Annals of Mathematical Sciences and Applications
Volume 0, Number 0, 1–54, 0000

Coordinate Friendly Structures, Algorithms and Applications *

Zhimin Peng, Tianyu Wu, Yangyang Xu, Ming Yan, and Wotao Yin

This paper focuses on the *coordinate update method*, which is useful for solving large-sized problems involving linear and nonlinear mappings, and smooth and nonsmooth functions. It decomposes a problem into simple subproblems, where each subproblem updates one, or a small block of, variables. The coordinate update method sits at a high level of abstraction and includes many special cases such as the Jacobi, Gauss-Seidel, alternated projection, as well as *coordinate descent* methods. They have found greatly many applications throughout computational sciences.

In this paper, we abstract many problems to the fixed-point problem $x = \mathcal{T}x$ and study the favorable structures in operator $\mathcal{T}$ that enable highly efficient coordinate updates: $x_i^{k+1} = (\mathcal{T}x^k)_i$. Such updates can be carried out in the sequential, parallel, and async-parallel fashions. This study leads to new coordinate update algorithms for a variety of problems in machine learning, image processing, as well as sub-areas of optimization. The obtained algorithms are scalable to very large instances through parallel and even asynchronous computing. We present numerical examples to illustrate how effective these algorithms are.

Keywords and phrases: coordinate update, fixed-point, operator splitting, parallel, asynchronous.

*This work is supported by NSF Grants DMS-1317602 and ECCS-1462398.

1. Introduction

This paper studies the *coordinate update method*, which reduces a large problem to smaller subproblems and, thus, is useful for solving large-sized problems. This method handles both linear and nonlinear maps, smooth and nonsmooth functions, and convex and nonconvex problems. Coordinate update algorithms generalize the coordinate descent algorithm by relaxing the form of an update from coordinate-wise minimization to other forms. The most common examples of these algorithms are the Jacobian and Gauss-Seidel algorithms for solving linear equations. Coordinate update algorithms are also commonly found for solving differential equations (e.g., *domain decomposition*) and optimization problems (e.g., *coordinate descent*).

After coordinate update algorithms are initially introduced in each topic area, their developments slowed down until recently, when modern applications in signal and image processing, statistical and machine learning, and data-driven tasks routinely involve a large amount of data; consequently, numerical methods of *small footprints* become increasingly popular. Since coordinate update algorithms decompose a large problem into much smaller subproblems, they tend to have low complexities and low memory requirements at each step. In addition, their implementations tend to be simpler and more amenable to taking advantages of existing numerical packages.

The coordinate update method generates simple subproblems by fixing all but one variable, or a small block of variables. The updating variable can be selected in the *cyclic*, *random*, or *greedy* orders, making the method flexible and adaptive to specific problems. The form of subproblem varies depending on both the subproblem structure and the tradeoff between exactness and complexity. Different subproblems can be solved either sequentially in a single thread or concurrently in multiple threads, or even in an asynchronous parallel fashion. Therefore, coordinate update algorithms give rise to powerful numerical algorithms for large-scale problems.

Regardless its update order, a coordinate update algorithm is *computationally worthy* only if updating each coordinate, or each small block of coordinates, is much cheaper than updating all the coordinates together. When this assumption fails to hold, coordinate update is at a disadvantage to the full update (to all the coordinates). For example, given a C^2 function $f : \mathbb{R}^n \to \mathbb{R}$, consider the Newton update $x^{k+1} \leftarrow x^k - \big(\nabla^2 f(x^k)\big)^{-1} \nabla f(x^k)$. Since updating only a single x_i (keeping other components fixed), in general, still requires forming $\nabla^2 f(x)$ (taking $O(n^2)$ operations) and factorizing it (taking $O(n^3)$ operations), there is little save in computation compared

to updating all the components of x; hence, the Netwon's method is generally not amenable to coordinate update. Therefore, identifying the favorable structures is the key to develop coordinate update algorithms.

The existing coordinate-update literature mostly focuses on introducing new algorithms, analyzing their convergence, and applying them to specific (classes of) problems with skillful improvements. §1.1 below will review the literature. This paper, however, has a different focus: the components of an efficient coordinate-update algorithm and their structures. We do not limit our discussion to specific update orders, update forms, or classes of applications. We also largely ignore convergence guarantees though they are important. Instead, the developed coordinate update algorithms in this paper solve the abstract fixed-point problem

$$x = \mathcal{T}x, \tag{1}$$

where $\mathcal{T} : \mathbb{H} \to \mathbb{H}$ is an operator and $\mathbb{H}$ is a Hilbert space. For a problem defined on $\mathbb{H}$ and a given iterative algorithm for the problem, we can let $\mathcal{T}$ abstract each iteration of the algorithm:

$$x^{k+1} = \mathcal{T}x^k. \tag{2}$$

The limit of the sequence $\{x^k\}$ is a fixed point of $\mathcal{T}$ and also a solution to the underlying problem. The update scheme (2) generalizes iterative methods for solving linear equations, gradient descent, proximal gradient method, operator splitting methods, and many other methods.

We study the structures of $\mathcal{T}$ that make the following coordinate update algorithm *computationally worthy*

$$x_i^{k+1} = x_i^k - \eta_k(x^k - \mathcal{T}x^k)_i, \tag{3}$$

where x_i is a coordinate of x and η_k is a step size. We define the so-called **Coordinate Friendly (CF)** operator and provide examples.

We construct coordinate update algorithms based on CF operators for a variety of problems including, but not limited to, linear programming, second-order cone programming, variational image processing, support vector machine, empirical risk minimization, portfolio optimization, and nonnegative matrix factorization. For each problem, we present an algorithm in the form of (2) so that its coordinate update (3) is efficient. A final coordinate update algorithm can be obtained by plugging (3) into an algorithmic framework reviewed in §1.1.

The algorithms developed in this paper are generalizations to many algorithms that are recently developed primarily for empirical risk minimization problems in machine learning. The generalization empowers us to solve more problems, such as those with multiple functions and constraints, as well as saddle-point formulations and variational inequalities.

In addition, the developed coordinate update algorithms can run on multiple agents in a parallel and even asynchronous fashion. This gives rise to parallel and asynchronous extensions to existing algorithms including the Alternating Direction Method of Multipliers (ADMM), primal-dual splitting algorithms, and others.

The paper is organized as follows. §1.1 reviews the existing frameworks of coordinate update algorithms. §2 discusses the underlying structures of CF operators. §3 reviews operator splitting methods and presents composite CF operators. §4 reviews primal-dual splitting methods. Based on the results of previous sections, §5 introduces coordinate update algorithms for various applications, some of which have been tested with numerical results presented in §6.

Throughout this paper, all functions f, g, h are proper closed convex and can take the extended value ∞, and all sets X, Y, Z are nonempty closed convex. The indicator function $\iota_X(x)$ returns 0 if $x \in X$, and ∞ elsewhere.

1.1. Coordinate Update Algorithmic Frameworks

This subsection reviews the *sequential* and *parallel* algorithmic frameworks for coordinate updates, as well as the relevant literature. They give rise to coordinate update algorithms once their components such as $(\mathcal{T}x^k)_i$ and $(x^k - \mathcal{T}x^k)_i$ are realized for specific problems.

1.1.1. Sequential Update. In this framework, there is a sequence of coordinate indices $i_1, i_2, \ldots$, and at each iteration $k = 1, 2, \ldots$, only the i_kth coordinate is updated:

$$\begin{cases} x_i^{k+1} = x_i^k - \eta_k (x^k - \mathcal{T}x^k)_i, & i = i_k, \\ x_i^{k+1} = x_i^k, & \text{for all } i \neq i_k. \end{cases}$$

Sequential updates have been long known for special types of problems and their corresponding operators $\mathcal{T}$, for example, the Gauss-Seidel iteration for solving linear equations, alternating projection [72, 4] for finding a point in the intersection of two sets, ADMM [31, 30] for solving monotropic programs, and Douglas-Rachford Splitting (DRS) [26] for finding a zero to the sum

of two operators. (ADMM and DRS also introduce and update additional variables.)

In optimization, *coordinate descent* algorithms, at each iteration, minimize the function $f(x_1, \ldots, x_n)$ by fixing all but one variable. Let

$$x_{-i} := (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n),$$

collect all but the ith coordinate of x. Coordinate descent updates take one of the following forms:

$$(\mathcal{T}x^k)_i = \arg\min_{x_i} f(x_i, x^k_{-i}), \tag{4a}$$

$$(\mathcal{T}x^k)_i = \arg\min_{x_i} f(x_i, x^k_{-i}) + \frac{1}{2\eta_k}\|x_i - x^k_i\|^2, \tag{4b}$$

$$(\mathcal{T}x^k)_i = \arg\min_{x_i} \langle \nabla_i f(x^k), x_i\rangle + \frac{1}{2\eta_k}\|x_i - x^k_i\|^2, \tag{4c}$$

$$(\mathcal{T}x^k)_i = \arg\min_{x_i} \langle \nabla_i f^{\text{diff}}(x^k), x_i\rangle + f_i^{\text{prox}}(x_i) + \frac{1}{2\eta_k}\|x_i - x^k_i\|^2, \tag{4d}$$

which are called *direct* update, *proximal* update, *gradient* update, and *prox-gradient* update, respectively. The last update applies to the function

$$f(x) = f^{\text{diff}}(x) + \sum_{i=1}^{n} f_i^{\text{prox}}(x_i),$$

where f^{diff} is differentiable and each f_i^{prox} is proximable (its proximal map takes $O\big(\dim(x_i)\,\text{polylog}(\dim(x_i))\big)$ operations).

Sequential-update literature. Coordinate descent algorithms date back to the 1950s [34], when the *cyclic* update-order was used. Its convergence has been established under a variety of cases, for both convex and non-convex objective functions; see [74, 82, 54, 33, 43, 67, 32, 69, 55, 9, 35, 75]. Proximal updates are studied in [32, 1] and developed into prox-gradient updates in [71, 70, 13] and mixed updates in [78].

The *stochastic* update-order appeared in [46] and then [59, 42]. Recently, [79, 77] compared the convergence speeds of cyclic and stochastic update-orders. The gradient update has been relaxed to stochastic gradient update for large-scale problems in [21, 80].

The *greedy* update-order leads to fewer total iterations but is often impractical since it requires a lot of effort to calculate scores for all the coordinates. However, there are cases where calculating the scores is easy [11,

39, 76] and the save in the total iterations significantly outweighs the effort [71, 25, 52, 61].

A simple test. Although our purpose is not to compare different update-orders, we feel necessary to make them concrete for the reader. For simplicity, we adopt the least squares problem

$$\underset{x}{\text{minimize}}\ \frac{1}{2}\|Ax-b\|^2,$$

where $A \in \mathbb{R}^{p\times m}$ and $b \in \mathbb{R}^p$ are Gaussian random, to numerically demonstrate the advantages of coordinate updates over the full update of gradient descent:

$$x^{k+1} = x^k - \eta_k A^\top (Ax^k - b).$$

The four coordinate update schemes are: cyclic, cyclic permutation, random, and greedy under the Gauss-Southwell rule. In the full update, the step size η_k is set to the theoretical upper bound $\frac{2}{\|A\|_2^2}$. For each coordinate update to x_i, the step size η_k is set to $\frac{1}{(A^\top A)_{ii}}$. All of the full and coordinate updates have the same *per-epoch* complexity, so we plot the objective errors in Figure 1. Note that the performance of coordinate update algorithms depend on many factors such as the condition number, the level of coupling among different coordinates, whether greedy selections can be efficiently made, as well as the amount of data move needed. The demonstration here is limited.

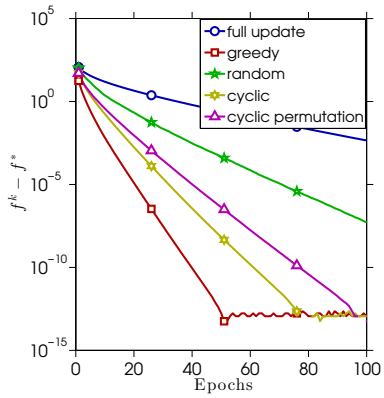
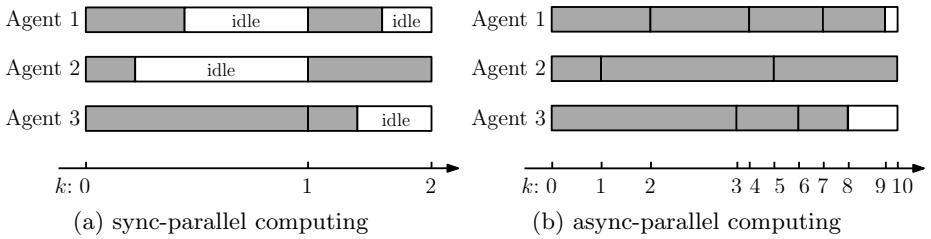


Figure 1: Gradient descent: the coordinate updates are faster than the full update since the former can take larger steps at each step.



(a) sync-parallel computing (b) async-parallel computing

Figure 2: Sync-parallel computing (left) versus async-parallel computing (right). On the left, all the agents must wait at idle (white boxes) until the slowest agent has finished.

1.1.2. Parallel Update. We discuss both synchronous (sync) and asynchronous (async) versions of parallel updates.

Sync-parallel (Jacobi) update. In this framework, there is a sequence of index sets $\mathbb{I}_1, \mathbb{I}_2, \ldots$ (which are subsets of $\{1, \ldots, n\}$), and at each iteration $k = 1, 2, \ldots$, the coordinates in $\mathbb{I}_k$ are updated in parallel by multiple agents:

$$\begin{cases} x_i^{k+1} = x_i^k - \eta_k (x^k - \mathcal{T} x^k)_i, & i \in \mathbb{I}_k, \\ x_i^{k+1} = x_i^k, & i \notin \mathbb{I}_k. \end{cases}$$

The counter k increases after all coordinate updates in $\mathbb{I}_k$ are completed. If $\mathbb{I}_k = \{1, \ldots, n\}$, then all the coordinates are updated; hence, each iteration reduces to the standard update: $x^{k+1} = x^k - \eta_k (x^k - \mathcal{T} x^k)$.

Async-parallel update. In this framework, a set of agents perform simultaneous updates that are aligned up in time. Each of them continuously applies (5) below to the variable x in the shared memory (or locally storing x and communicating with other agents):

$$\begin{cases} x_i^{k+1} = x_i^k - \eta_k \left((\mathcal{I} - \mathcal{T}) x^{k-d_k} \right)_i, & i = i_k, \\ x_i^{k+1} = x_i^k, & \text{for all } i \neq i_k. \end{cases} \tag{5}$$

Here, k increases whenever one agent completes an update, and d_k is the asynchronous delay.

At each sync-parallel iteration, synchronization must wait for the completion of the last (slowest) update. Async-parallel updates eliminate such idle time, spread out memory access and communication, and is generally more fault-tolerant. However, async-parallel is less stable and more difficult to analyze because of the asynchronous delay.

Asynchronous delay occurs if, after an agent reads x yet before it completes updating x_{i_k}, other agents also make updates to x. In (5), the agent reads x^{k-d_k} and commits the update to $x^k_{i_k}$. The delay d_k equals the number of updates by other agents during this period. Here we assume the *consistent* case, i.e., x read by one agent is in the history of $\{x^k\}$. For the *inconsistent* case, please see [51, Section 1.2] for more details.

Parallel-update literature. Async-parallel methods can be traced back to [17] for linear systems. For function minimization, [12] introduced an async-parallel gradient-projection method. Convergence rates are obtained in [66]. Recently, [14, 58] developed parallel randomized methods.

For fixed-point problems, async-parallel methods date back to [3] in 1978. In the pre-2010 methods [2, 10, 7, 27] and the review [29], each agent updates its own subset of coordinates. Convergence is established under the *P-contraction* condition and its variants [10]. Papers [7, 8] show convergence for async-parallel iterations with simultaneous reading and writing to the same set of components. Unbounded but stochastic delays are considered in [64].

Recently, random coordinate selection appeared in [19] for fixed-point problems. The works [45, 56, 41, 40, 36] introduced async-parallel stochastic methods for function minimization. For fixed-point problems, [51] introduced async-parallel stochastic methods, as well as several applications.

1.1.3. Contributions of this paper. The paper systematically discusses the CF structures found in both single and composite operators underlying many interesting applications. We introduce approaches to recognize CF operators and develop coordinate-update algorithms based on them. We provide a variety of applications to illustrate our approaches. In particular, we obtain new coordinate-update algorithms for image deblurring, portfolio optimization, second order cone programming, as well as tensor decomposition. Our analysis also provides guidance to the implementation of coordinate-update algorithms by specifying how to compute certain operators and maintain certain quantities in memory.

This paper does *not* focus on the convergence perspective of coordinate update algorithms (or coordinate descent for function minimization). In general, in fixed-point algorithms, the iterate convergence is ensured by the monotonic decrease of the distance between the iterates and the solution set, while in minimization problems, the objective value convergence is ensured by the monotonic decrease of a certain energy function. The reader is referred to the existing literature for details.

In our fixed-point setting, the operator $\mathcal{T}$ generally needs to be nonexpansive (under a certain metric). The operator splitting methods reviewed in §3 below generate nonexpansive operators $\mathcal{T}$ for many problems considered in this paper. The convergence of the resulting stochastic sequential and (async)-parallel coordinate-update algorithms, as well as their step size selections, is referred to the recent works [19, 51]. On the other hand, the structure properties of operators discussed in this paper are irrelevant to nonexpansiveness or convexity. Hence, the algorithms developed can be still applied to nonconvex problems without guarantees.

2. Coordinate Friendly Operators

2.1. Notation

For convenience, we do not distinguish *a coordinate* from *a block of coordinates* throughout this paper. A coordinate is treated as the unit of variables that are updated together each time. We assume that there are m coordinates and their spaces are $\mathbb{H}_1, \ldots, \mathbb{H}_m$. For simplicity, we assume that they are finite-dimensional real Hilbert spaces, though most results hold for general Hilbert spaces. For brevity, we also let

$$x = (x_1, \ldots, x_m) \in \mathbb{H} := \mathbb{H}_1 \times \cdots \times \mathbb{H}_m \quad \text{and} \quad x_i \in \mathbb{H}_i,\ i = 1, \ldots, m.$$

A function maps from $\mathbb{H}$ to $\mathbb{R}$, and an operator maps from $\mathbb{H}$ to $\mathbb{G}$, where the definition of $\mathbb{G}$ depends on the context. The operator $\mathcal{T}$ and those operators in the splitting methods in §3.2 map from $\mathbb{H}$ to $\mathbb{H}$.

Several definitions below use $x, x^+ \in \mathbb{H}$ that *differ by one coordinate*: for some $i \in \{1, \ldots, m\}$ and $\delta_i \in \mathbb{H}$ that is supported on $\mathbb{H}_i$,

$$x^+ = x + \delta_i. \tag{6}$$

Note that $x_j^+ = x_j$ for all $j \neq i$.

Definition 1 (number of operations)**.** *We let* $\mathfrak{M}[a \mapsto b]$ *denote* the number of basic operations *that it takes to compute the quantity* b *from the input* a.

For example, $\mathfrak{M}[x \mapsto (\mathcal{T}x)_i]$ denotes the number of operations to compute the ith component of $\mathcal{T}x$ given a general point x. We explore the possibility to compute $(\mathcal{T}x)_i$ with much fewer operations than what is needed to first compute $\mathcal{T}x$ and then take its ith component.

For a given matrix A, let $A_{i,:}$ and $A_{:,j}$ be its ith row and jth column, respectively. Let $A^\top$ be the transpose of A and $A_{i,:}^\top$ be $(A^\top)_{i,:}$, i.e., the ith row of the transpose of A.

2.2. Single Coordinate Friendly Operators

This subsection studies a few kinds of CF operators and then formally define the CF operator. We motivate the first kind through an example.

Example 1 (least squares)**.** *We consider the least squares problem*

$$\underset{x}{\text{minimize}}\, f(x) := \frac{1}{2}\|Ax-b\|^2, \tag{7}$$

where $A \in \mathbb{R}^{p\times m}$ and $b \in \mathbb{R}^p$. In this example, assume that $m = \Theta(p)$, namely, m and p are of the same order. We compare the full update of gradient descent and its coordinate update for problem (7).[1] *The full update is $x^{k+1} = \mathcal{T}x^k$ where*

$$\mathcal{T}x := x - \eta\nabla f(x) = x - \eta A^\top Ax + \eta A^\top b. \tag{8}$$

Assuming that $A^\top A$ and $A^\top b$ are made available by pre-computing, we have $\mathfrak{M}[x \mapsto \mathcal{T}x] = O(m^2)$. The coordinate update at the kth iteration performs

$$x_{i_k}^{k+1} = \mathcal{T}_{i_k}x^k = x_{i_k}^k - \eta\nabla_{i_k} f(x^k),$$

and $x_j^{k+1} = x_j^k, \forall j \neq i_k$, where i_k is some selected coordinate.

Since for all i, $\left(A^\top(Ax-b)\right)_i = (A^\top A)_{i,:}\cdot x - (A^\top b)_i$, we have $\mathfrak{M}[x \mapsto \mathcal{T}_i x] = O(m) = O(\frac{1}{m}\mathfrak{M}[x \mapsto \mathcal{T}x])$. Therefore, the coordinate gradient descent is computationally worthy.

The operator $\mathcal{T}$ in the above example is a special *Type-I CF* operator.

Definition 2 (Type-I CF)**.** *For an operator $\mathcal{T} : \mathbb{H} \to \mathbb{H}$, let $\mathfrak{M}[x \mapsto (\mathcal{T}x)_i]$ be the number of operations for computing the ith coordinate of $\mathcal{T}x$ given x and $\mathfrak{M}[x \mapsto \mathcal{T}x]$ the number of operations for computing $\mathcal{T}x$ given x.*

We say $\mathcal{T}$ is Type-I CF *(denoted as $\mathcal{F}_1$) if for any $x \in \mathbb{H}$ and $\forall\, i \in \{1,\ldots,m\}$, it holds*

$$\mathfrak{M}[x \mapsto (\mathcal{T}x)_i] = O\left(\frac{1}{m}\mathfrak{M}[x \mapsto \mathcal{T}x]\right).$$

Example 2 (least squares II)**.** *We can implement the coordinate update in Example 1 in a different manner by maintaining the result $\mathcal{T}x^k$ throughout,*

[1] Although gradient descent is seldom used to solve least squares, it often appears as a part in first-order algorithms for problems involving a least squares term.

which works when $m = \Theta(n)$ or $p \gg m$. The full update is unchanged. At each iteration, we immediately obtain $x_{i_k}^{k+1} = (\mathcal{T}x^k)_{i_k}$ but need to refresh $\mathcal{T}x^k$ to $\mathcal{T}x^{k+1}$. Since x^{k+1} and x^k only differ by just one coordinate, the refreshing only requires multiplying the i_kth column of $A^\top A$ by $x_{i_k}^{k+1} - x_{i_k}^k$, which is much cheaper than multiplying $A^\top A$ by x^{k+1}. The refreshing takes $O(m)$ operations, which is $O(\frac{1}{m}\mathfrak{M}[x \mapsto \mathcal{T}(x)])$. Therefore,

$$\mathfrak{M}\left[\{x^k, \mathcal{T}x^k, x^{k+1}\} \mapsto \mathcal{T}x^{k+1}\right] = O\left(\frac{1}{m}\mathfrak{M}\left[x^{k+1} \mapsto \mathcal{T}x^{k+1}\right]\right).$$

The operator $\mathcal{T}$ in the above example is a special *Type-II CF* operator.

Definition 3 (Type-II CF)**.** *An operator $\mathcal{T}$ is called* Type-II CF *(denoted as $\mathcal{F}_2$) if, for any x, i, δ_i, x^+ satisfying (6), the following holds*

$$\mathfrak{M}\left[\{x, \mathcal{T}x, x^+\} \mapsto \mathcal{T}x^+\right] = O\left(\frac{1}{m}\mathfrak{M}\left[x^+ \mapsto \mathcal{T}x^+\right]\right). \tag{9}$$

Sometimes, maintaining certain quantities other than $\mathcal{T}x$ can also make the coordinate update computationally worthy.

Example 3 (least squares III)**.** *For the case $p \ll m$, i.e., the system $Ax = b$ is highly underdetermined, we should avoid pre-computing $A^\top A$ because multiplying A and then $A^\top$ is cheaper. Therefore, we change the implementations of both the full and coordinate updates in Example 1. In particular, the full update*

$$x^{k+1} = \mathcal{T}x^k = x^k - \eta\nabla f(x^k) = x^k - \eta A^\top(Ax^k - b),$$

pre-multiplies x^k by A and then $A^\top$. Hence, $\mathfrak{M}\left[x^k \mapsto \mathcal{T}(x^k)\right] = O(mp)$.

We change the coordinate update to maintain the intermediate quantity Ax^k. The coordinate update computes

$$(\mathcal{T}x^k)_{i_k} = x_{i_k}^k - \eta(A^\top Ax^k - A^\top b)_{i_k},$$

by pre-multiplying Ax^k by $A_{i_k,:}^\top$ and refreshing Ax^k to Ax^{k+1} by adding $(x_{i_k}^{k+1} - x_{i_k}^k)A_{:,i_k}$ to Ax^k. Both steps take $O(p)$ operations, so

$$\mathfrak{M}\left[\{x^k, Ax^k\} \mapsto \{x^{k+1}, Ax^{k+1}\}\right] = O(p) = O\left(\frac{1}{m}\mathfrak{M}\left[x^k \mapsto \mathcal{T}x^k\right]\right).$$

Combining Type-I and Type-II CF operators with the last example, we arrive at the following definition:

Definition 4 (CF operator)**.** *We say that an operator $\mathcal{T}:\mathbb{H}\to\mathbb{H}$ is* CF *if, for any x, i, δ_i, x^+ satisfying (6), the following holds*

$$\mathfrak{M}\left[\{x,\mathcal{M}(x)\}\mapsto\{x^+,\mathcal{M}(x^+)\}\right]=O\left(\frac{1}{m}\mathfrak{M}\left[x\mapsto\mathcal{T}x\right]\right), \tag{10}$$

where $\mathcal{M}(x)$ is some quantity, possibly non-existing, that is maintained in the memory to facilitate each coordinate update and refreshed to $\mathcal{M}(x^+)$.

The left-hand side of (10) measures the cost of performing one coordinate update (including the cost of updating $\mathcal{M}(x)$ to $\mathcal{M}(x^+)$) while the right-hand side measures the average per-coordinate cost of updating all the coordinates together. When (10) holds, $\mathcal{T}$ is amenable to coordinate updates.

By definition, a Type-I CF operator $\mathcal{T}$ is CF without maintaining any quantity, i.e., $\mathcal{M}(x)=\emptyset$.

A Type-II CF operator $\mathcal{T}$ satisfies (10) with $\mathcal{M}(x)=\mathcal{T}x$, so it is also CF. Indeed, given any x and i, we can compute x^+ by immediately letting $x_i^+=(\mathcal{T}x)_i$ (at $O(1)$ cost) and keeping $x_j^+=x_j$, $\forall j\neq i$; then, by (9), we update $\mathcal{T}x$ to $\mathcal{T}x^+$ at a low cost. Formally, letting $\mathcal{M}(x)=\mathcal{T}x$,

$$\begin{aligned}
&\mathfrak{M}\left[\{x,\mathcal{M}(x)\}\mapsto\{x^+,\mathcal{M}(x^+)\}\right]\\
\leq&\mathfrak{M}\left[\{x,\mathcal{T}x\}\mapsto x^+\right]+\mathfrak{M}\left[\{x,\mathcal{T}x,x^+\}\mapsto\mathcal{T}x^+\right]\\
\overset{(9)}{=}&O(1)+O\left(\frac{1}{m}\mathfrak{M}\left[x^+\mapsto\mathcal{T}x^+\right]\right)\\
=&O\left(\frac{1}{m}\mathfrak{M}\left[x\mapsto\mathcal{T}x\right]\right).
\end{aligned}$$

In general, the set of CF operators is much larger than the union of Type-I and Type-II CF operators.

Another important subclass of CF operators are operators $\mathcal{T}:\mathbb{H}\to\mathbb{H}$ where $(\mathcal{T}x)_i$ only depends on one, or a few, entries out of $x_1,\ldots,x_m$. Based on how many input coordinates they depend on, we partition them into three subclasses.

Definition 5 (Separable operator)**.** *Consider $\mathfrak{T}:=\{\mathcal{T}:\mathbb{H}\to\mathbb{H}\}$. We have the partition $\mathfrak{T}=\mathcal{C}_1\cup\mathcal{C}_2\cup\mathcal{C}_3$, where*

- separable operators: *$\mathcal{T}\in\mathcal{C}_1$ if, for any index i, there exists $\mathcal{T}_i:\mathbb{H}_i\to\mathbb{H}_i$ such that $(\mathcal{T}x)_i=\mathcal{T}_ix_i$, that is, $(\mathcal{T}x)_i$ only depends on x_i.*

- nearly-separable operators: $\mathcal{T} \in \mathcal{C}_2$ *if, for any index* i*, there exists* $\mathcal{T}_i$ *and index set* I_i *such that* $(\mathcal{T}x)_i = \mathcal{T}_i(\{x_j\}_{j\in I_i})$ *with* $|I_i| \ll m$*, that is, each* $(\mathcal{T}x)_i$ *depends on a few coordinates of* x*.*
- non-separable operators: $\mathcal{C}_3 := \mathfrak{T} \setminus (\mathcal{C}_1 \cup \mathcal{C}_2)$*. If* $\mathcal{T} \in \mathcal{C}_3$*, there exists some* i *such that* $(\mathcal{T}x)_i$ *depends on many coordinates of* x*.*

Throughout the paper, we assume the coordinate update of a (nearly-) separable operator costs roughly the same for all coordinates. Under this assumption, separable operators are both Type-I CF and Type-II CF, and nearly-separable operators are Type-I CF.[2]

2.3. Examples of CF Operators

In this subsection, we give examples of CF operators arising in different areas including linear algebra, optimization, and machine learning.

Example 4 ((block) diagonal matrix)**.** *Consider the diagonal matrix*

$$A = \begin{bmatrix} a_{1,1} & & 0 \\ & \ddots & \\ 0 & & a_{m,m} \end{bmatrix} \in \mathbb{R}^{m\times m}.$$

Clearly $\mathcal{T} : x \mapsto Ax$ *is separable.*

Example 5 (gradient and proximal maps of a separable function)**.** *Consider a* separable *function*

$$f(x) = \sum_{i=1}^{m} f_i(x_i).$$

Then, both ∇f *and* $\mathbf{prox}_{\gamma f}$ *are separable, in particular,*

$$(\nabla f(x))_i = \nabla f_i(x_i) \quad \textit{and} \quad (\mathbf{prox}_{\gamma f}(x))_i = \mathbf{prox}_{\gamma f_i}(x_i).$$

Here, $\mathbf{prox}_{\gamma f}(x)$ *(* $\gamma > 0$*) is the* proximal operator *that we define in* (68)*.*

[2]Not all nearly-separable operators are Type-II CF. Indeed, consider a sparse matrix $A \in \mathbb{R}^{m\times m}$ whose non-zero entries are only located in the last column. Let $\mathcal{T}x = Ax$ and $x^+ = x+\delta_m$. As x^+ and x differ by the last entry, $\mathcal{T}x^+ = \mathcal{T}x+(x_m^+ - x_m)A_{:,m}$ takes m operations. Therefore, we have $\mathfrak{M}[\{x, \mathcal{T}x, x^+\} \mapsto \mathcal{T}x^+] = O(m)$. Since $\mathcal{T}x^+ = x_m^+ A_{:,m}$ takes m operations, we also have $\mathfrak{M}[x^+ \mapsto \mathcal{T}x^+] = O(m)$. Therefore, (9) is violated, and there is no benefit from maintaining $\mathcal{T}x$.

Example 6 (projection to box constraints)**.** *Consider the "box" set* $B := \{x : a_i \le x_i \le b_i,\ i = 1, \dots, m\} \subset \mathbb{R}^m$*. Then, the projection operator* $\mathbf{proj}_B$ *is separable. Indeed,*

$$\big(\mathbf{proj}_B(x)\big)_i = \max(b_i, \min(a_i, x_i)).$$

Example 7 (sparse matrices)**.** *If every row of the matrix* $A \in \mathbb{R}^{m\times m}$ *is sparse,* $\mathcal{T} : x \mapsto Ax$ *is nearly-separable.*

Examples of sparse matrices arise from various finite difference schemes for differential equations, problems defined on sparse graphs. When most pairs of a set of random variables are conditionally independent, their inverse covariance matrix is sparse.

Example 8 (sum of sparsely supported functions)**.** *Let* E *be a class of index sets and every* $e \in E$ *be a small subset of* $\{1, ..., m\}$*,* $|e| \ll m$*. In addition* $\#\{e : i \in e\} \ll \#\{e\}$ *for all* $i \in \{1, \dots, m\}$*. Let* $x_e := (x_i)_{i\in e}$*, and*

$$f(x) = \sum_{e\in E} f_e(x_e).$$

The gradient map ∇f *is nearly-separable.*

An application of this example arises in wireless communication over a graph of m *nodes. Let each* x_i *be the spectrum assignment to node* i*, each* e *be a neighborhood of nodes, and each* f_e *be a utility function. The input of* f_e *is* x_e *since the utility depends on the spectra assignments in the neighborhood.*

In machine learning, if each observation only involves with a few features, then each function of the optimization objective will depend on a small number of components of x*.*

Example 9 (square hinge loss function)**.** *Consider for* $a, x \in \mathbb{R}^m$*,*

$$f(x) := \frac{1}{2}\big(\max(0, 1 - \beta a^\top x)\big)^2,$$

which is the squared hinge loss function. Consider the operator

$$\mathcal{T}x := \nabla f(x) = -\beta \max(0, 1 - \beta a^\top x)a. \tag{11}$$

Let us maintain $\mathcal{M}(x) = a^\top x$*. For arbitrary* x *and* i*, let*

$$x_i^+ := (\mathcal{T}x)_i = -\beta \max(0, 1 - \beta a^\top x)a_i$$

and $x_j^+ := x_j, \forall j \neq i$. *Then, computing* x_i^+ *from* x *and* $a^\top x$ *takes* $O(1)$ *(as* $a^\top x$ *is maintained), and computing* $a^\top x^+$ *from* $x_i^+ - x_i$ *and* $a^\top x$ *costs* $O(1)$. *Formally, we have*

$$\begin{aligned}&\mathfrak{M}\left[\{x, a^\top x\} \mapsto \{x^+, a^\top x^+\}\right]\\ =&\mathfrak{M}\left[\{x, a^\top x\} \mapsto x^+\right] + \mathfrak{M}\left[\{a^\top x, x_i^+ - x_i\} \mapsto a^\top x^+\right]\\ =&O(1) + O(1) = O(1).\end{aligned}$$

On the other hand, $\mathfrak{M}[x \mapsto \mathcal{T}x] = O(m)$. *Therefore,* (10) *holds, and* $\mathcal{T}$ *defined in* (11) *is CF.*

3. Composite Coordinate Friendly Operators

There are many popular operator-splitting based algorithms, e.g., proximal gradient method, ADM, and primal-dual method, which reduce the original problem to simpler subproblems, each corresponding to a part of the objective or constraints. Coordinate updates can be combined with operator splitting to further simplify their subproblems and even offer better parallelism. Most operator splitting algorithms are sequential compositions of two or more operators. This section studies when their compositions are CF operators.

3.1. Combinations of Operators

We start by an example with numerous applications. It is a generalization of Example 9.

Example 10 (scalar map pre-composing affine function)**.** *Let* $a_j \in \mathbb{R}^m, b_j \in \mathbb{R}$, *and* $\phi_j : \mathbb{R} \to \mathbb{R}$ *be differentiable functions,* $j = 1, \ldots, p$. *Let*

$$f(x) = \sum_{j=1}^{p} \phi_j(a_j^\top x + b_j).$$

Assume that evaluating ϕ_j' *costs* $O(1)$ *for all* j. *Then,* ∇f *is CF. Indeed, let*

$$\mathcal{T}_1 y := A^\top y, \quad \mathcal{T}_2 y := \mathrm{Diag}(\phi_1'(y_1), \ldots, \phi_p'(y_p)), \quad \mathcal{T}_3 x := Ax + b,$$

where $A = [a_1^\top; a_2^\top; \ldots, a_p^\top] \in \mathbb{R}^{p\times m}$ *and* $b = [b_1; b_2; \ldots; b_p] \in \mathbb{R}^{p\times 1}$. *Then* $\nabla f(x) = \mathcal{T}_1 \circ \mathcal{T}_2 \circ \mathcal{T}_3 x$. *For any* x *and* $i \in \{1, \ldots, m\}$, *let* $x_i^+ = \nabla_i f(x)$ *and*

$x_j^+ = x_j, \forall j \neq i$, *and let* $\mathcal{M}(x) = \mathcal{T}_3 x$. *We can first compute* $\mathcal{T}_2 \circ \mathcal{T}_3 x$ *from* $\mathcal{T}_3 x$ *for* $O(p)$ *operations, then compute* $\nabla_i f(x)$ *and thus* x^+ *from* $\{x, \mathcal{T}_2 \circ \mathcal{T}_3 x\}$ *for* $O(p)$ *operations, and finally update the maintained* $\mathcal{T}_3 x$ *to* $\mathcal{T}_3 x^+$ *from* $\{x, x^+, \mathcal{T}_3 x\}$ *for another* $O(p)$ *operations. Formally,*

$$\begin{aligned}
&\mathfrak{M}\left[\{x, \mathcal{T}_3 x\} \mapsto \{x^+, \mathcal{T}_3 x^+\}\right] \\
=&\mathfrak{M}\left[\mathcal{T}_3 x \mapsto \mathcal{T}_2 \circ \mathcal{T}_3 x\right] + \mathfrak{M}\left[\{x, \mathcal{T}_2 \circ \mathcal{T}_3 x\} \mapsto x^+\right] + \mathfrak{M}\left[\{x, \mathcal{T}_3 x, x^+\} \mapsto \{\mathcal{T}_3 x^+\}\right] \\
=&O(p) + O(p) + O(p) = O(p).
\end{aligned}$$

Since $\mathfrak{M}[x \mapsto \nabla f(x)] = O(pm)$, *therefore* $\nabla f = \mathcal{T}_1 \circ \mathcal{T}_2 \circ \mathcal{T}_3$ *is CF.*

If $p = m$, $\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3$ *all map from* $\mathbb{R}^m$ *to* $\mathbb{R}^m$. *Then, it is easy to check that* $\mathcal{T}_1$ *is Type-I CF,* $\mathcal{T}_2$ *is separable, and* $\mathcal{T}_3$ *is Type-II CF. The last one is crucial since not maintaining* $\mathcal{T}_3 x$ *would disqualify* $\mathcal{T}$ *from CF. Indeed, to obtain* $(\mathcal{T} x)_i$, *we must multiple* $A_i^\top$ *to all the entries of* $\mathcal{T}_2 \circ \mathcal{T}_3 x$, *which in turn needs all the entries of* $\mathcal{T}_3 x$, *computing which from scratch costs* $O(pm)$.

There are some rules to preserve Type-I and Type-II CF. For example, $\mathcal{T}_1 \circ \mathcal{T}_2$ *is still Type-I CF, and* $\mathcal{T}_2 \circ \mathcal{T}_3$ *is still CF but there are counter examples where* $\mathcal{T}_2 \circ \mathcal{T}_3$ *can be neither Type-I nor Type-II CF. Such properties are important for developing efficient coordinate update algorithms for complicated problems; we will fomarlize them in the following.*

The operators $\mathcal{T}_2$ and $\mathcal{T}_3$ in the above example are prototypes of *cheap* and *easy-to-maintain* operators from $\mathbb{H}$ to $\mathbb{G}$ that arise in operator compositions.

Definition 6 (cheap operator)**.** *For a composite operator* $\mathcal{T} = \mathcal{T}_1 \circ \cdots \circ \mathcal{T}_p$, *an operator* $\mathcal{T}_i : \mathbb{H} \to \mathbb{G}$ *is cheap if* $\mathfrak{M}[x \mapsto \mathcal{T}_i x]$ *is less than or equal to the number of remaining coordinate-update operations, in order of magnitude.*

Definition 7 (easy-to-maintain operator)**.** *For a composite operator* $\mathcal{T} = \mathcal{T}_1 \circ \cdots \circ \mathcal{T}_p$, *an operator* $\mathcal{T}_j : \mathbb{H} \to \mathbb{G}$ *is* easy-to-maintain*, if for any* x, i, δ_i, x^+ *satisfying* (6), $\mathfrak{M}[\{x, \mathcal{T}_j x, x^+\} \mapsto \mathcal{T}_j x^+]$ *is less than or equal to the number of remaining coordinate-update operations, in order of magnitude,* or *belongs to* $O(\frac{1}{\dim \mathbb{G}} \mathfrak{M}[x^+ \mapsto \mathcal{T} x^+])$.

The splitting schemes in §3.2 below will be based on $\mathcal{T}_1 + \mathcal{T}_2$ or $\mathcal{T}_1 \circ \mathcal{T}_2$. If $\mathcal{T}_1$ and $\mathcal{T}_2$ are both CF, $\mathcal{T}_1 + \mathcal{T}_2$ must be CF while $\mathcal{T}_1 \circ \mathcal{T}_2$ is not necessarily so. This subsection discusses how $\mathcal{T}_1 \circ \mathcal{T}_2$ inherits the properties from $\mathcal{T}_1$ and $\mathcal{T}_2$, and we summarize the results in Tables 1 and 2.

The combination $\mathcal{T}_1 \circ \mathcal{T}_2$ generally inherits the *weaker* property from $\mathcal{T}_1$ and $\mathcal{T}_2$.

Case	$\mathcal{T}_1 \in$	$\mathcal{T}_2 \in$	$(\mathcal{T}_1 \circ \mathcal{T}_2) \in$
1	$\mathcal{C}_1$ (separable)	$\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$	$\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$, respectively
2	$\mathcal{C}_2$ (nearly-sep.)	$\mathcal{C}_1$, $\mathcal{C}_3$	$\mathcal{C}_2$, $\mathcal{C}_3$, resp.
3	$\mathcal{C}_2$	$\mathcal{C}_2$	$\mathcal{C}_2$ or $\mathcal{C}_3$, case by case
4	$\mathcal{C}_3$ (non-sep.)	$\mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3$	$\mathcal{C}_3$

Table 1: $\mathcal{T}_1 \circ \mathcal{T}_2$ inherits the weaker separability properties from those of $\mathcal{T}_1$ and $\mathcal{T}_2$.

Case	$\mathcal{T}_1 \in$	$\mathcal{T}_2 \in$	$(\mathcal{T}_1 \circ \mathcal{T}_2) \in$	Example
5	$\mathcal{C}_1 \cup \mathcal{C}_2$	$\mathcal{F}$, $\mathcal{F}_1$	$\mathcal{F}$, $\mathcal{F}_1$, resp.	Examples 11 and 13
6	$\mathcal{F}$, $\mathcal{F}_2$	$\mathcal{C}_1$	$\mathcal{F}$, $\mathcal{F}_2$, resp.	Example 10
7	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}$	Example 12
8	cheap	$\mathcal{F}_2$	$\mathcal{F}$	Example 13
9	$\mathcal{F}_1$	cheap	$\mathcal{F}_1$	Examples 10 and 13

Table 2: Summary of how $\mathcal{T}_1 \circ \mathcal{T}_2$ inherits CF properties from those of $\mathcal{T}_1$ and $\mathcal{T}_2$.

The separable ($\mathcal{C}_1$) property is preserved by composition. If $\mathcal{T}_1, \ldots, \mathcal{T}_n$ are separable, then $\mathcal{T}_1 \circ \cdots \circ \mathcal{T}_n$ is separable. However, combining nearly-separable ($\mathcal{C}_2$) operators may not yield a nearly-separable operator since composition introduces cross dependence on the input entries. Therefore, composition of nearly-separable operators can either nearly-separable or non-separable.

Next, we discuss how $\mathcal{T}_1 \circ \mathcal{T}_2$ inherits the CF properties from $\mathcal{T}_1$ and $\mathcal{T}_2$, and the results are summarized in Table 2. For simplicity, we only use matrix-vector multiplication as examples in this subsection; more examples will be given later.

- If $\mathcal{T}_1$ is separable or nearly-separable ($\mathcal{C}_1 \cup \mathcal{C}_2$), then as long as $\mathcal{T}_2$ is CF ($\mathcal{F}$), $\mathcal{T}_1 \circ \mathcal{T}_2$ remains CF. In addition, if $\mathcal{T}_2$ is Type-I CF ($\mathcal{F}_1$), so is $\mathcal{T}_1 \circ \mathcal{T}_2$; see Example 11.

  Example 11. *Let $A \in \mathbb{R}^{m\times m}$ be* sparse *and $B \in \mathbb{R}^{m\times m}$* dense. *Then $\mathcal{T}_1 x = Ax$ is nearly-separable and $\mathcal{T}_2 x = Bx$ is Type-I CF. For any i, let $\mathbb{I}_i$ index the set of nonzeros on the ith row of A. We first compute*[3] *$(Bx)_{\mathbb{I}_i}$ that costs $O(|\mathbb{I}_i|m)$ and then $a_{i,\mathbb{I}_i}(Bx)_{\mathbb{I}_i}$ that costs $O(|\mathbb{I}_i|)$, where $a_{i,\mathbb{I}_i}$ is formed by the nonzeros entries on the ith row of A.*

[3]For this example, one can of course pre-compute AB and claim that $(\mathcal{T}_1 \circ \mathcal{T}_2)$ is Type-I CF. Our arguments keep A and B separate and only use the nearly-separability of $\mathcal{T}_1$ and Type-I CF of $\mathcal{T}_2$, so our result holds for any such composition even when $\mathcal{T}_1$ and $\mathcal{T}_2$ are nonlinear.

Assume $O(|\mathbb{I}_i|) = O(1)$, $\forall i$. *We have, from the above discussion, that* $\mathfrak{M}\left[x \mapsto (\mathcal{T}_1 \circ \mathcal{T}_2 x)_i\right] = O(m)$, *while* $\mathfrak{M}\left[x \mapsto \mathcal{T}_1 \circ \mathcal{T}_2 x\right] = O(m^2)$. *Hence,* $\mathcal{T}_1 \circ \mathcal{T}_2$ *is Type-I CF.*

- Assume that $\mathcal{T}_2$ is separable ($\mathcal{C}_1$). It is easy to see that if $\mathcal{T}_1$ is CF ($\mathcal{F}$), then $\mathcal{T}_1 \circ \mathcal{T}_2$ remains CF. In addition if $\mathcal{T}_1$ is Type-II CF ($\mathcal{F}_2$), so is $\mathcal{T}_1 \circ \mathcal{T}_2$; see Examples 10.
  Note that if $\mathcal{T}_2$ is nearly-separable, in general we do not have CF properties for $\mathcal{T}_1 \circ \mathcal{T}_2$. This is because $\mathcal{T}_2 x$ and $\mathcal{T}_2 x^+$ can be totally different (so updating $\mathcal{T}_2 x$ is expensive) even if x and x^+ differ only at one coordinate; see the example in footnote 2.
- Assume that $\mathcal{T}_1$ is Type-I CF ($\mathcal{F}_1$). Then if $\mathcal{T}_2$ is Type-II CF ($\mathcal{F}_2$), $\mathcal{T}_1 \circ \mathcal{T}_2$ must be CF ($\mathcal{F}$); see the next example.

  Example 12. *Let* $A, B \in \mathbb{R}^{m \times m}$ *be dense. Then* $\mathcal{T}_1 x = Ax$ *is Type-I CF and* $\mathcal{T}_2 x = Bx$ *Type-II CF (by maintaining* Bx*; see Example 2). For any* x *and* i, *let* x^+ *satisfy* (6). *Maintaining* $\mathcal{T}_2 x$, *we can compute* $\mathcal{T}_2 x^+$ *for* $O(m)$ *operations and then* $(\mathcal{T}_1 \circ \mathcal{T}_2 x^+)_j$ *for* $O(m)$ *operations for any* j. *On the other hand, computing* $\mathcal{T}_1 \circ \mathcal{T}_2 x^+$ *without maintaining* $\mathcal{T}_2 x$ *costs* $O(m^2)$.

- Assume that one of $\mathcal{T}_1$ and $\mathcal{T}_2$ is cheap. If $\mathcal{T}_2$ is cheap, then as long as $\mathcal{T}_1$ is Type-I CF ($\mathcal{F}_1$), $\mathcal{T}_1 \circ \mathcal{T}_2$ is Type-I CF. If $\mathcal{T}_1$ is cheap, then as long as $\mathcal{T}_2$ is Type-II CF ($\mathcal{F}_2$), $\mathcal{T}_1 \circ \mathcal{T}_2$ is CF ($\mathcal{F}$); see Example 13.

We will see more examples of the above cases in the rest of the paper.

3.2. Operator Splitting Schemes

Before we apply our discussions above to operator splitting schemes for new algorithms, we review several major operator splitting schemes and discuss their CF properties by using the results from §3.1. For basic operator concepts like monotonicity and cocoercivity, please refer to Appendix A. The definitions of resolvent and reflective-resolvent operators $\mathcal{J}_{\mathcal{A}}$ and $\mathcal{R}_{\mathcal{A}}$ are also given there, in (66) and (67), respectively.

Consider the following problem: given three operators $\mathcal{A}, \mathcal{B}, \mathcal{C}$, possibly set-valued,

$$\text{find } x \in \mathbb{H} \qquad \text{such that} \qquad 0 \in \mathcal{A}x + \mathcal{B}x + \mathcal{C}x. \tag{12}$$

This is a high-level abstraction of many problems. The study began in the 1960s, and since then a large number of algorithms and applications have been introduced. We review a few general methods in this subsection.

When $\mathcal{A}, \mathcal{B}$ are maximally monotone (think it as the subdifferential ∂f of a proper convex function f) and $\mathcal{C}$ is β-cocoercive (think it as the gradient ∇f of a $1/\beta$-Lipschitz differentiable function f), a solution can be found by the iteration (2) with $\mathcal{T} = \mathcal{T}_{3\mathrm{S}}$, introduced recently in [24], where

$$\mathcal{T}_{3\mathrm{S}} := \mathcal{I} - \mathcal{J}_{\gamma\mathcal{B}} + \mathcal{J}_{\gamma\mathcal{A}} \circ (2\mathcal{J}_{\gamma\mathcal{B}} - \mathcal{I} - \gamma\mathcal{C} \circ \mathcal{J}_{\gamma\mathcal{B}}). \tag{13}$$

Indeed, by setting $\gamma \in (0, 2\beta)$, $\mathcal{T}_{3S}$ is $(\frac{2\beta}{4\beta-\gamma})$-averaged (think it as a weaker property than the Picard contraction that does not guarantee $\mathcal{T}$ having a fixed point). Following the standard convergence result (cf. textbook [5]), provided that $\mathcal{T}$ has a fixed point, the sequence from (2) converges to a fixed-point x^* of $\mathcal{T}_{3\mathrm{S}}$. Instead of x^*, $\mathcal{J}_{\gamma\mathcal{B}}(x^*)$ is a solution to (12). Applying the results in §3.1, $\mathcal{T}_{3\mathrm{S}}$ is CF if $\mathcal{J}_{\gamma\mathcal{A}}$ is separable ($\mathcal{C}_1$), $\mathcal{J}_{\gamma\mathcal{B}}$ is Type-II CF ($\mathcal{F}_2$), and $\mathcal{C}$ is Type-I CF ($\mathcal{F}_1$).

We give a few special cases of $\mathcal{T}_{3\mathrm{S}}$ below, which have much longer history. They all converge to a fixed point x^* whenever a solution exists and γ is properly chosen. Whenever $\mathcal{B} \neq 0$, $\mathcal{J}_{\gamma\mathcal{B}}(x^*)$ replaces x^* as a solution to (12).

Forward-Backward Splitting (FBS): Letting $\mathcal{B} = 0$ yields $\mathcal{J}_{\gamma\mathcal{B}} = \mathcal{I}$. Then, $\mathcal{T}_{3\mathrm{S}}$ reduces to FBS [49]:

$$\mathcal{T}_{\mathrm{FBS}} := \mathcal{J}_{\gamma\mathcal{A}} \circ (\mathcal{I} - \gamma\mathcal{C}) \tag{14}$$

for solving the problem $0 \in \mathcal{A}x + \mathcal{C}x$.

Backward-Forward Splitting (BFS): Letting $\mathcal{A} = 0$ yields $\mathcal{J}_{\gamma\mathcal{A}} = \mathcal{I}$. Then, $\mathcal{T}_{3\mathrm{S}}$ reduces to BFS:

$$\mathcal{T}_{\mathrm{BFS}} := (\mathcal{I} - \gamma\mathcal{C}) \circ \mathcal{J}_{\gamma\mathcal{B}} \tag{15}$$

for solving the problem $0 \in \mathcal{B}x + \mathcal{C}x$. When $\mathcal{A} = \mathcal{B}$, $\mathcal{T}_{\mathrm{FBS}}$ and $\mathcal{T}_{\mathrm{BFS}}$ apply the same pair of operators in the opposite orders, and they solve the same problem. Iterations based on $\mathcal{T}_{\mathrm{BFS}}$ are rarely used in the literature because they need an extra application of $\mathcal{J}_{\gamma B}$ before returning the solution, so $\mathcal{T}_{\mathrm{BFS}}$ is seemingly an unnecessary variant of $\mathcal{T}_{\mathrm{FBS}}$. However, they become different in coordinate update algorithms; in particular, $\mathcal{T}_{\mathrm{BFS}}$ is CF (but $\mathcal{T}_{\mathrm{FBS}}$ is generally not) when $\mathcal{J}_{\gamma\mathcal{B}}$ is Type-II CF ($\mathcal{F}_2$) and $\mathcal{C}$ is Type-I CF ($\mathcal{F}_1$). Therefore, $\mathcal{T}_{\mathrm{BFS}}$ is worth discussing alone.

Douglas-Rachford Splitting (DRS): Letting $\mathcal{C} = 0$, $\mathcal{T}_{3\mathrm{S}}$ reduces to

$$\mathcal{T}_{\mathrm{DRS}} := \mathcal{I} - \mathcal{J}_{\gamma\mathcal{B}} + \mathcal{J}_{\gamma\mathcal{A}} \circ (2\mathcal{J}_{\gamma\mathcal{B}} - \mathcal{I}) = \frac{1}{2}(\mathcal{I} + \mathcal{R}_{\gamma\mathcal{A}} \circ \mathcal{R}_{\gamma\mathcal{B}}) \tag{16}$$

introduced in [26] for solving the problem $0 \in \mathcal{A}x+\mathcal{B}x$. A more general splitting is the Relaxed Peaceman-Rachford Splitting (RPRS) with $\lambda \in [0,1]$:

$$\mathcal{T}_{\text{RPRS}} = (1-\lambda)\,\mathcal{I} + \lambda\,\mathcal{R}_{\gamma\mathcal{A}} \circ \mathcal{R}_{\gamma\mathcal{B}}, \tag{17}$$

which recovers $\mathcal{T}_{\text{DRS}}$ by setting $\lambda = \frac{1}{2}$ and Peaceman-Rachford Splitting (PRS) [50] by $\lambda = 1$.

Forward-Douglas-Rachford Splitting (FDRS): Let V be a linear subspace, and $\mathcal{N}_V$ and $\mathcal{P}_V$ be the normal cone and projection operator, respectively. The FDRS [15]

$$\mathcal{T}_{\text{FDRS}} = \mathcal{I} - \mathcal{P}_V + \mathcal{J}_{\gamma\mathcal{A}} \circ (2\mathcal{P}_V - \mathcal{I} - \gamma\mathcal{P}_V \circ \tilde{\mathcal{C}} \circ \mathcal{P}_V),$$

aims at finding a x such that $0 \in \mathcal{A}\,x + \tilde{\mathcal{C}}\,x + \mathcal{N}_V\,x$. If an optimal x exists, we have $x \in V$ and $\mathcal{N}_V x$ is the orthogonal complement of V. Therefore, the problem is equivalent to finding x such that $0 \in \mathcal{A}\,x + \mathcal{P}_V \circ \tilde{\mathcal{C}} \circ \mathcal{P}_V\,x + \mathcal{N}_V\,x$. Thus, $\mathcal{T}_{\text{3S}}$ recovers $\mathcal{T}_{\text{FDRS}}$ by letting $\mathcal{B} = \mathcal{N}_V$ and $\mathcal{C} = \mathcal{P}_V \circ \tilde{\mathcal{C}} \circ \mathcal{P}_V$.

Forward-Backward-Forward Splitting (FBFS): Composing $\mathcal{T}_{\text{FBS}}$ with one more forward step gives $\mathcal{T}_{\text{FBFS}}$ introduced in [68]:

$$\mathcal{T}_{\text{FBFS}} = -\gamma\mathcal{C} + (\mathcal{I} - \gamma\mathcal{C})\mathcal{J}_{\gamma\mathcal{A}}(\mathcal{I} - \gamma\mathcal{C}). \tag{18}$$

$\mathcal{T}_{\text{FBFS}}$ is not a special case of $\mathcal{T}_{\text{3S}}$. At the expense of one more application of $(\mathcal{I} - \gamma\mathcal{C})$, $\mathcal{T}_{\text{FBFS}}$ relaxes the convergence condition of $\mathcal{T}_{\text{FBS}}$ from the cocoercivity of $\mathcal{C}$ to its monotonicity (for example, a nonzero skew symmetric matrix is monotonic but not cocoercive). From Table 2, we know that $\mathcal{T}_{\text{FBFS}}$ is CF if both $\mathcal{C}$ and $\mathcal{J}_{\gamma\mathcal{A}}$ are separable.

3.2.1. Examples in Optimization.

Consider the optimization problem

$$\underset{x \in X}{\text{minimize}}\ f(x) + g(x), \tag{19}$$

where X is the feasible set and f and g are objective functions. We present examples of operator splitting methods discussed above.

Example 13 (proximal gradient method)**.** *Let $X = \mathbb{R}^m$, f be differentiable, and g be proximable in (19). Letting $\mathcal{A} = \partial g$ and $\mathcal{C} = \nabla f$ in (14) gives $\mathcal{J}_{\gamma\mathcal{A}} = \mathbf{prox}_{\gamma g}$ and reduces $x^{k+1} = \mathcal{T}_{\text{FBS}}(x^k)$ to prox-gradient iteration:*

$$x^{k+1} = \mathbf{prox}_{\gamma g}(x^k - \gamma\nabla f(x^k)). \tag{20}$$

A special case of (20) *with* $g = \iota_X$ *is the projected gradient iteration:*

$$x^{k+1} = \mathcal{P}_X(x^k - \gamma \nabla f(x^k)). \tag{21}$$

If ∇f *is CF and* $\mathbf{prox}_{\gamma g}$ *is (nearly-)separable (e.g.,* $g(x) = \|x\|_1$ *or the indicator function of a box constraint) or if* ∇f *is Type-II CF and* $\mathbf{prox}_{\gamma g}$ *is cheap (e.g.,* $\nabla f(x) = Ax - b$ *and* $g = \|x\|_2$*), the FBS iteration in* (20) *is CF. In the latter case, we can also apply the BFS iteration* (15) *(i.e, compute* $\mathbf{prox}_{\gamma g}$ *and then the gradient update), which is also CF.*

Example 14 (Alternating Direction Method of Multipliers (ADMM))**.** *Setting* $X = \mathbb{R}^m$ *simplifies* (19) *to*

$$\underset{x,y}{\text{minimize}} \;\; f(x) + g(y), \quad \text{subject to } x - y = 0. \tag{22}$$

The ADMM method iterates:

$$x^{k+1} = \mathbf{prox}_{\gamma f}(y^k - \gamma s^k), \tag{23a}$$

$$y^{k+1} = \mathbf{prox}_{\gamma g}(x^{k+1} + \gamma s^k), \tag{23b}$$

$$s^{k+1} = s^k + \frac{1}{\gamma}(x^{k+1} - y^{k+1}). \tag{23c}$$

(The iteration can be generalized to handle the constraint $Ax - By = b$*.) The dual problem of* (22) *is* $\min_s f^*(-s) + g^*(s)$*, where* f^* *is the convex conjugate of* f*. Letting* $\mathcal{A} = -\partial f^*(-\cdot)$ *and* $\mathcal{B} = \partial g^*$ *in* (16) *recovers the iteration* (23) *through (see the derivation in Appendix B)*

$$t^{k+1} = \mathcal{T}_{\text{DRS}}(t^k) = t^k - \mathcal{J}_{\gamma\mathcal{B}}(t^k) + \mathcal{J}_{\gamma\mathcal{A}} \circ (2\mathcal{J}_{\gamma\mathcal{B}} - \mathcal{I})(t^k).$$

From the results in §3.1, a sufficient condition for the above iteration to be CF is that $\mathcal{J}_{\gamma\mathcal{A}}$ *is (nearly-)separable and* $\mathcal{J}_{\gamma\mathcal{B}}$ *being CF.*

4. Primal-Dual Coordinate Friendly Operators

We study how to solve the problem

$$\underset{x \in \mathbb{H}}{\text{minimize}} \; f(x) + g(x) + h(Ax), \tag{24}$$

with primal-dual splitting algorithms and their coordinate updates, where f is differentiable and A is a "p-by-m" linear operator from $\mathbb{H} = \mathbb{H}_1 \times \cdots \times \mathbb{H}_m$ to $\mathbb{G} = \mathbb{G}_1 \times \cdots \times \mathbb{G}_p$. Problem (24) abstracts many applications in image processing and machine learning.

Example 15 (image deblurring/denoising)**.** *Let u^0 be an image, where $u_i^0 \in [0, 255]$, and B be the blurring linear operator. Let $\|\nabla u\|_1$ be the anisotropic total variation of u. Suppose that b is a noisy observation of Bu^0. Then, we can try to recover u^0 by solving*

$$\underset{u}{\text{minimize}}\ \frac{1}{2}\|Bu - b\|^2 + \iota_{[0,255]}(u) + \lambda\|\nabla u\|_1, \tag{25}$$

which can be written in the form of (24) with $f = \frac{1}{2}\|B \cdot -b\|^2$, $g = \iota_{[0,255]}$, $A = \nabla$, and $h = \lambda\|\cdot\|_1$.

More examples with formulation (24) will be given in §4.2. In general, primal-dual methods are capable of solving complicated problems involving constraints and the compositions of proximable and linear maps.

In many applications, although h is proximable, $h \circ A$ is generally non-proximable and non-differentiable. To avoid using slow subgradient methods, we can consider the primal-dual splitting approaches to separate h and A so that $\mathbf{prox}_h$ can be applied. The problem (24) is equivalent (for convex case) to finding x such that

$$0 \in (\nabla f + \partial g + A^\top \circ \partial h \circ A)(x). \tag{26}$$

Introducing the dual variable $s \in \mathbb{G}$ and applying the biconjugation property: $s \in \partial h(Ax) \Leftrightarrow Ax \in \partial h^*(s)$, yields the equivalent condition

$$0 \in \Bigg(\underbrace{\begin{bmatrix} \nabla f \\ 0 \end{bmatrix}}_{\text{operator } \mathcal{A}} + \underbrace{\begin{bmatrix} \partial g \\ \partial h^* \end{bmatrix} + \begin{bmatrix} 0 & A^\top \\ -A & 0 \end{bmatrix}}_{\text{operator } \mathcal{B}} \Bigg) \underbrace{\begin{bmatrix} x \\ s \end{bmatrix}}_{z}, \tag{27}$$

which we shorten as $0 \in \mathcal{A}z + \mathcal{B}z$.

Problem (27) can be solved iteratively by the Condat-Vũ algorithm [20, 73]:

$$\begin{cases} s^{k+1} = \mathbf{prox}_{\gamma h^*}(s^k + \gamma A x^k), \\ x^{k+1} = \mathbf{prox}_{\eta g}(x^k - \eta(\nabla f(x^k) + A^\top(2s^{k+1} - s^k))), \end{cases} \tag{28}$$

which explicitly applies A and $A^\top$ and updates s, x in a Gauss-Seidel style [4]. We introduce an operator $\mathcal{T}_{\text{CV}} : \mathbb{H} \times \mathbb{G} \to \mathbb{H} \times \mathbb{G}$ and write

$$\text{iteration (28)} \quad \Longleftrightarrow \quad z^{k+1} = \mathcal{T}_{\text{CV}}(z^k).$$

[4] By the Moreau identity: $\mathbf{prox}_{\gamma h^*} = \mathcal{I} - \gamma \mathbf{prox}_{\frac{1}{\gamma}h}(\frac{\cdot}{\gamma})$, one can compute $\mathbf{prox}_{\frac{1}{\gamma}h}$ instead of $\mathbf{prox}_{\gamma h^*}$. The latter inherits the same separability properties from the former.

Switching the orders of x and s yields the following algorithm:

$$(29)\qquad \begin{cases} x^{k+1} = \mathbf{prox}_{\eta g}(x^k - \eta(\nabla f(x^k) + A^\top s^k)), \\ s^{k+1} = \mathbf{prox}_{\gamma h^*}(s^k + \gamma A(2x^{k+1} - x^k)), \end{cases} \quad \text{as } z^{k+1} = \mathcal{T}'_{\mathrm{CV}} z^k.$$

It is known from [18, 22] that both (28) and (29) reduce to iterations of nonexpansive operators (under a special metric), i.e., $\mathcal{T}_{\mathrm{CV}}$ is nonexpansive; see Appendix C for the reasoning.

Remark 1. *Similar primal-dual algorithms can be used to solve other problems such as saddle point problems [38, 44, 16] and variational inequalities [65]. Our coordinate update algorithms below apply to these problems as well.*

4.1. Primal-dual Coordinate Update Algorithms

In this subsection, we make the following assumption.

Assumption 1. *Functions g and h^* are separable and proximable. Specifically,*

$$g(x) = \sum_{i=1}^{m} g_i(x_i) \quad \text{and} \quad h^*(y) = \sum_{j=1}^{p} h_i^*(y_i).$$

Furthermore, ∇f is CF.

Proposition 1. *Under Assumption 1, the followings hold:*

(a) when $p = O(m)$, $\mathcal{T}_{\mathrm{CV}}$ in (28) is CF, more specifically,

$$\mathfrak{M}\left[\{z^k, Ax\} \mapsto \{z^+, Ax^+\}\right] = O\left(\frac{1}{m+p}\mathfrak{M}\left[z^k \mapsto \mathcal{T}_{\mathrm{CV}} z^k\right]\right);$$

(b) when $m \ll p$ and $\mathfrak{M}[x \mapsto \nabla f(x)] = O(m)$, $\mathcal{T}'_{\mathrm{CV}}$ in (29) is CF, more specifically,

$$\mathfrak{M}\left[\{z^k, A^\top s\} \mapsto \{z^+, A^\top s^+\}\right] = O\left(\frac{1}{m+p}\mathfrak{M}\left[z^k \mapsto \mathcal{T}'_{\mathrm{CV}} z^k\right]\right).$$

Proof. Computing $z^{k+1} = \mathcal{T}_{\mathrm{CV}} z^k$ involves evaluating ∇f, $\mathbf{prox}_g$, and $\mathbf{prox}_{h^*}$, applying A and $A^\top$, and adding vectors. Formally, $\mathfrak{M}\left[z^k \mapsto \mathcal{T}_{\mathrm{CV}} z^k\right] = O(mp + m + p) + \mathfrak{M}[x \to \nabla f(x)]$, and $\mathfrak{M}\left[z^k \mapsto \mathcal{T}'_{\mathrm{CV}} z^k\right]$ is the same.
(a) We assume $\nabla f \in \mathcal{F}_1$ for simplicity, and other cases are similar.

1. If $(\mathcal{T}_{\rm CV} z^k)_j = s_i^{k+1}$, computing it involves: adding s_i^k and $\gamma(Ax^k)_i$, and evaluating $\mathbf{prox}_{\gamma h_i^*}$. In this case $\mathfrak{M}\left[\{z^k, Ax\} \mapsto \{z^+, Ax^+\}\right] = O(1)$.
2. If $(\mathcal{T}_{\rm CV} z^k)_j = x_i^{k+1}$, computing it involves evaluating: the entire s^{k+1} for $O(p)$ operations, $(A^\top(2s^{k+1} - s^k))_i$ for $O(p)$ operations, $\mathbf{prox}_{\eta g_i}$ for $O(1)$ operations, $\nabla_i f(x^k)$ for $O(\frac{1}{m}\mathfrak{M}\left[x \mapsto \nabla f(x)\right])$ operations, as well as updating Ax^+ for $O(p)$ operations. In this case $\mathfrak{M}\left[\{z^k, Ax\} \mapsto \{z^+, Ax^+\}\right] = O(p + \frac{1}{m}\mathfrak{M}\left[x \mapsto \nabla f(x)\right])$.

Therefore, $\mathfrak{M}\left[\{z^k, Ax\} \mapsto \{z^+, Ax^+\}\right] = O\big(\frac{1}{m+p}\mathfrak{M}\left[z^k \mapsto \mathcal{T}_{\rm CV} z^k\right]\big)$.

(b) When $m \ll p$ and $\mathfrak{M}\left[x \mapsto \nabla f(x)\right] = O(m)$, by arguments similar to the above,
$\mathfrak{M}\left[\{z^k, A^\top s\} \mapsto \{z^+, A^\top s^+\}\right] = O(1) + \mathfrak{M}\left[x \mapsto \nabla_i f(x)\right]$ if $(\mathcal{T}'_{\rm CV} z^k)_j = x_i^{k+1}$;
and $\mathfrak{M}\left[\{z^k, A^\top s\} \mapsto \{z^+, A^\top s^+\}\right] = O(m) + \mathfrak{M}\left[x \mapsto \nabla f(x)\right]$ if $(\mathcal{T}'_{\rm CV} z^k)_j = s_i^{k+1}$.
In both cases $\mathfrak{M}\left[\{z^k, A^\top s\} \mapsto \{z^+, A^\top s^+\}\right] = O(\frac{1}{m+p}\mathfrak{M}\left[z^k \mapsto \mathcal{T}'_{\rm CV} z^k\right])$. □

4.2. Extended Monotropic Programming

The extended monotropic program is the problem

$$\begin{array}{ll} \underset{x \in \mathbb{H}}{\text{minimize}} & g_1(x_1) + g_2(x_2) + \cdots + g_m(x_m) + f(x), \\ \text{subject to} & A_1 x_1 + A_2 x_2 + \cdots + A_m x_m = b, \end{array} \tag{30}$$

where $x = (x_1, \ldots, x_m) \in \mathbb{H} = \mathbb{H}_1 \times \ldots \times \mathbb{H}_m$ with $\mathbb{H}_i$ being Euclidean spaces. It generalizes linear, quadratic, second-order cone programs by allowing general objective functions g_i and f. It is a special case of (24) by letting $g(x) = \sum_{i=1}^m g_i(x_i)$, $A = [A_1, \cdots, A_m]$ and $h = \iota_{\{b\}}$.

Example 16 (quadratic programming)**.** *Consider the quadratic program*

$$\underset{x \in \mathbb{R}^m}{\text{minimize}}\ \frac{1}{2} x^\top U x + c^\top x, \ \text{subject to}\ Ax = b,\ x \in X, \tag{31}$$

where U is a symmetric positive semidefinite matrix, $X = \{x : x_i \geq 0\ \forall i\}$. Then, (31) is a special case of (30) with $g_i(x_i) = \iota_{\cdot \geq 0}(x_i)$, $f(x) = \frac{1}{2} x^\top U x + c^\top x$ and $h = \iota_{\{b\}}$.

Example 17 (Second Order Cone Programming (SOCP))**.** *The SOCP*

$$\begin{array}{ll} \underset{x \in \mathbb{R}^m}{\text{minimize}} & c^\top x, \quad \text{subject to } Ax = b, \\ & x \in X = Q_1 \times \cdots \times Q_n, \end{array}$$

(where the number of cones n may not be equal to the number of blocks m,) can be written in the form of (30)*:* $\text{minimize}_{x\in\mathbb{R}^m}\ \iota_X(x) + c^\top x + \iota_{\{b\}}(Ax)$.

Applying iteration (28) to problem (30) and eliminating s^{k+1} from the second row yield the Jacobi-style update (denoted as $\mathcal{T}_{\text{emp}}$):

$$(32)\quad \begin{cases} s^{k+1} = s^k + \gamma(Ax^k - b), \\ x^{k+1} = \mathbf{prox}_{\eta g}(x^k - \eta(\nabla f(x^k) + A^\top s^k + 2\gamma A^\top Ax^k - 2\gamma A^\top b)). \end{cases}$$

Note that x^{k+1} no longer depends on s^{k+1}, making it more convenient to perform parallel coordinate updates.

Remark 2. *In general, when the s update is affine, we can plug the s update into the x update and decouple s^{k+1} and x^{k+1}. It is the case when h is affine and quadratic in problem* (24)*.*

One sufficient condition for $\mathcal{T}_{\text{emp}}$ to be CF is $\mathbf{prox}_g \in \mathcal{C}_1$ i.e., separable. Indeed, we have $\mathcal{T}_{\text{emp}} = \mathcal{T}_1 \circ \mathcal{T}_2$, where

$$\mathcal{T}_1 = \begin{bmatrix} \mathcal{I} & 0 \\ 0 & \mathbf{prox}_{\eta g} \end{bmatrix}, \mathcal{T}_2 \begin{bmatrix} s \\ x \end{bmatrix} = \begin{bmatrix} s + \gamma(Ax - b) \\ x - \eta(\nabla f(x) + A^\top s + 2\gamma A^\top Ax - 2\gamma A^\top b) \end{bmatrix}.$$

Case 5 of Table 2 gives the CF of $\mathcal{T}_{\text{emp}}$. When $O(m) = O(p)$, the separability condition on $\mathbf{prox}_g$ can be relaxed to $\mathbf{prox}_g \in \mathcal{F}_1$ since in this case $\mathcal{T}_2 \in \mathcal{F}_2$, and we can apply Case 7 of Table 2 (by maintaining $\nabla f(x)$, $A^\top s$, Ax and $A^\top Ax$.)

4.3. Overlapping-block Coordinate Updates

In the coordinate update scheme proposed in §4.1, updating x_i involves computing s^{k+1} but the result is discarded (see the proof of Proposition 1, item 1). This is because x_i's and s_j's are coupled through the matrix A.

We define, for each i, $\mathbb{J}(i) \subset \{1, 2, \ldots, p\}$ as the set of indices j such that $A^\top_{i,j} \neq 0$, and, for each j, $\mathbb{I}(j) \subset \{1, 2, \ldots, m\}$ as the set of indices of i such that $A_{j,i} \neq 0$. We also let $m_j := |\mathbb{I}(j)|$, and assume $m_j \neq 0, \forall j = 1, 2, \ldots, p$ without loss of generality.

We arrange the coordinates of $z = [x; s]$ into m (overlapping) blocks. The ith block consists of the coordinate x_i and s_j for all $j \in \mathbb{J}(i)$. We propose a block coordinate update scheme based on (28). Because the blocks overlap, the s_j update is relaxed with parameters $\rho_{i,j} \geq 0$ that satisfy

$$\sum_{i\in\mathbb{I}(j)} \rho_{i,j} = 1, \quad \forall j = 1, 2, \ldots, p,$$

so that the aggregated effect is to update s_j without scaling. (Following the KM iteration [37], we can also assign a relaxation parameter η_k for the x_i update; then, the s_j update should be relaxed with $\rho_{i,j}\eta_k$.)

Our update scheme is proposed as:

$$
(33)\quad \begin{cases} \text{when the } i\text{th coordinate is chosen, compute} \\ \tilde{s_j}^{k+1} = \mathbf{prox}_{\gamma h_j^*}(s_j^k + \gamma(Ax^k)_j), \text{ for all } j \in \mathbb{J}(i), \\ \tilde{x_i}^{k+1} = \mathbf{prox}_{\eta g_i}(x_i^k - \eta(\nabla_i f(x^k) + \sum_{j\in\mathbb{J}(i)} A_{i,j}^\top(2\tilde{s}_j^{k+1} - s_j^k))), \\ \text{update } x_i^{k+1} = x_i^k + (\tilde{x}_i^{k+1} - x_i^k), \\ \text{update } s_j^{k+1} = s_j^k + \rho_{i,j}(\tilde{s}_j^{k+1} - s_j^k), \text{ for all } j \in \mathbb{J}(i). \end{cases}
$$

Remark 3. *The use of relaxation parameters $\rho_{i,j}$ makes our scheme different from that in [53].*

Following the assumptions and arguments in §4.1, if we maintain Ax, the cost for each block coordinate update is $O(p) + \mathfrak{M}\,[x \mapsto \nabla_i f(x)]$, which is $O(\frac{1}{m}\mathfrak{M}\,[z \mapsto \mathcal{T}_{\mathrm{CV}} z])$. Therefore the coordinate update scheme (33) is computationally worthy.

Typical choices of $\rho_{i,j}$ include: (1) one of the $\rho_{i,j}$'s is 1 for each j, others all equal to 0. This can be viewed as assigning s_j fully to a block containing x_i. (2) $\rho_{i,j} = \frac{1}{m_j}$ for all $i \in \mathbb{I}(j)$.

Remark 4. *In their paper [28], Fercoq and Bianchi proposed a different primal-dual coordinate update algorithm. They produced a new matrix $\bar{A}$ based on A, with only one nonzero entry in each row, i.e. $m_j = 1$ for each j. They also modify h to $\bar{h}$ so that the problem*

$$
(34)\qquad \underset{x\in\mathbb{H}}{\text{minimize}}\ f(x) + g(x) + \bar{h}(\bar{A}x)
$$

has the same solution as (24). Then they solve (34) by the scheme (33). Because they have $m_j = 1$, every dual variable coordinate is only associated with one primal variable coordinate. They create non-overlap blocks of z by duplicating each dual variable coordinate s_j multiple times. The computation cost for each block coordinate update of their algorithm is the same as (33), but more memory is needed since duplicated copies of each s_j are stored and updated.

4.4. Async-parallel Primal Dual Coordinate Update Algorithms and Their Convergence

In this section we propose some primal dual coordinate update algorithms based on [51] and state their convergence theorems. We assume a shared memory architecture and allow delays and inconsistent reading (see below for explanations).

Algorithm 1: Async-parallel primal dual coordinate update algorithm using $\mathcal{T}_{\rm CV}$

Input : $z^0 \in \mathbb{H} \times \mathbb{G}$, $K > 0$, a discrete distribution $(q_1, \ldots, q_{m+p})$ with $\sum_{i=1}^{m+p} q_i = 1$ and $q_i > 0, \forall i$,
set global iteration counter $k = 0$;
while $k < K$, *every agent asynchronously and continuously* **do**
  select $i_k \in \{1, \ldots, m+p\}$ with $\text{Prob}(i_k = i) = q_i$;
  perform an update to z_{i_k} according to (35);
  update the global counter $k \leftarrow k + 1$;

Whenever an agent updates a coordinate, the global iteration number k increases by one.

The kth update is applied to z_{i_k}, with $i_k, k = 1, 2, ...$ being independent random variables. $z_i = x_i$ when $i \leq m$ and $z_i = s_{i-m}$ when $i > m$. Each coordinate update has the form:

$$\text{(35)} \qquad \begin{cases} z_{i_k}^{k+1} = z_{i_k}^k - \frac{\eta_k}{(m+p)q_{i_k}} (\hat{z}_{i_k} - (\mathcal{T}_{\rm CV}\hat{z}^k)_{i_k}), \\ z_i^{k+1} = z_i^k, \quad \forall i \neq i_k, \end{cases}$$

where η_k is the step size, z^k denotes the state of z in global memory just before the update (35) is applied, and $\hat{z}^k$ is the result that z in global memory is read by an agent to its local cache. Due to possible updates to z by other agents, $\hat{z}^k$ can be different from z^k. We refer the reader to [51, Section 1.2] for more details.

We also propose an async-parallel primal dual algorithm using the overlapping-block coordinate update (33):

Algorithm 2: Async-parallel primal dual overlapping-block coordinate update algorithm using $\mathcal{T}_{\text{CV}}$

Input : $z^0 \in \mathbb{H} \times \mathbb{G}$, $K > 0$, a discrete distribution $(q_1, \ldots, q_m)$ with $\sum_{i=1}^m q_i = 1$ and $q_i > 0, \forall i$,

set global iteration counter $k = 0$;

while $k < K$, *every agent asynchronously and continuously* **do**

select $i_k \in \{1, \ldots, m\}$ with $\text{Prob}(i_k = i) = q_i$;

Compute $\tilde{s}_j^{k+1}, \forall j \in \mathbb{J}(i_k)$ and $\tilde{x}_{i_k}^{k+1}$ according to (33);

update $x_i^{k+1} = x_i^k + \frac{\eta_k}{mq_{i_k}}(\tilde{x}_i^{k+1} - x_i^k)$;

update $s_j^{k+1} = s_j^k + \frac{\rho_{i,j}\eta_k}{mq_{i_k}}(\tilde{s}_j^{k+1} - s_j^k)$, for all $j \in \mathbb{J}(i_k)$;

update the global counter $k \leftarrow k + 1$;

Here we still allow inconsistent delays, so the $\tilde{x}_{i_k}$ and $\tilde{s}_j^{k+1}$ are computed using some $\hat{z}^k$.

Remark 5. *For the shared memory architecture, it is recommended to set all but one $\rho_{i,j}$'s to 0.*

Now we state the convergence theorem for Algorithm 1 and 2.

Theorem 1. *Let Z^* be the set of optimal solutions of problem (24) and let $(z^k)_{k\geq 0} \subset \mathbb{H} \times \mathbb{G}$ be the sequence generated by Algorithm 1 or Algorithm 2 under the following conditions:*

(i) f, g, h^* are closed proper convex functions and f is differentiable with ∇f β-Lipschitz continuous;

(ii) the delay for every coordinate is bounded by a positive number τ, i.e. for every $1 \leq i \leq m + p$, $\hat{z}_i^k = z_i^{k-d_i^k}$ for some $0 \leq d_i^k \leq \tau$;

(iii) $\eta_k \in [\eta_{\min}, \eta_{\max}]$ for certain $\eta_{\min}, \eta_{\max} > 0$.

We have with probability 1, $(z^k)_{k\geq 0}$ converges to a Z^*-valued random variable.

The explicit formula for $\eta_{\min}, \eta_{\max}$, the proof and some other remarks are included in Appendix D.

Remark 6. *When there is only one agent and we have no delay, the async-parallel algorithms we propose become stochastic coordinate update algorithms, and their convergence is a direct consequence of Theorem 1.*

5. Applications

In this section, we provide examples to illustrate how to develop coordinate update algorithms based on CF operators. The applications are categorized into five different areas of applications. The first subsection discusses three well-known machine learning problems: empirical risk minimization, Support Vector Machine (SVM), and group Lasso. The second subsection discusses image processing problems including image deblurring, image denoising, and Computed Tomography (CT) image recovery. The remaining subsections provide applications in finance, distributed computing as well as certain stylized optimization models.

For each problem, we only describe the coordinate update. The final algorithm is obtained after plugging the update in a coordinate update framework in §1.1 along with initialization, parameter selection, and termination.

5.1. Machine Learning

5.1.1. Empirical Risk Minimization. We consider the following regularized empirical risk minimization problem

$$\underset{x\in\mathbb{R}^m}{\text{minimize}} \ \frac{1}{p}\sum_{j=1}^{p}\phi_j(a_j^\top x)+f(x)+g(x), \tag{36}$$

where a_j's are sample vectors, ϕ_j's are loss functions, and $f+g$ is a regularization function, where f is differentiable and g is proximable. The problem aims to learn a response vector x from a_j's so that the total loss measured by $\sum_j \phi_j$ is small. When prior information about x such as sparsity is provided, it is modeled with a regularization function $f+g$ such that the prior information is kept by small values of $f(x)+g(x)$. The need for coordinate update algorithms arise in many applications of (36) where there are a very large number of samples, namely, $p \gg m$.

We define $A = [a_1, a_2, ..., a_p]^\top$, with a_i being the ith row of A, and $h(y) := \frac{1}{p}\sum_{i=1}^{p}\phi_i(y_i)$. Hence, $h(Ax) = \frac{1}{p}\sum_{i=1}^{p}\phi_i(a_i^\top x)$, and problem (36) reduces to form (24). We can apply the primal-dual update scheme (29) to solve this problem, for which we introduce the dual variable $s = (s_1, ..., s_p)^\top$. Since $p \gg m$, we use $1+p$ coordinates, where the 0th coordinate is $x \in \mathbb{R}^m$ and the jth coordinate is s_j, $j = 1, \ldots, p$. At each iteration, a coordinate is

updated:

$$
(37)\qquad \begin{cases} \text{if } x \text{ is chosen, then compute} \\ \qquad x^{k+1} = \mathbf{prox}_{\eta g}(x^k - \eta(\nabla f(x^k) + A^\top s^k)), \\ \text{if } s_j \text{ is chosen, then compute} \\ \qquad \tilde{x}^{k+1} = \mathbf{prox}_{\eta g}(x^k - \eta(\nabla f(x^k) + A^\top s^k)), \\ \qquad \text{and} \\ \qquad s_j^{k+1} = \frac{1}{p}\mathbf{prox}_{p\gamma\phi_j^*}(ps_j^k + p\gamma a_j^\top(2\tilde{x}^{k+1} - x^k)). \end{cases}
$$

We maintain $A^\top s \in \mathbb{R}^m$ in the memory. Depending on the structure of ∇f, we can compute it each time or maintain it. When $\mathbf{prox}_g \in \mathcal{F}_1$, we can further apply coordinate updates to x_i.

5.1.2. Support Vector Machine. Given the training data $\{(a_i, \beta_i)\}_{i=1}^m$ with $\beta_i \in \{+1, -1\}$, $\forall i$, the kernel support vector machine [62] is

$$
(38)\qquad \begin{array}{ll} \underset{x,\xi,y}{\text{minimize}} & \frac{1}{2}\|x\|_2^2 + C\sum_{i=1}^m \xi_i, \\ \text{s.t.} & \beta_i(x^\top\phi(a_i) - y) \geq 1 - \xi_i,\ \xi_i \geq 0,\ i = 1, \ldots, m, \end{array}
$$

where ϕ is a vector-to-vector map, mapping each data a_i to a point in a (possibly) higher-dimensional space. If $\phi(a) = a$, then (38) reduces to the linear support vector machine. The model (38) can be interpreted as finding a hyperplane $\{w : x^\top w - y = 0\}$ to roughly separate two sets of points $\{\phi(a_i) : \beta_i = 1\}$ and $\{\phi(a_i) : \beta_i = -1\}$.

The dual problem of (38) is

$$
(39)\qquad \underset{s}{\text{minimize}}\, \frac{1}{2}s^\top Qs - e^\top s,\ \text{subject to } 0 \leq s_i \leq C,\ \forall i,\ \sum_i \beta_i s_i = 0,
$$

where $Q_{ij} = \beta_i\beta_j k(a_i, a_j)$, $k(\cdot,\cdot)$ is a so-called kernel function, and $e = (1, ..., 1)^\top$. If $\phi(a) = a$, then $k(a_i, a_j) = a_i^\top a_j$.

Unbiased case. If $y = 0$ is enforced in (38), then the solution hyperplane $\{w : x^\top w = 0\}$ passes through the origin and is called *unbiased*. Consequently, the dual problem (39) will no longer have the linear constraint $\sum_i \beta_i s_i = 0$, leaving it with the coordinate-wise separable box constraints $0 \leq s_i \leq C$. To solve (39), we can apply the FBS scheme (14). Letting $d(s) := \frac{1}{2}s^\top Qs - e^\top s$, the coordinate update based on FBS is

$$
s_i^{k+1} = \mathbf{proj}_{[0,C]}(s_i^k - \gamma_i \nabla_i d(s^k)),
$$

where we can take $\gamma_i = \frac{1}{Q_{ii}}$.

Biased (general) case. In this case, the mode (38) has $y \in \mathbb{R}$, so the hyperplane $\{w : x^\top w - y = 0\}$ may not pass the origin and is called *biased*. Then, the dual problem (39) retains the linear constraint $\sum_i \beta_i s_i = 0$. In this case, we apply the primal-dual splitting scheme (28) or the three-operator splitting scheme (13).

The coordinate update based on the primal-dual splitting is:

$$t^{k+1} = t^k + \gamma \sum_{i=1}^m \beta_i s_i^k, \tag{40a}$$

$$s_i^{k+1} = \mathbf{proj}_{[0,C]}\left(s_i^k - \eta\big(\nabla_i d(s^k) + \beta_i(2t^{k+1} - t^k)\big)\right), \tag{40b}$$

where t, s are the primal and dual variables, respectively. Note that we can let $w := \sum_{i=1}^m \beta_i s_i$ and maintain it. With variable w and substituting (40a) into (40b), we can equivalently write (40) into

$$t^{k+1} = t^k + \gamma w^k, \tag{41a}$$

$$s_i^{k+1} = \mathbf{proj}_{[0,C]}\left(s_i^k - \eta\big(q_i^\top s^k - 1 + \beta_i(2\gamma w^k + t^k)\big)\right),\ i = 1, \ldots, m. \tag{41b}$$

In parallel computing, whenever a processor updates some s_i, the w variable must be also renewed as $w^{k+1} = w^k + \beta_i(s_i^{k+1} - s_i^k)$.

We can also apply the three-operator splitting (13) as follows. Let $D_1 := [0, C]^m$ and $D_2 := \{s : \sum_{i=1}^m \beta_i s_i = 0\}$. The full update is

$$s^{k+1} = \mathbf{proj}_{D_2}(u^k), \tag{42a}$$

$$u^{k+1} = u^k + \eta_k\left(\mathbf{proj}_{D_1}\big(2s^{k+1} - u^k - \gamma(Qs^{k+1} - e)\big) - s^{k+1}\right), \tag{42b}$$

where s is just an intermediate variable. Let $\tilde{\beta} := \frac{\beta}{\|\beta\|_2}$ and $w := \tilde{\beta}^\top u$. Then $\mathbf{proj}_{D_2}(u) = (I - \tilde{\beta}\tilde{\beta}^\top)u$. Hence, $s^{k+1} = u^k - w^k\tilde{\beta}$. Plugging it into (42b) yields the coordinate update:

$$s_i^{k+1} = u_i^k - w^k\tilde{\beta}_i,$$

$$u_i^{k+1} = u_i^k + \eta_k\left(\mathbf{proj}_{[0,C]}\left(2s_i^{k+1} - u_i^k - \gamma\big(q_i^\top u^k - w^k(q_i^\top\tilde{\beta}) - 1\big)\right) - s_i^{k+1}\right).$$

We can maintain w and renew it as $w^{k+1} = w^k + \tilde{\beta}_i(u_i^{k+1} - u_i^k)$ whenever u_i is updated. The intermediate variable s_i is not kept throughout iterations.

5.1.3. Group Lasso. The group Lasso regression problem [81] is

$$\underset{x\in\mathbb{R}^n}{\text{minimize}}\, f(x) + \sum_{i=1}^{m} \lambda_i \|x_i\|_2, \tag{44}$$

where f is a differentiable convex function, often bearing the form $\frac{1}{2}\|Ax - b\|_2^2$, and $x_i \in \mathbb{R}^{n_i}$ is a subvector of $x \in \mathbb{R}^n$ supported on $\mathbb{I}_i \subset \{1, \ldots, n\}$, and $\cup_i \mathbb{I}_i = \{1, \ldots, n\}$. If $\mathbb{I}_i \cap \mathbb{I}_j = \emptyset$, $\forall i \neq j$, it is called *non-overlapping group Lasso*, and if there are two different groups $\mathbb{I}_i$ and $\mathbb{I}_j$ with a non-empty intersection, it is called *overlapping group Lasso*. The model finds a coefficient vector x that minimizes the fitting (or loss) function $f(x)$ and that is group sparse: all but a few x_i are zero.

Let U_i be formed by the columns of the identity matrix I in $\mathbb{I}_i$, and let $U = [U_1, \ldots, U_m]^\top \in \mathbb{R}^{(\Sigma_i n_i)\times n}$. Then, $x_i = U_i^\top x$. Let $h_i(y_i) = \lambda_i \|y_i\|_2$, $y_i \in \mathbb{R}^{n_i}$ for $i = 1, \ldots, m$, and $h(y) = \sum_{i=1}^m h_i(y_i)$ for $y = [y_1; \ldots; y_m] \in \mathbb{R}^{\Sigma_i n_i}$. In this way, (44) becomes

$$\underset{x}{\text{minimize}}\, f(x) + h(Ux). \tag{45}$$

Non-overlapping case. In this case, we have $\mathbb{I}_i \cap \mathbb{I}_j = \emptyset$, $\forall i \neq j$, and can apply the FBS scheme (14) to (45). Specifically, let $\mathcal{T}_1 = \partial(h \circ U)$ and $\mathcal{T}_2 = \nabla f$. The FBS full update is

$$x^{k+1} = \mathcal{J}_{\gamma\mathcal{T}_1} \circ (\mathcal{I} - \gamma\mathcal{T}_2)(x^k).$$

For $i \in \{1, \ldots, m\}$, the corresponding coordinate update is

$$x_i^{k+1} = \arg\min_{x_i} \frac{1}{2}\|x_i - x_i^k + \gamma\nabla_i f(x^k)\|_2^2 + \gamma h_i(x_i) \tag{46a}$$

$$= \max\left(\|x_i^k - \gamma\nabla_i f(x^k)\|_2 - \gamma, 0\right) \frac{x_i^k - \gamma\nabla_i f(x^k)}{\|x_i^k - \gamma\nabla_i f(x^k)\|_2}, \tag{46b}$$

where the step size can be taken to $\gamma = \frac{1}{\|A\|_2^2}$. When ∇f is either cheap or easy-to-maintain, the coordinate update in (46) is inexpensive.

Overlapping case. This case allows $\mathbb{I}_i \cap \mathbb{I}_j \neq \emptyset$ for some $i \neq j$, causing the evaluation of $\mathcal{J}_{\gamma\mathcal{T}_1}$ be generally difficult. However, we can apply the

primal-dual update (28) to this problem as

$$s^{k+1} = \mathbf{prox}_{\gamma h^*}(s^k + \gamma U x^k), \tag{47a}$$

$$x^{k+1} = x^k - \eta(\nabla f(x^k) + U^\top(2s^{k+1} - s^k)), \tag{47b}$$

where s is the dual variable. Note that

$$h^*(s) = \begin{cases} 0, & \text{if } \|s_i\|_2 \le \lambda_i, \ \forall i, \\ +\infty, & \text{otherwise,} \end{cases}$$

is cheap. Hence, for $i \in \{1, \ldots, m\}$, the coordinate update based on (47) is

$$s_i^{k+1} = \mathbf{proj}_{B_{\lambda_i}}(s_i^k + \gamma x_i^k), \tag{48a}$$

$$x_i^{k+1} = x_i^k - \eta(\nabla_i f(x^k) + (2\mathbf{proj}_{B_{\lambda_i}}(s_i^k + \gamma x_i^k) - s_i^k), \tag{48b}$$

where B_λ is the Euclidean ball of radius λ. When ∇f is easy-to-maintain, the coordinate update in (48) is inexpensive.

5.2. Imaging

5.2.1. DRS for Image Processing in the Primal-dual Form [47]. Many convex image processing problems have the general form

$$\underset{x}{\text{minimize}}\, f(x) + g(Ax),$$

where A is a matrix such as a dictionary, sampling operator, or finite difference operator. We can reduce the problem to the system: $0 \in \mathcal{A}(z) + \mathcal{B}(z)$, where $z = [x; s]$,

$$\mathcal{A}(z) := \begin{bmatrix} \partial f(x) \\ \partial g^*(s) \end{bmatrix}, \quad \text{and} \quad \mathcal{B}(z) := \begin{bmatrix} 0 & A^T \\ -A & 0 \end{bmatrix} \begin{bmatrix} x \\ s \end{bmatrix}.$$

(See Appendix C for the reduction.) The work [47] gives their resolvents

$$\mathcal{J}_{\gamma\mathcal{A}} = \begin{bmatrix} \mathbf{prox}_{\gamma f} \\ \mathbf{prox}_{\gamma g^*} \end{bmatrix},$$

$$\mathcal{J}_{\gamma\mathcal{B}} = (I + \gamma\mathcal{B})^{-1} = \begin{bmatrix} 0 & 0 \\ 0 & I \end{bmatrix} + \begin{bmatrix} I \\ \gamma A \end{bmatrix} (I + \gamma^2 A^T A)^{-1} \begin{bmatrix} I \\ -\gamma A \end{bmatrix}^T,$$

where $\mathcal{J}_{\gamma\mathcal{A}}$ is often cheap or separable and we can *explicitly form* $\mathcal{J}_{\gamma\mathcal{B}}$ as a matrix or implement it based on a fast transform. The resulting DRS

operator is CF when $\mathcal{J}_{\gamma\mathcal{B}}$ is CF. We leave the DRS coordinate update to the reader.

5.2.2. Total Variation Image Processing. We consider the following Total Variation (TV) image processing model

$$\underset{x}{\text{minimize}} \ \lambda\|x\|_{\text{TV}} + \frac{1}{2}\|A\,x - b\|^2, \tag{49}$$

where $x \in \mathbb{R}^n$ is the vector representation of the unknown image, A is an $m \times n$ matrix describing the transformation from the image to the measurements, and $b \in \mathbb{R}^m$ is the given measurements with noise. Let (∇_i^h, ∇_i^v) be the discrete gradient at pixel i. Then the TV semi-norm $\|\cdot\|_{\text{TV}}$ in the isotropic and anisotropic fashions are, respectively,

$$\|x\|_{\text{TV}} = \sum_i \sqrt{(\nabla_i^h x)^2 + (\nabla_i^v x)^2}, \tag{50a}$$

$$\|x\|_{\text{TV}} = \|\nabla x\|_1 = \sum_i \left(|\nabla_i^h x| + |\nabla_i^v x|\right). \tag{50b}$$

Consider applications in which A is a sparse matrix. For example, in CT image reconstruction, A models the discrete Radon transform. Each row describes a line integral, and as each line only intersects with a few pixels, there are many zeros in each row. For image deblurring, A models a convolution kernel, and each row is sparse if the size of the kernel is small.

For simplicity, we use the anisotropic TV for analysis and in the numerical experiment in § 6.2. It is slightly more complicated for the isotropic TV. Introducing the following notation:

$$B := \begin{pmatrix} \nabla \\ A \end{pmatrix}, \quad h(p,q) := \lambda\|p\|_1 + \frac{1}{2}\|q - b\|^2,$$

we can reformulate (49) as

$$\underset{x}{\text{minimize}} \ h(B\,x) = h(\nabla x, A\,x),$$

which reduces to the form of (24) with $f = g = 0$. Based on its definition, the convex conjugate of $h(p,q)$ and its proximal operator are, respectively,

$$h^*(s,t) = \iota_{\|\cdot\|_\infty \le \lambda}(s) + \frac{1}{2}\|t + b\|^2 - \frac{1}{2}\|b\|^2, \tag{51}$$

$$\mathbf{prox}_{\gamma h^*}(s,t) = \mathbf{proj}_{\|\cdot\|_\infty \le \lambda}(s) + \frac{1}{1+\gamma}(t - \gamma b). \tag{52}$$

Applying (29) gives the following full update:

$$x^{k+1} = x^k - \eta(\nabla^\top s^k + A^T t^k), \tag{53a}$$

$$s^{k+1} = \mathbf{proj}_{\|\cdot\|_\infty \le \lambda}\left(s^k + \gamma\nabla(x^k - 2\eta(\nabla^\top s^k + A^\top t^k))\right), \tag{53b}$$

$$t^{k+1} = \frac{1}{1+\gamma}\left(t^k + \gamma A(x^k - 2\eta(\nabla^\top s^k + A^\top t^k)) - \gamma b\right). \tag{53c}$$

To perform the coordinate updates as described in §4, we can maintain $\nabla^\top s^k$ and $A^\top t^k$. Whenever a coordinate of (s,t) is updated, the corresponding $\nabla^\top s^k$ (or $A^\top t^k$) should also be updated.

5.2.3. 3D Mesh Denoising. Follow an example in [57], we consider a 3D mesh described by their nodes $\bar{x}_i = (\bar{x}_i^X, \bar{x}_i^Y, \bar{x}_i^Z), i = 1, 2, ..., n$, and the adjacency matrix $A \in \mathbb{R}^{n\times n}$. We let $\mathcal{V}_i$ be the set of neighbours of node i. Noisy mesh nodes $z_i, i = 1, 2, ..., n$ with the same adjacency matrix A are observed. We try to recover the original mesh nodes by solving the following optimization problem [57]:

$$\underset{x}{\text{minimize}} \sum_{i=1}^n f_i(x_i) + \sum_{i=1}^n g_i(x_i) + \sum_{i,j:j\in\mathcal{V}_i} h_{i,j}(x_i - x_j), \tag{54}$$

where f_i's are differentiable data fidelity terms, g_i's are box constraints, and $\sum_{i,j:j\in\mathcal{V}_i} h_{i,j}(x_i - x_j)$ is the total variation on the mesh.

We introduce a dual variable s with coordinates $s_{i,j}$, for all ordered pairs of adjacent nodes (i,j), and, based on the overlapping-block coordinate updating scheme (33), perform coordinate update:

$$\begin{cases} \text{when } x_i \text{ is chosen, compute} \\ \quad \tilde{s}_{i,j}^{k+1} = \mathbf{prox}_{\gamma h_{i,j}^*}(s_{i,j}^k + \gamma x_i^k - \gamma x_j^k), \forall j \in \mathcal{V}_i, \\ \quad \tilde{s}_{j,i}^{k+1} = \mathbf{prox}_{\gamma h_{j,i}^*}(s_{j,i}^k + \gamma x_j^k - \gamma x_i^k), \forall j \in \mathcal{V}_i, \\ \quad \text{and update} \\ \quad x_i{}^{k+1} = \mathbf{prox}_{\eta g_i}(x_i^k - \eta(\nabla f_i(x_i^k) + \sum_{j\in\mathcal{V}_i}(2\tilde{s}_{i,j}^{k+1} - 2\tilde{s}_{j,i}^{k+1} - s_{i,j}^k + s_{j,i}^k))), \\ \quad s_{i,j}^{k+1} = s_{i,j}^k + \frac{1}{2}(\tilde{s}_{i,j}^{k+1} - s_{i,j}^k), \forall j \in \mathcal{V}_i, \\ \quad s_{j,i}^{k+1} = s_{j,i}^k + \frac{1}{2}(\tilde{s}_{j,i}^{k+1} - s_{j,i}^k), \forall j \in \mathcal{V}_i. \end{cases}$$

5.3. Finance

5.3.1. Portfolio Optimization. Assume that we have one unit of capital and m assets to invest on. The ith asset has an expected return rate $\xi_i \ge 0$.

Our goal is to find a portfolio with the minimal risk such that the expected return is no less than c. This problem can be formulated as

$$\begin{aligned}&\underset{x}{\text{minimize}}\ x^\top Qx,\\ &\text{subject to } x\geq 0, \sum_{i=1}^m x_i\leq 1, \sum_{i=1}^m \xi_i x_i\geq c\end{aligned}$$

where Q is the covariance matrix. Let $a_1=e/\sqrt{m}$, $b_1=1/\sqrt{m}$, $a_2=\xi/\|\xi\|_2$, and $b_2=c/\|\xi\|_2$. The above problem is rewritten as

$$\underset{x}{\text{minimize}}\, x^\top Qx,\ \text{subject to}\ \ x\geq 0,\ a_1^\top x\leq b_1,\ a_2^\top x\geq b_2. \tag{55}$$

We apply the three-operator splitting scheme (13) to (55). Let $f(x)=\frac{1}{2}x^\top Qx$, $D_1=\{x: x\geq 0\}$, $D_2=\{x: a_1^\top x\leq b_1, a_2^\top x\geq b_2\}$, $D_{21}=\{x: a_1^\top x=b_1\}$, and $D_{22}=\{x: a_2^\top x=b_2\}$. The full update is

$$y^{k+1} = \mathbf{proj}_{D_2}(x^k), \tag{56a}$$
$$x^{k+1} = x^k+\eta_k\big(\mathbf{proj}_{D_1}(2y^{k+1}-x^k-\gamma\nabla f(y^{k+1}))-y^{k+1}\big), \tag{56b}$$

where y is an intermediate variable. As the projection to D_1 is simple, we discuss how to evaluate the projection to D_2. Assume that a_1 and a_2 are neither perpendicular nor co-linear, i.e., $a_1^\top a_2\neq 0$ and $a_1\neq\lambda a_2$ for any scalar λ. In addition, assume $a_1^\top a_2>0$ for simplicity. Let $a_3=a_2-\frac{1}{a_1^\top a_2}a_1$, $b_3=b_2-\frac{1}{a_1^\top a_2}b_1$, $a_4=a_1-\frac{1}{a_1^\top a_2}a_2$, and $b_4=b_1-\frac{1}{a_1^\top a_2}b_2$. Then we can partition the whole space into four areas by the four hyperplanes $a_i^\top x=b_i$, $i=1,\ldots,4$. Let $P_i=\{x: a_i^\top x\leq b_i, a_{i+1}^\top x\geq b_{i+1}\}$, $i=1,2,3$ and $P_4=\{x: a_4^\top x\leq b_4, a_1^\top x\geq b_1\}$. Then

$$\mathbf{proj}_{D_2}(x)=\begin{cases}x, & \text{if } x\in P_1,\\ \mathbf{proj}_{D_{22}}(x), & \text{if } x\in P_2,\\ \mathbf{proj}_{D_{21}\cap D_{22}}(x), & \text{if } x\in P_3,\\ \mathbf{proj}_{D_{21}}(x), & \text{if } x\in P_4.\end{cases}$$

Let $w_i=a_i^\top x-b_i, i=1,2$, and maintain w_1,w_2. Let $\tilde{a}_2=\frac{a_2-a_1(a_1^\top a_2)}{1-(a_1^\top a_2)^2}$, $\tilde{a}_1=\frac{a_1-a_2(a_1^\top a_2)}{1-(a_1^\top a_2)^2}$. Then

$$\begin{aligned}&\mathbf{proj}_{D_{21}}(x)=x-w_1a_1,\\ &\mathbf{proj}_{D_{22}}(x)=x-w_2a_2,\\ &\mathbf{proj}_{D_{21}\cap D_{22}}(x)=x-w_1\tilde{a}_1-w_2\tilde{a}_2,\end{aligned}$$

Hence, the coordinate update of (56) is

$$
\begin{aligned}
&(57a)\\
&x^k \in P_1: \ x_i^{k+1} = (1-\eta_k)x_i^k + \eta_k \max(0, x_i^k - \gamma q_i^\top x^k),\\
&x^k \in P_2: \ x_i^{k+1} = (1-\eta_k)x_i^k + \eta_k w_2^k a_{i2} + \eta_k \max\big(0,\\
&(57b)\qquad\qquad x_i^k - \gamma q_i^\top x^k - w_2^k(2a_{i2} - \gamma q_i^\top a_2)\big),\\
&x^k \in P_3: \ x_i^{k+1} = (1-\eta_k)x_i^k + \eta_k \left(w_1^k \tilde{a}_{i1} + w_2^k \tilde{a}_{i2}\right) + \eta_k \max\big(0,\\
&(57c)\qquad\qquad x_i^k - \gamma q_i^\top x^k - w_1^k(2\tilde{a}_{i1} - \gamma q_i^\top \tilde{a}_1) - w_2^k(2\tilde{a}_{i2} - \gamma q_i^\top \tilde{a}_2)\big),\\
&x^k \in P_4: \ x_i^{k+1} = (1-\eta_k)x_i^k + \eta_k w_1^k a_{i1} + \eta_k \max\big(0,\\
&(57d)\qquad\qquad x_i^k - \gamma q_i^\top x^k - w_1^k(2a_{i1} - \gamma q_i^\top a_1)\big),
\end{aligned}
$$

where q_i is the ith column of Q. After updating x_i, we renew $w_j^{k+1} = w_j^k + a_{ij}(x_i^{k+1} - x_i^k), j = 1, 2$. Note that checking x^k in some P_j requires only $O(1)$ operations by using w_1 and w_2, so the coordinate update in (57) is inexpensive.

5.4. Distributed Computing

5.4.1. Network. Consider that m worker agents and one master agent form a star-shaped network. The $m+1$ agents collaboratively solve the consensus problem:

$$\underset{x_c}{\text{minimize}} \sum_{i=1}^{m} f_i(x_c),$$

where $x_c \in R^d$ is the common variable and each proximable function f_i is held privately by agent i. The problem can be reformulated as

$$\underset{x_1,\cdots,x_m,y\in\mathbb{R}^d}{\text{minimize}} F(x) := \sum_{i=1}^{m} f_i(x_i), \quad \text{subject to } x_i = y, \tag{58}$$

which has the KKT condition:

$$0 \in \underbrace{\begin{bmatrix} \partial F(x) \\ 0 \\ 0 \end{bmatrix}}_{\text{operator } \mathcal{A}} + \underbrace{\begin{bmatrix} 0 & 0 & I \\ 0 & 0 & -e^T \\ I & -e & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ s \end{bmatrix}}_{\text{operator } \mathcal{C}}. \tag{59}$$

Applying the FBFS scheme (18) to (59) yields the following full update:

$$x_i^{k+1} = \mathbf{prox}_{\gamma f_i}(x_i^k - \gamma s_i^k) + \gamma^2 x_i^k - \gamma^2 y^k - 2\gamma s_i^k, \tag{60a}$$

$$y^{k+1} = (1 + m\gamma^2)y^k + 3\gamma \sum_j s_j^k - \gamma^2 \sum_j x_j^k, \tag{60b}$$

$$s_i^{k+1} = s_i^k - 2\gamma x_i^k - \gamma \mathbf{prox}_{\gamma f_i}(x_i^k - \gamma s_i^k) + 3\gamma y^k + \gamma^2 \sum_j s_j^k, \tag{60c}$$

where (60a) and (60c) are applied to all $i = 1, \ldots, m$. Hence, for each i, we group x_i and s_i together and assign them on agent i. We let the master agent maintain $\sum_j s_j$ and $\sum_j x_j$. Therefore, in the FBFS coordinate update, updating any (x_i, s_i) needs only y and $\sum_j s_j$ from the master agent, and updating y is done on the master agent.

5.5. Dimension Reduction

5.5.1. Nonnegative Matrix Factorization. Nonnegative matrix factorization (NMF) is an important dimension reduction method for nonnegative data. It was proposed by Paatero and his coworkers in [48]. Given a nonnegative matrix $A \in \mathbb{R}_+^{p\times n}$, NMF aims at finding two nonnegative matrices $W \in \mathbb{R}_+^{p\times r}$ and $H \in \mathbb{R}_+^{n\times r}$ such that $WH^\top \approx A$, where r is user-specified depending on the applications, and usually $r \ll \min(p, n)$. A widely used model is

$$\begin{aligned} &\underset{W,H}{\text{minimize}}\, F(W, H) := \frac{1}{2}\|WH^\top - A\|_F^2, \\ &\text{subject to } W \in \mathbb{R}_+^{p\times r},\ H \in \mathbb{R}_+^{n\times r}. \end{aligned} \tag{61}$$

Applying the projected gradient method (21) to (61), we have

$$W^{k+1} = \max\left(0, W^k - \eta_k \nabla_W F(W^k, H^k)\right), \tag{62a}$$

$$H^{k+1} = \max\left(0, H^k - \eta_k \nabla_H F(W^k, H^k)\right). \tag{62b}$$

In general, we do not know the Lipschitz constant of ∇F, so we have to choose η_k by line search such that the Armijo condition is satisfied.

Partitioning the variables into $2r$ block coordinates: $(w_1, \ldots, w_r, h_1, \ldots, h_r)$, we can apply the coordinate update based on the projected-gradient method:

$$\text{(63a)} \quad w_{i_k}^{k+1} = \max\left(0, w_{i_k}^k - \eta_k \nabla_{w_{i_k}} F(W^k, H^k)\right), \qquad \text{if } 1 \le i_k \le r,$$
$$\text{(63b)} \quad h_{i_k-r}^{k+1} = \max\left(0, h_{i_k-r}^k - \eta_k \nabla_{h_{i_k-r}} F(W^k, H^k)\right), \quad \text{if } r+1 \le i_k \le 2r,$$

where i_k is the selected coordinate at the k-th iteration. It is easy to see that $\nabla_{w_i} F(W^k, H^k)$ and $\nabla_{h_i} F(W^k, H^k)$ are both Lipschitz continuous with constants $\|h_i^k\|_2^2$ and $\|w_i^k\|_2^2$ respectively. Hence, we can set

$$\eta_k = \begin{cases} \frac{1}{\|h_{i_k}^k\|_2^2}, & \text{if } 1 \le i_k \le r, \\ \frac{1}{\|w_{i_k-r}^k\|_2^2}, & \text{if } r+1 \le i_k \le 2r. \end{cases}$$

However, it is possible to have $w_i^k = 0$ or $h_i^k = 0$ for some i and k, and thus the setting in the above formula may have trouble of being divided by zero. To overcome this problem, one can first modify the problem (61) by restricting W to have unit-norm columns and then apply the coordinate update method in (63). Note that the modification does not change the optimal value since $WH^\top = (WD)(HD^{-1})^\top$ for any $r \times r$ invertible diagonal matrix D. We refer the readers to [79] for more details.

Note that

$$\nabla_W F(W, H) = (WH^\top - A)H, \quad \nabla_H F(W, H) = (WH^\top - A)^\top W$$

and

$$\nabla_{w_i} F(W, H) = (WH^\top - A)h_i, \quad \nabla_{h_i} F(W, H) = (WH^\top - A)^\top w_i, \ \forall i.$$

Therefore, the coordinate updates given in (63) are computationally worthy (by maintaining the residual $W^k(H^k)^\top - A$).

5.6. Stylized Optimization

5.6.1. Second-Order Cone Programming. SOCP extends LP by incorporating second-order cones. A second-order cone in $\mathbb{R}^n$ is

$$Q = \left\{(x_1, x_2, \ldots, x_n) \in \mathbb{R}^n : \|(x_2, \ldots, x_n)\|_2 \le x_1\right\}.$$

Given a point $v \in \mathbb{R}^n$, let $\rho_1^v := \|(v_2, \ldots, v_n)\|_2$ and $\rho_2^v := \frac{1}{2}(v_1 + \rho_1^v)$. Then, the projection of v to Q returns 0 if $v_1 < -\rho_1$, returns v if $v_1 \ge \rho_1$, and

returns $(\rho_2^v, \frac{\rho_2^v}{\rho_1^v} \cdot (v_2, \ldots, v_n))$ otherwise. Therefore, if we define the scalar couple:

$$(\xi_1^v, \xi_2^v) = \begin{cases} (0,0), & v_1 < -\rho_2^v, \\ (1,1), & v_1 \geq \rho_2^v, \\ \left(\rho_2^v, \frac{\rho_2^v}{\rho_1^v}\right), & \text{otherwise}, \end{cases}$$

then we have $u = \mathbf{proj}_Q(v) = \left(\xi_1^v v_1,\, \xi_2^v \cdot (v_2, \ldots, v_n)\right)$. Based on this, we have

Proposition 2. *1. Let $v \in \mathbb{R}^n$ and $v^+ := v + \nu e_i$ for any $\nu \in \mathbb{R}$. Then, given $\rho_1^v, \rho_2^v, \xi_1^v, \xi_2^v$ defined above, it takes $O(1)$ operations to obtain $\rho_1^{v^+}, \rho_2^{v^+}, \xi_1^{v^+}, \xi_2^{v^+}$.*

2. Let $v \in \mathbb{R}^n$ and $A = [a_1\ A_2] \in \mathbb{R}^{m\times n}$, where $a_1 \in \mathbb{R}^m, A_2 \in \mathbb{R}^{m\times(n-1)}$. Given $\rho_1^v, \rho_2^v, \xi_1^v, \xi_2^v$, we have

$$A(2 \cdot \mathbf{proj}_Q(v) - v) = ((2\xi_1^v - 1)v_1) \cdot a_1 + (2\xi_2^v - 1) \cdot A_2(v_2, \ldots, v_n)^T.$$

By the proposition, if $\mathcal{T}_1$ is an affine operator, then in the composition $\mathcal{T}_1 \circ \mathbf{proj}_Q$, the computation of $\mathbf{proj}_Q$ is cheap as long as we maintain $\rho_1^v, \rho_2^v, \xi_1^v, \xi_2^v$.

Given $x, c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, and $A \in \mathbb{R}^{m\times n}$, the standard-form SOCP is

$$\text{(64a)} \qquad \underset{x}{\text{minimize}}\ c^\top x, \quad \text{subject to } Ax = b,$$

$$\text{(64b)} \qquad x \in X = Q_1 \times \cdots \times Q_{\bar{n}},$$

where each Q_i is a second-order cone, and $\bar{n} \neq n$ in general. The problem (64) is equivalent to

$$\underset{x}{\text{minimize}}\ \left(c^\top x + \iota_{A\cdot=b}(x)\right) + \iota_X(x),$$

to which we can apply the DRS iteration $z^{k+1} = \mathcal{T}_{\text{DRS}}(z^k)$ (see (16)), in which $\mathcal{J}_1 = \mathbf{proj}_X$ and $\mathcal{T}_2$ is a linear operator given by

$$\mathcal{J}_2(x) = \arg\min_y\ c^\top y + \frac{1}{2\gamma}\|y - x\|^2 \quad \text{subject to } Ay = b.$$

Assume that the matrix A has full row-rank (otherwise, $Ax = b$ has either redundant rows or no solution). Then, in (16), we have $\mathcal{R}_2(x) = Bx + d$, where $B := I - 2A^\top(AA^\top)^{-1}A$ and $d := 2A^\top(AA^\top)^{-1}(b + \gamma Ac) - 2\gamma c$.

It is easy to apply coordinate updates to $z^{k+1} = \mathcal{T}_{\text{DRS}}(z^k)$ following Proposition 2. Specifically, by maintaining the scalars $\rho_1^v, \rho_2^v, \xi_1^v, \xi_2^v$ for each

$v = x_i \in Q_i$ during coordinate updates, the computation of the projection can be completely avoided. We pre-compute $(AA^\top)^{-1}$ and cache the matrix B and vector d. Then, $\mathcal{T}_{\text{DRS}}$ is CF.

It is trivial to extend this method for SOCPs with a quadratic objective:

$$\underset{x}{\text{minimize}} \ c^\top x + \frac{1}{2} x^\top C x, \quad \text{subject to } Ax = b, \ x \in X = Q_1 \times \cdots \times Q_{\bar{n}},$$

because $\mathcal{J}_2$ is still linear. Clearly, this method applies to linear programs as they are special SOCPs.

Note that many LPs and SOCPs have sparse matrices A, which deserve further investigation. In particular, we may prefer not to form $(AA^\top)^{-1}$ and use the results in §4.2 instead.

6. Numerical Experiments

We illustrate the behavior of coordinate update algorithms for solving portfolio optimization, image processing, and machine learning problems. Our primary goal is to show the efficiency of coordinate update compared to the corresponding full update algorithms. We will also illustrate that asynchronous parallel coordinate update algorithms are more scalable than their synchronous parallel counterparts.

Our first two experiments run on Mac OSX 10.9 with 2.4 GHz Intel Core i5 and 8 Gigabytes of RAM. The experiments were coded in Matlab. The sparse logistic regression experiment runs on 1 to 16 threads on a machine with two 2.5Ghz 10-core Intel Xeon E5-2670v2 (20 cores in total) and 64 Gigabytes of RAM. The experiment was coded in C++ with OpenMP enabled. We use the Eigen library[5] for sparse matrix operations.

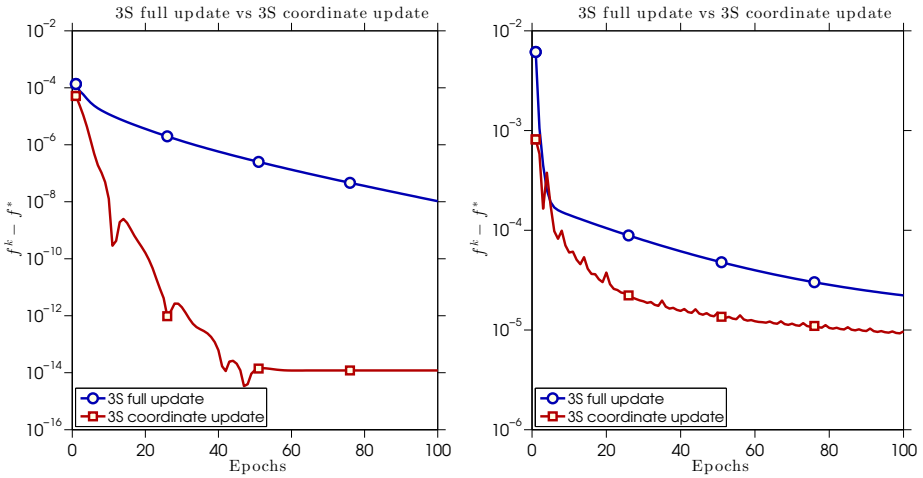
6.1. Portfolio Optimization

In this subsection, we compare the performance of the 3S splitting scheme (56) with the corresponding coordinate update algorithm (57) for solving the portfolio optimization problem (55). In this problem, our goal is to distribute our investment resources to all the assets so that the investment risk is minimized and the expected return is greater than r. This test uses two datasets, which are summarized in Table 3. The NASDAQ dataset is collected through Yahoo! Finance. We collected one year (from 10/31/2014 to 10/31/2015) of historical closing prices for 2730 stocks.

[5] http://eigen.tuxfamily.org

	Synthetic data	NASDAQ data
Num. assets (N)	1000	2730
Expected return rate	0.02	0.02
Asset return rate	`3 * rand(N, 1) - 1`	mean of 30 days return rate
Risk	covariance matrix + $0.01 \cdot I$	positive definite matrix

Table 3: Two datasets for portfolio optimization



(a) Synthesis dataset (b) NASDAQ dataset

Figure 3: Compare the convergence of 3S full update with 3S coordinate update algorithms.

In our numerical experiments, for comparison purposes, we first obtain a high accurate solution by solving (55) with an interior point solver. For both full update and coordinate update, η_k is set to 0.8. However, we use different γ. For 3S full update, we used the step size parameter $\gamma_1 = \frac{2}{\|Q\|_2}$, and for 3S coordinate update, $\gamma_2 = \frac{2}{\max\{Q_{11},\ldots,Q_{NN}\}}$. In general, coordinate update can benefit from more relaxed parameters. The results are reported in Figure 3. We can observe that the coordinate update method converges much faster than the 3S method for the synthetic data. This is due to the fact that γ_2 is much larger than γ_1. However, for the NASDAQ dataset, $\gamma_1 \approx \gamma_2$, so 3S coordinate update is only moderately faster than 3S full update.

6.2. Computed Tomography Image Reconstruction

We compare the performance of algorithm (53) and its corresponding coordinate version on Computed Tomography (CT) image reconstruction. We generate a thorax phantom of size 284×284 to simulate spectral CT measurements. We then apply the Siddon's algorithm [63] to form the sinogram data. There are 90 parallel beam projections and, for each projection, there are 362 measurements. Then the sinogram data is corrupted with Gaussian noise. We formulate the image reconstruction problem in the form of (49). The primal-dual full update corresponds to (53). For coordinate update, the block size for x is set to 284, which corresponds to a column of the image. The dual variables s, t are also partitioned into 284 blocks accordingly. A block of x and the corresponding blocks of s and t are bundled together as a single block. In each iteration, a bundled block is randomly chosen and updated. The reconstruction results are shown in Figure 4. After 100 epochs, the image recovered by the coordinate version is better than that by (53). As shown in Figure 4d, the coordinate version converges faster than (53).

6.3. ℓ_1 Regularized Logistic Regression

In this subsection, we compare the performance of sync-parallel coordinate update and the async-parallel coordinate update for solving the sparse logistic regression problem

$$\underset{x\in\mathbb{R}^n}{\text{minimize}}\ \lambda\|x\|_1 + \frac{1}{N}\sum_{i=1}^{N}\log\left(1+\exp(-b_i\cdot a_i^\top x)\right), \tag{65}$$

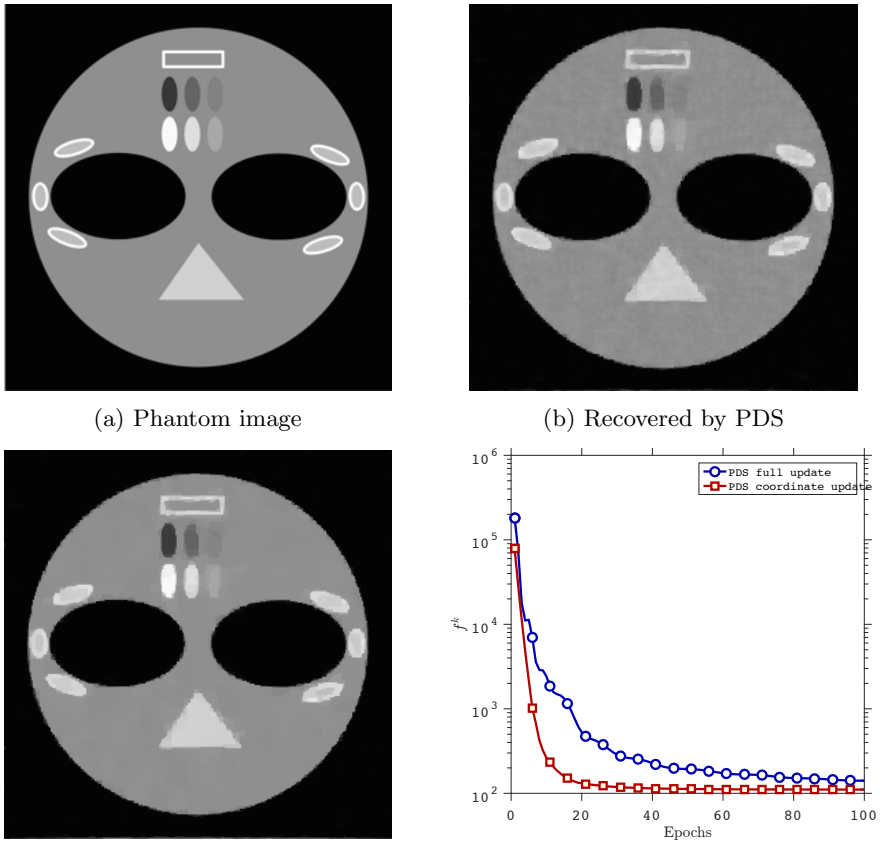
where $\{(a_i, b_i)\}_{i=1}^N$ is the set of sample-label pairs with $b_i \in \{1, -1\}$, $\lambda = 0.0001$, and n and N represent the numbers of features and samples, respectively. This test uses the datasets[6]: real-sim and news20, which are summarized in Table 4.

Table 4: Two datasets for sparse logistic regression

Name	# samples	# features
real-sim	72, 309	20, 958
news20	19,996	1,355,191

We let each coordinate hold roughly 50 features. Since the total number of features is not divisible by 50, some coordinates have 51 features. We let

[6] http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/



(a) Phantom image (b) Recovered by PDS

(c) Recovered by PDS coord (d) Objective function value

Figure 4: CT image reconstruction.

each thread draw a coordinate uniformly at random at each iteration. We stop all the tests after 10 epochs since they have nearly identical progress per epoch. The step size is set to $\eta_k = 0.9$, $\forall k$. Let $A = [a_1, \ldots, a_N]^\top$ and $b = [b_1, ..., b_N]^\top$. In global memory, we store A, b and x. We also store the product Ax in global memory so that the forward step can be efficiently computed. Whenever a coordinate of x gets updated, Ax is immediately updated at a low cost. Note that if Ax is *not* stored in global memory, every coordinate update will have to compute Ax from scratch, which involves the entire x and will be very expensive.

Table 5 gives the running times of the sync-parallel and async-parallel implementations on the two datasets. We can observe that async-parallel

achieves almost-linear speedup, but sync-parallel scales very poorly as we explain below.

In the sync-parallel implementation, all the running cores have to wait for the last core to finish an iteration, and therefore if a core has a large load, it slows down the iteration. Although every core is (randomly) assigned to roughly the same number of features (either 50 or 51 components of x) at each iteration, their a_i's have very different numbers of nonzeros, and the core with the largest number of nonzeros is the slowest. (Sparse matrix computation is used for both datasets, which are very large.) As more cores are used, despite that they altogether do more work at each iteration, the per-iteration time reduces as the slowest core tends to be slower. On the other hand, async-parallel coordinate update does not suffer from the load imbalance. Its performance grows nearly linear with the number of cores.

Finally, we have observed that the progress toward solving (65) is mainly a function of the number of epochs and does not change appreciably when the number of cores increases or between sync-parallel and async-parallel. Therefore, we always stop at 10 epochs.

# cores	real-sim				news20			
	time (s)		speedup		time (s)		speedup	
	async	sync	async	sync	async	sync	async	sync
1	81.6	82.1	1.0	1.0	591.1	591.3	1.0	1.0
2	45.9	80.6	1.8	1.0	304.2	590.1	1.9	1.0
4	21.6	63.0	3.8	1.3	150.4	557.0	3.9	1.1
8	16.1	61.4	5.1	1.3	78.3	525.1	7.5	1.1
16	7.1	46.4	11.5	1.8	41.6	493.2	14.2	1.2

Table 5: Running times of async-parallel and sync-parallel FBS implementations for ℓ_1 regularized logistic regression on two datasets. Sync-parallel has very poor speedup due to the large distribution of coordinate sparsity and thus the large load imbalance across cores.

7. Conclusions

We have presented a coordinate update method for fixed-point iterations, which updates one coordinate (or a few variables) at every iteration and can be applied to solve linear systems, optimization problems, saddle point problems, variational inequalities, and so on. We proposed a new concept called CF operator. When an operator is CF, its coordinate update is computationally worthy and often preferable over the full update method, in particular in

a parallel computing setting. We gave examples of CF operators and also discussed how the properties can be preserved by composing two or more such operators such as in operator splitting and primal-dual splitting schemes. In addition, we have developed CF algorithms for problems arising in several different areas including machine learning, imaging, finance, and distributed computing. Numerical experiments on portfolio optimization, logistic regression, and CT imaging have been provided to demonstrate the superiority of CF methods over their counterparts that updates all coordinates at every iteration.

References

[1] Attouch, H., Bolte, J., Redont, P., Soubeyran, A.: Proximal alternating minimization and projection methods for nonconvex problems: An approach based on the kurdyka-lojasiewicz inequality. Mathematics of Operations Research **35**(2), 438–457 (2010)

[2] Bahi, J., Miellou, J.C., Rhofir, K.: Asynchronous multisplitting methods for nonlinear fixed point problems. Numerical Algorithms **15**(3-4), 315–345 (1997)

[3] Baudet, G.M.: Asynchronous iterative methods for multiprocessors. J. ACM **25**(2), 226–244 (1978).

[4] Bauschke, H.H., Borwein, J.M.: On the convergence of von neumann's alternating projection algorithm for two sets. Set-Valued Analysis **1**(2), 185–212 (1993)

[5] Bauschke, H.H., Combettes, P.L.: Convex analysis and monotone operator theory in Hilbert spaces. Springer Science & Business Media (2011)

[6] Bauschke, H.H., Combettes, P.L.: Convex analysis and monotone operator theory in Hilbert spaces. Springer Science & Business Media (2011)

[7] Baz, D.E., Frommer, A., Spiteri, P.: Asynchronous iterations with flexible communication: contracting operators. Journal of Computational and Applied Mathematics **176**(1), 91 – 103 (2005)

[8] Baz, D.E., Gazen, D., Jarraya, M., Spiteri, P., Miellou, J.: Flexible communication for parallel asynchronous methods with application to a nonlinear optimization problem. In: E. D'Hollander, F. Peters, G. Joubert, U. Trottenberg, R. Volpel (eds.) Parallel ComputingFundamen-

tals, Applications and New Directions, *Advances in Parallel Computing*, vol. 12, pp. 429 – 436. North-Holland (1998)

[9] Beck, A., Tetruashvili, L.: On the convergence of block coordinate descent type methods. SIAM Journal on Optimization **23**(4), 2037–2060 (2013)

[10] Bertsekas, D.P.: Distributed asynchronous computation of fixed points. Mathematical Programming **27**(1), 107–120 (1983)

[11] Bertsekas, D.P.: Nonlinear programming (1999)

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

[13] Bolte, J., Sabach, S., Teboulle, M.: Proximal alternating linearized minimization for nonconvex and nonsmooth problems. Mathematical Programming **146**(1-2), 459–494 (2014)

[14] Bradley, J.K., Kyrola, A., Bickson, D., Guestrin, C.: Parallel coordinate descent for l1-regularized loss minimization. arXiv preprint arXiv:1105.5379 (2011)

[15] Briceño-Arias, L.M.: Forward-douglas–rachford splitting and forward-partial inverse method for solving monotone inclusions. Optimization **64**(5), 1239–1261 (2015)

[16] Briceno-Arias, L.M., Combettes, P.L.: Monotone operator methods for nash equilibria in non-potential games. In: Computational and Analytical Mathematics, pp. 143–159. Springer (2013)

[17] Chazan, D., Miranker, W.: Chaotic relaxation. Linear algebra and its applications **2**(2), 199–222 (1969)

[18] Combettes, P.L., Condat, L., Pesquet, J.C., Vu, B.C.: A forward-backward view of some primal-dual optimization methods in image recovery. In: Image Processing (ICIP), 2014 IEEE International Conference on, pp. 4141–4145. IEEE (2014)

[19] Combettes, P.L., Pesquet, J.C.: Stochastic quasi-fejr block-coordinate fixed point iterations with random sweeping. SIAM Journal on Optimization **25**(2), 1221–1248 (2015).

[20] Condat, L.: A primal–dual splitting method for convex optimization involving lipschitzian, proximable and linear composite terms. Journal of Optimization Theory and Applications **158**(2), 460–479 (2013)

[21] Dang, C.D., Lan, G.: Stochastic Block Mirror Descent Methods for Nonsmooth and Stochastic Optimization. SIAM Journal on Optimization **25**(2), 856–881 (2015).

[22] Davis, D.: Convergence rate analysis of primal-dual splitting schemes. arXiv preprint arXiv:1408.4419 (2014)

[23] Davis, D.: An $o(n \log(n))$ algorithm for projecting onto the ordered weighted ℓ_1 norm ball. arXiv preprint arXiv:1505.00870 (2015)

[24] Davis, D., Yin, W.: A three-operator splitting scheme and its optimization applications. arXiv preprint arXiv:1504.01032 (2015)

[25] Dhillon, I.S., Ravikumar, P.K., Tewari, A.: Nearest neighbor based greedy coordinate descent. In: Advances in Neural Information Processing Systems, pp. 2160–2168 (2011)

[26] Douglas, J., Rachford, H.H.: On the numerical solution of heat conduction problems in two and three space variables. Transactions of the American mathematical Society pp. 421–439 (1956)

[27] El Baz, D., Gazen, D., Jarraya, M., Spiteri, P., Miellou, J.C.: Flexible communication for parallel asynchronous methods with application to a nonlinear optimization problem. Advances in Parallel Computing **12**, 429–436 (1998)

[28] Fercoq, O., Bianchi, P.: A coordinate descent primal-dual algorithm with large step size and possibly non separable functions. arXiv preprint arXiv:1508.04625 (2015)

[29] Frommer, A., Szyld, D.B.: On asynchronous iterations. Journal of Computational and Applied Mathematics **123**(12), 201 – 216 (2000). Numerical Analysis 2000. Vol. III: Linear Algebra

[30] Gabay, D., Mercier, B.: A dual algorithm for the solution of nonlinear variational problems via finite element approximation. Computers & Mathematics with Applications **2**(1), 17–40 (1976)

[31] Glowinski, R., Marroco, A.: Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de dirichlet non linéaires. Revue française d'automatique, informatique, recherche opérationnelle. Analyse numérique **9**(2), 41–76 (1975)

[32] Grippo, L., Sciandrone, M.: On the convergence of the block nonlinear gauss-seidel method under convex constraints. Operations Research Letters **26**(3), 127–136 (2000)

[33] Han, S.: A successive projection method. Mathematical Programming **40**(1), 1–14 (1988)

[34] Hildreth, C.: A quadratic programming procedure. Naval research logistics quarterly **4**(1), 79–85 (1957)

[35] Hong, M., Wang, X., Razaviyayn, M., Luo, Z.Q.: Iteration complexity analysis of block coordinate descent methods. arXiv preprint arXiv:1310.6957v2 (2015)

[36] Hsieh, C.J., Yu, H.F., Dhillon, I.S.: Passcode: Parallel asynchronous stochastic dual co-ordinate descent. arXiv preprint arXiv:1504.01365 (2015)

[37] Krasnosel'skii, M.A.: Two remarks on the method of successive approximations. Uspekhi Matematicheskikh Nauk **10**(1), 123–127 (1955)

[38] Lebedev, V., Tynjanskiı, N.: Duality theory of concave-convex games. In: Soviet Math. Dokl, vol. 8, pp. 752–756 (1967)

[39] Li, Y., Osher, S.: Coordinate descent optimization for 1 minimization with application to compressed sensing; a greedy algorithm. Inverse Problems and Imaging **3**(3), 487–503 (2009)

[40] Liu, J., Wright, S.J.: Asynchronous stochastic coordinate descent: Parallelism and convergence properties. SIAM Journal on Optimization **25**(1), 351–376 (2015)

[41] Liu, J., Wright, S.J., Ré, C., Bittorf, V., Sridhar, S.: An asynchronous parallel stochastic coordinate descent algorithm. Journal of Machine Learning Research **16**, 285–322 (2015)

[42] Lu, Z., Xiao, L.: On the complexity analysis of randomized block-coordinate descent methods. Mathematical Programming **152**(1-2), 615–642 (2015).

[43] Luo, Z.Q., Tseng, P.: On the convergence of the coordinate descent method for convex differentiable minimization. Journal of Optimization Theory and Applications **72**(1), 7–35 (1992)

[44] McLinden, L.: An extension of fenchels duality theorem to saddle functions and dual minimax problems. Pacific Journal of Mathematics **50**(1), 135–158 (1974)

[45] Nedić, A., Bertsekas, D.P., Borkar, V.S.: Distributed asynchronous incremental subgradient methods. Studies in Computational Mathematics **8**, 381–407 (2001)

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

[47] O'Connor, D., Vandenberghe, L.: Primal-dual decomposition by operator splitting and applications to image deblurring. SIAM Journal on Imaging Sciences **7**(3), 1724–1754 (2014)

[48] Paatero, P., Tapper, U.: Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. Environmetrics **5**(2), 111–126 (1994)

[49] Passty, G.B.: Ergodic convergence to a zero of the sum of monotone operators in hilbert space. Journal of Mathematical Analysis and Applications **72**(2), 383–390 (1979)

[50] Peaceman, D.W., Rachford Jr, H.H.: The numerical solution of parabolic and elliptic differential equations. Journal of the Society for Industrial and Applied Mathematics **3**(1), 28–41 (1955)

[51] Peng, Z., Xu, Y., Yan, M., Yin, W.: ARock: an Algorithmic Framework for Asynchronous Parallel Coordinate Updates. ArXiv e-prints (2015)

[52] Peng, Z., Yan, M., Yin, W.: Parallel and distributed sparse optimization. In: Signals, Systems and Computers, 2013 Asilomar Conference on, pp. 659–646. IEEE (2013)

[53] Pesquet, J.C., Repetti, A.: A class of randomized primal-dual algorithms for distributed optimization. arXiv preprint arXiv:1406.6404 (2014)

[54] Polak, E., Sargent, R., Sebastian, D.: On the convergence of sequential minimization algorithms. Journal of Optimization Theory and Applications **12**(6), 567–575 (1973)

[55] Razaviyayn, M., Hong, M., Luo, Z.Q.: A unified convergence analysis of block successive minimization methods for nonsmooth optimization. SIAM Journal on Optimization **23**(2), 1126–1153 (2013)

[56] Recht, B., Re, C., Wright, S., Niu, F.: Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In: Advances in Neural Information Processing Systems, pp. 693–701 (2011)

[57] Repetti, A., Chouzenoux, E., Pesquet, J.C.: A random block-coordinate primal-dual proximal algorithm with application to 3d mesh denoising. In: Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on, pp. 3561–3565. IEEE (2015)

[58] Richtárik, P., Takáč, M.: Parallel coordinate descent methods for big data optimization. Mathematical Programming pp. 1–52 (2012)

[59] Richtárik, P., Takáč, M.: Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. Mathematical Programming **144**(1-2), 1–38 (2014)

[60] Rockafellar, R.T.: Convex analysis, vol. 28. Princeton university press (1997)

[61] Schmidt, M., Friedlander, M.: Coordinate descent converges faster with the gauss-southwell rule than random selection. In: NIPS OPT-ML workshop (2014)

[62] Scholkopf, B., Smola, A.J.: Learning with kernels: support vector machines, regularization, optimization, and beyond. MIT press (2001)

[63] Siddon, R.L.: Fast calculation of the exact radiological path for a three-dimensional ct array. Medical Physics **12**(2), 252–255 (1985).

[64] Strikwerda, J.C.: A probabilistic analysis of asynchronous iteration. Linear Algebra and its Applications **349**(13), 125 – 154 (2002)

[65] Tseng, P.: Applications of a splitting algorithm to decomposition in convex programming and variational inequalities. SIAM Journal on Control and Optimization **29**(1), 119–138 (1991)

[66] Tseng, P.: On the rate of convergence of a partially asynchronous gradient projection algorithm. SIAM Journal on Optimization **1**(4), 603–619 (1991)

[67] Tseng, P.: Dual coordinate ascent methods for non-strictly convex minimization. Mathematical Programming **59**(1), 231–247 (1993)

[68] Tseng, P.: A modified forward-backward splitting method for maximal monotone mappings. SIAM J. Control and Optimization **38**(2), 431–446 (2000).

[69] Tseng, P.: Convergence of a block coordinate descent method for nondifferentiable minimization. Journal of Optimization Theory and Applications **109**(3), 475–494 (2001)

[70] Tseng, P., Yun, S.: Block-coordinate gradient descent method for linearly constrained nonsmooth separable optimization. Journal of optimization theory and applications **140**(3), 513–535 (2009)

[71] Tseng, P., Yun, S.: A coordinate gradient descent method for nonsmooth separable minimization. Mathematical Programming **117**(1-2), 387–423 (2009)

[72] Von Neumann, J.: On rings of operators. reduction theory. Annals of Mathematics pp. 401–485 (1949)

[73] Vũ, B.C.: A splitting algorithm for dual monotone inclusions involving cocoercive operators. Advances in Computational Mathematics **38**(3), 667–681 (2013)

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

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

[76] Wu, T.T., Lange, K.: Coordinate descent algorithms for lasso penalized regression. The Annals of Applied Statistics pp. 224–244 (2008)

[77] Xu, Y.: Alternating proximal gradient method for sparse nonnegative Tucker decomposition. Mathematical Programming Computation **7**(1), 39–70 (2015).

[78] Xu, Y., Yin, W.: A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion. SIAM Journal on imaging sciences **6**(3), 1758–1789 (2013)

[79] Xu, Y., Yin, W.: A globally convergent algorithm for nonconvex optimization based on block coordinate update. arXiv preprint arXiv:1410.1386 (2014)

[80] Xu, Y., Yin, W.: Block Stochastic Gradient Iteration for Convex and Nonconvex Optimization. SIAM Journal on Optimization **25**(3), 1686–1716 (2015).

[81] Yuan, M., Lin, Y.: Model selection and estimation in regression with grouped variables. Journal of the Royal Statistical Society: Series B (Statistical Methodology) **68**(1), 49–67 (2006)

[82] Zadeh, N.: A note on the cyclic coordinate ascent method. Management Science **16**(9), 642–644 (1970)

Appendix A. Some Key Concepts of Operator

In this section, we go over a few key concepts in monotone operator theory and operator splitting theory.

Definition 8 (monotone operator)**.** *A* set-valued *operator* $\mathcal{T} : \mathbb{H} \rightrightarrows \mathbb{H}$ *is* monotone *if* $\langle x - y, u - v\rangle \geq 0,\ \forall x, y \in \mathbb{H},\ u \in \mathcal{T}x,\ v \in \mathcal{T}y$. *Furthermore,* $\mathcal{T}$ *is* maximally monotone *if its graph* $\mathrm{Grph}(T) = \{(x, u) \in \mathbb{H} \times \mathbb{H} : u \in \mathcal{T}x\}$ *is not strictly contained in the graph of any other monotone operator.*

Example 18. *An important maximally monotone operator is the subdifferential* ∂f *of a closed proper convex function* f.

Definition 9 (nonexpansive operator)**.** *An operator* $\mathcal{T} : \mathbb{H} \to \mathbb{H}$ *is* nonexpansive *if* $\|\mathcal{T}x - \mathcal{T}y\| \leq \|x - y\|,\ \forall x, y \in \mathbb{H}$. *We say* $\mathcal{T}$ *is averaged, or* α*-averaged, if there is one nonexpansive operator* $\mathcal{R}$ *such that* $\mathcal{T} = (1 - \alpha)\mathcal{I} + \alpha\mathcal{R}$ *for some* $0 < \alpha < 1$. *A* $\frac{1}{2}$*-averaged operator* $\mathcal{T}$ *is also called* firmly-nonexpansive.

By definition, a nonexpansive operator is single-valued. Let $\mathcal{T}$ be averaged. If $\mathcal{T}$ has a fixed point, the iteration (2) converges to a fixed point; otherwise, the iteration diverges unboundedly. Now let $\mathcal{T}$ be nonexpansive. The convergence is guaranteed [37] after damping: $x^{k+1} = x^k - \eta(x^k - \mathcal{T}x^k)$, for any $0 < \eta < 1$.

Example 19. *A common firmly-nonexpansive operator is the resolvent of a maximally monotone map* $\mathcal{T}$, *written as*

$$\mathcal{J}_{\mathcal{A}} := (\mathcal{I} + \mathcal{A})^{-1}. \tag{66}$$

Given $x \in \mathbb{H}$, $\mathcal{J}_{\mathcal{A}}(x) = \{y : x \in y + \mathcal{A}y\}$. *(By monotonicity of* $\mathcal{A}$, $\mathcal{J}_{\mathcal{A}}$ *is a singleton, and by maximality of* $\mathcal{A}$, $\mathcal{J}_{\mathcal{A}}(x)$ *is well defined for all* $x \in \mathbb{H}$. *) A reflective resolvent is*

$$\mathcal{R}_{\mathcal{A}} := 2\mathcal{J}_{\mathcal{A}} - \mathcal{I}. \tag{67}$$

Definition 10 (proximal map)**.** *The* proximal map *for a function* f *is a special resolvent and defined as:*

$$\mathbf{prox}_{\gamma f}(y) = \arg\min_x \big\{f(x) + \frac{1}{2\gamma}\|x - y\|^2\big\}, \tag{68}$$

where $\gamma > 0$. *The first-order variational condition of the minimization yields* $\mathbf{prox}_{\gamma f}(y) = (\mathcal{I} + \gamma\partial f)^{-1}$; *hence,* $\mathbf{prox}_{\gamma f}$ *is firmly-nonexpansive. When*

$x \in \mathbb{R}^m$ *and* $\mathbf{prox}_{\gamma f}$ *can be computed in* $O(m)$ *or* $O(m \log m)$*, we call* f proximable*.*

Examples of proximal functions include $\ell_1, \ell_2, \ell_\infty$*-norms, several matrix norms, the owl-norm [23], (piece-wise) linear functions, certain quadratic functions, and many more.*

Example 20. *A special proximal map is the projection map. Let* X *be a nonempty closed convex set, and* ι_S *be its indicator function. Minimizing* $\iota_S(x)$ *enforces* $x \in S$*, so* $\mathbf{prox}_{\gamma \iota_S}$ *reduces to the projection map* $\mathbf{proj}_S$ *for any* $\gamma > 0$*. Therefore,* $\mathbf{proj}_S$ *is also firmly nonexpansive.*

Definition 11 (β-cocoercive operator)**.** *An operator* $\mathcal{T} : \mathbb{H} \to \mathbb{H}$ *is* β-cocoercive *if* $\langle x - y, \mathcal{T}x - \mathcal{T}y \rangle \geq \beta \|\mathcal{T}x - \mathcal{T}y\|^2, \ \forall x, y \in \mathbb{H}$.

Example 21. *A special example of cocoercive operator is the gradient of a smooth function. Let* f *be a differentiable function. Then* ∇f *is* β*-Lipschitz continuous* if and only if ∇f *is* $\frac{1}{\beta}$*-cocoercive [5, Corollary 18.16].*

Appendix B. Derivation of ADMM from the DRS Update

We derive the ADMM update in (23) from the DRS update

$$s^k = \mathcal{J}_{\eta \mathcal{B}}(t^k), \tag{69a}$$

$$t^{k+1} = \left(\frac{1}{2}(2\mathcal{J}_{\eta \mathcal{A}} - \mathcal{I}) \circ (2\mathcal{J}_{\eta \mathcal{B}} - \mathcal{I}) + \frac{1}{2}\mathcal{I} \right)(t^k), \tag{69b}$$

where $\mathcal{A} = -\partial f^*(-\cdot)$ and $\mathcal{B} = \partial g^*$.

Note (69a) is equivalent to $t^k \in s^k + \eta \partial g^*(s^k)$, i.e., there is $y^k \in \partial g^*(s^k)$ such that $t^k = s^k + \eta y^k$, so

$$t^k - \eta y^k = s^k \in \partial g(y^k). \tag{70}$$

In addition, (69b) can be written as

$$\begin{aligned} t^{k+1} &= \mathcal{J}_{\eta \mathcal{A}}(2s^k - t^k) + t^k - s^k \\ &= s^k + (\mathcal{J}_{\eta \mathcal{A}} - \mathcal{I})(2s^k - t^k) \\ &= s^k + (\mathcal{I} - (\mathcal{I} + \eta \partial f^*)^{-1})(t^k - 2s^k) \\ &= s^k + \eta (\eta \mathcal{I} + \partial f)^{-1}(t^k - 2s^k) \\ &= s^k + \eta (\eta \mathcal{I} + \partial f)^{-1}(\eta y^k - s^k), \end{aligned} \tag{71}$$

where in the fourth equality, we have used Moreau's Identity [60]: $(\mathcal{I} + \partial h)^{-1} + (\mathcal{I} + \partial h^*)^{-1} = \mathcal{I}$ for any closed convex function h. Let

$$x^{k+1} = (\eta\mathcal{I} + \partial f)^{-1}(\eta y^k - s^k) = (\mathcal{I} + \frac{1}{\eta}\partial f)^{-1}(y^k - \frac{1}{\eta}s^k). \tag{72}$$

Then (71) becomes

$$t^{k+1} = s^k + \eta x^{k+1},$$

and

$$s^{k+1} \overset{(70)}{=} t^{k+1} - \eta y^{k+1} = s^k + \eta x^{k+1} - \eta y^{k+1}, \tag{73}$$

which together with $s^{k+1} \in \partial g(y^{k+1})$ gives

$$y^{k+1} = (\eta\mathcal{I} + \partial g)^{-1}(t^k + \eta x^{k+1}) = (\mathcal{I} + \frac{1}{\eta}\partial g)^{-1}(x^{k+1} + \frac{1}{\eta}s^k). \tag{74}$$

Hence, from (72), (73), and (74), the ADMM update in (23) is equivalent to the DRS update in (69) with $\eta = \frac{1}{\gamma}$.

Appendix C. Representing the Condat-Vũ Algorithm as a Nonexpansive Operator

We show how to derive the Condat-Vũ algorithm (28) by applying a forward-backward operator to the optimality condition (27):

$$0 \in \underbrace{\begin{bmatrix}\nabla f(x)\\ 0\end{bmatrix}}_{\text{operator } \mathcal{A}} + \underbrace{\begin{bmatrix}\partial g(x)\\ \partial h^*(s)\end{bmatrix} + \begin{bmatrix}0 & A^\top\\ -A & 0\end{bmatrix}\begin{bmatrix}x\\ s\end{bmatrix}}_{\text{operator } \mathcal{B}}. \tag{75}$$

It can be written as $0 \in \mathcal{A}z + \mathcal{B}z$ after we define $z = \begin{bmatrix}x\\ s\end{bmatrix}$. Let M be a symmetric positive definite matrix, we have

$$\begin{aligned}
&0 \in \mathcal{A}z + \mathcal{B}z\\
\Leftrightarrow &Mz - \mathcal{A}z \in Mz + \mathcal{B}z\\
\Leftrightarrow &z - M^{-1}\mathcal{A}z \in z + M^{-1}\mathcal{B}z\\
\Leftrightarrow &z = (\mathcal{I} + M^{-1}\mathcal{B})^{-1} \circ (\mathcal{I} - M^{-1}\mathcal{A})z.
\end{aligned}$$

Convergence and other results can be found in [22]. The last equivalent relation is due to $M^{-1}\mathcal{B}$ being a maximally monotone operator. We let

$$M = \begin{bmatrix} \frac{1}{\eta} I & A^\top \\ A & \frac{1}{\gamma} I \end{bmatrix} \succ 0$$

and iterate

$$z^{k+1} = \mathcal{T} z^k = (\mathcal{I} + M^{-1}\mathcal{B})^{-1} \circ (\mathcal{I} - M^{-1}\mathcal{A}) z^k.$$

We have $Mz^{k+1} + \mathcal{B}z^{k+1} = Mz^k - \mathcal{A}z^k$:

$$\begin{cases} \frac{1}{\eta} x^k + A^\top s^k - \nabla f(x^k) & \in \frac{1}{\eta} x^{k+1} + A^\top s^{k+1} + A^\top s^{k+1} + \nabla g(x^{k+1}), \\ \frac{1}{\gamma} s^k + A\, x^k & \in \frac{1}{\gamma} s^{k+1} + A\, x^{k+1} - A\, x^{k+1} + \nabla h^*(s^{k+1}), \end{cases}$$

which is equivalent to

$$\begin{cases} s^{k+1} = \mathbf{prox}_{\gamma h^*}(s^k + \gamma A x^k), \\ x^{k+1} = \mathbf{prox}_{\eta g}(x^k - \eta(\nabla f(x^k) + A^\top(2s^{k+1} - s^k))). \end{cases}$$

Now we derived the Condat-Vũ algorithm. With proper choices of η and γ, the forward-backward operator $\mathcal{T} = (\mathcal{I} + M^{-1}\mathcal{B})^{-1} \circ (\mathcal{I} - M^{-1}\mathcal{A})$ can be shown to be α-averaged if we use the inner product $\langle z_1, z_2 \rangle_M = z_1^\top M z_2$ and norm $\|z\|_M = \sqrt{z^\top M z}$ on the space of $z = \begin{bmatrix} x \\ s \end{bmatrix}$. More details can be found in [22].

If we change the matrix M to $\begin{bmatrix} \frac{1}{\eta} I & -A^\top \\ -A & \frac{1}{\gamma} I \end{bmatrix}$, the other algorithm (29) can be derived in the same way.

Appendix D. Proof of convergence for async-parallel primal dual coordinate update algorithms

We will introduce some notations first. First since $\hat{z}_i^k$ can be related to z_i^k through the interim changes applied to z_i, we let $J_i(k) \subset \{k-1, ..., k-\tau\}$ be the index set of these interim changes. Since the global counter is increased after each coordinate update, we have $J_i(k) \cap J_j(k) = \emptyset, \forall i \neq j$. Let $J(k) := \cup_i J_i(k)$ and $|J(k)|$ be the number of elements in $J(k)$.

Define $S = I - \mathcal{T}_{\mathrm{CV}}$, by Lemma 1 below, T is nonexpansive in the norm induced by a symmetric positive definite matrix M (see Appendix C) if and only if S is 1/2-cocoercive in the norm M.

Lemma 1. *(Using any norm), operator $T : \mathbb{R}^{m+p} \to \mathbb{R}^{m+p}$ is nonexpansive if and only if $S = I - T$ is 1/2-cocoercive, i.e.,*

$$\langle x - y, Sx - Sy\rangle_M \geq \frac{1}{2}\|Sx - Sy\|_M^2, \quad \forall\ x, y \in \mathbb{R}^{m+p}. \tag{76}$$

The proof is the same as that of [6, Proposition 4.33]. It is shown in [22] that with proper choice of η and γ, $\mathcal{T}_{\mathrm{CV}}$ is nonexpansive in the norm induced by M.

With the definition of S, in both Algorithm 1 and Algorithm 2 the coordinate changes when i_k us picked can be written as $z^{k+1} = \hat{z} - \frac{\eta_k}{mq_{i_k}} S_{i_k}\hat{z}^k$, where in Algorithm 1, $S_i\hat{z}^k$ is just the i-th component of $S\hat{z}^k$ and in Algorithm 2

$$S_i\hat{z}^k = \begin{bmatrix} 0 & & & & & & \\ & \ddots & & & & & \\ & & I_{\mathbb{H}_i} & & & & \\ & & & 0 & & & \\ & & & & \rho_{i,1} I_{\mathbb{G}_1} & & \\ & & & & & \ddots & \\ & & & & & & \rho_{i,p} I_{\mathbb{G}_p} \end{bmatrix} S\hat{z}^k.$$

The next lemma is due to simple calculation.

Lemma 2. *We have for both Algorithm 1 and Algorithm 2,*

$$\sum_{i=1}^{m+p} S_i\hat{z}^k = S\hat{z}^k$$

$$\sum_{i=1}^{m+p} \|s_i\hat{z}^k\|^2 \leq \|S\hat{z}^k\|^2$$

We define

$$\bar{z}^{k+1} := z^k - \eta_k S\hat{z}^k. \tag{77}$$

At last we let $p_{\min} = \min_i p_i > 0$. We state the complete theorem here:

Theorem 2. *Let Z^* be the set of optimal solutions of Problem (24) and let $(z^k)_{k\geq 0} \subset \mathbb{R}^{m+p}$ be the sequence generated by Algorithm 1 or Algorithm 2 (with proper choice of η and γ such that $\mathcal{T}_{\mathrm{CV}}$ is nonexpansive in the norm induced by M), under the following conditions:*

(i) f, g, h^* *are closed proper convex function and* ∇f *is* β*-Lipschitz continuous;*

(ii) the delay for every coordinate is bounded by a positive number τ*, i.e. for every* $1 \le i \le m+p$*,* $\hat{z}_i^k = z^{k-d_{i_k}}$ *for some* $0 \le d_{i_k} \le \tau$*;*

(iii) $\eta_k \in [\eta_{\min}, \eta_{\max}]$ *for certain* $0 < \eta_{\max} < \frac{cmp_{\min}}{2\tau\kappa\sqrt{p_{\min}}+\kappa^2}$ *and any* $0 < \eta_{\min} \le \eta_{\max}$*;*

We have $(z^k)_{k\ge 0}$ *converges to a* Z^**-valued random variable a.s..*

The proof directly follows from [51, Section 3]. Here we present the most important steps and the key modifications.

The next lemma shows that the conditional expectation of the distance between z^{k+1} and any $z^* \in \mathbf{Fix}\mathcal{T}_{\mathrm{CV}} = Z^*$ for given $\mathcal{Z}^k = \{z^0, z^1, \cdots, z^k\}$ has an upper bound that depends on $\mathcal{Z}^k$ and z^* only.

Lemma 3. *Let* $(x^k)_{k\ge 0}$ *be the sequence generated by Algorithm 1. Let* $\lambda_{max}, \lambda_{min}$ *be the maximal and minimal eigenvalues of the matrix* U*, and* $\kappa = \frac{\lambda_{max}}{\lambda_{min}}$ *be the condition number. Then for any* $x^* \in \mathbf{Fix}T$*, we have*

(78)
$$\mathbb{E}\big(\|x^{k+1}-x^*\|_U^2 \,\big|\, \mathcal{X}^k\big) \le \|x^k - x^*\|_U^2 + \frac{\gamma}{m}\sum_{d\in J(k)} \|x^d - x^{d+1}\|_U^2 + \frac{1}{m}\left(\frac{|J(k)|}{\gamma} + \frac{\kappa^2}{mp_{\min}} - \frac{1}{\eta_k}\right)\|x^k - \bar{x}^{k+1}\|_U^2$$

where $\gamma > 0$ *(to be optimized later) and* $\mathbb{E}(\cdot \,|\, \mathcal{X}^k)$ *denotes expectation conditional on* $\mathcal{X}^k$*.*

Proof. We have

(79)
$$\begin{aligned}
& \mathbb{E}\left(\|x^{k+1}-x^*\|_U^2 \,|\, \mathcal{X}^k\right) \\
\overset{(35)}{=}\ & \mathbb{E}\left(\|x^k - \tfrac{\eta_k}{mp_{i_k}} S_{i_k}\hat{x}^k - x^*\|_U^2 \,|\, \mathcal{X}^k\right) \\
=\ & \|x^k - x^*\|_U^2 + \mathbb{E}\left(\tfrac{2\eta_k}{mp_{i_k}}\left\langle S_{i_k}\hat{x}^k, x^* - x^k\right\rangle_U + \tfrac{\eta_k^2}{m^2p_{i_k}^2}\|S_{i_k}\hat{x}^k\|_U^2 \,\big|\, \mathcal{X}^k\right) \\
=\ & \|x^k - x^*\|_U^2 + \tfrac{2\eta_k}{m}\textstyle\sum_{i=1}^m \left\langle US_i\hat{x}^k, x^* - x^k\right\rangle_U + \tfrac{\eta_k^2}{m^2}\sum_{i=1}^m \tfrac{1}{p_i}\|S_i\hat{x}^k\|_U^2 \\
=\ & \|x^k - x^*\|_U^2 + \tfrac{2\eta_k}{m}\left\langle S\hat{x}^k, x^* - x^k\right\rangle_U + \tfrac{\eta_k^2}{m^2}\textstyle\sum_{i=1}^m \tfrac{1}{p_i}\|S_i\hat{x}^k\|_U^2,
\end{aligned}$$

where the third equality holds because the probability of choosing i is p_i.

Note that

(80)
$$\begin{aligned}\textstyle\sum_{i=1}^m \frac{1}{p_i}\|S_i\hat{x}^k\|_U^2 \le \frac{1}{p_{\min}}\sum_{i=1}^m \|S_i\hat{x}^k\|_U^2 &\le \frac{\lambda_{max}^2}{p_{\min}}\sum_{i=1}^m \|S_i\hat{x}^k\|^2\\ &\le \frac{\lambda_{max}^2}{p_{\min}}\|S\hat{x}^k\|^2 \overset{(77)}{=} \tfrac{\lambda_{max}^2}{\eta_k^2 p_{\min}}\ \|x^k-\bar{x}^{k+1}\|^2 \le \frac{\lambda_{max}^2}{\lambda_{min}^2\eta_k^2 p_{\min}}\|x^k-\bar{x}^{k+1}\|_U^2\\ &= \frac{\kappa^2}{\eta_k^2 p_{\min}}\|x^k-\bar{x}^{k+1}\|_U^2,\end{aligned}$$

and

(81)
$$\begin{aligned}&\langle S\hat{x}^k, x^*-x^k\rangle_U\\ =&\langle S\hat{x}^k, x^*-\hat{x}^k+\textstyle\sum_{d\in J(k)}(x^d-x^{d+1})\rangle_U\\ \overset{(77)}{=}&\langle S\hat{x}^k, x^*-\hat{x}^k\rangle_U+\tfrac{1}{\eta_k}\textstyle\sum_{d\in J(k)}\langle x^k-\bar{x}^{k+1}, x^d-x^{d+1}\rangle_U\\ \le&\langle S\hat{x}^k-Sx^*, x^*-\hat{x}^k\rangle_U+\tfrac{1}{2\eta_k}\textstyle\sum_{d\in J(k)}\left(\frac{1}{\gamma}\|x^k-\bar{x}^{k+1}\|_U^2+\gamma\|x^d-x^{d+1}\|_U^2\right)\\ \overset{(76)}{\le}&-\tfrac{1}{2}\|S\hat{x}^k\|_U^2+\tfrac{1}{2\eta_k}\textstyle\sum_{d\in J(k)}(\frac{1}{\gamma}\|x^k-\bar{x}^{k+1}\|_U^2+\gamma\|x^d-x^{d+1}\|_U^2)\\ \overset{(77)}{=}&-\tfrac{1}{2\eta_k^2}\|x^k-\bar{x}^{k+1}\|_U^2+\tfrac{|J(k)|}{2\gamma\eta_k}\|x^k-\bar{x}^{k+1}\|_U^2+\tfrac{\gamma}{2\eta_k}\textstyle\sum_{d\in J(k)}\|x^d-x^{d+1}\|_U^2,\end{aligned}$$

where the first inequality follows from the Young's inequality. Plugging (80) and (81) into (79) gives the desired result. □

Let $\mathcal{H}^{\tau+1}=\prod_{i=0}^{\tau}\mathcal{H}$ be a product space and $\langle\cdot\,|\,\cdot\rangle$ be the induced inner product:

$$\langle(z^0,\ldots,z^\tau)\,|\,(y^0,\ldots,y^\tau)\rangle=\sum_{i=0}^{\tau}\langle z^i,y^i\rangle,\quad \forall(z^0,\ldots,z^\tau),(y^0,\ldots,y^\tau)\in\mathcal{H}^{\tau+1}.$$

Define a $(\tau+1)\times(\tau+1)$ matrix M' by

$$M':=\begin{bmatrix}1&0&\cdots&0\\0&0&\cdots&0\\\vdots&\vdots&\ddots&\vdots\\0&0&\cdots&0\end{bmatrix}+\sqrt{p_{\min}}/\kappa\begin{bmatrix}\tau&-\tau&&&&&\\-\tau&2\tau-1&1-\tau&&&&\\&1-\tau&2\tau-3&2-\tau&&&\\&&\ddots&\ddots&\ddots&&\\&&&-2&3&-1&\\&&&&-1&1\end{bmatrix},$$

and let $M = M' \otimes U_{\mathcal{H}}$. Here $U_{\mathcal{H}}$ is the operator U on $\mathcal{H}$, and $\otimes$ represents the Kronecker product. For a given $(y^0, \cdots, y^\tau) \in \mathcal{H}^{\tau+1}$, $(z^0, \cdots, z^\tau) = M(y^0, \cdots, y^\tau)$ is given by:

$$
\begin{aligned}
&z^0 = Uy^0 + \sqrt{p_{\min}}/\kappa\tau U(y^0 - y^1),\\
&z^i = \sqrt{p_{\min}}/\kappa((i-\tau-1)Uy^{i-1} + (2\tau - 2i + 1)Uy^i + (i-\tau)Uy^{i+1}), \text{ if } 1 \le i \le \tau - 1,\\
&z^\tau = \sqrt{p_{\min}}/\kappa U(y^\tau - y^{\tau-1}).
\end{aligned}
$$

Then M is a self-adjoint and positive definite linear operator since M' is symmetric and positive definite, and we define $\langle \cdot \,|\, \cdot \rangle_M = \langle \cdot \,|\, M \cdot \rangle$ as the M-weighted inner product and $\|\cdot\|_M$ the induced norm.

Let

$$
\mathbf{x}^k = (x^k, x^{k-1}, \ldots, x^{k-\tau}) \in \mathcal{H}^{\tau+1},\ k \ge 0, \text{ and } \mathbf{x}^* = (x^*, x^*, \ldots, x^*) \in \mathbf{X}^* \subseteq \mathcal{H}^{\tau+1},
$$

where we set $x^k = x^0$ for $k < 0$. With

(82)
$$
\xi_k(\mathbf{x}^*) := \|\mathbf{x}^k - \mathbf{x}^*\|_M^2 = \|x^k - x^*\|_U^2 + \sqrt{p_{\min}}/\kappa \sum_{i=k-\tau}^{k-1} (i-(k-\tau)+1)\|x^i - x^{i+1}\|_U^2, \quad \forall\, k \ge 0,
$$

we have the following fundamental inequality:

Theorem 3 (Fundamental inequality)**.** *Let $(x^k)_{k\ge 0}$ be the sequence generated by ARock. Then for any $\mathbf{x}^* \in \mathbf{X}^*$, it holds that*

(83)
$$
\mathbb{E}\left(\xi_{k+1}(\mathbf{x}^*) \,\middle|\, \mathcal{X}^k\right) \le \xi_k(\mathbf{x}^*) + \frac{1}{m}\left(\frac{2\tau\kappa}{m\sqrt{p_{\min}}} + \frac{\kappa^2}{m p_{\min}} - \frac{1}{\eta_k}\right)\|\bar{x}^{k+1} - x^k\|_U^2.
$$

The proof of the fundamental inequality of the other part of the proof is omitted. Interested readers are referred to the orginal paper [51] for the complete procedure.

Zhimin Peng
PO Box 951555
UCLA Math Department
Los Angeles, CA 90095
E-mail address: zhimin.peng@math.ucla.edu

Tianyu Wu
PO Box 951555
UCLA Math Department
Los Angeles, CA 90095
E-mail address: wuty11@math.ucla.edu

Yangyang Xu
207 Church St SE
University of Minnesota, Twin Cities
Minneapolis, MN 55455
E-mail address: yangyang@ima.umn.edu

Ming Yan
Department of Computational Mathematics, Science and Engineering
Department of Mathematics
Michigan State University
East Lansing, MI 48824
E-mail address: yanm@math.msu.edu

Wotao Yin
PO Box 951555
UCLA Math Department
Los Angeles, CA 90095
E-mail address: wotaoyin@math.ucla.edu

Received January 1, 2016