior and does not allow for much parallelization speedup beyond this point, but at least does not deteriorate like the naive variant. Pegasos and the aggressive variant do continue showing speedups beyond $\tau \approx \frac{1}{\sigma^2}$. The experiments clearly demonstrate the aggressive modification allows SDCA to continue enjoying roughly the same empirical speedups as Pegasos, even for large mini-batch sizes, maintaining an advantage throughout. It is interesting to note that the aggressive variant continues improving even past the point of failure of the naive variant, thus establishing that it is empirically important to adjust the step-size to achieve a balance between safety and progress.

In Figure 5.2 we depict the evolution of solutions using the various methods for two specific data sets. Here we can again see the relative behavior of the methods, as well as clearly see the failure of the naive approach, which past some point causes the objective to deteriorate and does not converge to the optimal solution.

5.7 Summary and Discussion

Contribution. Our contribution in this chapter is twofold:

1. we identify the spectral norm of the data, and through it $\beta_{\mathcal{T}}$, as the important quantity controlling guarantees for mini-batched/parallelized Pegasos (primal method) and SDCA (dual method). We provide the first analysis of mini-batched Pegasos, with the non-smooth hinge-loss, that shows speedups, and we analyze for the first time mini-batched SDCA with guarantees expressed in terms of the primal problem (hence, our mini-batched SDCA is a primal-dual method);
2. based on our analysis, we present novel variants of mini-batched SDCA which are necessary for achieving speedups similar to those of Pegasos, and thus open the door to effective mini-batching using the often-empirically-better SDCA.

Related work. Our safe SDCA mini-batching approach is similar to the parallel coordinate descent methods of [5] and [54], but we provide an analysis in terms of the primal SVM objective, which is the more relevant object of interest. Furthermore, [5]'s analysis does *not* use a step-size and is thus limited only to small enough mini-batches—if the spectral norm is unknown and too large a mini-batch is used, their method might not converge. [53]'s method does incorporate a *fixed* step-size, similar to our safe variant, but as we discuss this step-size might be too conservative for achieving the true potential of mini-batching. In the context of SVMs with mini-batches, the analysis of [13, 14] is valid in small dimensions d, implying nearly linear speedups up to a batch size that gets worse as d increases. On the other hand, our analysis is dimension independent, allows for infinite dims (as in the kernelized case) or extremely high d (e.g., when the feature vectors are very sparse).

Table 5.1: Datasets and regularization parameters λ used; "%" is percent of features which are non-zero. *cov* is the forest covertype dataset of [60], *astro-ph* consists of abstracts of papers from physics also of [60], *rcv1* is from the Reuters collection and *news20* is from the 20 news groups both obtained from libsvm collection [27].

Data	# train	# test	# dim	%	$10^3\lambda$
cov	522,911	58,101	54	22	0.010
rcv1	20,242	677,399	47,236	0.16	0.100
ast.-ph	29,882	32,487	99,757	0.08	0.050
news20	15,020	4,976	1,355,191	0.04	0.125

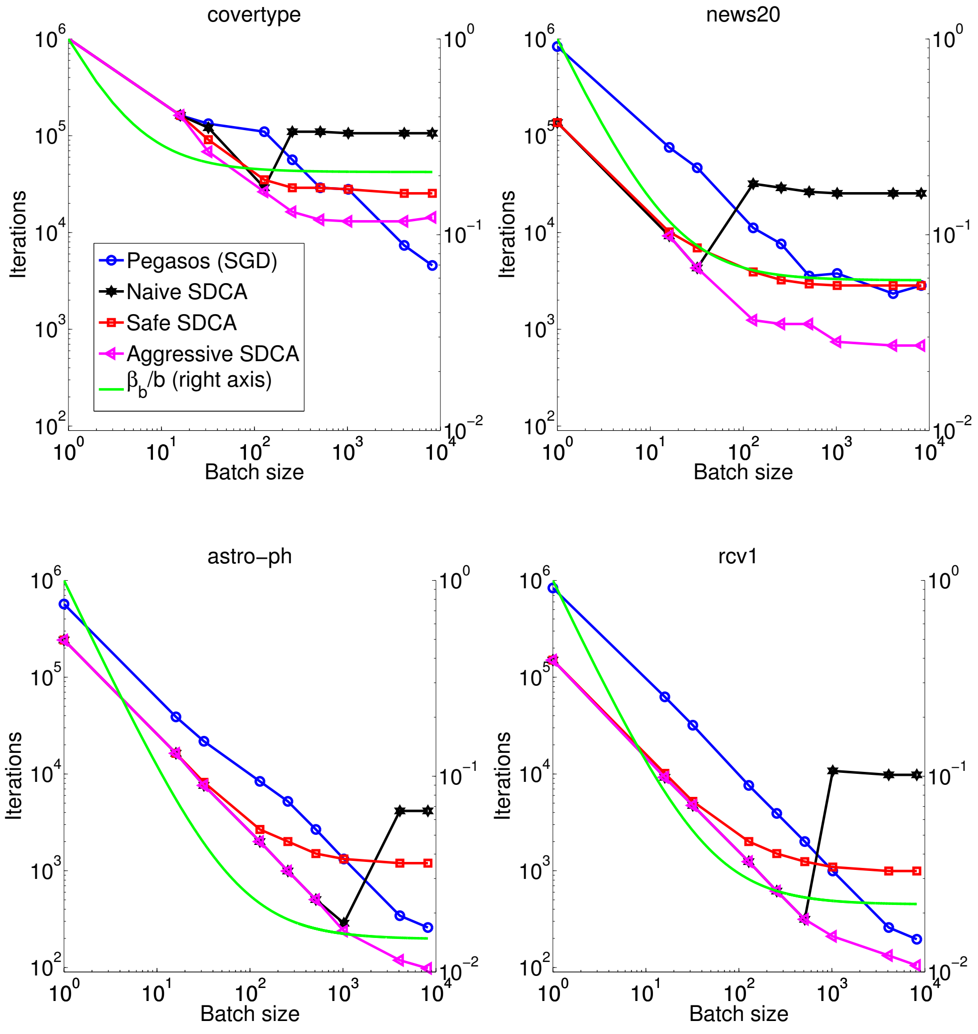


Figure 5.1: Number of iterations (left vertical axis) needed to find a 0.001-accurate *primal* solution for different mini-batch sizes b (horizontal axis). The leading factor in our analysis, β_τ/b, is plotted on the right vertical axis.

Generality. We chose to focus on Pegasos and SDCA with regularized hinge-loss minimization, but all our results remain unchanged for any Lipschitz continuous loss functions. Furthermore, Lemma 17 can also be used to establish identical speedups for mini-batched SGD optimization of $\min_{\|w\|\leq B} \hat{L}(w)$, as well as for direct stochastic approximation of the population objective (generalization error) $\min L(w)$. In considering the population objective, the sample size is essentially infinite, we sample with replacements (from the population), σ^2 is a bound on the second moment of the data distribution, and $\beta_\tau = 1 + (\tau - 1)\sigma^2$.

Experiments. Our experiments confirm the empirical advantages of SDCA over Pegasos, previously observed without mini-batching. However, we also point out that in order to perform mini-batched SDCA effectively, a step-size is needed, detracting from one of the main advantages

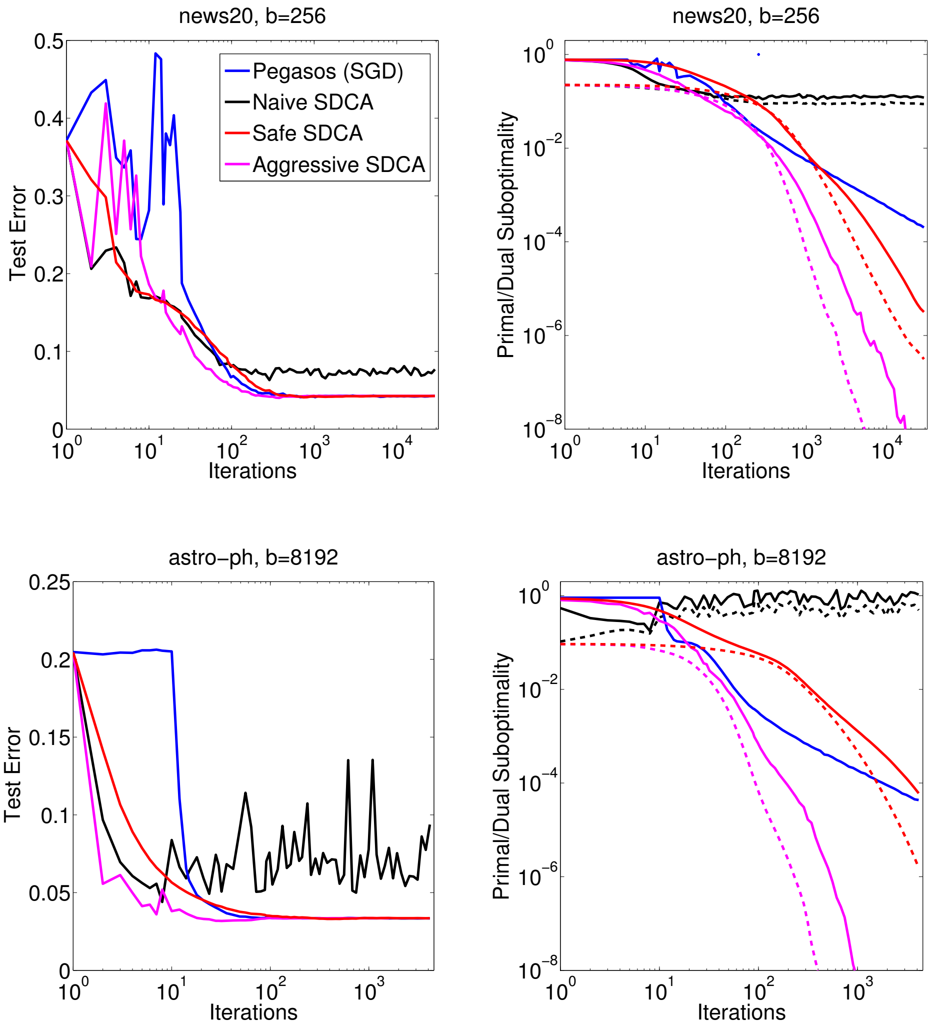


Figure 5.2: Evolutions of primal (solid) and dual (dashed) sub-optimality and test error for news20 and astro-ph datasets. Instead of tail averaging, in the experiments we used decaying averaging with $\bar{w}_t = 0.9\bar{w}_{t-1} + 0.1w_t$.

of SDCA over Pegasos. Furthermore, in the safe variant, this stepsize needs to be set according to the spectral norm (or bound on the spectral norm), with too small a setting for β (i.e., too large steps) possibly leading to non-convergence, and too large a setting for β yielding reduced speedups. In contrast, the Pegasos stepsize is *independent* of the spectral norm, and in a sense Pegasos adapts implicitly (see, e.g., its behavior compared to aggressive SDCA in the experiments). We do provide a more aggressive variant of SDCA, which does match Pegasos's speedups empirically, but this requires an explicit heuristic adaptation of the stepsize.

Parallel Implementation. In this Chapter we analyzed the iteration complexity, and behavior of the iterates, of mini-batched Pegasos and SDCA. Unlike "pure" (τ=1) Pegasos and SDCA, which are not amenable to parallelization, using mini-batches does provide opportunities for it. Of course, actually achieving good parallelization speedups on a specific architecture in practice requires an efficient parallel, possibly distributed, implementation of the iterations. In this regard, we point out that the core computation required for both Pegasos and SDCA is that of computing $\sum_{i\in S} g_i(\langle w, x_i\rangle)x_i$, where g is some scalar function. Parallelizing such computations efficiently in a distributed environment has been studied by e.g., [11, 10]; their methods can be used here too. Alternatively, one could also consider asynchronous or delayed updates Chapter 4, [1, 50].

5.8 Proof of Theorem 24

The proof of Theorem 24 follows mostly along the path of [64], crucially using Lemma 18, and with a few other required modifications detailed below.

We will prove the theorem for a general L-Lipschitz continuous loss function $\ell(\cdot)$. For consistency with [64], we will also allow example-specific loss functions ℓ_i, $i = 1, 2, \ldots, n$, and only require each ℓ_i be individually Lipschitz continuous, and thus refer to the primal and dual problems (expressed slightly differently but equivalently):

$$\min_{w\in\mathbb{R}^d} \left[P(w) \overset{\text{def}}{=} \frac{1}{n}\sum_{i=1}^n \ell_i(\langle w, x_i\rangle) + \frac{\lambda}{2}\|w\|^2 \right], \tag{5.24}$$

$$\max_{\alpha\in\mathbb{R}^n} \left[D(\alpha) \overset{\text{def}}{=} -\frac{1}{n}\sum_{i=1}^n \ell_i^*(-\alpha^{(i)}) - \frac{\lambda}{2}\left\|\frac{1}{\lambda n}X^T\alpha\right\|_2^2 \right], \tag{5.25}$$

where $\ell_i^*(u) = \max_z(zu - \ell_i(z))$ is the Fenchel conjugate of ℓ_i. In the above we dropped without loss of generality the labels y_i since we can always substitute $x_i \leftarrow y_i x_i$. For the hinge loss $\ell_i(a) = [1-a]_+$ we have $\ell_i^*(-a) = -a$ for $a \in [0,1]$ and $\ell_i^*(-a) = \infty$ otherwise, thus encoding the box constraints. Recall also (from (5.4)) that $w(\alpha) = \frac{1}{\lambda n}\sum_{i=1}^n \alpha^{(i)} x_i$ and so $\|w(\alpha)\|^2 = \frac{1}{\lambda^2 n^2}\alpha^T X X^T \alpha = \left\|\frac{1}{\lambda n}X^T\alpha\right\|^2$.

The separable approximation $H(\delta, \alpha)$ defined in (5.17) now has the more general form:

$$H(\delta,\alpha) := -\frac{1}{n}\sum_{i=1}^n \ell_i^*(-(\alpha^{(i)}+\delta^{(i)})) - \frac{\lambda}{2}\left(\|w(\alpha)\|^2 + \beta_\tau \frac{1}{\lambda n}\sum_{i=1}^n \|x_i\|^2 (\delta^{(i)})^2 + 2\left(\frac{1}{\lambda n}\delta\right)^T X w(\alpha) \right) \tag{5.26}$$

and all the properties mentioned in Section 5.5, including Lemma 18, still hold.

Our goal here is to get a bound on the duality gap, which we will denote by

$$\mathcal{G}(\alpha) \overset{\text{def}}{=} P(w(\alpha)) - D(\alpha) = \frac{1}{n}\sum_{i=1}^n \left[\ell_i(\langle w(\alpha), x_i\rangle) + \ell_i^*(-\alpha^{(i)}) + \alpha^{(i)}\langle w(\alpha), x_i\rangle \right]. \tag{5.27}$$

The analysis now rests on the following lemma, paralleling Lemma 1 of [64], which bounds the expected improvement in the dual objective after a single iteration in terms of the duality gap:

Lemma 19. *For any t and any $s \in [0,1]$ we have*

$$\mathbf{E}_{S_t}[D(\alpha_{t+1})] - D(\alpha_t) \geq \tau \left(\frac{s}{n}\mathcal{G}(\alpha_t) - \left(\frac{s}{n}\right)^2 \frac{\beta_\tau}{2\lambda} G_t \right), \tag{5.28}$$

where

$$G_t \overset{def}{=} \frac{1}{n}\sum_{i=1}^{n} \|x_i\|^2 (\chi_t^{(i)} - \alpha_t^{(i)})^2 \leq G, \tag{5.29}$$

with $G = 4L$ for general L-Lipschitz continuous loss, and $G = 1$ for the hinge loss, and $-\chi_t^{(i)} \in \ell_i'(\langle w(\alpha_t), x_i\rangle)$.

Proof. The situation here is trickier then in the case $\tau = 1$ considered by [64], and we will first bound the right hand side of (5.28) by $H - D$ and then use the fact that δ_t is a minimizer of $H(\cdot, \alpha)$:

$$-\frac{n}{\tau}\left(\mathbf{E}[D(\alpha_{t+1})] - D(\alpha_t)\right) = -\frac{n}{\tau}\left(\mathbf{E}[D(\alpha_t + (\delta_t)_{[S_t]})] - D(\alpha_t)\right) \overset{\text{(Lemma 18)}}{\leq} -H(\delta_t, \alpha_t) + D(\alpha_t)$$
$$= \frac{1}{n}\sum_{i=1}^{n}\left(\ell_i^*(-(\alpha_t^{(i)} + \delta_t^{(i)})) - \ell_i^*(-\alpha_t^{(i)})\right) + \frac{\lambda}{2}\left(\beta_\tau \left\|\frac{1}{\lambda n}\delta_t\right\|_X^2 + 2\left(\frac{1}{\lambda n}\delta_t\right)^T X w(\alpha_t)\right),$$

where we denote $\|u\|_X^2 \overset{\text{def}}{=} \sum_{i=1}^n u_i^2 \|x^{(i)}\|^2$. We will now use the optimally of δ_t to upper bound the above, noting that if we replace δ_t with any quantity, and in particular with $s(\chi_{(t)} - \alpha_t)$, we can only decrease $H(\cdot, \alpha_t)$, and thus increase the right-hand-side above:

$$\leq \frac{1}{n}\sum_{i=1}^{n}\left[\ell_i^*(-(\alpha_t^{(i)} + s(\chi_t^{(i)} - \alpha_t^{(i)}))) - \ell_i^*(-\alpha_t^{(i)})\right]$$
$$+ \frac{\lambda}{2}\left(\beta_\tau \left\|\frac{1}{\lambda n}s(\chi_t - \alpha_t)\right\|_X^2 + 2\left(\frac{1}{\lambda n}s(\chi_t - \alpha_t)\right)^T X w(\alpha_t)\right)$$

Now from convexity we have $\ell_i^*(-(\alpha_t^{(i)} + s(\chi_t^{(i)} - \alpha_t^{(i)}))) \leq s\ell_i^*(-\chi_t^{(i)}) + (1-s)\ell_i^*(-\alpha_t^{(i)})$, and so:

$$\begin{aligned}
&\leq \frac{1}{n}\sum_{i=1}^{n}\left(s\ell_i^*(-\chi_t^{(i)}) + s\chi_t^{(i)}\langle w(\alpha_t), x_i\rangle - s\ell_i^*(-\alpha_t^{(i)})\right) \\
&\qquad + \frac{\lambda}{2}\left(\beta_\tau\left\|\frac{1}{\lambda n}s(\chi_{(t)} - \alpha_t)\right\|_X^2 + 2\left(\frac{1}{\lambda n}s(-\alpha_t)\right)^T Xw(\alpha_t)\right)
\end{aligned}$$

and from conjugacy we have $\ell_i^*(-\chi_t^{(i)}) = -\chi_t^{(i)}\langle w(\alpha_t), x_i\rangle - \ell_i(\langle w(\alpha_t), x_i\rangle)$, and so:

$$\begin{aligned}
&\leq \frac{s}{n}\sum_{i=1}^{n}\left(-\chi_t^{(i)}\langle w(\alpha_t), x_i\rangle - \ell_i\left(\langle w(\alpha_t), x_i\rangle\right) + \chi_t^{(i)}\langle w(\alpha_t), x_i\rangle - \ell_i^*(-\alpha_t^{(i)})\right) \\
&\qquad + \frac{\lambda}{2}\left(\beta_\tau\left\|\frac{1}{\lambda n}s(\chi_{(t)} - \alpha_t)\right\|_X^2 + 2\left(\frac{1}{\lambda n}s(-\alpha_t)\right)^T Xw(\alpha_t)\right) \\
&\leq \frac{s}{n}\sum_{i=1}^{n}\left(-\ell_i\left(\langle w(\alpha_t), x_i\rangle\right) - \ell_i^*(-\alpha_t^{(i)}) - \alpha_t^{(i)}\langle w(\alpha_t), x_i\rangle\right) + \frac{\lambda}{2}\beta_\tau\left\|\frac{1}{\lambda n}s(\chi_{(t)} - \alpha_t)\right\|_X^2 \\
&\overset{(5.27)}{=} -s\mathcal{G}(\alpha_t) + \frac{1}{2\lambda}\left(\frac{s}{n}\right)^2\left(\beta_\tau\left\|(\chi_t - \alpha_t)\right\|_X^2\right).
\end{aligned}$$

Multiplying both sides of the resulting inequality by $\frac{-b}{n}$ we obtain (5.28). To get the bound on G_t, recall that $\ell(\cdot)$ is L-Lipschitz continuous, hence $-L \leq \chi_i^{(t)} \leq L$. Furthermore, α_t is dual feasible, hence $\ell_i^*(-\alpha_t^{(i)}) < \infty$ and so $(-\alpha_t^{(i)})$ is a (sub)derivative of ℓ_i and so we also have $-L \leq \alpha_t^{(i)} \leq L$ and for each i, and $(\chi_t^{(i)} - \alpha_t^{(i)})^2 \leq 4L$. For the hinge loss we have $0 \leq \chi_t^{(i)}, \alpha_t^{(i)} \leq 1$, and so $(\chi_t^{(i)} - \alpha_t^{(i)})^2 \leq 1$. □

We are now ready to prove the theorem.

Proof of Theorem 24. We will bound the change in the dual sub-optimality $\epsilon_t^D \overset{\text{def}}{=} D(\alpha^*) - D(\alpha_t)$:

$$\begin{aligned}
\mathbf{E}_{S_t}[\epsilon_{t+1}^D] = \mathbf{E}[D(\alpha_t) - D(\alpha_{t+1}) + \epsilon_t^D] &\overset{\text{(Lemma 19)}}{\leq} -\tau\left(\frac{s}{n}\mathcal{G}(\alpha_t) - \left(\frac{s}{n}\right)^2\frac{\beta_\tau}{2\lambda}G\right) + \epsilon_t^D \\
&\overset{\epsilon_t^D \leq \mathcal{G}(\alpha_t)}{\leq} -\tau\frac{s}{n}\epsilon_t^D + \tau\left(\frac{s}{n}\right)^2\frac{\beta_\tau}{2\lambda}G + \epsilon_t^D = \left(1 - \tau\frac{s}{n}\right)\epsilon_t^D + \tau\left(\frac{s}{n}\right)^2\frac{\beta_\tau}{2\lambda}G. \quad (5.30)
\end{aligned}$$

Unrolling this recurrence, we have:

$$\mathbf{E}[\epsilon_t^D] \leq \left(1 - \tau\frac{s}{n}\right)^t\epsilon_0^D + \tau\left(\frac{s}{n}\right)^2\frac{\beta_\tau}{2\lambda}G\sum_{i=0}^{t-1}(1 - \tau\frac{s}{n})^i \leq \left(1 - \tau\frac{s}{n}\right)^t\epsilon_0^D + \left(\frac{s}{n}\right)\frac{\beta_\tau G}{2\lambda}.$$

Setting $s = 1$ and

$$t_0 := \left[\left\lceil \frac{n}{\tau} \log(2\lambda n \epsilon_0^D / (G\beta_\tau)) \right\rceil\right]_+ \tag{5.31}$$

yields:

$$\mathbf{E}[\epsilon_{t_0}^D] \leq \left(1 - \frac{\tau}{n}\right)^{t_0} \epsilon_0^D + \frac{s}{n}\frac{\beta_\tau G}{2\lambda} \leq \frac{G\beta_\tau}{2\lambda n \epsilon_D}\epsilon^D + \frac{1}{n}\frac{\beta_\tau G}{2\lambda} = \frac{\beta_\tau G}{\lambda n}. \tag{5.32}$$

Following the proof of [64] we will now show by induction that

$$\forall t \geq t_0 : \mathbf{E}[\epsilon_t^D] \leq \frac{2\beta_\tau G}{\lambda(2n + \tau(t - t_0))}. \tag{5.33}$$

Clearly, (5.32) implies that (5.33) holds for $t = t_0$. Now, if it holds for some $t \geq t_0$, we show that it also holds for $t + 1$. Using $s = \frac{2n}{2n + \tau(t - t_0)}$ in (5.30) we have:

$$\begin{aligned}
\mathbf{E}[\epsilon_{t+1}^D] &\overset{(5.30)}{\leq} \left(1 - \tau\frac{s}{n}\right)\mathbf{E}[\epsilon_t^D] + \tau\left(\frac{s}{n}\right)^2 \frac{\beta_\tau}{2\lambda}G \overset{(5.33)}{\leq} \left(1 - \tau\frac{s}{n}\right)\frac{2\beta_\tau G}{\lambda(2n + \tau(t - t_0))} + \tau\left(\frac{s}{n}\right)^2\frac{\beta_\tau}{2\lambda}G \\
&= \left(1 - \tau\frac{2}{2n + \tau(t - t_0)}\right)\frac{2\beta_\tau G}{\lambda(2n + \tau(t - t_0))} + \tau\left(\frac{2}{2n + \tau(t - t_0)}\right)^2\frac{\beta_\tau}{2\lambda}G \\
&= \frac{2G\beta_\tau}{\lambda(2n + \tau(t - t_0) + \tau)}\frac{(2n + \tau(t - t_0) + \tau)(2n + \tau(t - t_0) - \tau)}{(2n + \tau(t - t_0))^2} \leq \frac{2G\beta_\tau}{\lambda(2n + \tau(t - t_0) + \tau)},
\end{aligned} \tag{5.34}$$

where in the last inequality we used the arithmetic-geometric mean inequality. This establishes (5.33).

Now, for the average $\bar{\alpha}$ defined in (5.21) we have:

$$\mathbf{E}[\mathcal{G}(\bar{\alpha})] = \mathbf{E}\left[\mathcal{G}\left(\sum_{t=T_0}^{T-1}\frac{1}{T - T_0}\alpha_t\right)\right] \leq \frac{1}{T - T_0}\mathbf{E}\left[\sum_{t=T_0}^{T-1}\mathcal{G}(\alpha_t)\right] \tag{5.35}$$

Applying Lemma 19 with $s = \frac{n}{\tau(T - T_0)}$:

$$\begin{aligned}
&\leq \frac{nb(T - T_0)}{n\tau}\frac{1}{T - T_0}\left(\mathbf{E}[D(\alpha_T)] - \mathbf{E}[D(\alpha_{T_0})]\right) + \frac{G\beta_\tau n}{2n\tau(T - T_0)\lambda} \\
&\leq (D(\alpha^*) - \mathbf{E}[D(\alpha_{T_0})]) + \frac{G\beta_\tau}{2\tau(T - T_0)\lambda} \\
&\overset{(5.33)}{\leq} \left(\frac{2\beta_\tau G}{\lambda(2n + \tau(T_0 - t_0))}\right) + \frac{G\beta_\tau}{2\tau(T - T_0)\lambda}
\end{aligned}$$

and if $T \geq \lceil \frac{n}{b} \rceil + T_0$ and $T_0 \geq t_0$:

$$\leq \frac{\beta_\tau G}{\tau\lambda}\left(\frac{2}{2\frac{n}{\tau}+(T_0-t_0)}+\frac{1}{2(T-T_0)}\right). \tag{5.36}$$

Now, we can ensure the above is at most ϵ if we require:

$$T_0 - t_0 \geq \frac{\beta_\tau}{\tau}\left(\frac{4G}{\lambda\epsilon} - 2\frac{n}{\beta_\tau}\right), \tag{5.37}$$

$$T - T_0 \geq \frac{\beta_\tau}{\tau}\frac{G}{\lambda\epsilon_{\mathcal{G}}}. \tag{5.38}$$

Combining the requirements (5.31), (5.37) and (5.38) with $T \geq \lceil \frac{n}{\tau} \rceil + T_0$ and $T_0 \geq t_0$, and recalling that for the hinge loss $G = 1$ and with $\alpha_0 = \mathbf{0}$ we have $\epsilon_0^D = D(\alpha^*) - D(\mathbf{0}) \leq 1 - 0 = 1$ gives the requirements in Theorem 24. □

Bibliography

[1] Alekh Agarwal and John Duchi. Distributed delayed stochastic optimization. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.

[2] Dimitri P. Bertsekas. *Nonlinear Programming.* Athena Scientific, 2nd edition, September 1999.

[3] Yatao Bian, Xiong Li, and Yuncai Liu. Parallel coordinate descent newton for large-scale l1-regularized minimization. *arXiv1306:4080v1*, June 2013.

[4] Mathieu Blondel, Kazuhiro Seki, and Kuniaki Uehara. Block coordinate descent algorithms for large-scale sparse multiclass classification. *Machine Learning*, pages 1–22, 2013.

[5] Joseph K. Bradley, Aapo Kyrola, Danny Bickson, and Carlos Guestrin. Parallel coordinate descent for L1-regularized loss minimization. In *28th International Conference on Machine Learning (ICML)*, 2011.

[6] Adrian A. Canutescu and Roland L. Dunbrack. Cyclic coordinate descent: A robotics algorithm for protein loop closure. *Protein Science*, 12:963–972, 2003.

[7] Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. Coordinate descent method for large-scale l2-loss linear support vector machines. *Journal of Machine Learning Research*, 9:1369–1398, 2008.

[8] Andrew Cotter, Ohad Shamir, Nathan Srebro, and Karthik Sridharan. Better mini-batch algorithms via accelerated gradient methods. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.

[9] Cong D Dang and Guanghui Lan. Stochastic block mirror descent methods for nonsmooth and stochastic optimization. *arXiv:1309.2249*, 2013.

[10] John Langford Daniel Hsu, Nikos Karampatziakis and Alex J. Smola. Parallel online learning. *arXiv:1103.4204*, 2011.

[11] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13:165–202, 2012.

[12] Inderjit Dhillon, Pradeep Ravikumar, and Ambuj Tewari. Nearest neighbor based greedy coordinate descent. In *Advances in Neural Information Processing Systems (NIPS)*, volume 24, pages 2160–2168, 2011.

[13] John Duchi, Peter L. Bartlett, and Martin J. Wainwright. Randomized smoothing for (parallel) stochastic optimization. In *International Conference on Machine Learning (ICML)*, 2012.

[14] John Duchi, Peter L. Bartlett, and Martin J. Wainwright. Randomized smoothing for stochastic optimization. *SIAM Journal on Optimization*, 22(2):674–701, June 2012.

[15] Olivier Fercoq. Parallel coordinate descent for the AdaBoost problem. In *International Conference on Machine Learning and Applications (ICMLA)*, 2013.

[16] Olivier Fercoq and Peter Richtárik. Accelerated, parallel and proximal coordinate descent. *arXiv:1312.5799*, 2013.

[17] Olivier Fercoq and Peter Richtárik. Smooth minimization of nonsmooth functions with parallel coordinate descent methods. *arXiv:1309.5885*, 2013.

[18] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. A note on the group lasso and a sparse group lasso. *arXiv preprint arXiv:1001.0736*, 2010.

[19] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S Sathiya Keerthi, and Sellamanickam Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *International Conference on Machine Learning (ICML)*, pages 408–415, 2008.

[20] Jakub Konečný and Peter Richtárik. Semi-stochastic gradient descent methods. *arXiv:1312.1666*, 2013.

[21] Simon Lacoste-Julien, Martin Jaggi, Mark Schmidt, and Patrick Pletcher. Block-coordinate frank-wolfe optimization for structural svms. In *International Conference on Machine Learning (ICML)*, 2013.

[22] Yin Tat Lee and Aaron Sidford. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. *arXiv:1305.1922*, 2013.

[23] Dennis Leventhal and Adrian S. Lewis. Randomized methods for linear constraints: Convergence rates and conditioning. *Mathematics of Operations Research*, 35(3):641–654, 2010.

[24] Adrian S. Lewis and Stephen J. Wright. A proximal method for composite minimization. *arXiv:0812.0423*, 2008.

[25] Yingying Li and Stanley Osher. Coordinate descent optimization for ℓ_1 minimization with application to compressed sensing; a greedy algorithm. *Inverse Problems and Imaging*, 3(3):487–503, 2009.

[26] Yingying Li and Stanley Osher. Coordinate descent optimization for l_1 minimization with application to compressed sensing; a greedy algorithm. *Inverse Probl. Imaging*, 3:487–503, August 2009.

[27] Libsvm. *Datasets.* http://www.csie.ntu.edu.tw/~cjlin/ libsvmtools/datasets/binary.html.

[28] Ji Liu, Stephen J Wright, Christopher Ré, and Victor Bittorf. An asynchronous parallel stochastic coordinate descent algorithm. *arXiv:1311.1873*, 2013.

[29] Zhaosong Lu and Lin Xiao. On the complexity analysis of randomized block-coordinate descent methods. *arXiv:1305.4723*, 2013.

[30] Zhaosong Lu and Lin Xiao. Randomized block coordinate non-monotone gradient methods for a class of nonlinear programming. *arXiv:1306.5918*, 2013.

[31] Zhi-Quan Luo and Paul Tseng. A coordinate gradient descent method for nonsmooth separable minimization. *Journal of optimization theory and applications*, 72(1), January 2002.

[32] Lukas Meier, Sara Van De Geer, and Peter Buhlmann. The group lasso for logistic regression. *Journal of the Royal Statistical Society B*, 70:53–71, 2008.

[33] Indraneel Mukherjee, Kevin Canini, Rafael Frongillo, and Yoram Singer. Parallel boosting with momentum. In *European Conference on Machine Learning (ECML)*, 2013.

[34] Ion Necoara. A random coordinate descent method for large-scale resource allocation problems. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pages 4474–4479. IEEE, 2012.

[35] Ion Necoara. Suboptimal distributed MPC based on a block-coordinate descent method with feasibility and stability guarantees. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pages 4480–4485. IEEE, 2012.

[36] Ion Necoara. Random coordinate descent algorithms for multi-agent convex optimization over networks. 2013.

[37] Ion Necoara and Dragos Clipici. A computationally efficient parallel coordinate descent algorithm for MPC: Implementation on a PLC. In *Control Conference (ECC), 2013 European*, pages 3596–3601. IEEE, 2013.

[38] Ion Necoara and Dragos Clipici. Efficient parallel coordinate descent algorithm for convex optimization problems with separable constraints: application to distributed MPC. *Journal of Process Control*, 23:243–253, 2013.

[39] Ion Necoara, Yurii Nesterov, and Francois Glineur. Efficiency of randomized coordinate descent methods on optimization problems with linearly coupled constraints. Technical report, Politehnica University of Bucharest, 2012.

[40] Ion Necoara and Andrei Patrascu. A random coordinate descent algorithm for optimization problems with composite objective function and linear coupled constraints. *Computational Optimization and Applications*, pages 1–31, 2013.

[41] Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Springer Netherlands, 1st edition, 2004.

[42] Yurii Nesterov. Gradient methods for minimizing composite objective function. CORE Discussion Papers 2007/076, Universite catholique de Louvain, Center for Operations Research and Econometrics (CORE), September 2007.

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

[44] Yurii Nesterov. Subgradient methods for huge-scale optimization problems. CORE Discussion Papers 2012/002, Universit catholique de Louvain, Center for Operations Research and Econometrics (CORE), 2012.

[45] Andrei Patrascu and Ion Necoara. Efficient random coordinate descent algorithms for large-scale structured nonconvex optimization. *arXiv preprint arXiv:1305.4027*, 2013.

[46] Andrei Patrascu and Ion Necoara. A random coordinate descent algorithm for large-scale sparse nonconvex optimization. In *Control Conference (ECC), 2013 European*, pages 2789–2794. IEEE, 2013.

[47] Zhimin Peng, Ming Yan, and Wotao Yin. Parallel and distributed sparse optimization. Technical report, 2013.

[48] Zhiwei Qin, Katya Scheinberg, and Donald Goldfarb. Efficient block-coordinate descent algorithms for the group lasso. *Mathematical Programming Computation*, 5(2):143–169, 2013.

[49] Alexander Rakhlin, Ohad Shamir, and Karthik Sridharan. Making gradient descent optimal for strongly convex stochastic optimization. In *International Conference on Machine Learning (ICML)*, 2012.

[50] Benjamin Recht, Christopher Ré, Stephen Wright, and Feng Niu. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems (NIPS)*, pages 693–701. 2011.

[51] Peter Richtárik and Martin Takáč. Efficiency of randomized coordinate descent methods on minimization problems with a composite objective function. *4th Workshop on Signal Processing with Adaptive Sparse Structured, Representations*, 2011.

[52] Peter Richtárik and Martin Takáč. Efficient serial and parallel coordinate descent methods for huge-scale truss topology design. In Diethard Klatte, Hans-Jakob Lüthi, and Karl Schmedders, editors, *Operations Research Proceedings 2011*, pages 27–32. Springer Berlin Heidelberg, 2012.

[53] Peter Richtárik and Martin Takáč. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming, Series A*, pages 1–38, 2012.

[54] Peter Richtárik and Martin Takáč. Parallel coordinate descent methods for big data optimization. *submitted to Mathematical Programming, Series A, arXiv:1212.0873*, 2012.

[55] Peter Richtárik and Martin Takáč. Distributed coordinate descent method for learning with big data. *arXiv:1310.2059*, 2013.

[56] Peter Richtárik and Martin Takáč. On optimal probabilities on stochastic coordinate descent methods. *arXiv:1310.3438*, 2013.

[57] Andrzej Ruszczynski. On convergence of an augmented Lagrangian decomposition method for sparse convex optimization. *Mathematics of Operations Research*, 20(3):634–656, 1995.

[58] Ankan Saha and Ambuj Tewari. On the nonasymptotic convergence of cyclic coordinate descent methods. *SIAM Journal on Optimization*, 23(1):576–601, 2013.

[59] Chad Scherrer, Ambuj Tewari, Mahantesh Halappanavar, and David J Haglin. Feature clustering for accelerating parallel coordinate descent. In *Advances in Neural Information Processing Systems (NIPS)*, pages 28–36, 2012.

[60] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical Programming: Series A and B- Special Issue on Optimization and Machine Learning*, pages 3–30, 2011.

[61] Shai Shalev-Shwartz and Ambuj Tewari. Stochastic methods for ℓ_1-regularized loss minimization. *Journal of Machine Learning Research*, 12:1865–1892, 2011.

[62] Shai Shalev-Shwartz and Tong Zhang. Accelerated mini-batch stochastic dual coordinate ascent. *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[63] Shai Shalev-Shwartz and Tong Zhang. Proximal stochastic dual coordinate ascent. *arXiv:1211:2717*, 2012.

[64] Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14:567–599, 2013.

[65] Thomas Strohmer and Roman Vershynin. A randomized kaczmarz algorithm with exponential convergence. *Journal of Fourier Analysis and Applications*, 15:262–278, 2009.

[66] Martin Takáč, Avleen Singh Bijral, Peter Richtárik, and Nathan Srebro. Mini-batch primal and dual methods for SVMs. *International Conference on Machine Learning (ICML)*, 2013.

[67] Martin Takáč, Peter Richtárik, and Nathan Srebro. Primal-dual parallel stochastic coordinate descent for optimization in machine learning. Manuscript, 2014.

[68] Rachael Tappenden, Peter Richtárik, and Burak Büke. Separable approximations and decomposition methods for the augmented Lagrangian. *arXiv:1308.6774*, 2013.

[69] Rachael Tappenden, Peter Richtárik, and Jacek Gondzio. Inexact coordinate descent: complexity and preconditioning. *arXiv:1304.5530*, 2013.

[70] Rachael Tappenden, Martin Takáč, and Peter Richtárik. Improved complexity analysis of randomized parallel coordinate descent methods. Manuscript, 2014.

[71] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society B*, 58:268–288, 1996.

[72] Paul Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of Optimization Theory and Applications*, 109:475–494, June 2001.

[73] Paul Tseng and Sangwoon Yun. Block-coordinate gradient descent method for linearly constrained nonsmooth separable optimization. *Journal of Optimization Theory and Applications*, 140:513–535, 2009.

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

[75] Zaiwen Wen, Donald Goldfarb, and Katya Scheinberg. Block coordinate descent methods for semidefinite programming. In Miguel F. Anjos and Jean B. Lasserre, editors, *Handbook on Semidefinite, Cone and Polynomial Optimization: Theory, Algorithms, Software and Applications*. Springer, forthcoming.

[76] Stephen J. Wright. Accelerated block-coordinate relaxation for regularized optimization. *SIAM Journal on Optimization*, 22(1):159–186, February 2012.

[77] Stephen J. Wright, Robert D. Nowak, and Mário A. T. Figueiredo. Sparse reconstruction by separable approximation. *Trans. Sig. Proc.*, 57:2479–2493, July 2009.

[78] Tong Tong Wu and Kenneth Lange. Coordinate descent algorithms for lasso penalized regression. *The Annals of Applied Statistics*, 2(1):224–244, 2008.

[79] Jinchao Xu. Iterative methods by space decomposition and subspace correction. *SIAM Review*, 34(4):581–613, 1992.

[80] Hsiang-Fu Yu, Cho-Jui Hsieh, Si Si, and Inderjit Dhillon. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In *IEEE 12th International Conference on Data Mining*, pages 765–774, 2012.

[81] Guo-Xun Yuan, Chia-Hua Ho, and Chih-Jen Lin. Recent advances of large-scale linear classification. *Proceedings of the IEEE*, 100(9):2584–2603, 2012.

[82] Guo-Xun Yuan and Chih-Jen Lin. A comparison of optimization methods and software for large-scale l1-regularized linear classification. *Journal of Machine Learning Research*, 11(1):3183–3234, 2010.

[83] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society B*, 68:49–67, 2006.

[84] Sangwoon Yun and Kim-Chuan Toh. A coordinate gradient descent method for l1-regularized convex minimization. *Computational Optimization and Applications*, 48:273–307, 2011.

[85] Michael Zargham, Alejandro Ribeiro, Asuman Ozdaglar, and Ali Jadbabaie. Accelerated dual descent for network optimization. In *American Control Conference (ACC), 2011*, pages 2663–2668. IEEE, 2011.

[86] Tong Zhang. Solving large scale linear prediction using stochastic gradient descent algorithms. In *International Conference on Machine Learning (ICML)*, 2004.

[87] Hui Zhou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society B*, 67:301–320, 2005.

A

Notation Glossary

A.1 Notation Introduced in Chapter 1

	Optimization problem	
N	dimension of the optimization variable	(1.1)
x, h	vectors in $\mathbb{R}^N$	
F	$F = f + \Psi$ (loss / objective function)	(1.1)
F^*	optimal value, we assume $F^* > -\infty$	
f	smooth convex function ($f : \mathbb{R}^N \to \mathbb{R}$)	(1.1)
Ψ	convex block separable function ($\Psi : \mathbb{R}^N \to \mathbb{R} \cup \{+\infty\}$)	(1.1)
$\mathcal{R}_W(x)$	level set radius	(1.18)
	Block structure	
n	number of blocks	
$[n]$	$[n] = \{1, 2, \dots, n\}$ (the set of blocks)	Sec 1.5.1
N_i	dimension of block i ($N_1 + \cdots + N_n = N$)	Sec 1.5.1
U_i	an $N_i \times N$ column submatrix of the $N \times N$ identity matrix	Prop 1
$x^{(i)}$	$x^{(i)} = U_i^T x \in \mathbb{R}^{N_i}$ (block i of vector x)	Prop 1
$\nabla_i f(x)$	$\nabla_i f(x) = U_i^T \nabla f(x)$ (block gradient of f associated with block i)	(1.6)
L_i	block Lipschitz constant of the gradient of f	(1.6)
L	$L = (L_1, \dots, L_n)^T \in \mathbb{R}^n$ (vector of block Lipschitz constants)	
w	$w = (w_1, \dots, w_n)^T \in \mathbb{R}^n$ (vector of positive weights)	
W	$W = \mathrm{diag}(w) = \mathrm{diag}(w_1, \dots, w_n) \in \mathbb{R}^{n \times n}$	
$\mathit{support}(h)$	$\mathit{support}(h) = \{i \in [n] \ : \ x^{(i)} \neq 0\}$ (set of nonzero blocks of x)	
B_i	an $N_i \times N_i$ positive definite matrix	
$\lVert \cdot \rVert_{(i)}$	$\lVert x^{(i)} \rVert_{(i)} = \langle B_i x^{(i)}, x^{(i)} \rangle^{1/2}$ (norm associated with block of i)	
$\lVert x \rVert_w$	$\lVert x \rVert_w = (\sum_{i=1}^n w_i \lVert x^{(i)} \rVert_{(i)}^2)^{1/2}$ (weighted norm associated with x)	(1.5)
Ψ_i	i-th componet of $\Psi = \Psi_1 + \cdots + \Psi_n$	(1.9)
$\mu_\Psi(W)$	strong convexity constant of Ψ with respect to the norm $\lVert \cdot \rVert_w$	(1.10)
$\mu_f(W)$	strong convexity constant of f with respect to the norm $\lVert \cdot \rVert_w$	(1.10)
$w \odot z$	Hadamard product, $w \odot z = (w_1 z_1, \dots, w_n z_n)^T$	
u^{-1}	$u^{-1} \overset{\text{def}}{=} (1/u_1, \dots, 1/u_n)^T$	
$\langle x, y \rangle_w$	$\langle x, y \rangle_w \overset{\text{def}}{=} \sum_{i=1}^n w_i \langle x^{(i)}, y^{(i)} \rangle$	(1.4)
S, J	subsets of $\{1, 2, \dots, n\}$	
$x_{[S]}$	vector in $\mathbb{R}^N$ formed from x by zeroing out blocks $x^{(i)}$ for $i \notin S$	(1.19),(1.20)

A.2 Notation Introduced in Chapter 2

	Algorithm	
$V_i(x,t^{(i)})$	$V_i(x,t^{(i)}) \overset{\text{def}}{=} \langle \nabla_i f(x), t^{(i)}\rangle + \frac{L_i}{2}\|t^{(i)}\|_{(i)}^2 + \Psi_i(x^{(i)}+t^{(i)})$	(2.2)
$H(x,t)$	$H(x,t) = f(x) + \sum_{i=1}^n V_i(x,t^{(i)}) = f(x) + \langle \nabla f(x), t\rangle + \frac{1}{2}\|t\|_L^2 + \Psi(x+t)$	(2.5)
$T(x)$	$T(x) = \arg\min_{t\in\mathbb{R}^N} H(x,t)$	(2.6)
$T^{(i)}$	$(T(x))^{(i)} = \arg\min_{t^{(i)}\in\mathbb{R}^{N_i}} H(x, U_i t^{(i)})$	
$s^\#$	$s^\# \in \arg\min_t \left\{u(s) \overset{\text{def}}{=} -\langle s,t\rangle + \frac{1}{2}\|t\|^2\right\}$	(2.25)
	Other	
a_i	i-th column of matrix A	Sec 2.7

A.3 Notation Introduced in Chapter 3

	Block samplings	
ω	degree of partial separability of f	(3.1),(3.2)
$\hat{S}, S_k$	block samplings (random subsets of $\{1,2,\dots,n\}$)	
τ	# of blocks updated in 1 iteration (when $\mathbf{Prob}(\lvert\hat{S}\rvert = \tau) = 1$)	
$\mathbf{E}[\lvert\hat{S}\rvert]$	average # of blocks updated in 1 iteration (when $\mathbf{Var}[\lvert\hat{S}\rvert] > 0$)	
$p(S)$	$p(S) = \mathbf{Prob}(\hat{S} = S)$	(3.7)
p_i	$p_i = \mathbf{Prob}(i \in \hat{S})$	(3.8)
p	$p = (p_1,\dots,p_n)^T \in \mathbb{R}^n$	(3.8)
	Algorithm	
β	stepsize parameter depending on f and $\hat{S}$	
$H_{\beta,w}(x,h)$	$H_{\beta,w}(x,h) = f(x) + \langle \nabla f(x), h\rangle + \frac{\beta}{2}\|h\|_w^2 + \Psi(x+h)$	(3.5)
$h(x)$	$h(x) = \arg\min_{h\in\mathbb{R}^N} H_{\beta,w}(x,h)$	(3.4)
$h^{(i)}(x)$	$h^{(i)}(x) = (h(x))^{(i)} = \arg\min_{t\in\mathbb{R}^{N_i}} \langle \nabla_i f(x), t\rangle + \frac{\beta w_i}{2}\|t\|_{(i)}^2 + \Psi_i(x^{(i)}+t)$	(3.4)
x_{k+1}	$x_{k+1} = x_k + \sum_{i\in S_k} U_i h^{(i)}(x_k)$ (x_k is the kth iterate of PCDM)	
	Other	
$\mathcal{R}_W(x_0,x^*)$	$\mathcal{R}_W(x_0,x^*) \overset{\text{def}}{=} \max_x\{\|x-x^*\|_w \ : \ F(x) \le F(x_0)\} < +\infty$	(3.62)

A.4 Notation Introduced in Chapter 4

	Problem	
c	# of compute nodes (computers)	
$\mathbf{M}$	matrix in upperbound of f: $f(x+h) \le f(x) + (\nabla f(x))^T h + \frac{1}{2}h^T\mathbf{M}h$	(4.1)
$\|x\|_{\mathbf{M}}^2$	$\|x\|_{\mathbf{M}}^2 \overset{\text{def}}{=} \sum_{i=1}^N \mathbf{M}_{ii}(x^{(i)})^2$	
$\mathcal{P}_l$	$\mathcal{P}_1,\dots,\mathcal{P}_c$ is partition the N coordinates into c sets	
s	$s = \frac{N}{c}$, number of coordinates in each node	
$\mathbf{Q}$	$\mathbf{Q} \overset{\text{def}}{=} (D^{\mathbf{M}})^{-1/2}\mathbf{M}(D^{\mathbf{M}})^{-1/2}$, where $D^{\mathbf{M}} = \text{Diag}(\mathbf{M})$	(4.3)
σ'	$\sigma' \overset{\text{def}}{=} \max\{x^T\mathbf{Q}x \ : \ x\in\mathbb{R}^N, \ x^T B^{\mathbf{Q}} x \le 1\}$	(4.5)
σ	$\sigma \overset{\text{def}}{=} \max\{x^T\mathbf{Q}x \ : \ x\in\mathbb{R}^N, \ x^T x \le 1\}$	(4.6)

A.5 Notation Introduced in Chapter 5

Problem		
$\{(x_i, y_i)\}_{i=1}^n$	training data, $x_i \in \mathbb{R}^d$ and $y_i \in \pm 1$, $\max_i \|x_i\| \leq 1$	
X	matrix of training examples, $X = [x_1, \ldots, x_n]$	
$\hat{L}(w)$	(empirical) average hinge loss, $\hat{L}(w) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n \ell(y_i \langle w, x_i \rangle)$	
$\hat{L}_S(w)$	average hinge loss on examples in S, $\hat{L}_S(w) \stackrel{\text{def}}{=} \frac{1}{\tau} \sum_{i \in S} \ell(y_i \langle w, x_i \rangle)$	(5.5)
$\ell(z)$	hinge loss, $\ell(z) \stackrel{\text{def}}{=} [1 - z]_+ = \max\{0, 1 - z\}$	
$P(w)$	$P(w) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n \ell(y_i \langle w, x_i \rangle) + \frac{\lambda}{2} \|w\|^2$	(5.1)
$D(\alpha)$	$D(\alpha) \stackrel{\text{def}}{=} \frac{-\alpha^T Q \alpha}{2\lambda n^2} + \frac{1}{n} \sum_{i=1}^n \alpha^{(i)}$	(5.2)
Q	Gram matrix of the (labeled) data, $Q \in \mathbb{R}^{n \times n}$, $Q_{i,j} = y_i y_j \langle x_i, x_j \rangle$	(5.3)
σ^2	$\sigma^2 \geq \frac{1}{n} \|X\|^2 = \frac{1}{n} \|Q\|$	(5.10)
β_τ	$\beta_\tau \stackrel{\text{def}}{=} 1 + \frac{(\tau - 1)(n\sigma^2 - 1)}{n - 1}$	(5.11)
$w(\alpha)$	associate primal solution $w(\alpha) \stackrel{\text{def}}{=} \frac{1}{\lambda n} \sum_{i=1}^n \alpha^{(i)} y_i x_i$	(5.4)