Parallel block coordinate descent methods with identification strategies *

Ronaldo Lopes[1], Sandra A. Santos[2], and Paulo J. S. Silva[2]

[1] *Universidade Estadual de Maringá (UEM), Centro de Ciências Exatas (CCE), Av. Colombo, 5790 - Zona 7, 87020-900, Maringá, Paraná, Brazil Email:* ronaldolps3@gmail.com

[2] *Universidade Estadual de Campinas (Unicamp), Instituto de Matemática, Estatística e Computação Científica (IMECC), Rua Sergio Buarque de Holanda, 651, 13083-859, Campinas, São Paulo, Brazil Emails:* sandra@ime.unicamp.br, pjssilva@ime.unicamp.br

July 29th, 2025

Abstract

This work presents a parallel variant of the algorithm introduced in [*Acceleration of block coordinate descent methods with identification strategies*, Comput. Optim. Appl. 72(3):609–640, 2019] to minimize the sum of a partially separable smooth convex function and a possibly non-smooth block-separable convex function under simple constraints. It achieves better efficiency by using a strategy to identify the nonzero coordinates that allows the computational effort to be focused on using a nonuniform probability distribution in the selection of the blocks. Parallelization is achieved by extending the theoretical results from Richtárik and Takáč [*Parallel coordinate descent methods for big data optimization*, Math. Prog. Ser. A 156:433–484, 2016]. We present convergence results and comparative numerical experiments on regularized regression problems using both synthetic and real data.

Key words: Block coordinate descent, active-set identification, large-scale optimization, parallel computation and ℓ_1 regularization.

AMS Classification: 60K05, 65Y05, 49M37, 90C30, 90C06, 90C25

1 Introduction

The contemporary need to address problems with huge data sets has renewed interest in simple first-order optimization algorithms such as coordinate descent

*Partially supported by FAPESP grants 2023/08706-1, 2018/24293-0, 2014/14228-6, 2013/07375-0, and CNPq grants 312394/2023-3, 305010/2020-4.

methods, particularly when problem structure can be further explored [1, 15, 19]. Another room for improvement is the use of parallelism [2, 12, 14, 21, 23, 25], where decomposition strategies, communication, and distributed memory architectures are crucial [3, 20, 22, 6].

Previously, [13] proposed an acceleration of block coordinate descent methods (BCDMs) using identification strategies. The main idea was to use a nonuniform probability distribution to select free blocks that are more likely to induce a larger reduction of the objective function at each iteration. The resulting method was called Active BCDM. This approach was combined with an extra second-order step in the subspace of the free blocks, using a blocked Hessian, reducing the running time required to solve ℓ_1-regularized problems. In [17], the authors develop an exhaustive study concerning algorithmic choices that influence the efficiency of BCDMs, namely the block partitioning strategy, the block selection rule, and the block update rule. In the current work, we investigate how identification strategies can be used in parallel variants of BCDM.

A naive parallel BCDM implementation suffers from several challenges, notably the need to predefine subgroups of blocks to be updated in parallel. This would enable the precomputation of each subgroup's Lipschitz constants of the gradient. Without this precomputation, the descent directions and the Lipschitz constants, or line searches, would need to be computed for each new selection, which would limit computational performance.

These issues were addressed by the theoretical results in [21]. In this paper, the authors introduced the concept of partial separability, which relaxes the notion of function separability. Then, they developed a framework where the blocks can be updated in parallel using the original Lipschitz continuity constants of the gradient of each block. Hence, the blocks updated in parallel are no longer fixed in advance. The method also avoids extra function evaluations to enforce a descent criterion. Instead, it penalizes the curvature of the model used to compute the descent directions, leading to improved efficiency.

However, the results in [21] assume that the blocks are selected using a uniform distribution, limiting the direct application of the identification strategy described in [13]. The main theoretical contribution of the present text is to overcome this limitation and allow nonuniform distributions to be employed. This opens up the path to incorporate identification techniques in parallel block coordinate descent methods. We then present a convergence theory for our method, named Parallel Active BCDM (Algorithm 2) and compare its efficiency with the (uniform) parallel block coordinate descent method, `PCDM`. The numerical comparison uses randomly generated artificial tests and a library of real-world problems [13, Tables 2 and 3]. The numerical experiments unveil essential differences in the behavior of parallel block coordinate descent methods depending on the relation between dimensions of the data matrix when applied to regression problems with regularization (Lasso) [24].

The remainder of this text is organized as follows: Section 2 contains the problem statement and details the main ingredients of Active BCDM, whose (sequential) version is detailed in Section 3. Then, its parallel version is introduced in Section 4, along with the associated convergence results. Finally, computa-

tional experiments are presented and analyzed in Section 5. Our conclusions are stated in Section 6, along with lines for future research.

2 Background

We consider the problem

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & f(x) + \psi(x) \\ \text{s.t.} \quad & l \leq x \leq u, \end{aligned} \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $\psi : \mathbb{R}^n \to \mathbb{R}$ are both convex, with ψ possibly nonsmooth. In addition, ψ is assumed to have a block-separable structure given by

$$\psi(x) = \sum_{i=1}^{m} \psi_i(x_{(i)}), \tag{2}$$

where $x_{(i)} \in \mathbb{R}^{p_i}$ is a subset of coordinates of the vector $x \in \mathbb{R}^n$ with $\sum_{i=1}^m p_i = n$. The vectors l and u have n coordinates in $[-\infty, +\infty]$, with components $l_j < u_j$, $j = 1, \ldots, n$. Infinite values denote the absence of a bound.

Based on the block decomposition of the vector x in m coordinate subsets described in (2), we define a set of matrices $U_i \in \mathbb{R}^{n\times p_i}$ whose columns are canonical vectors of $\mathbb{R}^n$, and such that

$$x = \sum_{i=1}^{m} U_i x_{(i)} \quad \text{and} \quad x_{(i)} = U_i^T x.$$

Let $B_i \in \mathbb{S}^{p_i}_{++}$, $i = 1, \ldots, m$ be a set of positive definite matrices of order p_i. We then define the respective primal and dual norms in $\mathbb{R}^{p_i}$ by

$$\|x_{(i)}\|_{(i)} := \sqrt{x_{(i)}^T B_i x_{(i)}} \text{ and } \|x_{(i)}\|^*_{(i)} := \sqrt{x_{(i)}^T B_i^{-1} x_{(i)}}, \ \forall i \in \{1, \ldots, m\}.$$

Such matrices may be used to provide scaling information for the problem variables, particularly if they are diagonal. They may also be used to provide second-order information to the models.

The gradient of the smooth function f is assumed to be Lipschitz continuous by blocks, that is, for each $i = 1, \ldots, m$, there exists a constant $L_i > 0$ such that

$$\|\nabla_i f(x + U_i h_i) - \nabla_i f(x)\|^*_{(i)} \leq L_i \|h_i\|_{(i)}, \ h_i \in \mathbb{R}^{p_i}, \ i = 1, \ldots, m, \tag{3}$$

for all x such that $l \leq x \leq u$ and $\nabla_i f(x) = U_i^T \nabla f(x)$.

Let $B \in \mathbb{R}^{n\times n}$ be the block diagonal positive definite matrix stated as

$$B := \text{diag}(L_1 B_1, \cdots, L_m B_m) \in \mathbb{R}^{n\times n},$$

where, for $i = 1, \ldots, m$, the scalars L_i are as in (3) and the matrices B_i are fixed above. We then define the following primal and dual norms in $\mathbb{R}^n$:

$$\|z\|_B = \sqrt{z^T B z} \quad \text{and} \quad \|z\|_B^* = \sqrt{z^T B^{-1} z}.$$

For the vectors x, y, l, u, which can be indexed by coordinates or blocks, we have adopted the convention that the subindex (i) refers to the ith block, whereas i refers to the ith component. The remaining vectors, matrices, functions, and scalars used in the text, namely, h_i, w_i, p_i, s_i, B_i, U_i, ψ_i, $\nabla_i f$, L_i, are only indexed by blocks. Therefore, we simply use i to refer to the ith block and avoid overloading the notation.

Throughout the text, $\|\cdot\|$ refers to either the Euclidean vector norm or it's induced matrix norm. The cardinality of the set $\mathcal{I}$ is denoted by $|\mathcal{I}|$. Given a positive integer p, we define the set of indices $[p] := \{1, \ldots, p\}$. The feasible set of problem (1) is denoted by

$$\mathcal{X} := \{x \in \mathbb{R}^n \mid l \leq x \leq u\}, \tag{4}$$

and its objective function by $F(x) := f(x) + \psi(x)$.

3 On the sequential Active BCDM

Our objective is to parallelize the block coordinate descent method with the identification of active variables (Active BCDM) introduced in [13], see Algorithm 1. Let us start describing its essential components.

Given a vector $x \in \mathbb{R}^n$ and a block index i, the block descent direction employed in the Active BCDM, $h_i(x)$, is the minimizer of the subproblem

$$\begin{aligned} \min_{h \in \mathbb{R}^{p_i}} \quad & \nabla_i f(x)^T h + \frac{L_i}{2}\|h\|_{(i)}^2 + \psi_i(x_{(i)} + h) - \psi_i(x_{(i)}), \\ \text{s.t.} \quad & l \leq x + U_i h \leq u. \end{aligned} \tag{5}$$

Such descent directions were introduced in (5) and they are supported theoretically by the fact that null descent directions for all blocks are equivalent to the stationarity of the current iterate [13, Lemma 2].

In [13], the blocks are chosen from a nonuniform probability distribution. The coordinate blocks are split into two groups: one, denoted by $\mathcal{J}$, containing a subset of the blocks for which the variables are likely to activate the bound constraints of the problem, and the other, denoted by $\mathcal{I}$, with the remaining blocks. The blocks in $\mathcal{I}$ are updated by a probability distribution that is δ_{DP} times more likely than the blocks in the set $\mathcal{J}$, since the objective function value is expected to have a larger decrease for variables in $\mathcal{I}$ than for those in $\mathcal{J}$. Such a strategy has proved effective for problems with $\psi(x) = \lambda\|x\|_1$, where $\lambda > 0$ is a regularization parameter. Both sets $\mathcal{I}$ and $\mathcal{J}$ are updated along the method after a pre-established cycle of iterations. The strategy employed to classify a block of coordinates as active is based upon an identification function [8].

Algorithm 1 BCDM with Identification of Active Variables (Active BCDM)

1: Choose an initial point $l \leq x^0 \leq u$, a block separation of ψ function described as $x = \sum_{i=1}^{m} U_i x_{(i)}$, an initial cycle size c_0, the parameters $\ell_{\max} \in \mathbb{N}$, $\varepsilon \in \mathbb{R}_+$ and two natural numbers δ_F, δ_{DP}. Initialize a vector $v \in \mathbb{R}^n$ with 2ε in all positions, the sets $\mathcal{I} = [m]$, $\mathcal{J} = \emptyset$, the cycle size $c_s = c_0$, and the counter $\ell = 0$.
2: Calculate the Lipschitz constants of the gradient by blocks satisfying (3).
3: **repeat**
4: **for** $k = \ell + 1, \ldots, \ell + c_s$ **do**
5: Choose a block i that satisfies the probability distribution

$$\mathbb{P}(i) = \begin{cases} \dfrac{\delta_{DP}}{\delta_{DP}|\mathcal{I}| + |\mathcal{J}|}, & \text{if } i \in \mathcal{I}, \\ \dfrac{1}{\delta_{DP}|\mathcal{I}| + |\mathcal{J}|}, & \text{if } i \in \mathcal{J}. \end{cases}$$

6: Find $h_i \equiv h_i(x^k)$, a solution to the subproblem (5)
7: Set $x^{k+1} = x^k + U_i h_i$ and $v_{(i)} = h_i$.
8: **end for**
9: Set $\ell = \ell + c_s$.
10: Obtain the set $\mathcal{C}(x^\ell) \subset [m]$, where

$$\mathcal{C}(x^\ell) = \left\{ i \mid g_j(x^\ell) \geq \rho_\alpha(x^\ell), \forall x_j^\ell \text{ s.t. } j \text{ belongs to the } i\text{th-block} \right\},$$

with the identification function $\rho_\alpha(x)$ described in (8).
11: Define $\mathcal{J} \subset \mathcal{C}(x^\ell)$ and $\mathcal{I} = [m] \setminus \mathcal{J}$. Set $c_s = \max\{\min\{\delta_F|\mathcal{I}|, m\}, c_0\}$.
12: **until** $\|v\| \leq \varepsilon$ or $\ell \geq \ell_{\max}$

Such functions can unveil, in a neighborhood of a stationary point x^*, which constraints are active at x^*.

One example of an identification function that can be used in line 10 of Active BCDM, Algorithm 1, was given in [13, Propositions 2 and 3]. To present it, we start by describing the feasible set $\mathcal{X}$ using the function $g : \mathbb{R}^n \to \mathbb{R}^{2n}$, defined by

$$g_i(x) := \begin{cases} l_i - x_i, & \text{if } 1 \leq i \leq n, \\ x_i - u_i, & \text{if } n+1 \leq i \leq 2n. \end{cases}$$

Then, the feasible set $\mathcal{X}$ may be rewritten as

$$\mathcal{X} = \{x \in \mathbb{R}^n \mid g(x) \leq 0\}.$$

Given $x \in \mathcal{X}$ and $\beta \in \mathbb{R}_+$, $h^\beta(x) \in \mathbb{R}^n$ is defined as the unique minimizer of

$$\begin{aligned} \min_{h \in \mathbb{R}^n} \quad & \sum_{i=1}^{m} \left(\nabla_i f(x)^T h_i + \frac{\beta L_i}{2} \|h_i\|_{(i)}^2 + \psi_i(x_{(i)} + h_i) \right) \\ \text{s.t.} \quad & l_i \leq x_{(i)} + h_i \leq u_i, \ i \in [m]. \end{aligned} \tag{6}$$

Additionally, let $h(x) \in \mathbb{R}^n$ be defined as $h^\beta(x)$, for $\beta = 1$, that is, the solution of

$$\begin{aligned} \min_{h \in \mathbb{R}^n} \quad & \sum_{i=1}^{m} \left(\nabla_i f(x)^T h_i + \frac{L_i}{2} \|h_i\|_{(i)}^2 + \psi_i(x_{(i)} + h_i) \right) \\ \text{s.t.} \quad & l_i \leq x_{(i)} + h_i \leq u_i, \ i \in [m]. \end{aligned} \tag{7}$$

Notice that problems (5) and (7) are closely related, since the block separability of (7) implies that the ith block of $h(x)$ is the solution $h_i(x)$ of (5).

Finally, we draw inspiration from an idea introduced in [9] to define a function that identifies the active constraints at the stationary points of an optimization problem, namely

$$\rho_\alpha(x) := -\|h(x)\|^\alpha, \tag{8}$$

where $\alpha \in (0, 1)$. Under an error-bound assumption, it was shown in [13, Propositions 2 and 3] that ρ_α is an identification function for (1) that is highly effective when used in Active BCDM in the context of ℓ_1-regularization problems.

4 Parallel Active BCDM

The parallelization of the Active BCDM Algorithm is not straightforward. An initial idea would be to distribute the task of the `for` loop to the available threads, choosing τ blocks obeying the nonuniform probability distribution to be updated in parallel. However, to ensure descent of the objective function, it would be necessary to calculate the Lipschitz constant for each selected group of blocks and an associated descent direction. This would greatly affect efficiency.

To maintain the computational cost of the parallel iterations as close as the serial version, both methods use the same Lipschitz constants for a fixed block coordinate structure. Moreover, it is desirable that both variations should employ the same nonuniform distribution, which privileges updating the blocks of inactive variables. To achieve this, we have adapted the theoretical framework based upon [21].

We start with a concept presented in [21, Section 1.5] which extends the block separability notion to the smooth part of the objective function of problem (1). This enables the characterization of a wider class of smooth functions f that might benefit from the parallelism.

Definition 1. *The convex and smooth function f is **partially separable of degree** ω if there exists a finite number of smooth functions f_S such that*

$$f(x) = \sum_{S \in \mathcal{S}} f_S(x),$$

where $\mathcal{S}$ is a finite collection of nonempty subsets of $[m]$, f_S are differentiable convex functions that only depend on blocks $x_{(i)}$ for $i \in S$ and

$$|S| \leq \omega, \ \forall S \in \mathcal{S}.$$

When applying block coordinate descent methods to a partially separable function, we must determine the value of ω as it influences the convergence of the methods. Specifically, we aim to find the minimal value ω that satisfies Definition 1; however, we may need to consider larger values for ω to ensure that the coordinate blocks of f_S, for all $S \in \mathcal{S}$, satisfy a desirable property.

The partial separability notion is useful because the structure of the smooth component of relevant problems, like Lasso and ℓ_1-regularized logistic regression, allows for a simple computation of the degree ω; for further details see [21, Section 1.5].

4.1 Descent directions for the method

Throughout this subsection, we assume that the point $x \in \mathcal{X}$, the number $\tau > 0$ of threads that will work in parallel, together with the disjoint sets of indices $\mathcal{I}$ and $\mathcal{J}$ that split the groups of coordinate blocks ($\mathcal{I} \cup \mathcal{J} = [m]$) are all fixed.

Let us also recall the concept of a multiset [11, Definition 1], a generalization of the notion of a set that allows repetitions of its elements. In the finite case, a multiset can be defined as a set of tuples, $\{(x_1, c_1), (x_2, x_2), \dots (x_k, c_k)\}$, where, for $i = 1, \dots, k$, x_i represents the element in the multiset and c_i is a positive integer representing the cardinality of x_i in the multiset. In other words, c_i represents the number of times x_i appears repeated in the multiset. The cardinality of a multiset is the sum of the cardinalities of its elements. From now on, we will adopt a simplified abuse of notation and denote a multiset using the notation of a set with the elements x_i appearing c_i times. For example, we will denote the multiset $\{(1,2),(2,1),(3,4)\}$ by $\{1,1,2,3,3,3,3\}$.

Now, let $\mathcal{B}$ be randomly chosen from $[m]$, with cardinality τ, allowing possible repetitions, and following the probability distribution of Algorithm 1. Observe that this definition allows the algorithm to select a block of coordinates multiple times in a single step. The objective is to select $\mathcal{B}$ and a constant $\beta \geq 1$, such that updating $x + \sum_{i \in \mathcal{B}} U_i h_i^\beta$ produces descent for F in expectation, where $h_i^\beta \in \mathbb{R}^{p_i}$ is the solution of

$$\begin{aligned} \min_{h \in \mathbb{R}^{p_i}} \quad & \nabla_i f(x)^T h + \frac{\beta L_i}{2} \|h\|_{(i)}^2 + \psi_i(x_{(i)} + h) - \psi_i(x_{(i)}), \\ \text{s.t.} \quad & l \leq x + U_i h \leq u. \end{aligned} \tag{9}$$

To reach this goal, a few auxiliary definitions and results are provided.

Definition 2 (**Intersection keeping all values).** *Let I and J be a set and a multiset of indexes in $[m]$, respectively. The multiset of elements of J that are in I is stated as (I, J), and the cardinality of such a multiset is denoted by $|(I, J)|$.*

For the sake of illustration, let $I = \{1, 3, 5\}$ and $J = \{1, 1, 2, 3, 4\}$. In this case, $(I, J) = \{1, 1, 3\}$ and $|(I, J)| = 3$, since, considering the repetitions, the multiset J contains three elements of I.

Proposition 1. *Let $\mathcal{B}$ be a multiset randomly chosen with elements from $[m]$, with cardinality τ, following the probability distribution of Algorithm 1, and let $S \subset [m]$ be a nonempty set. For $k \in \mathbb{N}$, with $1 \leq k \leq \tau$, it holds*

$$\mathbb{P}(i \in \mathcal{B} \mid |(S,\mathcal{B})| = k) = \begin{cases} \dfrac{k\delta_{DP}}{\delta_{DP}|\mathcal{I} \cap S| + |\mathcal{J} \cap S|}, & \text{if } i \in \mathcal{I} \cap S; \\ \\ \dfrac{k}{\delta_{DP}|\mathcal{I} \cap S| + |\mathcal{J} \cap S|}, & \text{if } i \in \mathcal{J} \cap S. \end{cases} \tag{10}$$

Proof. First, notice that from the probability distribution of Algorithm 1 we have

$$\mathbb{P}(i) = \begin{cases} \dfrac{\delta_{DP}}{\delta_{DP}|\mathcal{I}| + |\mathcal{J}|}, & \text{if } i \in \mathcal{I}, \\ \\ \dfrac{1}{\delta_{DP}|\mathcal{I}| + |\mathcal{J}|}, & \text{if } i \in \mathcal{J}. \end{cases} \tag{11}$$

Additionally, observe that the integer $k \in \mathbb{N}$ such that $1 \leq k \leq \tau$ is constant along the proof.

Let $i \in \mathcal{I} \cap S$. To reach expression (10), we will rest upon the conditional probability formula:

$$\mathbb{P}(i \in \mathcal{B} \mid |(S,\mathcal{B})| = k) = \frac{\mathbb{P}(i \in \mathcal{B} \text{ and } |(S,\mathcal{B})| = k)}{\mathbb{P}(|(S,\mathcal{B})| = k)}. \tag{12}$$

For computing the numerator above, let us calculate the following values:

1. Probability of the index i to be chosen in a certain position.

$$p_{\mathcal{I}} \overset{(11)}{=} \frac{\delta_{DP}}{\delta_{DP}|\mathcal{I}| + |\mathcal{J}|}.$$

2. Probability of an element of S to be chosen in a certain position. This is obtained by adding up the probabilities of each element of S to be in a certain position of $\mathcal{B}$, that is,

$$p_1 = \sum_{i \in \mathcal{I} \cap S} \frac{\delta_{DP}}{\delta_{DP}|\mathcal{I}| + |\mathcal{J}|} + \sum_{j \in \mathcal{J} \cap S} \frac{1}{\delta_{DP}|\mathcal{I}| + |\mathcal{J}|} = \frac{\delta_{DP}|\mathcal{I} \cap S| + |\mathcal{J} \cap S|}{\delta_{DP}|\mathcal{I}| + |\mathcal{J}|}.$$

3. Probability of an element that does not belong to S to be chosen in a certain position. This is obtained by adding up the probabilities of each element that does not belong to S to be in a certain position of $\mathcal{B}$, namely,

$$\begin{aligned} p_2 &= \sum_{i \in \mathcal{I} \setminus S} \frac{\delta_{DP}}{\delta_{DP}|\mathcal{I}| + |\mathcal{J}|} + \sum_{j \in \mathcal{J} \setminus S} \frac{1}{\delta_{DP}|\mathcal{I}| + |\mathcal{J}|} \\ &= \frac{\delta_{DP}(|\mathcal{I}| - |\mathcal{I} \cap S|) + |\mathcal{J}| - |\mathcal{J} \cap S|}{\delta_{DP}|\mathcal{I}| + |\mathcal{J}|}. \end{aligned}$$

Now, to obtain the probability that $i \in \mathcal{B}$ and $|(S, \mathcal{B})| = k$, one should organize the previous values. Assume the index i is chosen in the first position, the next $k-1$ positions are elements of S and the last $\tau - k$ positions are elements of $[m] \setminus S$:

$$p_{\mathcal{I}} \overbrace{p_1 \dots p_1}^{k-1} \underbrace{p_2 \dots p_2}_{\tau-k}. \tag{13}$$

Notice that (13) amounts to the probability of a specific event in which the index i appears in the first position, and the k first terms are elements of S. To account for the remaining events of $i \in \mathcal{B}$ and $|(S, \mathcal{B})| = k$, we must introduce two additional factors. First, $\binom{\tau}{k}$, the number of possibilities of distributing the k elements of S in the τ positions of the random multiset $\mathcal{B}$. Second, $\binom{k}{1}$, the number of possibilities of setting i among the k elements of S that belong to $\mathcal{B}$. Therefore,

$$\mathbb{P}(i \in \mathcal{B} \text{ and } |(S, \mathcal{B})| = k) = \binom{k}{1}\binom{\tau}{k} p_{\mathcal{I}} \overbrace{p_1 \dots p_1}^{k-1} \underbrace{p_2 \dots p_2}_{\tau-k}. \tag{14}$$

Now, let us compute the probability of $|(S, \mathcal{B})| = k$, i.e., the denominator of (12). We start by evaluating the probability of the particular event of $\mathcal{B}$ in which the first k indices are elements of S, and the last $\tau - k$ elements do not belong to S:

$$\overbrace{p_1 \dots p_1}^{k} \underbrace{p_2 \dots p_2}_{\tau-k}. \tag{15}$$

As in the previous reasoning, the remaining related events must be considered. Indeed, the probability (15) has to be multiplied by $\binom{\tau}{k}$, the number of possibilities of distributing the k elements of S in the τ positions of the random multiset $\mathcal{B}$. Hence,

$$\mathbb{P}(|(S, \mathcal{B})| = k) = \binom{\tau}{k} \overbrace{p_1 \dots p_1}^{k} \underbrace{p_2 \dots p_2}_{\tau-k}. \tag{16}$$

Replacing (14) and (16) in (12), we obtain

$$\begin{aligned}
\mathbb{P}(i \in \mathcal{B} \mid |(S, \mathcal{B})| = k) &= \frac{\binom{k}{1}\binom{\tau}{k} p_{\mathcal{I}} p_1^{k-1} p_2^{\tau-k}}{\binom{\tau}{k} p_1^k p_2^{\tau-k}} \\
&= \frac{k p_{\mathcal{I}}}{p_1} \\
&= \frac{k \frac{\delta_{DP}}{\delta_{DP}|\mathcal{I}|+|\mathcal{J}|}}{\frac{\delta_{DP}|\mathcal{I}\cap S|+|\mathcal{J}\cap S|}{\delta_{DP}|\mathcal{I}|+|\mathcal{J}|}} \\
&= \frac{k\delta_{DP}}{\delta_{DP}|\mathcal{I} \cap S| + |\mathcal{J} \cap S|}.
\end{aligned}$$

Now, assuming that $i \in \mathcal{J} \cap S$, we proceed as before, using the fact that the probability of the index i being chosen in a certain position is given by

$$p_{\mathcal{J}} \overset{(11)}{=} \frac{1}{\delta_{DP}|\mathcal{I}| + |\mathcal{J}|}.$$

Arguing as in the previous case, we have

$$\begin{aligned}
\mathbb{P}(i \in \mathcal{B} \mid |(S,\mathcal{B})| = k) &= \frac{\binom{k}{1}\binom{\tau}{k} p_{\mathcal{J}} p_1^{k-1} p_2^{\tau-k}}{\binom{\tau}{k} p_1^k p_2^{\tau-k}} \\
&= \frac{k p_{\mathcal{J}}}{p_1} \\
&= \frac{k \frac{1}{\delta_{DP}|\mathcal{I}|+|\mathcal{J}|}}{\frac{\delta_{DP}|\mathcal{I}\cap S|+|\mathcal{J}\cap S|}{\delta_{DP}|\mathcal{I}|+|\mathcal{J}|}} \\
&= \frac{k}{\delta_{DP}|\mathcal{I}\cap S| + |\mathcal{J}\cap S|},
\end{aligned}$$

the proof is complete. □

It is worth mentioning that formulas (10) and (16) are also valid for $k = 0$. This case is not included in Proposition 1 because the theoretical reasoning to prove (10) differs from the analysis regarding $1 \leq k \leq \tau$. Nevertheless, the aforementioned probability values for $k = 0$ are easily verifiable: $|(S,\mathcal{B})| = 0$ implies that $S \cap \mathcal{B} = \emptyset$, and thus, if either $i \in \mathcal{I} \cap S$ or $i \in \mathcal{J} \cap S$ then $i \notin \mathcal{B}$, so that $\mathbb{P}(i \in \mathcal{B} \mid |(S,\mathcal{B})| = 0) = 0$. Moreover, it is not difficult to see that $\mathbb{P}(|(S,\mathcal{B})| = 0) = p_2^{\tau}$.

Before the main result of this section, we will show that, given $\mathcal{I}$ and $\mathcal{J}$, two fixed and disjoint sets of indices such that $\mathcal{I} \cup \mathcal{J} = [m]$, it is always possible to extend the partial separability description of $f(x) = \sum_{S\in\mathcal{S}} f_S(x)$ in such way that it conforms to

$$\begin{aligned}
|\mathcal{I} \cap S_1| = |\mathcal{I} \cap S_2|, \ \forall\ S_1, S_2 \in \mathcal{S}, \\
|\mathcal{J} \cap S_1| = |\mathcal{J} \cap S_2|, \ \forall\ S_1, S_2 \in \mathcal{S},
\end{aligned}$$

the respective degree of partial separability is at most doubled, and it can be easily computed from the original ω, $|\mathcal{I}|$ and $|\mathcal{J}|$.

To achieve this, let $\mathcal{I}$ and $\mathcal{J}$ be fixed and set $\mathcal{S}' = \emptyset$. Select $S \in \mathcal{S}$. As $|S| \leq \omega$, it follows that

$$|S \cap \mathcal{I}| \leq \min\{|\mathcal{I}|, \omega\}.$$

If the equality holds, define $S' = S$. Otherwise, there are $\min\{|\mathcal{I}|, \omega\} - |S \cap \mathcal{I}|$ block indexes in $\mathcal{I}$ that can be added to S, creating an S' such that $|S' \cap \mathcal{I}| = \min\{|\mathcal{I}|, \omega\}$. Similarly, we can define S' from S, adding extra elements from $\mathcal{J}$ whenever necessary, such that $|S' \cap \mathcal{J}| = \min\{|\mathcal{J}|, \omega\}$. Finally, define $S' = S \cup I \cup J$. As $I \subset \mathcal{I}$ and $J \subset \mathcal{J}$, these two sets are disjoint, and S' obeys

$$|S' \cap \mathcal{I}| = \min\{|\mathcal{I}|, \omega\} \quad \text{and} \quad |S' \cap \mathcal{J}| = \min\{|\mathcal{J}|, \omega\}.$$

Clearly $S' \supset S$. Now, define $f_{S'} = f_S$. It is true that the function $f_{S'}$ only depends on variables that are in S', as it only depends on the variables in S. Moreover,

$$|S'| = |S' \cap \mathcal{I}| + |S' \cap \mathcal{J}| = \min\{|\mathcal{I}|, \omega\} + \min\{|\mathcal{J}|, \omega\} \leq 2\omega. \tag{17}$$

Finally, repeat the process to all $S \in \mathcal{S}$ to obtain $\mathcal{S}' = \{S'_1, S'_2, \ldots, S'_{|\mathcal{S}|}\}$, a partial separability decomposition of f that has the desired properties.

The following example illustrates the procedure above. Let $m = 7$, $\mathcal{I} = \{1, 2\}$ and $\mathcal{J} = \{3, 4, 5, 6, 7\}$ and $\mathcal{S}$ be composed of the sets $S_1 = \{1, 4\}$, $S_2 = \{2\}$, $S_3 = \{2, 5, 6\}$, and $S_4 = \{1, 3, 7\}$. In this case, $\omega = 3$, $\min\{|\mathcal{I}|, \omega\} = 2$ and $\min\{|\mathcal{J}|, \omega\} = 3$.

For S_1, we have $|S_1 \cap \mathcal{I}| = 1 < 2 = \min\{|\mathcal{I}|, \omega\}$, and we can define $I_1 = \{2\}$. As for $|S_1 \cap \mathcal{J}| = 1 < 3 = \min\{|\mathcal{J}|, \omega\}$ and we choose J_1 as $\{3, 5\}$. Hence, $S'_1 = S_1 \cup I_1 \cup J_1 = \{1, 2, 3, 4, 5\}$. Similarly, we get $S'_2 = \{1, 2, 3, 4, 5\}$, $S'_3 = \{1, 2, 3, 5, 6\}$, $S'_4 = \{1, 2, 3, 4, 7\}$, and $\mathcal{S}' = \{S'_1, S'_2, S'_3, S'_4\}$. All intersections of the elements of $\mathcal{S}'$ with $\mathcal{I}$ and $\mathcal{J}$ have the same cardinality. The partial separability degree associated with $\mathcal{S}'$ is $5 = \min\{|\mathcal{I}|, \omega\} + \min\{|\mathcal{J}|, \omega\} < 2\omega$.

We summarize these ideas in the following lemma. Its proof is basically the discussion that precedes the example.

Lemma 1. *Lef $f = \sum_{S \in \mathcal{S}} f_S$ be a partially separable decomposition of f with degree ω and let $\mathcal{I}$ and $\mathcal{J}$ be two fixed and disjoint sets such that $\mathcal{I} \cup \mathcal{J} = [m]$. It is possible to extend the sets in $\mathcal{S}$ to form a new decomposition $\mathcal{S}'$ such that, for all $S' \in \mathcal{S}'$,*

$$|\mathcal{I} \cap S'| = \min\{|\mathcal{I}|, \omega\} \quad \textit{and} \quad |\mathcal{J} \cap S'| = \min\{|\mathcal{J}|, \omega\}.$$

Hence, the decomposition $\mathcal{S}'$ has degree $\omega' = \min\{|\mathcal{I}|, \omega\} + \min\{|\mathcal{J}|, \omega\} \leq 2\omega$.

We are now ready to state the main result of this subsection. It presents the expected decrease of the objective function of problem (1) whenever blocks of coordinates are updated following the probability distribution of Algorithm 1.

Lemma 2. *Consider $x \in \mathcal{X}$, $h^\beta = [h_1^\beta, h_2^\beta, \ldots, h_m^\beta]^T \in \mathbb{R}^n$, in which h_i^β is the solution of (9), and a random multiset $\mathcal{B}$ as in Proposition 1, with $|\mathcal{B}| = \tau$. Additionally, suppose that f is a partially separable function of degree ω, and let $\mathcal{I}$ and $\mathcal{J}$ be two fixed and disjoint sets of indices such that $\mathcal{I} \cup \mathcal{J} = [m]$. Then,*

$$\begin{aligned} \mathbb{E}\left[f\left(x + \sum_{i \in \mathcal{B}} U_i h_i^\beta\right)\right] \quad \leq \quad & f(x) + \sum_{i \in \mathcal{I}} \frac{\tau \delta_{DP}}{q} \left(\nabla_i f(x)^T h_i^\beta + \frac{\beta L_i}{2} \|h_i^\beta\|_{(i)}^2\right) \\ & + \sum_{i \in \mathcal{J}} \frac{\tau}{q} \left(\nabla_i f(x)^T h_i^\beta + \frac{\beta L_i}{2} \|h_i^\beta\|_{(i)}^2\right), \end{aligned} \tag{18}$$

with $\beta = \dfrac{(\tau - 1)(\delta_{DP} \min\{|\mathcal{I}|, \omega\} + \min\{|\mathcal{J}|, \omega\})}{q} + 1$, *and* $q = \delta_{DP}|\mathcal{I}| + |\mathcal{J}|$.

Proof. Without loss of generality, we start by applying Lemma 1 to build a new set $\mathcal{S}'$ such that $f(x) = \sum_{S' \in \mathcal{S}'} f_{S'}(x)$, which satisfies

$$\begin{cases} |\mathcal{I} \cap S'| = \min\{|\mathcal{I}|, \omega\}, \ \forall \ S' \in \mathcal{S}', \\ |\mathcal{J} \cap S'| = \min\{|\mathcal{J}|, \omega\}, \ \forall \ S' \in \mathcal{S}', \end{cases} \tag{19}$$

with partial separability degree $\omega' = \min\{|\mathcal{I}|, \omega\} + \min\{|\mathcal{J}|, \omega\}$.

Now, to simplify the notation, let $H := \sum\limits_{i \in \mathcal{B}} U_i h_i^\beta$, and define the functions

$$\phi(h) := f(x+h) - f(x) - \nabla f(x)^T h,$$

$$\phi_{S'}(h) := f_{S'}(x+h) - f_{S'}(x) - \nabla f_{S'}(x)^T h, \forall \ S' \in \mathcal{S}',$$

in which $f_{S'}, \ S' \in \mathcal{S}'$, are those functions that come from the partial separability of f (cf. Definition 1). Since

$$\begin{aligned} \mathbb{E}\left[\phi(H)\right] &= \mathbb{E}\left[f(x+H) - f(x) - \nabla f(x)^T H\right] \\ &= \mathbb{E}\left[f(x+H)\right] - f(x) - \sum_{i=1}^{m} \nabla_i f(x)^T h_i^\beta \mathbb{P}(i \in \mathcal{B}) \\ &= \mathbb{E}\left[f(x+H)\right] - f(x) - \sum_{i \in \mathcal{I}} \frac{\tau \delta_{DP}}{q} \nabla_i f(x)^T h_i^\beta + \\ &\quad - \sum_{i \in \mathcal{J}} \frac{\tau}{q} \nabla_i f(x)^T h_i^\beta, \end{aligned}$$

to obtain the desired result, it is enough to ensure that

$$\mathbb{E}\left[\phi(H)\right] \leq \sum_{i \in \mathcal{I}} \frac{\tau \delta_{DP}}{q} \left(\frac{\beta L_i}{2} \|h_i^\beta\|_{(i)}^2 \right) + \sum_{i \in \mathcal{J}} \frac{\tau}{q} \left(\frac{\beta L_i}{2} \|h_i^\beta\|_{(i)}^2 \right), \tag{20}$$

for some $\beta \in \mathbb{R}$.

Resting upon the sets $S' \in \mathcal{S}'$ from Lemma 1, the expected value of function ϕ may be expressed as the following sum of conditional expectations:

$$\mathbb{E}\left[\phi\left(H\right)\right] = \sum_{k=0}^{\tau} \sum_{S' \in \mathcal{S}'} \mathbb{P}(|(S', \mathcal{B})| = k) \mathbb{E}\left[\phi_{S'}\left(H\right) \ | \ |(S', \mathcal{B})| = k\right]. \tag{21}$$

The relations in (19) guarantee that the cardinalities $|\mathcal{I} \cap S'|$ and $|\mathcal{J} \cap S'|$ do not depend on the sets $S' \in \mathcal{S}'$. From (16), the probability $\mathbb{P}(|(S', \mathcal{B})| = k)$ is also independent on $S' \in \mathcal{S}'$ for all fixed k, $0 \leq k \leq \tau$. Therefore, rewriting (21) we obtain

$$\mathbb{E}\left[\phi\left(H\right)\right] \leq \sum_{k=0}^{\tau} \mathbb{P}(|(S', \mathcal{B})| = k) \sum_{S' \in \mathcal{S}'} \mathbb{E}\left[\phi_{S'}(H) \ \middle| \ |(S', \mathcal{B})| = k\right]. \tag{22}$$

For $k = 0$, and for all $S' \in \mathcal{S}'$, the expected value of $\phi_{S'}$ in (22) may be expressed as

$$\begin{aligned}
\mathbb{E}\left[\phi_{S'}\left(H\right) \mid |(S', \mathcal{B})| = 0\right] &= \mathbb{E}\left[\phi_{S'}\left(\sum_{i \in (S', \mathcal{B})} U_i h_i^\beta\right) \;\middle|\; |(S', \mathcal{B})| = 0\right] \\
&= \mathbb{E}\left[\phi_{S'}\left(\sum_{i \in \emptyset} U_i h_i^\beta\right)\right] \\
&= \phi_{S'}(0) \\
&= f_{S'}(x+0) - f_{S'}(x) - \nabla f_{S'}(x)^T 0 \\
&= 0. \qquad (23)
\end{aligned}$$

For each fixed k, $1 \leq k \leq \tau$, from the convexity of the function $\phi_{S'}$ we have

$$\begin{aligned}
\mathbb{E}\left[\phi_{S'}\left(H\right) \mid |(S', \mathcal{B})| = k\right] &= \mathbb{E}\left[\phi_{S'}\left(\frac{1}{k}\sum_{i \in (S', \mathcal{B})} k U_i h_i^\beta\right) \;\middle|\; |(S', \mathcal{B})| = k\right] \\
&\leq \mathbb{E}\left[\frac{1}{k}\sum_{i \in (S', \mathcal{B})} \phi_{S'}\left(k U_i h_i^\beta\right) \;\middle|\; |(S', \mathcal{B})| = k\right] \\
&= \frac{1}{k}\mathbb{E}\left[\sum_{i \in (S', \mathcal{B})} \phi_{S'}\left(k U_i h_i^\beta\right) \;\middle|\; |(S', \mathcal{B})| = k\right] \\
&= \frac{1}{k}\sum_{i \in S'} \phi_{S'}\left(k U_i h_i^\beta\right) \mathbb{P}(i \in \mathcal{B} \mid |(S', \mathcal{B})| = k) \\
&\overset{(10)}{=} \frac{1}{k}\left[\sum_{i \in S' \cap \mathcal{I}} \phi_{S'}\left(k U_i h_i^\beta\right) \frac{k\delta_{DP}}{\delta_{DP}|\mathcal{I} \cap S'| + |\mathcal{J} \cap S'|} + \right. \\
&\quad \left. + \sum_{i \in S' \cap \mathcal{J}} \phi_{S'}\left(k U_i h_i^\beta\right) \frac{k}{\delta_{DP}|\mathcal{I} \cap S'| + |\mathcal{J} \cap S'|}\right] \\
&= z\left[\sum_{i \in S' \cap \mathcal{I}} \delta_{DP} \phi_{S'}\left(k U_i h_i^\beta\right) + \sum_{i \in S' \cap \mathcal{J}} \phi_{S'}\left(k U_i h_i^\beta\right)\right], \qquad (24)
\end{aligned}$$

for all $S' \in \mathcal{S}'$ and $z := \dfrac{1}{\delta_{DP}|\mathcal{I} \cap S'| + |\mathcal{J} \cap S'|}$.

Applying expressions (23) and (24) to (22) yields

$$\begin{aligned}\mathbb{E}\left[\phi\left(H\right)\right] &\leq \sum_{k=1}^{\tau}\mathbb{P}(|(S',\mathcal{B})|=k)\sum_{S'\in\mathcal{S}'} z\left[\sum_{i\in S'\cap\mathcal{I}}\delta_{DP}\phi_{S'}\left(kU_ih_i^{\beta}\right)+\right.\\ &\qquad\left.+\sum_{i\in S'\cap\mathcal{J}}\phi_{S'}\left(kU_ih_i^{\beta}\right)\right]\\ &= z\sum_{k=1}^{\tau}\mathbb{P}(|(S',\mathcal{B})|=k)\left[\sum_{i\in\mathcal{I}}\delta_{DP}\phi\left(kU_ih_i^{\beta}\right)+\sum_{i\in\mathcal{J}}\phi\left(kU_ih_i^{\beta}\right)\right]\\ &\leq z\sum_{k=1}^{\tau}\mathbb{P}(|(S',\mathcal{B})|=k)\left[\sum_{i\in\mathcal{I}}\delta_{DP}\frac{L_i}{2}\|kh_i^{\beta}\|_{(i)}^2+\sum_{i\in\mathcal{J}}\frac{L_i}{2}\|kh_i^{\beta}\|_{(i)}^2\right]\\ &= z\sum_{k=1}^{\tau}\mathbb{P}(|(S',\mathcal{B})|=k)k^2\left[\sum_{i\in\mathcal{I}}\delta_{DP}\frac{L_i}{2}\|h_i^{\beta}\|_{(i)}^2+\sum_{i\in\mathcal{J}}\frac{L_i}{2}\|h_i^{\beta}\|_{(i)}^2\right].\end{aligned} \tag{25}$$

Now, from [21, (24)], it holds

$$\begin{aligned}\mathbb{E}\left[\left(|(S',\mathcal{B})|\right)^2\right] &= \sum_{k=1}^{\tau}\mathbb{P}(|(S',\mathcal{B})|=k)k^2\\ &= \tau p_1(\tau p_1+p_2)=\tau p_1((\tau-1)p_1+1),\end{aligned} \tag{26}$$

since $p_2=1-p_1$, with p_1 and p_2 as established in Proposition 1.

Through the definition of p_1 and (19), we obtain

$$p_1=\frac{\delta_{DP}|\mathcal{I}\cap S'|+|\mathcal{J}\cap S'|}{\delta_{DP}|\mathcal{I}|+|\mathcal{J}|}=\frac{\delta_{DP}\min\{|\mathcal{I}|,\omega\}+\min\{|\mathcal{J}|,\omega\}}{q}. \tag{27}$$

Using (26), (27) and $q=\delta_{DP}|\mathcal{I}|+|\mathcal{J}|$, inequality (25) becomes

$$\begin{aligned}\mathbb{E}\left[\phi\left(H\right)\right] &\leq z\tau p_1((\tau-1)p_1+1)\left[\sum_{i\in\mathcal{I}}\delta_{DP}\frac{L_i}{2}\|h_i^{\beta}\|_{(i)}^2+\sum_{i\in\mathcal{J}}\frac{L_i}{2}\|h_i^{\beta}\|_{(i)}^2\right]\\ &= \frac{\tau}{q}((\tau-1)p_1+1)\left[\sum_{i\in\mathcal{I}}\delta_{DP}\frac{L_i}{2}\|h_i^{\beta}\|_{(i)}^2+\sum_{i\in\mathcal{J}}\frac{L_i}{2}\|h_i^{\beta}\|_{(i)}^2\right]\\ &\overset{(27)}{=} \frac{\tau\delta_{DP}}{q}\sum_{i\in\mathcal{I}}\frac{\beta L_i}{2}\|h_i^{\beta}\|_{(i)}^2+\frac{\tau}{q}\sum_{i\in\mathcal{J}}\frac{\beta L_i}{2}\|h_i^{\beta}\|_{(i)}^2,\end{aligned} \tag{28}$$

with $\beta=\dfrac{(\tau-1)(\delta_{DP}\min\{|\mathcal{I}|,\omega\}+\min\{|\mathcal{J}|,\omega\})}{q}+1$.

As (28) is exactly inequality (20), the proof is complete. □

Owing to Lemma 2, an expression for the expected decrease of function $F=f+\psi$ may be attained by updating simultaneously τ blocks of coordinates, with each of these blocks obeying the probability distribution described in Algorithm 1.

Let $\mathcal{B}$ be a multiset of independent and identically distributed (i.i.d.) random variables, as previously defined, and let $h_i^\beta \in \mathbb{R}^{p_i}, i \in [m]$ be blocks of solution vectors of the subproblems (9). The expected value of F may be expressed as

$$\mathbb{E}\left[F\left(x+\sum_{i\in\mathcal{B}}U_i h_i^\beta\right)\right] = \mathbb{E}\left[f\left(x+\sum_{i\in\mathcal{B}}U_i h_i^\beta\right)\right] + \mathbb{E}\left[\psi\left(x+\sum_{i\in\mathcal{B}}U_i h_i^\beta\right)\right]. \tag{29}$$

Due to the separable structure of $\psi(x)$, we have

$$\begin{aligned}
\mathbb{E}\left[\psi\left(x+\sum_{i\in\mathcal{B}}U_i h_i^\beta\right)\right] &= \mathbb{E}\left[\sum_{i\in\mathcal{B}}\psi_i(x_{(i)}+h_i^\beta)+\sum_{i\notin\mathcal{B}}\psi_i(x_{(i)})\right] \\
&= \mathbb{E}\left[\sum_{i\in\mathcal{B}}(\psi_i(x_{(i)}+h_i^\beta)-\psi_i(x_{(i)}))+\sum_{i=1}^{m}\psi_i(x_{(i)})\right] \\
&= \sum_{i\in\mathcal{I}}\frac{\tau\delta_{DP}}{q}\psi_i(x_{(i)}+h_i^\beta)+\left(1-\frac{\tau\delta_{DP}}{q}\right)\psi_i(x_{(i)})+ \\
&\quad +\sum_{i\in\mathcal{J}}\frac{\tau}{q}\psi_i(x_{(i)}+h_i^\beta)+\left(1-\frac{\tau}{q}\right)\psi_i(x_{(i)}).
\end{aligned} \tag{30}$$

Using (18) and (30) in equality (29), we obtain

$$\begin{aligned}
\mathbb{E}\left[F\left(x+\sum_{i\in\mathcal{B}}U_i h_i^\beta\right)\right] &\leq F(x)+\sum_{i\in\mathcal{I}}\frac{\tau\delta_{DP}}{q}\left(\nabla_i f(x)^T h_i^\beta+\frac{\beta L_i}{2}\|h_i^\beta\|_{(i)}^2+\right. \\
&\quad \left.+\psi_i(x_{(i)}+h_i^\beta)-\psi_i(x_{(i)})\right)+\sum_{i\in\mathcal{J}}\frac{\tau}{q}\left(\nabla_i f(x)^T h_i^\beta+\right. \\
&\quad \left.+\frac{\beta L_i}{2}\|h_i^\beta\|_{(i)}^2+\psi_i(x_{(i)}+h_i^\beta)-\psi_i(x_{(i)})\right).
\end{aligned} \tag{31}$$

This ensures that solving τ subproblems (9) from a fixed vector $x \in \mathcal{X}$, as previously described, the direction $\sum_{i\in\mathcal{B}}U_i h_i^\beta(x)$ is expected to induce a decrease in F. Thus, we have shown to be theoretically sound to base an algorithm on this type of updating to compute descent directions for problem (1).

This leads directly to Algorithm 2 (Parallel Active BCDM). It is a parallel version of Algorithm 1, supported by the theoretical development of this subsection. Note that the algorithm does not depend on the specific sets in the partial separability decomposition of the objective $\mathcal{S}$. It only depends on the partial separability degree of f and the sizes of the sets $\mathcal{I}$ and $\mathcal{J}$. This explains the formula for β appearing in the method.

Algorithm 2 also uses other parameters as inputs. The parameter τ is the number of threads available for the method; δ_F controls the number of iterations the algorithm performs before reevaluating the identification function. Furthermore, δ_{DP} determines how many times the probability of selecting inactive blocks should be higher than that of active blocks.

In the outer loop (starting at line 7 of the algorithm), it selects τ coordinate blocks according to the probability distribution $\mathbb{P}(i)$ and update their coordinates in parallel using our proposed descent direction. At the end of each cycle of γ parallel iterations, the sets of active and inactive coordinate blocks are updated using the identification function (as shown in lines 15 and 16 of the algorithm).

Algorithm 2 Parallel BCDM with Identification of Active Variables (Parallel Active BCDM)

1: Choose an initial point $l \leq x^0 \leq u$, a block separation of ψ function described as $x = \sum_{i=1}^{m} U_i x_{(i)}$, an initial cycle size c_0, the parameters $\ell_{\max} \in \mathbb{N}$, $\varepsilon \in \mathbb{R}_+$, ω the partial separable degree of f, the number of threads τ and two natural numbers δ_F, δ_{DP}. Initialize a vector $v \in \mathbb{R}^n$ with 2ε in all positions, the sets $\mathcal{I} = [m]$, $\mathcal{J} = \emptyset$, the initial cycle size $c_s = c_0$, and the counters $\ell, j = 0$.
2: Calculate the Lipschitz constants of the gradient by blocks satisfying (3).
3: **repeat**
4: $q = \delta_{DP}|\mathcal{I}| + |\mathcal{J}|$;
5: $\gamma \longleftarrow \lfloor \frac{c_s}{\tau} \rfloor$;
6: $\beta = \dfrac{(\tau - 1)(\delta_{DP} \min\{|\mathcal{I}|, \omega\} + \min\{|\mathcal{J}|, \omega\})}{q} + 1$, with $q = \delta_{DP}|\mathcal{I}| + |\mathcal{J}|$;
7: **for** $k = (j-1)\gamma + 1, \ldots, j\gamma$ **do**
8: Choose a multiset $\mathcal{B}^k \subset [m]$ of τ blocks of coordinates, where each element of $\mathcal{B}^k$ satisfies the probability distribution

$$\mathbb{P}(i) = \begin{cases} \dfrac{\delta_{DP}}{\delta_{DP}|\mathcal{I}| + |\mathcal{J}|}, & \text{if } i \in \mathcal{I}, \\ \dfrac{1}{\delta_{DP}|\mathcal{I}| + |\mathcal{J}|}, & \text{if } i \in \mathcal{J}. \end{cases}$$

9: **for each** $i \in \mathcal{B}^k$ **do in parallel**
10: Find $h_i^\beta \equiv h_i^\beta(x^k)$, a solution to the subproblem (9)
11: Set $x^{k+1} = x^k + U_i h_i^\beta$ and $v_{(i)} = h_i^\beta$.
12: **end parallel for**
13: **end for**
14: Set $j = j + 1$ and $\ell = \ell + \gamma\tau$.
15: Obtain the set $\mathcal{C}(x^\ell) \subset [m]$, where

$$\mathcal{C}(x^\ell) = \left\{ i \mid g_j(x^\ell) \geq \rho_\alpha(x^\ell), \forall x_j^\ell \text{ s.t. } j \text{ belongs to the } i\text{th-block} \right\},$$

with the identification function $\rho_\alpha(x)$ described in (8).
16: Define $\mathcal{J} \subset \mathcal{C}(x^\ell)$ and $\mathcal{I} = [m] \setminus \mathcal{J}$. Set $c_s = \max\{\min\{\delta_F|\mathcal{I}|, m\}, c_0\}$.
17: **until** $\|v\| \leq \varepsilon$ or $\ell \geq \ell_{\max}$

4.2 Convergence results

Using the notation above, the global convergence and the complexity analysis of the Algorithm 2 are obtained as follows.

Definition 3. *Let* $Q(h, x, \gamma) : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}_+ \to \mathbb{R}$ *and* $G(h, x, \gamma) : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}_+ \to \mathbb{R}$ *be the functions*

$$Q(h, x, \gamma) := f(x) + \nabla f(x)^T h + \frac{\gamma}{2}\|h\|_2^2 + \psi(x + h)$$

and

$$G(h, x, \gamma) := \nabla f(x)^T h + \frac{\gamma}{2}\|h\|_2^2 + \psi(x + h) - \psi(h).$$

Additionally, we define $G_i(h_i, x, \gamma) : \mathbb{R}^{p_i} \times \mathbb{R}^n \times \mathbb{R}_+ \to \mathbb{R}$ *as*

$$G_i(h_i, x, \gamma) := \nabla_i f(x)^T h_i + \frac{\gamma}{2}\|h_i\|_2^2 + \psi_i(x_{(i)} + h_i) - \psi_i(x_{(i)})$$

and, for $i \in [m]$.

Notice that with this definition

$$\begin{aligned} h_i^\beta(x) = \underset{h_i \in \mathbb{R}^{p_i}}{\operatorname{argmin}} \quad & G_i(h_i, x, \beta) \\ \text{s.t.} \quad & l_i \le x_{(i)} + h_i \le u_i \end{aligned} \tag{32}$$

and

$$\begin{aligned} h^\beta(x) = \underset{h \in \mathbb{R}^n}{\operatorname{argmin}} \quad & G(h, x, \beta). \\ \text{s.t.} \quad & l \le x + h \le u \end{aligned} \tag{33}$$

Definition 4. $\mathcal{X}^*$ *is the set of minimizers of problem* (1).

Assumption 1. *The set* X^* *is not empty, and the minimum value of the problem is* $F^* = \min_{x \in \mathcal{X}} F(x)$.

The error bound condition stated next comes from [26, EB condition (47)]. This condition can be verified in some situations. For example, when f is strongly convex and has Lipschitz continuous gradient. Another is whenever f is a quadratic function (even nonconvex) and ψ is a polyhedral function, see [26, 27] for more examples.

Assumption 2. *Let* $\varrho \ge F^*$, *suppose there exist* $\epsilon_1, \epsilon_2 > 0$ *such that*

$$dist(x, \mathcal{X}^*) \le \epsilon_2 \|h(x)\|, \ \textit{whenever } F(x) \le \varrho, \ \|h(x)\| \le \epsilon_1,$$

where $dist(x, \mathcal{X}^*) = \min_{y \in \mathcal{X}^*} \|x - y\|$ *and* $h(x)$ *is the solution of* (7).

Assumption 3. *The gradient of* f *is globally Lipschitz continuous with respect to the Euclidean norm, i.e., there exists* $L_f > 0$ *such that*

$$\|\nabla f(y) - \nabla f(x)\| \le L_f \|y - x\|, \ \forall x, y \in \mathbb{R}^n.$$

A classical consequence of the last assumption is described next.

Corollary 1. *[5, Lemma 4.1.12] Under Assumption 3,*

$$|f(y) - f(x) - \nabla f(x)^T(y-x)| \leq \frac{L_f}{2}\|y-x\|^2.$$

Corollary 2. *A point $x^* \in \mathcal{X}^*$ if, and only if, $h_i(x^*) = 0$ for all $i \in [m]$, with $h(x)$ the solution of (7).*

Proof. Follows as an immediate consequence of [13, Lemma 2]. □

We now present some auxiliary results that will be useful below.

Lemma 3. *Let $\{x^k\}$ be the sequence generated by Algorithm 2 (Parallel Active BCDM) and $\mathcal{B}^k \subset [m]$ be a multiset of randomly generated i.i.d. indices that controls the choice of τ blocks of coordinates at the k-th iteration of the method, where each element of the multiset $\mathcal{B}^k$ is chosen by the probability distribution of Algorithm 2. Then,*

$$F(x^k) - \mathbb{E}\left[F(x^{k+1})|\mathcal{B}^k\right] \geq \frac{1}{2m\delta_{DP}}\|h(x^k)\|^2,$$

where $\mathbb{E}\left[F(x^{k+1})|\mathcal{B}^k\right] = \mathbb{E}\left[F\left(x^k + \sum\limits_{i\in\mathcal{B}^k} U_i h_i^{\beta_k}(x^k)\right)\right]$, with $h_i^{\beta_k}(x^k)$ solution of (9), $i \in [m]$, and $h(x)$ the solution of (7).

Proof. Due to the dynamics of Algorithm 2 and Equation (31), we have

$$\begin{aligned}
\mathbb{E}\left[F(x^{k+1})|\mathcal{B}^k\right] &\leq F(x^k) + \sum_{i\in\mathcal{I}} \frac{\tau\delta_{DP}}{q} G_i(h_i^{\beta_k}(x^k), x^k, \beta_k) + \\
&\quad + \sum_{i\in\mathcal{J}} \frac{\tau}{q} G_i(h_i^{\beta_k}(x^k), x^k, \beta_k) \\
&\leq F(x^k) + \sum_{i=1}^{m} \frac{\tau\delta_{DP}}{q} G_i(h_i^{\beta_k}(x^k), x^k, \beta_k). \qquad (34)
\end{aligned}$$

First, we use the fact that $|\mathcal{I}| + |\mathcal{J}| = m$, despite of the change of the sets $\mathcal{I}$ and $\mathcal{J}$ along the iterations. Second, since $G_i(h_i^{\beta_k}(x^k), x^k, \beta_k) \leq G_i(h_i^{L_i\beta_k}(x^k), x^k, L_i\beta_k) \leq 0$, where $h_i^{\beta_k}$ and $h_i^{L_i\beta_k}(x^k)$ are the minimizers of the subproblem (32), for all $i \in [m]$, each one with your specific β. Putting all this together, we get

$$\begin{aligned}
&\frac{\tau\delta_{DP}}{q} \geq \frac{1}{m\delta_{DP}} \\
\Leftrightarrow\quad &\frac{\tau\delta_{DP}}{q} G_i(h_i^{\beta_k}(x^k), x^k, \beta_k) \leq \frac{1}{m\delta_{DP}} G_i(h_i^{L_i\beta_k}(x^k), x^k, L_i\beta_k) \\
\Leftrightarrow\quad &\frac{\tau\delta_{DP}}{q} G_i(h_i^{\beta_k}(x^k), x^k, \beta_k) \leq \frac{1}{m\delta_{DP}} G_i(h_i^{\overline{\beta}}(x^k), x^k, \overline{\beta}), \qquad (35)
\end{aligned}$$

with

$$\overline{\beta} := \beta_{\max}\lambda_{\max}L_{\max}, \tag{36}$$

where $\beta_{\max} = 2(\tau-1)\omega(\delta_{DP}+1)/m$, $\lambda_{\max}$ is the largest eigenvalue of the matrix $B \in \mathbb{R}^{n\times n}$, and $L_{\max} = \max_{1\leq i\leq m}\{L_i\}$.

Through the expression (35), the inequality (34) admits the following upper bound

$$\begin{aligned}
\mathbb{E}\left[F(x^{k+1})|\mathcal{B}^k\right] &\overset{(35)}{\leq} F(x^k) + \frac{1}{m\delta_{DP}}\sum_{i=1}^{m} G_i(h_i^{\overline{\beta}}(x^k), x^k, \overline{\beta}) \\
&= F(x^k) + \frac{1}{m\delta_{DP}} G(h^{\overline{\beta}}(x^k), x^k, \overline{\beta}) \\
&= \left(1 - \frac{1}{m\delta_{DP}}\right) F(x^k) + \frac{1}{m\delta_{DP}}\left(Q(h^{\overline{\beta}}(x^k), x^k, \overline{\beta})\right). \qquad (37)
\end{aligned}$$

From the relationship (37) and because the function $Q(\cdot, x^k, \overline{\beta})$ is strongly convex with respect to the norm $\|\cdot\|_2$, it holds that

$$\begin{aligned}
F(x^k) - \mathbb{E}\left[F(x^{k+1})|\mathcal{B}^k\right] &\geq \frac{1}{m\delta_{DP}} F(x^k) - \frac{1}{m\delta_{DP}} Q(h^{\overline{\beta}}(x^k), x^k, \overline{\beta}) \\
&= \frac{1}{m\delta_{DP}}(Q(0, x^k, \overline{\beta}) - Q(h^{\overline{\beta}}(x^k), x^k, \overline{\beta}) \\
&\geq \frac{\overline{\beta}}{2m\delta_{DP}} \|h^{\overline{\beta}}(x^k)\|^2. \qquad (38)
\end{aligned}$$

Assuming $\overline{\beta} \geq 1$, which may be achieved without loss of generality by increasing the Lipschitz constants L_i or choosing conveniently the matrix B, and using [18, Lemma 4], we see that

$$\overline{\beta}\|h^{\overline{\beta}}(x^k)\|^2 \geq \|h(x^k)\|^2. \tag{39}$$

Putting the expression (38) and (39) together concludes the proof. □

Lemma 4. *Let $x \notin \mathcal{X}^*$. Then, there exist $\gamma, \delta > 0$ such that, for all feasible $y \in \mathbb{B}(x, \delta)$, we get $\|h(y)\| \geq \gamma\|h(x)\|$, with $h(\cdot)$ the solution of* (7).

Proof. Just apply the same arguments of [13, Lemma 4]. □

The following result establishes a global convergence property for Algorithm 2. We show that every sequence generated by the algorithm converges to a point $x^* \in \mathcal{X}$ by applying Corollary 2, thus guaranteeing that $h_i(x^*) = 0$ for all $i \in [m]$, which is equivalent to $h(x^*) = \sum_{i=1}^{m} U_i h_i(x^*) = 0$.

Theorem 1. *Let $\{x^k\}$ be an infinite sequence generated by Algorithm 2 (Parallel Active BCDM) under the hypotheses of Lemma 3. Then, $\|h(x^k)\| \to 0$, with $h(\cdot)$ the solution of* (7).

Proof. Without loss of generality, we admit that $x^k \notin \mathcal{X}^*$. If $x^p \in \mathcal{X}^*$, for some $p \in \mathbb{N}$, $\|h(x^k)\| = 0$ for all $k \geq p$. Suppose, by contradiction, that $\|h(x^k)\| \not\to 0$, then there exist $\mu > 0$ and a subsequence $\mathbb{N}_1 \subset \mathbb{N}$ such that

$$\|h(x^k)\| \geq \mu, \ k \in \mathbb{N}_1.$$

Combining the last expression with Lemma 3, yields the following inequality:

$$\mathbb{E}\left[F(x^{k+1})|\mathcal{B}^k\right] - F(x^k) \leq -\frac{\mu^2}{2m\delta_{DP}}, \ k \in \mathbb{N}_1. \tag{40}$$

Taking the expectation on $\{\mathcal{B}^k\}_{k\in\mathbb{N}}$ in inequality (40), since the random variables are i.i.d., we get

$$\mathbb{E}\left[F(x^{k+1})\right] - \mathbb{E}\left[F(x^k)\right] \leq -\frac{\mu^2}{2m\delta_{DP}}, \ k \in \mathbb{N}_1. \tag{41}$$

From the expression (31) and Assumption 1, we have that the sequence $\{\mathbb{E}\left[F(x^k)\right]\}_{k\in\mathbb{N}}$ is nonincreasing and bounded below, therefore, it converges. Its convergence along with (41) guarantees the contradiction, since the left side term of (41) goes to zero, while the right side term of (41) is negative. □

Finally, we present a complexity result for Algorithm 2, estimating the expected objective decrease in its sequences.

Theorem 2. *Let $\{x^k\}$ be a sequence generated by the Parallel Active BCDM Algorithm and suppose that the Assumptions 1, 2 and 3 are verified. Then, there exists $k_1 \in \mathbb{N}$ with the following linear convergence rate for the expected values of the objective function*

$$\mathbb{E}\left[F(x^k)\right] - F^* \leq \left(1 - \frac{1}{m\delta_{DP}(1 + (\overline{\beta} + L_f)\epsilon_2^2)}\right)^{k-k_1} (F(x^0) - F^*),$$

with $\overline{\beta}$ as in (36) *(Lemma 3), ϵ_2 as in Assumption 2 and L_f as in Assumption 3.*

Proof. From Theorem 1 we have $\|h(x^k)\| \to 0$. Hence, Assumption 2 ensures that, for a fixed $\varrho = F(x^0)$, there exist $\epsilon_2 > 0$ and $k_1 \in \mathbb{N}$ such that

$$\|x^k - z^k\| \leq \epsilon_2 \|h(x^k)\|, \ \forall \ k \geq k_1, \tag{42}$$

in which $z^k \in \mathcal{X}^*$ satisfies $\|x^k - z^k\| = \text{dist}(x^k, \mathcal{X}^*)$.

From Corollary 1, we have

$$f(x) + \nabla f(x)^T(y - x) \leq f(y) + \frac{L_f}{2}\|y - x\|^2, \ \forall \ y, x \in \mathbb{R}^n. \tag{43}$$

Adding $\frac{\overline{\beta}}{2}\|x-y\|^2+\psi(y)$ to both sides of inequality (43) yields

$$\begin{aligned} Q(y-x,x,\overline{\beta}) &= f(x)+\nabla f(x)^T(y-x)+\frac{\overline{\beta}}{2}\|x-y\|^2+\psi(y) \\ &\leq f(y)+\frac{\overline{\beta}+L_f}{2}\|x-y\|^2+\psi(y) \\ &= F(y)+\frac{\overline{\beta}+L_f}{2}\|x-y\|^2. \end{aligned} \tag{44}$$

As a consequence of (44), we obtain

$$Q(z^k-x^k,x^k,\overline{\beta}) \leq F^*+\frac{c}{2}\|x^k-z^k\|^2,\ c=\overline{\beta}+L_f. \tag{45}$$

Putting together the expressions (37), (45), and using that $\min_{s\in\mathcal{X}} Q(h,x^k,\overline{\beta}) = Q(h^{\overline{\beta}}(x^k),x^k,\overline{\beta})$, we have

$$\begin{aligned} \mathbb{E}\left[F(x^{k+1})|\mathcal{B}^k\right] &\leq \left(1-\frac{1}{m\delta_{DP}}\right)F(x^k)+\frac{1}{m\delta_{DP}}\left(F^*+\frac{c}{2}\|x^k-z^k\|^2\right) \\ &= F(x^k)-\frac{1}{m\delta_{DP}}(F(x^k)-F^*)+\frac{c}{2m\delta_{DP}}\|x^k-z^k\|^2, \end{aligned} \tag{46}$$

for all $k\geq k_1$.

Subtracting F^* from both sides of inequality (46) and using the expression (42), we obtain

$$\begin{aligned} \mathbb{E}\left[F(x^{k+1})|\mathcal{B}^k\right]-F^* &\leq \left(1-\frac{1}{m\delta_{DP}}\right)(F(x^k)-F^*)+ \\ &\quad +\frac{c\epsilon_2^2}{2m\delta_{DP}}\|h(x^k)\|^2, \end{aligned} \tag{47}$$

for all $k\geq k_1$.

In view of Lemma 3, the relationship (47) turns into

$$\begin{aligned} \mathbb{E}\left[F(x^{k+1})|\mathcal{B}^k\right]-F^* &\leq \left(1-\frac{1}{m\delta_{DP}}\right)(F(x^k)-F^*)+ \\ &\quad +c\epsilon_2^2(F(x^k)-\mathbb{E}\left[F(x^{k+1})|\mathcal{B}^k\right]) \\ &= \left(1-\frac{1}{m\delta_{DP}}+c\epsilon_2^2\right)(F(x^k)-F^*)+ \\ &\quad +c\epsilon_2^2(F^*-\mathbb{E}\left[F(x^{k+1})|\mathcal{B}^k\right]). \end{aligned}$$

This may be rewritten as

$$\mathbb{E}\left[F(x^{k+1})|\mathcal{B}^k\right]-F^* \leq \left(1-\frac{1}{m\delta_{DP}(1+c\epsilon_2^2)}\right)(F(x^k)-F^*), \tag{48}$$

for all $k\geq k_1$.

Taking both sides of (48) to the expectation conditioned to $\{\mathcal{B}^k\}_{k\in\mathbb{N}}$, which are i.i.d. random variables, yields

$$\mathbb{E}\left[F(x^{k+1})\right] - F^* \leq \left(1 - \frac{1}{m\delta_{DP}(1+c\epsilon_2^2)}\right)\left(\mathbb{E}\left[F(x^k)\right] - F^*\right), \qquad (49)$$

for all $k \geq k_1$.

The convergence result follows by applying the expression (49) repeatedly, since by (31), the sequence $\{\mathbb{E}\left[F(x^k)\right]\}$ is nonincreasing. □

The complexity result above includes the term $k - k_1$ in the exponent of the linear rate, meaning that linear convergence is only ensured from iteration k_1 on. This is needed to accommodate the nature of Assumption 2. This assumption only asserts the error bound when F and h are small enough, that we assume will happen at iteration k_1 in the proof of Theorem 2. However, in some special cases it is possible to estimate k_1. A notable example is when f is strongly convex with Lipschitz continuous gradients. In this case, [28, Theorem 4] shows that Assumption 2 holds with $\epsilon = \infty$ and independently of ρ for all $x \in \mathcal{X}$. Hence, in this case $k_1 = 0$, and Theorem 2 is a global linear convergence result in expectation. For other cases, estimating k_1 may be difficult, or even impossible, and problem dependent.

5 Numerical tests

This section presents the numerical behavior of the parallel version of Active BCDM, Algorithm 2, denoted `PA` from now on. In [13], the authors introduced its serial variant. It showed that active constraints identification combined with a nonuniform choice of coordinate blocks was very efficient and competitive with several well-established methods in the literature. Therefore, in this work, we will focus on how parallelism can accelerate this method and compare the effectiveness of the acceleration achieved by `PA` with the one achieved by its uniform counterpart.

To accomplish this, one might consider it enough to compare the behaviors of the new `PA` code and `PCDM`, the standard implementation of parallel uniform block selection from [21]. However, as `PA` evolved from the code in [13], it is implemented in Fortran 90, while `PCDM` is implemented in C++. This difference, and the fact that such codes were developed by different groups, could be partially responsible for the perceived differences in performance. We introduced our implementation of the uniform selection method in Fortran 90, `UBCDM`, to mitigate this. Note that Fortran 90 is considered to produce slightly faster executables due to its programming model that allows the compiler to optimize the generated binary better. This should be considered when interpreting the results below. We also point out that we needed to perform minimal changes to `PCDM` to enable repeated executions for each test instance so that the random selection of the blocks could be taken into account.

All experiments were performed on a Ryzen $97950X3D$ 16C/32T system with 128 GB of memory, running Ubuntu Linux 22.04.4 LTS. They were compiled using version 11.4.0 of the GNU compilers, and the parallelization is achieved using OpenMP. Since the CPU has 16 processing cores with independent floating point units, the experiments were limited to use a maximum of 16 threads always pinned to run in different cores. For each test case, the codes were executed with the number of threads τ within the set $\{1, 2, 4, 8, 16\}$.

Throughout the section, the class of problems tested is the well-known Lasso [24], given by

$$\min_x \frac{1}{2}\|Ax - b\|_2^2 + \lambda\|x\|_1, \quad \lambda > 0, \tag{50}$$

which can be reformulated as a problem with simple constraints

$$\begin{aligned} \min_{x_+, x_-} \quad & \frac{1}{2}\|A(x_+ - x_-) - b\|^2 + \lambda e^T(x_+ + x_-) \\ \text{s.t.} \quad & x_+ \geq 0 \\ & x_- \geq 0, \end{aligned} \tag{51}$$

where $e = (1, \ldots, 1)^T$. Note that all solutions $\overline{x}$ of (50) can be written in terms of solutions $(\overline{x}_+, \overline{x}_-)$ of (51), using $\overline{x} = \overline{x}_+ - \overline{x}_-$.

Adopting the same choices of [21], the initial point x^0 was set as the null vector; the initial cycle size c_0 was the problem dimension n, and $\ell_{\max} = 1000n$. Moreover, for updating the set $\mathcal{J}$, we have used the strategy described in [21, Section 6.1].

All algorithms evaluated in this study are variants of coordinate descent methods that operate on coordinate blocks of size one, i.e., individual variables. To account for the stochasticity inherent in the coordinate selection process, each algorithm was executed multiple times on the same problem instance. Specifically, 20 independent runs were performed for the test cases described in Subsection 5.1, and 100 runs for those in Subsection 5.2, provided that the cumulative execution time to solve the instance exceeded one second. Otherwise, a sufficient number of runs were performed to complete one second of cumulative time.

Each run was stopped once the objective value of an iteration reached a predefined target, which closely approximates the known optimal value. More details on how such a target was obtained are presented in the sections describing each test scenario below. Finally, the primary performance metric used in this study is the average running time of all independent runs.

Analogously to what was done in [21], even though the theory of Parallel Active BCDM was originally formulated for the synchronous case, the coordinate blocks are updated asynchronously. The updates of the blocks (of size 1) have a very low cost, compared to the effort necessary to synchronize the gradient update of the smooth part of (50) after the calculation of the descent directions. This makes synchronous implementation impractical.

In the serial case, the gradient can be efficiently updated at a low computational cost immediately following the modification of a single coordinate. However, in the asynchronous setting, multiple threads concurrently read from and write to the shared gradient vector, leading to a "race condition" that can cause the gradient to accumulate numerical errors over time. This gradient degradation can severely impact the algorithm's convergence.

To mitigate this issue, our implementation includes a correction routine that runs after each cycle that comprises n coordinate updates. This routine recomputes and corrects the shared gradient vector to restore numerical consistency. Additionally, this synchronization point is leveraged within the framework of Algorithm 2 to identify the active constraints, thus improving the algorithmic performance.

5.1 A controlled and highly favorable scenario

The initial experiments were performed in a controlled environment that is particularly favorable to coordinate descent methods. As theoretical results suggest, the smaller the degree of partial separability ω, the greater the expected acceleration for these methods. For problem (50), the value of ω depends on the number of nonzero elements in the rows of the matrix A.

To generate the initial test instances, we employed the random problem generator of [21], which the authors provide in a C++ implementation. This tool builds a class of test problems originally described in [16]. Using this generator, it is possible to construct a matrix A and a vector b for the problem (50), assuming $\lambda = 1$, based on the following input parameters: the dimensions of matrix A; the number of nonzero entries per row in matrix A, and the sparsity level of the optimal solution, that is, the number of its nonzero components. By default, the sparsity level is set to $\min\{10000, \frac{m}{2}\}$ and we used this value. Consequently, the value of the degree of partial separability ω is indirectly influenced by these inputs.

We generated six random problem instances (matrix A and vector b). Among these, three instances were constructed with 20 nonzero elements per column, while the remaining three contained 1000 nonzero elements per column. Table 1 summarizes the set of generated problems, including their identifiers, the dimensions of matrix A, the corresponding values of ω, and the proportion of zero components in the optimal generated solution x^*, denoted by $(nz(x^*))$.

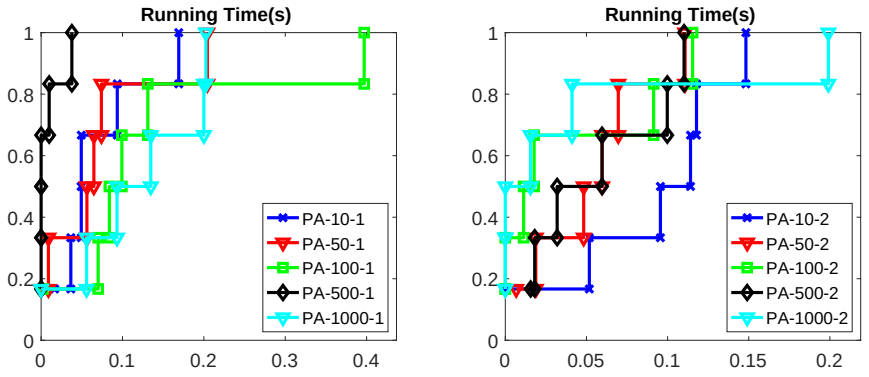
To define the target function value (F_{target}) for each of the six test problems, we employed our implementation of the uniform coordinate descent method, running for $1000n$ iterations with a block size equal to 1, obtaining a very precise approximation of the optimal value that we denote $\tilde{F}^*$. Finally, we constructed the target value so that the relative error between F_{target} and $\tilde{F}^*$ is 10^{-4}, that is, F_{target} is defined as

$$F_{target} = \tilde{F}^*(1 + 10^{-4}).$$

We begin analyzing the performance of Algorithm 2 to identify an effective

Name	#rows × #cols	ω	$nz(x^*)$
$Ne1$	100,000 × 200,000	258	95.0%
$Ne2$	200,000 × 100,000	85	90.0%
$Ne3$	2,000,000 × 1,000,000	90	99.0%
$Ne4$	100,000 × 200,000	71	95.0%
$Ne5$	200,000 × 100,000	27	90.0%
$Ne6$	2,000,000 × 1,000,000	30	99.0%

Table 1: Features of the artificially generated problems.



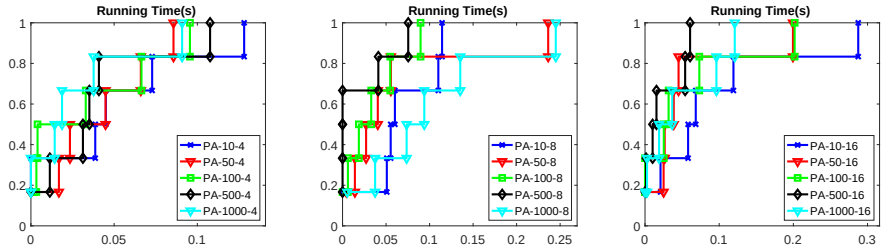
(a) 1 thread.

(b) 2 threads.

Figure 1: Performance profile of the average execution time between the variants of the Parallel Active BCDM method (`PA`) with 1 and 2 threads.

configuration. This process requires the appropriate selection of two key parameters: δ_{DP} and δ_F. The first governs the probability distribution employed by the method, that is, the relative frequency with which the method selects inactive coordinates compared to active ones. The second one determines the number of iterations between successive evaluations of the identification function. In this preliminary analysis, we fix $\delta_F = 1$ and evaluate the algorithm's performance with different values of $\delta_{DP} \in \{10, 50, 100, 500, 1000\}$. In the experimental results shown in the figures, each variant of the algorithm is labeled as `PA` followed by the corresponding value of δ_{DP} and the number of threads used. To put the different configurations in perspective, we have used $\log_2$ scaled performance profiles of the average execution time among the 20 independent runs of each problem instance solved by each variant of the algorithm. Such cumulative distribution plots depict the proportion of problems solved in the y-axis, within the factor of the fastest ($\log_2$ scaled) that is shown in the x-axis. We refer the reader to [7] for further details on this benchmarking tool.

Looking at Figures 1 and 2, we note that the performance of the methods is minimally affected by the change in the value of δ_{DP} for this set of problems. Due to the promising performance of the `PA-500` variant, this choice was adopted



(a) 4 threads. (b) 8 threads. (c) 16 threads.

Figure 2: Performance profile of the average execution time between the variants of the Parallel Active BCDM method (`PA`) with 4, 8, and 16 threads.

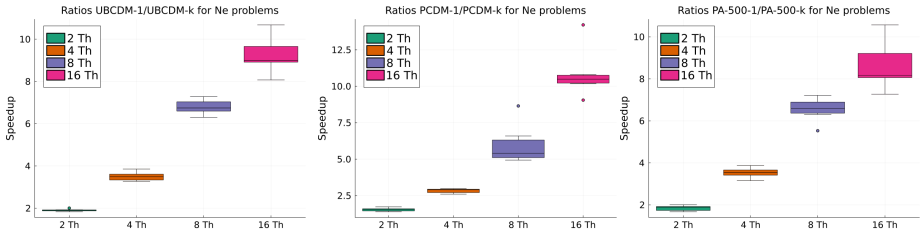


Figure 3: Boxplots comparing the speedup in running time between `UBCDM-1`/`UBCDM-`k (left), `PCDM-1`/`PCDM-`k (center) and `PA-500-1`/`PA-500-`k (right), with $k \in \{2, 4, 8, 16\}$ for Ne problems.

for the subsequent tests within this subsection.

Using the selected `PA` variant, we evaluate the multi-threaded speedup in execution time for the `UBCDM`, `PCDM`, and `PA-500` methods for the six randomly generated Ne test problems. To illustrate the performance of these methods, we present the results in the form of boxplots, as shown in Figure 3. Each boxplot represents the speedup ratio, defined as the average execution time of the serial version of a given method divided by the average execution time of its multi-threaded counterpart. This visualization provides a clear comparison of the parallel efficiency achieved by the different algorithmic implementations.

Figure 3 shows that all methods scale effectively in terms of speedup for the problems considered: as the number of threads increases, the observed speedup improves consistently. The `UBCDM` and `PA-500` methods exhibit similar speedup behavior for the tested configurations. In contrast, the `PCDM` method achieves a comparable average speedup when using 2, 4, and 8 threads but surpasses the other approaches in average speedup when executed with 16 threads.

We conclude this subsection by presenting two boxplots in Figure 4, which compare the execution time ratios between the `PCDM` and `UBCDM` and between the `UBCDM` and `PA-500` for different thread values. The results indicate that `UBCDM` achieves, on average, a two-times speedup relative to `PCDM`. In turn, for

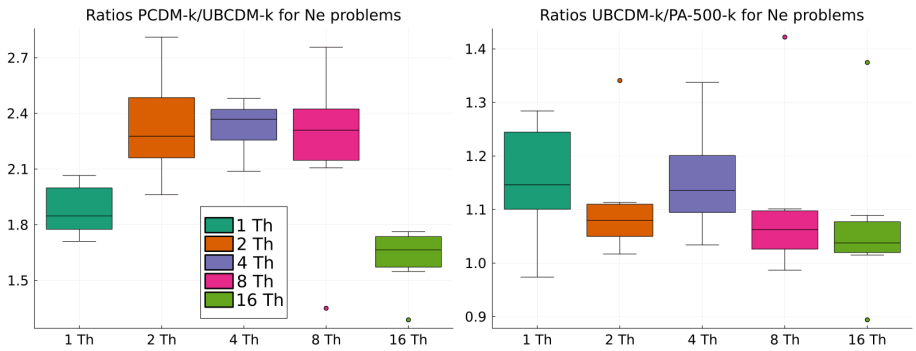


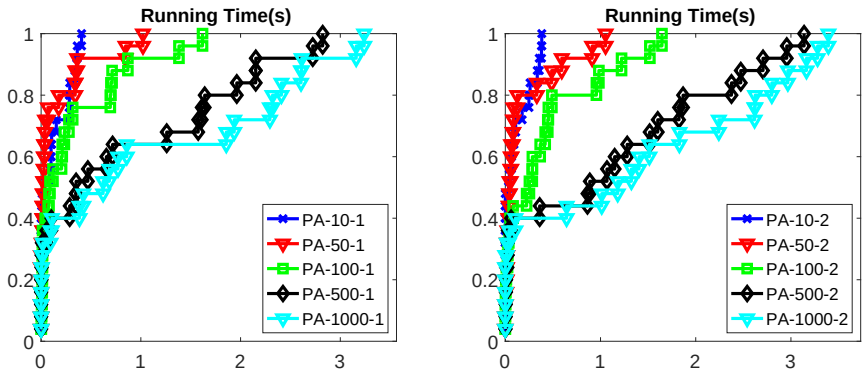
Figure 4: Boxplot comparing the running time ratio between `PCDM` and `UBCDM` (left) and between `UBCDM` and `PA-500` (right) for Ne problems, with 1, 2, 4, 8 and 16 threads.

this set of problems, `PA-500` demonstrates a modest performance advantage over `UBCDM`, with an average speedup slightly greater than one. Additionally, in Section 7.1 of the appendix, we present tables reporting the execution time and the average number of iterations for the most relevant methods discussed in this subsection, namely `UBCDM`, `PCDM`, and `PA-500`. The values displayed in such tables contrasting the pairs $(Ne1, Ne4)$, $(Ne2, Ne5)$, and $(Ne3, Ne6)$ corroborate that the smaller the degree of partial separability ω, the greater the expected acceleration for the coordinate descent methods, as suggested by the theoretical results.

It is important to caution the reader that, despite the favorable speedup presented in the plots of Figure 3, consistent with the principle discussed in [23], it should be interpreted as indicative of the best-case performance achievable by parallel coordinate descent methods. In real-world applications, problem instances often involve matrices A with less favorable structures, particularly with columns containing highly nonuniform quantities of nonzero elements. The structural uniformity present in the test cases used here may bias the results, obscuring several challenges that coordinate descent methods face when applied to more heterogeneous and irregular matrix structures. The experiments in the next subsection were performed to provide further insight into this matter.

5.2 A realistic scenario

In this subsection, we evaluate the performance of block coordinate descent methods in real-world problems. For this purpose, we use the set of 49 test instances introduced in [13, Tables 2 and 3]. These tables provide detailed information on each problem, including its name, source, matrix dimensions, target objective function value, and the number of zero coordinates in the optimal solution. The regularization parameter is chosen as $\lambda = 0.1\|A^T b\|_\infty$, where A and b denote the matrix and vector that define each problem, respectively.



(a) 1 thread. (b) 2 threads.

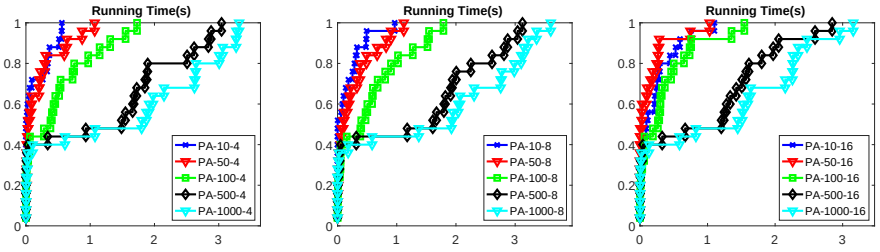
Figure 5: Performance profile of the average execution time between the variants of the Parallel Active BCDM method (`PA`) with 1 and 2 threads for the 25 SC problems.

Because these problems are derived from real applications and contain actual data, they present great structural diversity, making them particularly valuable for analysis.

A limitation of this test set is that most problems have a relatively small number of samples and/or variables compared to those analyzed in the previous subsection. Another distinguishing characteristic is the presence of two problem classes: those in which the matrix A has more columns than rows, called SC problems, and those in which there are more rows than columns, called SR problems. In the context of Lasso, where the primary objective is the selection of variables or characteristics, this distinction is particularly relevant. In the SC case, the problem includes many potentially redundant or insignificant features to be eliminated. In contrast, the SR case involves fewer variables from the outset, where a more refined selection is required. These differences were already important in [13], which reported that the behavior of the block coordinate descent method with identification differs notably between these two problem classes. Therefore, we present the results for each class separately to highlight the differences, which also impact the performance of the parallel implementations.

We begin by analyzing the SC-type problems. The parameter δ_{DP} is calibrated using the same range of values as considered in Subsection 7.1. The comparison is once more conducted using performance profiles now based on the average execution time among 100 independent runs of each problem instance for all `PA` method variants, on the set of 25 SC-type problems. These results are depicted in Figures 5 and 6. From the performance profiles, it is evident that the `PA-10` variant achieves the best overall performance among the tested configurations.

Figure 7 presents three boxplots, one for each method `PCDM`, `UBCDM` and



(a) 4 threads. (b) 8 threads. (c) 16 threads.

Figure 6: Performance profile of the average execution time between the variants of the Parallel Active BCDM method (`PA`) with 4, 8, and 16 threads for the 25 SC problems.

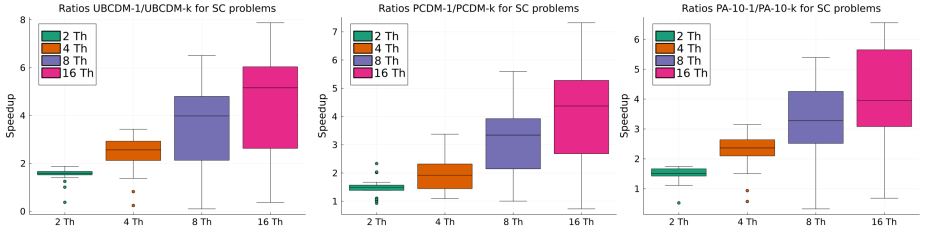


Figure 7: Boxplots comparing the speedup in running time between `UBCDM-1`/`UBCDM-`k (left), `PCDM-1`/`PCDM-`k (center) and `PA-10-1`/`PA-10-`k (right), with $k \in \{2, 4, 8, 16\}$ for SC problems.

`PA-10`, illustrating the execution time speedup achieved by each method relative to its corresponding single-threaded variant. Although the observed speedups are less pronounced than those reported for the Ne problems, the performance of all methods improves consistently with increasing number of threads. Notably, the `PCDM` method exhibits relatively modest gains when using 2 and 4 threads. With 8 and 16 threads, its speedup becomes more significant and surpasses the performance of the other two methods under these configurations.

We conclude the comparison of the SC problems with the boxplots shown in Figure 8, which depict the execution time performance for the evaluated methods. Specifically, the figure contrasts the performance of `PCDM` against `UBCDM`, as well as `UBCDM` against `PA-10`. Again, `UBCDM` method achieves an average speedup of approximately 2 relative to `PCDM`. In turn, `PA-10` attains an average speedup of nearly 3.4 when compared to `UBCDM`.

We now turn to the analysis of the experimental results for the problems of SR-type. The parameter δ_{DP} is adjusted using the same range of values employed previously. Performance profiles are used to compare the average execution times of all `PA` method variants for the 24 SR-type problems, as illustrated in Figures 9 and 10. Similarly to the SC case, the most promising configuration corresponds to $\delta_{DP} = 10$, which delivers the best overall performance.

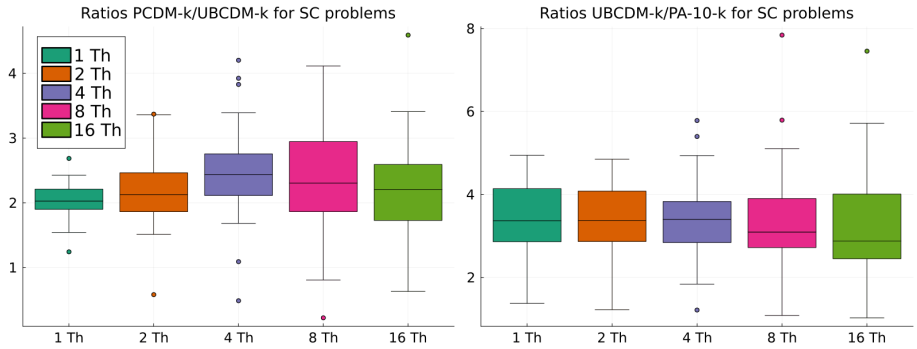


Figure 8: Boxplot comparing the running time ratio between `PCDM` and `UBCDM` (left) and between `UBCDM` and `PA-10` (right) for SC problems, with 1, 2, 4, 8 and 16 threads.

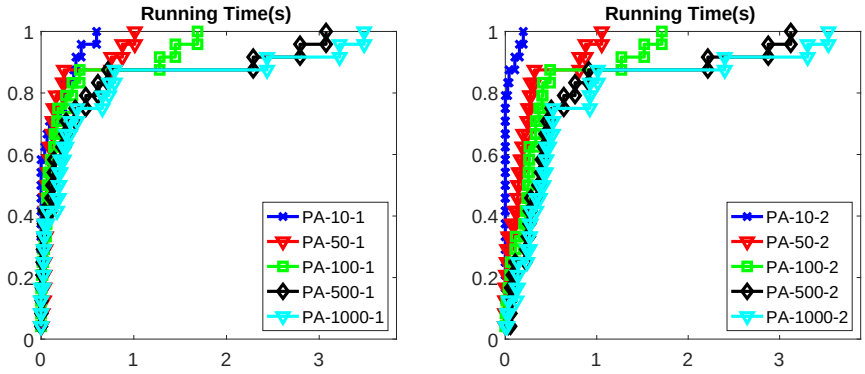
Figure 11 presents three boxplots showing the speedup of the multi-threaded variants when compared to its serial version. Unlike the previous cases, the multi-threaded variants did not outperform their serial counterparts in the majority of instances in this test set. This outcome is attributed to the higher $\frac{\omega}{n}$ ratio observed in this class of problems, which leads to increased values of β, thereby penalizing the magnitude of the descent directions when compared to the serial case. In several instances, the number of iterations required by the multi-threaded versions increased substantially compared to the serial versions, resulting in a significant reduction in the effective speedup. Please refer to the tables in the appendix for further details.

An analysis of the average speedups reveals that the `UBCDM` method achieved speedups of 1.12, 1.38, 1.61, and 1.61 for 2, 4, 8, and 16 threads, respectively. The `PCDM` method reached 1.19, 1.49, 2.00, and 2.50, while `PA-10` achieved 1.03, 1.25, 1.39, and 1.44 for the same thread counts.

As with the SC problems, Figure 11 presents boxplots comparing the execution time of the serial implementation of each method with its corresponding multi-threaded version. In contrast to previous cases, for the set of SR problems, the multi-threaded implementations did not outperform their serial counterparts in the majority of instances. Among the evaluated methods, `PCDM` demonstrated the most notable speedup under multi-threaded execution.

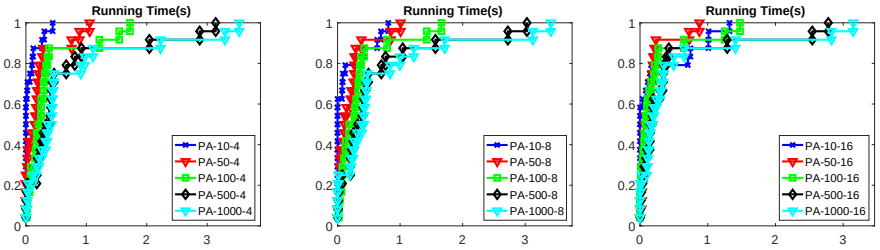
To conclude this section of experiments, we present the final two boxplots in Figure 12. It illustrates the execution time ratios between `UBCDM` and `PCDM`, as well as between `UBCDM` and `PA-10`, for the set of problems SR. The results indicate that `UBCDM` is, on average, approximately 1.5 times faster than `PCDM`, although this advantage slightly decreases as the number of threads increases to 16. In contrast, the performance gap between `PA-10` and `UBCDM` remains more stable, with `PA-10` consistently exhibiting nearly 2.4 times the demand of `UBCDM` for all configurations of threads.

We recall that Sections 7.2 and 7.3 of the appendix present tables with full



(a) 1 thread. (b) 2 threads.

Figure 9: Performance profile of the average execution time between the variants of the Parallel Active BCDM method (`PA`) with 1 and 2 threads for the 24 SR problems.



(a) 4 threads. (b) 8 threads. (c) 16 threads.

Figure 10: Performance profile of the average execution time between the variants of the Parallel Active BCDM method (`PA`) with 4, 8, and 16 threads for the 24 SR problems.

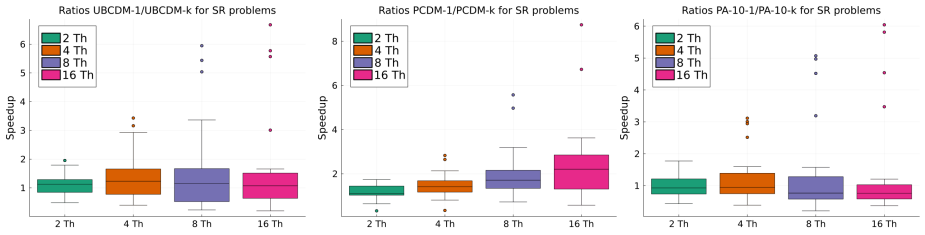


Figure 11: Boxplots comparing the speedup in running time between `UBCDM-1`/`UBCDM-`k (left), `PCDM-1`/`PCDM-`k (center) and `PA-10-1`/`PA-10-`k (right), with $k \in \{2, 4, 8, 16\}$ for SR problems.

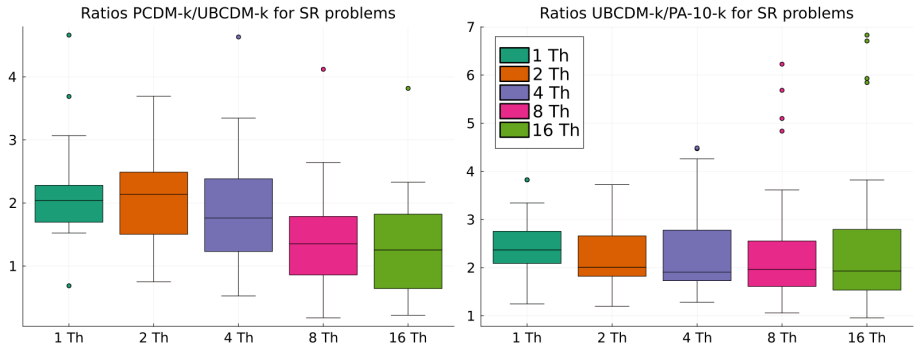


Figure 12: Boxplot comparing the running time ratio between `PCDM` and `UBCDM` (left) and between `UBCDM` and `PA-10` (right) for SR problems, with 1, 2, 4, 8 and 16 threads.

detailed information about the most relevant methods previously discussed for problems SC and SR.

6 Final remarks

In this paper, we introduce a parallel variation of the Active Block Coordinate Descent Method from [13] and show that its sequence of objective values converges to the optimal value in expectation, similar to the results of [21] for the uniform case without identification. The convergence analysis is accompanied by a high-performance implementation that is tested in different scenarios based on Lasso problems.

In the synthetic test set presented in [21], our implementation displays a consistent speedup as the number of threads increases. It can also outperform the implementation from [21] even in the uniform case, with further acceleration whenever identification is activated. In real-world tests, using the collection from [13], the parallel implementation is still faster than the serial one, but the improvement is more limited than in the synthetic case. For problems with sparse matrices that have a very unbalanced quantity of elements among the columns, the large amount of information may burden the computational effort unevenly among the threads. In case such columns take part in the problem solution, our identification strategy should benefit less from the parallelism than a uniform choice of blocks. Nevertheless, better speedups were achieved in problems where there are many columns (features) to be selected. In this scenario, identification has a favorable effect, decreasing the computational effort to approximate an optimal solution.

A future direction of research is to use ideas akin to relative smoothness [10] or the smooth approximation framework in [4] to relax the differentiability assumptions on the smooth part of the objective function in a setting that allows the use of identification of the active constraints.

Acknowledgments. We are thankful to the anonymous reviewers, whose insightful comments and questions helped us improve the presentation of our work.

References

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

[2] L. Cannelli, F. Facchinei, V. Kungurtsev, and G. Scutari. Asynchronous parallel algorithms for nonconvex optimization. *Mathematical Programming*, 184:121–154, 2020.

[3] Y.-G. Choi, S. Lee, and D. Yu. An efficient parallel block coordinate descent algorithm for large-scale precision matrix estimation using graphics processing units. *Computational Statistics*, 37:419–443, 2022.

[4] F. Chorobura and I. Necoara. Coordinate descent methods beyond smoothness and separability. *Computational Optimization and Applications*, 88(1):107–149, May 2024.

[5] J. E. Dennis, Jr. and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Society for Industrial and Applied Mathematics, Philadelphia, 1996.

[6] A. Devarakonda, K. Fountoulakis, J. Demmel, and M. W. Mahoney. Avoiding communication in primal and dual block coordinate descent methods. *SIAM Journal on Scientific Computing*, 41(1):C1–C27, 2019.

[7] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002.

[8] F. Facchinei, A. Fischer, and C. Kanzow. On the accurate identification of active constraints. *SIAM Journal on Optimization*, 9(1):14–32, 1998.

[9] F. Facchinei, A. Fischer, and C. Kanzow. On the accurate identification of active constraints. *SIAM Journal on Optimization*, 9(1):14–32, 1998.

[10] F. Hanzely and P. Richtárik. Fastest rates for stochastic mirror descent methods. *Computational Optimization and Applications*, 79(3):717–766, July 2021.

[11] J. Hickman. A note on the concept of multiset. *Bulletin of the Australian Mathematical Society*, 22(2):211–217, 1980.

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

[13] R. Lopes, S. A. Santos, and P. J. S. Silva. Accelerating block coordinate descent methods with identification strategies. *Computational Optimization and Applications*, 72(3):609–640, Apr 2019.

[14] I. Necoara and D. Clipici. Parallel random coordinate descent method for composite minimization: convergence analysis and error bounds. *SIAM Journal on Optimization*, 26(1):197–226, 2016.

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

[16] Y. Nesterov. Gradient methods for minimizing composite functions. *Mathematical programming*, 140(1):125–161, 2013.

[17] J. Nutini, I. Laradji, and M. Schmidt. Let's make block coordinate descent converge faster: Faster greedy rules, message-passing, active-set complexity, and superlinear convergence. *Journal of Machine Learning Research*, 23(131):1–74, 2022.

[18] A. Patrascu and I. Necoara. Efficient random coordinate descent algorithms for large-scale structured nonconvex optimization. *Journal of Global Optimization*, 61(1):19–46, 2015.

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

[20] P. Richtárik and M. Takáč. Distributed coordinate descent method for learning with big data. *Journal of Machine Learning Research*, 17(75):1–25, 2016.

[21] P. Richtárik and M. Takáč. Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, 156(1):433–484, 2016.

[22] R. Tappenden, P. Richtárik, and B. Buke. Separable approximations and decomposition methods for the augmented Lagrangian. *Optimization Methods and Software*, 30(3):643–668, 2015.

[23] R. Tappenden, M. Takáč, and P. Richtárik. On the complexity of parallel coordinate descent. *Optimization Methods and Software*, 33(2):372–395, 2018.

[24] R. Tibshirani. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58(1):267–288, 1996.

[25] C. Traoré, S. Salzo, and S. Villa. Convergence of an asynchronous block-coordinate forward-backward algorithm for convex composite optimization. *Computational Optimization and Applications*, 86:303–344, 2023.

[26] P. Tseng. Approximation accuracy, gradient methods, and error bound for structured convex optimization. *Mathematical Programming*, 125:263–295, 2010.

[27] P. Tseng and S. Yun. Block-coordinate gradient descent method for linearly constrained nonsmooth separable optimization. *Journal of Optimization Theory and Applications*, 140(3):513, Sep 2008.

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

7 Appendix

7.1 Tables - Datasets Ne

Problem \ Threads	$1T$	$2T$	$4T$	$8T$	$16T$
$Ne1$	1300.837	679.493	395.705	191.876	146.266
$Ne2$	4.778	2.511	1.364	0.729	0.592
$Ne3$	83.552	44.800	22.892	11.733	9.302
$Ne4$	292.040	159.256	89.240	46.335	29.576
$Ne5$	1.366	0.710	0.391	0.203	0.152
$Ne6$	29.755	14.832	7.721	4.086	2.787

Table 2: Running time of the method `UBCDM` with multiple threads applied to Ne problems.

Problem \ Ratio	$1T/2T$	$1T/4T$	$1T/8T$	$1T/16T$
$Ne1$	1.91442	3.28739	6.77956	8.89365
$Ne2$	1.90323	3.50227	6.55844	8.07393
$Ne3$	1.86500	3.64977	7.12090	8.98226
$Ne4$	1.83377	3.27252	6.30280	9.87430
$Ne5$	1.92308	3.49023	6.71706	9.00923
$Ne6$	2.00614	3.85353	7.28232	10.6769

Table 3: Running time ratio between `UBCDM-1`/`UBCDM-k` with `k` $\in \{2, 4, 8, 16\}$, applied to Ne problems.

Problem \ Threads	$1T$	$2T$	$4T$	$8T$	$16T$
$Ne1$	1.20×10^9	1.58×10^9	1.29×10^9	1.32×10^9	1.56×10^9
$Ne2$	6.71×10^6	6.70×10^6	6.74×10^6	6.65×10^6	6.83×10^6
$Ne3$	8.80×10^7	9.04×10^7	8.91×10^7	8.67×10^7	8.71×10^7
$Ne4$	1.40×10^9	1.41×10^9	1.51×10^9	1.52×10^9	1.46×10^9
$Ne5$	8.02×10^6	7.68×10^6	8.04×10^6	7.84×10^6	7.90×10^6
$Ne6$	8.25×10^7	8.41×10^7	8.51×10^7	8.34×10^7	8.44×10^7

Table 4: Average number of iterations of the method `UBCDM` with multiple threads applied to Ne problems.

Problem \ Threads	$1T$	$2T$	$4T$	$8T$	$16T$
$Ne1$	2662.8	1910.3	917.53	404.20	246.50
$Ne2$	8.8046	5.6280	3.3843	1.7840	0.9729
$Ne3$	146.49	103.62	55.365	27.569	14.400
$Ne4$	541.09	312.39	186.33	62.555	38.082
$Ne5$	2.8220	1.8062	0.9483	0.56064	0.26588
$Ne6$	50.905	31.652	17.265	9.2756	4.9133

Table 5: Running time of the method `PCDM` with multiple threads applied to Ne problems.

Problem \ Ratio	$1T/2T$	$1T/4T$	$1T/8T$	$1T/16T$
$Ne1$	1.39392	2.90214	6.58783	10.8024
$Ne2$	1.56443	2.60160	4.93531	9.05013
$Ne3$	1.41372	2.64590	5.31358	10.1729
$Ne4$	1.73210	2.90393	8.64983	14.2085
$Ne5$	1.56240	2.97595	5.03353	10.6138
$Ne6$	1.60827	2.94845	5.48805	10.3607

Table 6: Running time ratio between `PCDM-1`/`PCDM-k` with `k` $\in \{2, 4, 8, 16\}$, applied to Ne problems.

Problem \ Threads	$1T$	$2T$	$4T$	$8T$	$16T$
$Ne1$	2.0×10^9	2.2×10^9	1.9×10^9	1.5×10^9	1.8×10^9
$Ne2$	6.8×10^6	6.4×10^6	6.8×10^6	6.6×10^6	6.7×10^6
$Ne3$	9.0×10^7	8.7×10^7	9.1×10^7	8.7×10^7	8.3×10^7
$Ne4$	1.6×10^9	1.5×10^9	1.6×10^9	1.0×10^9	1.2×10^9
$Ne5$	8.7×10^6	8.7×10^6	8.1×10^6	8.7×10^6	7.2×10^6
$Ne6$	8.0×10^7	8.0×10^7	8.5×10^7	8.8×10^7	8.5×10^7

Table 7: Average number of iterations of the method `PCDM` with multiple threads applied to Ne problems.

Problem \ Threads	$1T$	$2T$	$4T$	$8T$	$16T$
$Ne1$	1021.017	610.287	323.761	184.617	140.427
$Ne2$	4.390	2.283	1.203	0.661	0.543
$Ne3$	85.811	42.805	22.142	11.891	10.404
$Ne4$	227.433	118.760	66.722	32.582	21.515
$Ne5$	1.182	0.699	0.344	0.187	0.147
$Ne6$	26.174	13.999	7.139	3.999	2.745

Table 8: Running time of the method `PA-500` with multiple threads applied to Ne problems.

Problem \ Ratio	$1T/2T$	$1T/4T$	$1T/8T$	$1T/16T$
$Ne1$	1.67301	3.15362	5.53045	7.27081
$Ne2$	1.92254	3.64808	6.63575	8.07794
$Ne3$	2.00472	3.87544	7.21640	8.24739
$Ne4$	1.91507	3.40863	6.98027	10.5709
$Ne5$	1.69169	3.43276	6.30851	8.05658
$Ne6$	1.86967	3.66637	6.54548	9.53374

Table 9: Running time ratio between `PA-500-1`/`PA-500-k` with `k` $\in \{2, 4, 8, 16\}$, applied to Ne problems.

Problem \ Threads	$1T$	$2T$	$4T$	$8T$	$16T$
$Ne1$	1.13×10^9	1.26×10^9	1.23×10^9	1.25×10^9	1.21×10^9
$Ne2$	5.41×10^6	5.33×10^6	5.23×10^6	5.26×10^6	5.35×10^6
$Ne3$	7.52×10^7	7.56×10^7	7.59×10^7	7.37×10^7	7.55×10^7
$Ne4$	1.06×10^9	1.02×10^9	1.10×10^9	1.10×10^{10}	9.39×10^8
$Ne5$	6.14×10^6	6.63×10^6	6.19×10^6	6.20×10^6	6.40×10^6
$Ne6$	6.81×10^7	6.95×10^7	7.11×10^7	7.15×10^7	6.80×10^7

Table 10: Average number of iterations of the method `PA-500` with multiple threads applied to Ne problems.

7.2 Tables - Datasets SC

Problem \ Threads	$1T$	$2T$	$4T$	$8T$	$16T$
SC1	0.145980	0.085250	0.045240	0.024530	0.018540
SC2	0.224420	0.129270	0.071410	0.039430	0.031690
SC3	0.282010	0.169530	0.096300	0.056070	0.041420
SC4	6.302630	3.607520	2.037880	1.313480	1.036560
SC5	6.317380	3.606250	2.112590	1.308840	1.046580
SC6	0.091400	0.244840	0.380470	0.904970	0.245390
SC7	0.118780	0.074830	0.048980	0.029830	0.023020
SC8	0.004292	0.004268	0.005229	0.007463	0.007828
SC9	0.004907	0.003343	0.002717	0.002770	0.002330
SC10	0.005533	0.003941	0.003191	0.003359	0.002676
SC11	0.100320	0.069290	0.051630	0.047080	0.038160
SC12	0.081420	0.053590	0.038330	0.039960	0.037020
SC13	2.188110	1.261490	0.732300	0.386250	0.326500
SC14	0.206540	0.124170	0.078240	0.049500	0.046150
SC15	0.046290	0.029180	0.017370	0.013160	0.010590
SC16	0.121760	0.075150	0.043570	0.025970	0.022590
SC17	0.059170	0.035510	0.020630	0.013530	0.011060
SC18	0.237920	0.152000	0.095620	0.071490	0.063080
SC19	0.087520	0.053470	0.037080	0.029250	0.024060
SC20	0.456530	0.243350	0.133080	0.070180	0.068070
SC21	0.076410	0.049050	0.033190	0.024900	0.019000
SC22	0.072410	0.047130	0.028210	0.017120	0.012390
SC23	0.092990	0.061140	0.038710	0.025610	0.018000
SC24	0.074610	0.048170	0.028420	0.017070	0.012660
SC25	0.220670	0.176830	0.160500	0.151380	0.141470

Table 11: Running time of the method `UBCDM` with multiple threads applied to SC* problems.

Problem \ Ratio	$1T/2T$	$1T/4T$	$1T/8T$	$1T/16T$
SC1	1.71238	3.22679	5.95108	7.87379
SC2	1.73606	3.14270	5.69161	7.08173
SC3	1.66348	2.92845	5.02961	6.80855
SC4	1.74708	3.09274	4.79842	6.08033
SC5	1.75179	2.99035	4.82670	6.03621
SC6	0.37330	0.24022	0.10099	0.37246
SC7	1.58733	2.42507	3.98190	5.15986
SC8	1.00562	0.82080	0.57510	0.54828
SC9	1.46784	1.80604	1.77148	2.10601
SC10	1.40396	1.73394	1.64722	2.06764
SC11	1.44783	1.94306	2.13084	2.62893
SC12	1.51931	2.12418	2.03754	2.19935
SC13	1.73454	2.98800	5.66501	6.70172
SC14	1.66336	2.63983	4.17253	4.47541
SC15	1.58636	2.66494	3.51748	4.37110
SC16	1.62023	2.79458	4.68849	5.39000
SC17	1.66629	2.86815	4.37324	5.34991
SC18	1.56526	2.48818	3.32802	3.77172
SC19	1.63681	2.36030	2.99214	3.63757
SC20	1.87602	3.43049	6.50513	6.70677
SC21	1.55780	2.30220	3.06867	4.02158
SC22	1.53639	2.56682	4.22956	5.84423
SC23	1.52094	2.40222	3.63100	5.16611
SC24	1.54889	2.62526	4.37083	5.89336
SC25	1.24792	1.37489	1.45772	1.55984

Table 12: Running time ratio between `UBCDM-1`/`UBCDM-k` with `k` $\in \{2, 4, 8, 16\}$, applied to SC* problems.

Problem \ Threads	$1T$	$2T$	$4T$	$8T$	$16T$
SC1	956825.60	950272.00	959447.04	956170.24	956825.60
SC2	870318.08	871628.80	878182.40	877527.04	882769.92
SC3	1246494.72	1247805.44	1255014.40	1256325.12	1275985.92
SC4	1032765.44	1079214.08	1097564.16	1187594.24	1390592.00
SC5	1032765.44	1079214.08	1111900.16	1186447.36	1401487.36
SC6	23470.08	31539.20	52838.40	96583.68	183746.56
SC7	657031.26	622336.54	674378.62	652694.42	635347.06
SC8	101970.84	112387.80	128864.19	165763.05	245936.64
SC9	97533.42	96913.34	99625.63	109179.60	128035.71
SC10	114514.46	116623.72	119904.16	136005.889	151275.06
SC11	997507.08	1042447.77	1146428.19	1251289.80	1537676.55
SC12	1078810.48	1143712.18	1176884.16	1414857.06	1890081.73
SC13	16655297.39	17563275.36	17739450.19	18322182.32	19189504.56
SC14	1171919.32	1150798.22	1266059.08	1358388.46	1681843.02
SC15	525097.76	537227.44	540357.68	633091.04	705869.12
SC16	471742.40	466846.96	476637.84	484648.56	552739.68
SC17	273367.64	260104.52	262683.46	290683.38	309841.22
SC18	1352837.72	1375908.61	1445612.15	1699882.81	2212841.96
SC19	724651.98	703465.62	785857.02	902774.34	1103260.08
SC20	5889029.12	5937018.88	6094699.52	5827328.00	5895884.80
SC21	589927.75	574562.19	623951.49	671694.48	790228.80
SC22	301811.20	297840.00	303548.60	304789.60	306775.20
SC23	438073.00	437824.80	441051.40	449490.20	466119.60
SC24	303300.40	299577.40	300322.00	301314.80	308264.40
SC25	1224809.85	1340415.35	1589271.40	2060820.15	3036165.50

Table 13: Average number of iterations of the method `UBCDM` with multiple threads applied to SC* problems.

Problem \ Threads	$1T$	$2T$	$4T$	$8T$	$16T$
SC1	0.295980	0.191120	0.123840	0.072854	0.048063
SC2	0.447870	0.290370	0.193110	0.111220	0.068896
SC3	0.509380	0.324010	0.217410	0.129210	0.079972
SC4	12.00500	5.951900	3.597600	2.472100	1.790400
SC5	11.98600	5.906300	3.552500	2.439900	1.764700
SC6	0.202090	0.142180	0.185070	0.201200	0.276520
SC7	0.254240	0.252200	0.205840	0.122730	0.105690
SC8	0.010252	0.011031	0.009100	0.009387	0.008898
SC9	0.011744	0.011236	0.009212	0.006492	0.007697
SC10	0.012542	0.012455	0.008793	0.007369	0.006571
SC11	0.154650	0.139830	0.115110	0.064811	0.053388
SC12	0.101140	0.094578	0.070004	0.048144	0.037675
SC13	5.173100	3.091400	2.803900	1.348100	1.022700
SC14	0.455560	0.305870	0.307030	0.164090	0.118520
SC15	0.112290	0.074498	0.058524	0.040661	0.033170
SC16	0.326960	0.140030	0.105670	0.058428	0.044678
SC17	0.112390	0.076469	0.059944	0.033605	0.025700
SC18	0.517640	0.323400	0.229500	0.153540	0.117300
SC19	0.173340	0.085238	0.096556	0.048751	0.054268
SC20	0.997850	0.700750	0.361880	0.254350	0.189010
SC21	0.157330	0.108840	0.088074	0.073356	0.064766
SC22	0.143000	0.094124	0.067822	0.042223	0.029270
SC23	0.167230	0.114150	0.081809	0.050927	0.032954
SC24	0.146940	0.096429	0.069298	0.042135	0.027952
SC25	0.370390	0.267260	0.175030	0.122200	0.089110

Table 14: Running time of the method `PCDM` with multiple threads applied to SC* problems.

Problem \ Ratio	$1T/2T$	$1T/4T$	$1T/8T$	$1T/16T$
SC1	1.54866	2.39002	4.06265	6.15817
SC2	1.54241	2.31925	4.02688	6.50067
SC3	1.57211	2.34295	3.94226	6.36948
SC4	2.01700	3.33695	4.85620	6.70521
SC5	2.02936	3.37396	4.91250	6.79209
SC6	1.42137	1.09197	1.00442	0.73083
SC7	1.00809	1.23513	2.07154	2.40553
SC8	0.92938	1.12659	1.09215	1.15217
SC9	1.04521	1.27486	1.80900	1.52579
SC10	1.00699	1.42636	1.70199	1.90869
SC11	1.10599	1.34350	2.38617	2.89672
SC12	1.06938	1.44477	2.10078	2.68454
SC13	1.67338	1.84497	3.83733	5.05828
SC14	1.48939	1.48376	2.77628	3.84374
SC15	1.50729	1.91870	2.76161	3.38529
SC16	2.33493	3.09416	5.59595	7.31814
SC17	1.46975	1.87492	3.34444	4.37315
SC18	1.60062	2.25551	3.37137	4.41296
SC19	2.03360	1.79523	3.55562	3.19415
SC20	1.42397	2.75741	3.92314	5.27935
SC21	1.44552	1.78634	2.14475	2.42921
SC22	1.51927	2.10846	3.38678	4.88555
SC23	1.46500	2.04415	3.28372	5.07465
SC24	1.52382	2.12041	3.48736	5.25687
SC25	1.38588	2.11615	3.03101	4.15655

Table 15: Running time ratio between `PCDM-1`/`PCDM-k` with `k` $\in \{2, 4, 8, 16\}$, applied to NE* problems.

Problem \ Threads	$1T$	$2T$	$4T$	$8T$	$16T$
SC1	917504.0	917504.0	917504.0	851968.0	851968.0
SC2	851968.0	851968.0	851968.0	786432.0	786432.0
SC3	1179648.0	1179648.0	1179648.0	1179648.0	1179648.0
SC4	1089536.0	860160.0	916930.56	1261568.0	1662976.0
SC5	1089536.0	860160.0	916930.56	1261568.0	1664122.88
SC6	20480.0	20480.0	45056.0	90112.0	184320.0
SC7	433684.0	650526.0	650526.0	216842.0	216842.0
SC8	103444.0	141765.3	133254.68	154084.54	153379.24
SC9	85261.0	113009.58	115567.41	75264.5353	155020.0
SC10	102011.0	128847.74	102481.82	122099.32	72898.63
SC11	793071.0	1058309.19	1317379.05	969309.0	1061833.95
SC12	721130.0	928815.44	1009582.0	935305.61	871125.04
SC13	16262292.0	14907101.0	24393438.0	16641745.48	18037592.21
SC14	1448304.0	1267266.0	1563564.86	1206920.0	1206920.0
SC15	665176.0	547792.0	586920.0	504751.2	627221.84
SC16	667560.0	400536.0	480198.16	355141.92	445040.0
SC17	221052.0	218104.64	257894.0	184210.0	201525.74
SC18	1718045.0	1473591.74	1201649.76	1183978.44	1161398.42
SC19	784680.0	470808.0	763101.3	417449.76	669724.38
SC20	5484544.0	4798976.0	3427840.0	3427840.0	2742272.0
SC21	603647.0	384687.77	493893.0	560294.17	579501.12
SC22	273020.0	273020.0	273020.0	273020.0	297840.0
SC23	397120.0	421940.0	420202.6	421940.0	421940.0
SC24	273020.0	273020.0	273020.0	270538.0	273020.0
SC25	1156055.0	1216900.0	1156055.0	1216900.0	1216900.0

Table 16: Average number of iterations of the method `PCDM` with multiple threads applied to SC* problems.

Problem \ Threads	$1T$	$2T$	$4T$	$8T$	$16T$
SC1	0.047730	0.027920	0.015170	0.008850	0.007564
SC2	0.076200	0.044280	0.024760	0.014420	0.013210
SC3	0.098590	0.059140	0.033900	0.019760	0.015040
SC4	1.383670	0.827150	0.540110	0.419150	0.429240
SC5	1.385970	0.827140	0.525250	0.423070	0.449980
SC6	0.037660	0.071650	0.065810	0.115420	0.032920
SC7	0.086440	0.061200	0.040360	0.027550	0.022440
SC8	0.000990	0.000880	0.001059	0.001462	0.001455
SC9	0.001271	0.000944	0.000799	0.000802	0.000684
SC10	0.001406	0.001065	0.000926	0.000975	0.000838
SC11	0.030770	0.020530	0.014080	0.011920	0.008833
SC12	0.019670	0.013050	0.009118	0.008638	0.007368
SC13	0.649910	0.384290	0.223760	0.134390	0.113470
SC14	0.044680	0.027690	0.018040	0.012070	0.010620
SC15	0.011370	0.007723	0.004850	0.003374	0.002860
SC16	0.049930	0.032550	0.018790	0.011710	0.009229
SC17	0.028680	0.018390	0.011220	0.006870	0.005567
SC18	0.054500	0.037260	0.024970	0.019670	0.015740
SC19	0.022050	0.014950	0.010710	0.008757	0.006732
SC20	0.130130	0.074630	0.042020	0.025900	0.021660
SC21	0.026680	0.018730	0.012710	0.009806	0.006764
SC22	0.026560	0.017380	0.010240	0.006296	0.004700
SC23	0.029080	0.019460	0.012280	0.009278	0.006623
SC24	0.027110	0.017750	0.010390	0.006369	0.004767
SC25	0.044630	0.040130	0.029740	0.026140	0.024750

Table 17: Running time of the method `PA-10` with multiple threads applied to SC* problems.

Problem \ Ratio	$2T/1T$	$4T/1T$	$8T/1T$	$16T/1T$
SC1	1.70953	3.14634	5.39322	6.31015
SC2	1.72087	3.07754	5.28433	5.76836
SC3	1.66706	2.90826	4.98937	6.55519
SC4	1.67282	2.56183	3.30113	3.22353
SC5	1.67562	2.63869	3.27598	3.08007
SC6	0.52561	0.57225	0.32628	1.14399
SC7	1.41242	2.14172	3.13757	3.85205
SC8	1.12500	0.93484	0.67715	0.68041
SC9	1.34640	1.59074	1.58479	1.85819
SC10	1.32019	1.51836	1.44205	1.6778
SC11	1.49878	2.18537	2.58138	3.48353
SC12	1.50728	2.15727	2.27715	2.66965
SC13	1.69120	2.90450	4.83600	5.72759
SC14	1.61358	2.47672	3.70174	4.20716
SC15	1.47223	2.34433	3.36989	3.97552
SC16	1.53395	2.65726	4.26388	5.41012
SC17	1.55954	2.55615	4.17467	5.15179
SC18	1.46269	2.18262	2.77072	3.46252
SC19	1.47492	2.05882	2.51799	3.27540
SC20	1.74367	3.09686	5.02432	6.00785
SC21	1.42445	2.09913	2.72078	3.94441
SC22	1.52819	2.59375	4.21855	5.65106
SC23	1.49435	2.36808	3.13430	4.39076
SC24	1.52732	2.60924	4.25656	5.68701
SC25	1.11214	1.50067	1.70735	1.80323

Table 18: Running time ratio between `PA-10-1`/`PA-10-k` with `k` $\in \{2, 4, 8, 16\}$, applied to SC* problems.

Problem \ Threads	$1T$	$2T$	$4T$	$8T$	$16T$
SC1	275251.20	273285.12	275251.20	284182.66	308955.43
SC2	265420.80	265420.80	270663.68	275251.20	312606.72
SC3	393216.00	393216.00	393216.00	393216.00	401080.32
SC4	202424.32	217333.76	248872.96	306216.96	415744.00
SC5	202424.32	220200.96	246005.76	309084.16	428359.68
SC6	8560.64	8765.44	11919.36	16506.88	25600.00
SC7	420673.48	427178.74	427178.74	425010.32	429347.16
SC8	18240.04	19085.32	21358.24	25840.38	35251.33
SC9	21224.15	21511.04	22349.13	23482.98	26844.67
SC10	23165.76	23649.64	24943.29	26929.13	31789.23
SC11	261713.43	263475.81	263475.81	270525.33	289865.13
SC12	220665.78	227155.95	230106.03	255503.82	305419.77
SC13	4146884.46	4065573.00	4052021.09	4038469.18	4160436.37
SC14	207590.24	200348.72	213021.38	225090.58	252246.28
SC15	105254.32	108655.45	110201.08	113036.44	124091.66
SC16	126836.40	126836.40	128171.52	131731.84	132695.41
SC17	85105.02	82894.50	85841.86	86048.78	89444.19
SC18	219909.76	220891.50	234144.99	260651.97	300412.44
SC19	133787.94	131826.24	142811.76	153524.35	173524.87
SC20	1446548.48	1467115.52	1460259.84	1487682.56	1508249.60
SC21	159692.07	156948.22	159692.07	163032.64	166484.95
SC22	99280.00	99280.00	99280.00	99436.10	99862.63
SC23	124100.00	124348.20	127326.60	148920.00	152043.05
SC24	99280.00	99280.00	99280.00	99280.00	99398.19
SC25	197137.80	242771.55	243380.00	304225.00	425915.00

Table 19: Average number of iterations of the method `PA-10` with multiple threads applied to SC* problems.

7.3 Tables - Datasets SR

Problem \ Threads	$1T$	$2T$	$4T$	$8T$	$16T$
SR1	0.060170	0.04674	0.03554	0.03524	0.03864
SR2	0.040430	0.03136	0.02424	0.02313	0.02439
SR3	0.072170	0.05771	0.04363	0.04360	0.0481
SR4	0.500270	0.49184	0.43086	0.42877	0.36076
SR5	0.006308	0.01027	0.01578	0.02741	0.00986
SR6	2.301980	2.36673	2.11020	2.01118	1.95690
SR7	0.076220	0.06340	0.05882	0.07350	0.06652
SR8	0.029830	0.03621	0.03600	0.05208	0.04367
SR9	0.098760	0.11589	0.11614	0.14765	0.12289
SR10	0.001636	0.00152	0.00139	0.00183	0.00194
SR11	0.027170	0.02030	0.01714	0.01695	0.02338
SR12	0.022830	0.01802	0.01500	0.01478	0.02053
SR13	0.023390	0.01873	0.01603	0.01620	0.02256
SR14	0.008403	0.00708	0.00609	0.00642	0.00840
SR15	0.002227	0.00222	0.00220	0.00299	0.00356
SR16	0.178370	0.11570	0.07262	0.05304	0.05929
SR17	4.230360	2.36061	1.33827	0.77799	0.73230
SR18	2.845190	1.64976	0.97123	0.56467	0.51093
SR19	0.067020	0.03432	0.01954	0.01127	0.01004
SR20	10.953400	22.7002	26.4181	30.1164	29.5181
SR21	0.001004	0.00156	0.00253	0.00423	0.00501
SR22	0.004052	0.00543	0.00865	0.01378	0.01590
SR23	0.021560	0.02746	0.04231	0.07728	0.07712
SR24	0.043040	0.04570	0.06990	0.11690	0.12221

Table 20: Running time of the method `UBCDM` with multiple threads applied to SR* problems.

Problem \ Ratio	$1T/2T$	$1T/4T$	$1T/8T$	$1T/16T$
SR1	1.28733	1.69302	1.70743	1.55719
SR2	1.28922	1.66790	1.74795	1.65765
SR3	1.25056	1.65414	1.65528	1.50042
SR4	1.01714	1.16110	1.16676	1.38671
SR5	0.61421	0.39974	0.23013	0.63956
SR6	0.97264	1.09088	1.14459	1.17634
SR7	1.20221	1.29582	1.03701	1.14582
SR8	0.82380	0.82861	0.57277	0.68307
SR9	0.85218	0.85035	0.66887	0.80364
SR10	1.07349	1.17276	0.89301	0.84069
SR11	1.33842	1.58518	1.60295	1.16210
SR12	1.26693	1.52200	1.54465	1.11203
SR13	1.24880	1.45914	1.44383	1.03679
SR14	1.18603	1.37822	1.30827	1.00000
SR15	0.99331	1.01227	0.74456	0.62450
SR16	1.54166	2.45621	3.36293	3.00843
SR17	1.79206	3.16107	5.43755	5.77366
SR18	1.72461	2.92947	5.03868	5.56865
SR19	1.95280	3.42989	5.94676	6.67530
SR20	0.48252	0.41461	0.36370	0.37107
SR21	0.64071	0.39543	0.23696	0.20019
SR22	0.74553	0.46816	0.29040	0.25484
SR23	0.78514	0.50957	0.27898	0.27956
SR24	0.94179	0.61573	0.36817	0.35218

Table 21: Running time ratio between `UBCDM-1`/`UBCDM-k` with `k` $\in \{2, 4, 8, 16\}$, applied to SR* problems.

Problem \ Threads	$1T$	$2T$	$4T$	$8T$	$16T$
SR1	3236.13	3563.31	4189.38	5496.87	8215.17
SR2	2264.43	2483.37	2959.38	3724.44	5259.48
SR3	4409.55	4933.53	5747.79	7575.57	11319.69
SR4	4089.96	4959.36	7284.06	10944.36	14779.80
SR5	2531.32	3043.80	4071.60	6001.20	9382.94
SR6	28794.72	38173.08	61296.33	96988.59	158033.97
SR7	9765.28	11109.28	14675.36	21209.44	35086.24
SR8	1168.24	1636.08	2424.20	4111.28	7293.68
SR9	3086.52	4118.92	6917.08	12427.96	23385.64
SR10	2244.42	2549.59	3266.32	4848.45	8007.34
SR11	2805.00	3252.00	4728.00	7566.00	13137.00
SR12	2610.00	3150.00	4458.00	7071.00	12126.00
SR13	3288.00	4002.00	5775.00	9282.00	16107.00
SR14	2621.85	3175.35	4445.46	6888.46	11984.87
SR15	3928.29	4767.91	6506.37	10046.40	17129.19
SR16	151316.76	157394.58	176256.78	229490.10	324220.26
SR17	488461.05	492307.20	508973.85	532050.75	614956.65
SR18	477756.00	491832.00	522468.00	549378.00	602784.00
SR19	820699.87	779612.20	816485.75	822806.93	819646.34
SR20	8634.66	16188.78	31116.24	61484.52	121368.24
SR21	6413.594	8491.262	12703.60	20979.521	39082.485
SR22	22566.968	28822.087	42961.655	69578.56	127390.40
SR23	63907.20	85311.04	131567.68	231283.20	406672.96
SR24	140052.00	165250.75	247478.25	392039.50	659146.25

Table 22: Average number of iterations of the method `UBCDM` with multiple threads applied to SR* problems.

Problem \ Threads	$1T$	$2T$	$4T$	$8T$	$16T$
SR1	0.12204	0.12296	0.11422	0.084881	0.07406
SR2	0.09174	0.08349	0.08109	0.05328	0.04713
SR3	0.15507	0.14088	0.13083	0.11519	0.10124
SR4	0.34371	1.0583	1.0015	0.42095	0.46745
SR5	0.01458	0.01119	0.01128	0.01088	0.01231
SR6	4.2099	2.5719	2.7691	3.1289	2.4748
SR7	0.18007	0.15135	0.12963	0.11744	0.15223
SR8	0.07767	0.06294	0.05484	0.06459	0.10173
SR9	0.17433	0.10103	0.08768	0.08335	0.05565
SR10	0.00762	0.00437	0.00357	0.00322	0.00289
SR11	0.04584	0.04309	0.03011	0.02309	0.01515
SR12	0.03725	0.03725	0.02650	0.01827	0.01363
SR13	0.03970	0.03610	0.02612	0.02095	0.01419
SR14	0.01734	0.01678	0.01231	0.00997	0.00698
SR15	0.00683	0.00597	0.00457	0.00401	0.00362
SR16	0.38503	0.24950	0.19520	0.13857	0.10617
SR17	6.4437	4.1870	2.2757	1.2959	0.95778
SR18	5.8721	3.5825	2.2175	1.0538	0.67153
SR19	0.12541	0.11877	0.09052	0.04643	0.03832
SR20	17.756	17.026	21.898	24.447	30.689
SR21	0.00370	0.00578	0.00435	0.00370	0.00305
SR22	0.00771	0.00773	0.00853	0.00693	0.00573
SR23	0.04416	0.03126	0.02228	0.01383	0.01703
SR24	0.07214	0.06993	0.04541	0.04876	0.02918

Table 23: Running time of the method `PCDM` with multiple threads applied to SR* problems.

Problem \ Ratio	$1T/2T$	$1T/4T$	$1T/8T$	$1T/16T$
SR1	0.99251	1.06846	1.43778	1.64785
SR2	1.09871	1.13125	1.72159	1.94616
SR3	1.10072	1.18528	1.34621	1.53171
SR4	0.32477	0.34319	0.81651	0.73528
SR5	1.30352	1.29266	1.33979	1.18475
SR6	1.63688	1.52031	1.34549	1.70111
SR7	1.18976	1.38911	1.53329	1.18288
SR8	1.23411	1.41631	1.20247	0.76355
SR9	1.72553	1.98825	2.09144	3.13216
SR10	1.74212	2.13343	2.36569	2.63819
SR11	1.06387	1.52238	1.98493	3.02488
SR12	1.00000	1.40575	2.03930	2.73233
SR13	1.09968	1.51991	1.89473	2.79659
SR14	1.03324	1.40814	1.73840	2.48232
SR15	1.14384	1.49311	1.69968	1.88493
SR16	1.54321	1.97249	2.77860	3.62654
SR17	1.53898	2.83152	4.97237	6.72775
SR18	1.63911	2.64807	5.57231	8.74436
SR19	1.05591	1.38542	2.70082	3.27194
SR20	1.04288	0.81085	0.72630	0.57857
SR21	0.64005	0.85129	0.99865	1.21165
SR22	0.99767	0.90393	1.11246	1.34495
SR23	1.41261	1.98223	3.19294	2.59324
SR24	1.03157	1.58875	1.4795	2.47207

Table 24: Running time ratio between `PCDM-1`/`PCDM-k` with `k` $\in \{2, 4, 8, 16\}$, applied to SR* problems.

Problem \ Threads	$1T$	$2T$	$4T$	$8T$	$16T$
SR1	2952.0	5170.92	6258.24	7165.98	7867.08
SR2	2214.0	3444.0	4466.13	4250.88	5033.16
SR3	4305.0	6642.0	8141.37	10830.15	12315.99
SR4	1260.0	8195.04	11085.48	8328.6	14467.32
SR5	2160.0	2280.6	3434.4	5601.6	7803.0
SR6	30831.0	31540.83	52591.95	106338.27	150892.65
SR7	10976.0	15444.8	20916.0	28716.8	40237.12
SR8	1360.0	1765.96	2415.36	4154.8	6886.36
SR9	2492.0	2484.88	3560.0	6724.84	7835.56
SR10	3223.0	2044.73	1758.0	2175.11	2608.81
SR11	1800.0	3294.0	3600.0	4500.0	4500.0
SR12	1500.0	3291.0	3303.0	3606.0	4263.0
SR13	2100.0	3597.0	3975.0	5397.0	5526.0
SR14	1800.0	3291.0	3600.0	4116.0	4332.0
SR15	3289.0	4167.46	3915.11	5372.43	5214.86
SR16	146706.0	125748.0	125748.0	125748.0	146706.0
SR17	384615.0	384615.0	341880.0	342307.35	427350.0
SR18	538200.0	496800.0	538200.0	401994.0	412344.0
SR19	632118.0	842824.0	1158883.0	632118.0	737471.0
SR20	7866.0	12696.0	25325.76	45410.28	86491.5
SR21	3081.0	8842.78	5135.0	6280.93	3717.33
SR22	9888.0	13184.0	23170.88	20232.83	16282.24
SR23	50720.0	50720.0	40576.0	20288.0	53357.44
SR24	79575.0	132625.0	106100.0	185675.0	106100.0

Table 25: Average number of iterations of the method `PCDM` with multiple threads applied to SR* problems.

Problem \ Threads	$1T$	$2T$	$4T$	$8T$	$16T$
SR1	0.02429	0.02678	0.02431	0.02913	0.0336
SR2	0.01820	0.02011	0.01835	0.02183	0.02546
SR3	0.02629	0.02958	0.02877	0.0334	0.04079
SR4	0.19722	0.29489	0.26163	0.27652	0.25714
SR5	0.00506	0.00859	0.01233	0.02394	0.00950
SR6	0.60193	0.90691	0.75186	0.70294	0.65568
SR7	0.02279	0.03072	0.03349	0.04509	0.04320
SR8	0.01299	0.01961	0.01952	0.02426	0.02445
SR9	0.05282	0.07261	0.06423	0.0702	0.06393
SR10	0.00097	0.00095	0.00095	0.00138	0.00128
SR11	0.01469	0.01099	0.00922	0.00933	0.01218
SR12	0.01219	0.00962	0.00801	0.00836	0.01094
SR13	0.01083	0.00961	0.00823	0.00885	0.0116
SR14	0.00455	0.00382	0.00338	0.00380	0.00455
SR15	0.00094	0.00106	0.00111	0.00166	0.00174
SR16	0.07538	0.04996	0.02997	0.02362	0.02170
SR17	1.75288	0.98794	0.56279	0.34571	0.29011
SR18	1.24088	0.71877	0.41257	0.24972	0.21352
SR19	0.02077	0.01222	0.00705	0.00459	0.00457
SR20	2.86415	6.61739	7.38814	8.33419	7.72298
SR21	0.00041	0.00049	0.00062	0.00083	0.00085
SR22	0.00141	0.00148	0.00193	0.00242	0.00237
SR23	0.00774	0.00737	0.00943	0.01241	0.01129
SR24	0.01813	0.01595	0.01641	0.02417	0.02061

Table 26: Running time of the method `PA-10` with multiple threads applied to SR* problems.

Problem \ Ratio	$2T/1T$	$4T/1T$	$8T/1T$	$16T/1T$
SR1	0.90702	0.99917	0.83384	0.72291
SR2	0.90502	0.99182	0.83371	0.71484
SR3	0.88877	0.91379	0.78712	0.64452
SR4	0.66879	0.75381	0.71322	0.76697
SR5	0.58917	0.41046	0.21140	0.53273
SR6	0.66371	0.80058	0.85630	0.91802
SR7	0.74186	0.68050	0.50543	0.52754
SR8	0.66241	0.66547	0.53544	0.53128
SR9	0.72744	0.82235	0.75242	0.82621
SR10	1.01981	1.01981	0.70818	0.76287
SR11	1.33667	1.59328	1.57398	1.20608
SR12	1.26649	1.52071	1.45691	1.11426
SR13	1.12695	1.31464	1.22262	0.93041
SR14	1.1921	1.34564	1.19805	1.00110
SR15	0.89359	0.85112	0.57065	0.54508
SR16	1.50881	2.51518	3.19136	3.47373
SR17	1.77428	3.11463	5.07038	6.04212
SR18	1.72639	3.00768	4.96909	5.81154
SR19	1.69967	2.94359	4.51915	4.53989
SR20	0.43282	0.38766	0.34366	0.37086
SR21	0.82828	0.65390	0.49338	0.47785
SR22	0.95020	0.72933	0.58250	0.59578
SR23	1.05031	0.82142	0.62417	0.68609
SR24	1.13668	1.10481	0.75010	0.87967

Table 27: Running time ratio between `PA-10-1`/`PA-10-k` with `k` $\in \{2, 4, 8, 16\}$, applied to SR* problems.

Problem \ Threads	$1T$	$2T$	$4T$	$8T$	$16T$
SR1	685.11	907.74	1291.50	2102.07	3713.37
SR2	562.11	709.71	1001.22	1627.29	2858.52
SR3	880.68	1132.83	1725.69	2740.44	5109.42
SR4	1120.14	1679.58	2613.24	4393.62	7247.52
SR5	1809.09	2384.62	3450.60	5583.60	8894.72
SR6	5384.67	7614.54	11464.83	19043.52	31971.03
SR7	1682.24	2261.28	3690.40	6588.96	11966.08
SR8	398.48	584.80	981.92	1743.52	3289.84
SR9	1384.84	1840.52	2961.92	5283.04	10096.16
SR10	783.63	881.53	1127.90	1667.35	2778.62
SR11	873.00	960.00	1337.62	2069.44	3576.00
SR12	852.00	949.04	1269.60	1995.00	3399.00
SR13	915.00	1122.86	1556.56	2490.27	4374.00
SR14	855.00	953.82	1296.28	1984.79	3415.91
SR15	989.48	1188.07	1616.00	2512.20	4249.96
SR16	45478.86	45059.70	45059.70	57215.34	63921.90
SR17	125640.91	126495.60	126922.95	129059.70	131623.80
SR18	122130.00	123786.00	124200.00	125028.00	129582.00
SR19	212813.06	214920.12	213673.69	213605.62	215035.58
SR20	1780.20	3207.12	5964.36	11659.62	23047.38
SR21	2165.59	2306.94	2855.00	3753.45	5649.38
SR22	6768.90	7111.13	9269.60	11867.20	17425.06
SR23	20444.06	20959.29	28441.12	36315.52	55589.12
SR24	53050.00	53050.00	56233.00	79575.00	106100.00

Table 28: Average number of iterations of the method `PA-10` with multiple threads applied to SR* problems.