Distributed Convolutional Dictionary Learning (DiCoDiLe): Pattern Discovery in Large Images and Signals

Moreau Thomas[1]*, Alexandre Gramfort[1]
[1]INRIA, Université Paris Saclay, Saclay, France

January 26, 2019

Abstract

Convolutional dictionary learning (CDL) estimates shift invariant basis adapted to multidimensional data. CDL has proven useful for image denoising or inpainting, as well as for pattern discovery on multivariate signals. As estimated patterns can be positioned anywhere in signals or images, optimization techniques face the difficulty of working in extremely high dimensions with millions of pixels or time samples, contrarily to standard patch-based dictionary learning. To address this optimization problem, this work proposes a distributed and asynchronous algorithm, employing locally greedy coordinate descent and an asynchronous locking mechanism that does not require a central server. This algorithm can be used to distribute the computation on a number of workers which scales linearly with the encoded signal's size. Experiments confirm the scaling properties which allows us to learn patterns on large scales images from the Hubble Space Telescope.

1 Introduction

The sparse convolutional linear model has been used successfully for various signal and image processing applications. It was first developed by Grosse et al. (2007) for music recognition and has then been used extensively for denoising and inpainting of images (Kavukcuoglu et al., 2010; Bristow et al., 2013; Wohlberg, 2014). The benefit of this model over traditional sparse coding technique is that it is shift-invariant. Instead of local patches, it considers the full signals which allows for a sparser reconstruction, as not all shifted versions of the same patches are present in the estimated dictionary (Bergstra et al., 2011).

Beyond denoising or inpainting, recent applications used this shift-invariant model as a way to learn and localize important patterns in signals or images. Yellin et al. (2017) used convolutional sparse coding to count the number of blood-cells present in holographic lens-free images. Jas et al. (2017) and Dupré la Tour et al. (2018) learned recurring patterns in univariate and multivariate signals from neuroscience. This model has also been used with astronomical data as a way to denoise (Starck et al., 2005) and localize predefined patterns from space satellite images (del Aguila Pla and Jalden, 2018).

Convolutional dictionary learning estimates the parameters of this model. It boils down to an optimization problem, where one needs to estimate the patterns (or atoms) of the basis as well as their activations, *i.e.* where the patterns are present in the data. The latter step, which is commonly referred to as convolutional sparse coding (CSC), is the critical bottleneck when it comes to applying CDL to large data. Chalasani et al. (2013) used an accelerated proximal gradient method based on Fast Iterative Soft-Thresholding Algorithm (FISTA; Beck and Teboulle 2009 that was adapted for convolutional problems. The Fast Convolutional

*Corresponding author thomas.moreau@inria.fr

Sparse Coding of Bristow et al. (2013) is based on Alternating Direction Method of Multipliers (ADMM). Both algorithms compute a convolution on the full data at each iteration. While it is possible to leverage the Fast Fourier Transform to reduce the complexity of this step (Wohlberg, 2016), the cost can be prohibitive for large signals or images. To avoid such costly global operations, Moreau et al. (2018) proposed to use locally greedy coordinate descent (LGCD) which requires only local computations to update the activations. Based on Greedy Coordinate Descent (GCD), this method has a lower per-iteration complexity and it relies on local computation in the data, like a patch based technique would do, yet solving the non-local problem.

To further reduce the computational burden for CDL, recent studies consider different parallelization strategies. Skau and Wohlberg (2018) proposed a method based on consensus ADMM to parallelize the CDL optimization by splitting the computation across the different atoms. This limits the number of cores that can be used to the number of atoms in the dictionary, and does not significantly reduce the burden for very large signals or images. An alternative parallel algorithm called DICOD, proposed by Moreau et al. (2018) distributes GCD updates when working with 1d signals. Their results could be used with multidimensional data such as images, yet only by splitting the data along one dimension. This limits the gain of their distributed optimization strategy when working with images. Also, the choice of GCD for the local updates make the algorithm inefficient when used with low number of workers.

In the present paper, we propose a distributed solver for CDL, which can fully leverage multidimensional data. For the CSC step, we present a coordinate selection rule based on LGCD that works in the asynchronous setting without a central server. We detail a soft-lock mechanism ensuring the convergence with a grid of workers, while splitting the data in all convolutional dimensions. Then, we explain how to also leverage the distributed set of workers to further reduce the cost of the dictionary update, by precomputing in parallel some quantities used in the gradient operator. Extensive numerical experiments confirm the performance of our algorithm, which is then used to learn patterns from a large astronomical image.

In Section 2, we describe the convolutional linear model and its parameter estimation through CDL. Then, Section 3 details the LGCD algorithm for convolutional sparse coding (CSC). Our distributed algorithm DiCoDiLe is introduced in Section 4. Section 5 presents results on simulations and images.

2 Convolutional Dictionary Learning

Notations. For a vector $U \in \mathbb{R}^P$ we denote $U_p \in \mathbb{R}$ its p^{th} coordinate. For finite domain $\Omega = \prod_{i=1}^d [0, T_i[$, where $[0, T_i[$ are the integers from 0 to $T_i - 1$, $|\Omega|$ is the size of this domain $\prod_{i=1}^d T_i$. We denote $\mathcal{X}_\Omega^P$ (resp. $\mathcal{X}_\Omega^{P\times Q}$) the space of observations defined on Ω with values in $\mathbb{R}^P$ (resp. $\mathbb{R}^{P\times Q}$). For instance, $\mathcal{X}_\Omega^3$ with $d = 2$ is the space of RGB-images with height and width T_1, T_2. The value of an observation $X \in \mathcal{X}_\Omega^P$ at position $\omega \in \Omega$ is given by $X[\omega] = X[\omega_1, \dots, \omega_d] \in \mathbb{R}^P$, and its restriction to the p^{th} coordinate at each position is denoted $X_p \in \mathcal{X}_\Omega^1$. All signals are 0-padded, *i.e.* for $\omega \notin \Omega$, $X[\omega] = 0$. The convolution operator is denoted $*$ and is defined for $X \in \mathcal{X}_\Omega^K$ and $\boldsymbol{Y} \in \mathcal{X}_\Omega^{K\times P}$ as the signal $X * \boldsymbol{Y} \in \mathcal{X}_\Omega^P$ with

$$(X * \boldsymbol{Y})[\omega] = \sum_{\tau\in\Omega} X[\omega - \tau] \cdot \boldsymbol{Y}[\tau] \ , \quad \forall \omega \in \Omega \ . \tag{1}$$

For any signal $X \in \mathcal{X}_\Omega^P$, the reversed signal is defined as $X^\curvearrowleft[\omega] = X[T_1 - \omega_1, \dots, T_d - \omega_d]^\mathsf{T}$ and the p-norm is defined as $\|X\|_p = \left(\sum_{\omega\in\Omega} \|X[\omega]\|_p^p\right)^{1/p}$. We will also denote ST the soft-thresholding operator defined as

$$\mathrm{ST}(u, \lambda) = \mathrm{sign}(u) \max(|u| - \lambda, 0) \ .$$

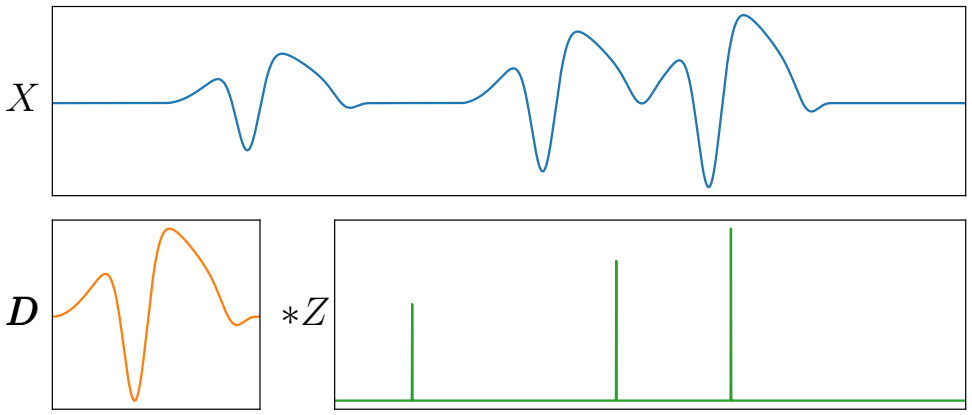

Figure 1: Decomposition of a noiseless univariate signal X (*blue*) as the convolution $Z * \boldsymbol{D}$ between a temporal pattern $\boldsymbol{D}$ (*orange*) and a sparse activation signal Z (*green*).

Convolutional Linear Model. For an observation $X \in \mathcal{X}_\Omega^P$, the convolutional linear model reads

$$X = Z * \boldsymbol{D} + \xi \,, \tag{2}$$

with $\boldsymbol{D} \in \mathcal{X}_\Theta^{K \times P}$ a dictionary of K atoms with the same number of dimensions P on a domain $\Theta = \prod_{i=1}^d [0, L_i[\subset \Omega$ and $Z \in \mathcal{X}_\Omega^K$ the set of K activation vectors associated with this dictionary. $\xi \in \mathcal{X}_\Omega^P$ is an additive noise term which is typically considered Gaussian and white. The dictionary elements $\boldsymbol{D}_k$ are patterns present in the data and typically have a much smaller support Θ than the full domain Ω. The activations Z_k encode the localization of the pattern k in the data. As patterns typically occur at few locations in each observation, the coding signal Z is considered to be sparse. Figure 1 illustrates the decomposition of a temporal signal with one pattern.

The parameters of this model are estimated by solving the following optimization problem, called convolutional dictionary learning (CDL)

$$\min_{\substack{Z, \boldsymbol{D} \\ \|\boldsymbol{D}_k\| \leq 1}} \underbrace{\frac{1}{2} \left\| X - Z * \boldsymbol{D} \right\|_2^2}_{F(Z, \boldsymbol{D})} + \underbrace{\lambda \|Z\|_1}_{G(Z)} \,. \tag{3}$$

where the first term measures how well the model fits the data, the second term enforces a sparsity constraint on the activation, and $\lambda > 0$ is a regularization parameter that balances the two. The constraint $\|D_k\|_2 \leq 1$ is used to avoid the scale ambiguity in our model – due to the ℓ_1 constraint. Indeed, the ℓ_1-norm of Z can be made arbitrarily small by rescaling D by $\alpha \in \mathbb{R}_+^*$ and Z by $\frac{1}{\alpha}$. The minimization (3) is not jointly convex in $(Z, \boldsymbol{D})$ but it is convex in each of its coordinates. It is usually solved with an alternated minimization algorithm, updating at each iteration one of the block of coordinate, Z or $\boldsymbol{D}$.

Convolutional Sparse Coding (CSC). Given a dictionary of patterns $\boldsymbol{D}$, CSC aims to retrieve the sparse decomposition Z^* associated to the signal X. When minimizing only over Z, (3) becomes

$$Z^\star = \operatorname*{argmin}_Z \frac{1}{2} \left\| X - Z * \boldsymbol{D} \right\|_2^2 + \lambda \left\| Z \right\|_1 \,. \tag{4}$$

Algorithm 1 Locally Greedy Coordinate Descent

Input: $\boldsymbol{D}, X$, parameter $\epsilon > 0$, sub-domains $\mathcal{C}_m$,
Initialization: $\forall (k, \omega) \in [1, K] \times \Omega$,
$Z_k[\omega] = 0, \quad \beta_k[\omega] = \left(\boldsymbol{D}_k^{\scriptscriptstyle\backslash} * X\right)[\omega]$
repeat
  for $m = 1 \dots M$ **do**
    $\forall (k, \omega) \in [1, K] \times \mathcal{C}_m, \quad Z'_k[\omega] = \dfrac{1}{\|\boldsymbol{D}_k\|_2^2} \mathrm{ST}(\beta_k^{(q)}[\omega], \lambda)$,
    Choose $(k_0, \omega_0) = \arg \max_{(k,\omega) \in [1,K] \times \mathcal{C}_m} |\Delta Z_k[\omega]|$
    Update β using (8) and $Z_{k_0}[\omega_0] \leftarrow Z'_{k_0}[\omega_0]$
  end for
until $\|\Delta Z\|_\infty < \epsilon$

The problem in (4) boils down to a LASSO problem with a Toeplitz design matrix $\boldsymbol{D}$. Therefore, classical LASSO optimization techniques can easily be applied with the same convergence guarantees. Chalasani et al. (2013) adapted FISTA (Beck and Teboulle, 2009) exploiting the convolution in the gradient computation to save time. Based on the ADMM algorithm (Gabay and Mercier, 1976), Bristow et al. (2013) solved the CSC problem with a fast computation of the convolution in the Fourier domain. For more details on these techniques, see Wohlberg (2016) and references therein. Finally, Kavukcuoglu et al. (2010) proposed an efficient adaptation of the greedy coordinate descent (Osher and Li, 2009) to efficiently solve the CSC, which has been later refined by Moreau et al. (2018) with locally greedy coordinate selection (LGCD).

It is useful to note that for $\lambda \geq \|X * \boldsymbol{D}^{\scriptscriptstyle\backslash}\|_\infty$, 0 minimizes the equation (4). The first order condition for (4) reads

$$\nabla_Z F(0, \boldsymbol{D}) = X * \boldsymbol{D}^{\scriptscriptstyle\backslash} \in \partial G(0) = [-\lambda, \lambda]^{K|\Omega|} . \tag{5}$$

This condition is always verified if $\lambda > \lambda_{\max} = \|X * \boldsymbol{D}^{\scriptscriptstyle\backslash}\|_\infty$. In the following, the regularization parameter λ is set as a fraction of this value.

Dictionary Update. Given a activation signal $Z^{(q)}$, the dictionary update aims at improving how the model reconstructs the signal X using the fixed codes, by solving

$$\boldsymbol{D}^* = \underset{\substack{\boldsymbol{D} \\ \|\boldsymbol{D}_k\|_2 \leq 1}}{\operatorname{argmin}} \frac{1}{2} \left\| X - Z * \boldsymbol{D} \right\|_2^2 . \tag{6}$$

This problem is smooth and convex and can be solved using classical algorithms. The block coordinate descent for dictionary learning (Mairal et al., 2010) can be easily extended to the convolutional cases. Recently, Yellin et al. (2017) proposed to adapt the K-SVD dictionary update method (Aharon et al., 2006) to the convolutional setting. Classical convex optimization algorithms, such as projected gradient descent (PGD) and its accelerated version (APGD), can also be used for this step (Combettes and Bauschke, 2011). These last two algorithms are often referred to as ISTA and FISTA (Chalasani et al., 2013).

3 Locally Greedy Coordinate Descent for CSC

Coordinate descent (CD) is an algorithm which updates a single coefficient at each iteration. For (4), it is possible to compute efficiently, and in closed form, the optimal update of a given coordinate when all the others are kept fixed. Denoting q the iteration number, and $Z'_{k_0}[\omega_0]$ (atom k_0 at position ω_0) the updated version of $Z^{(q)}_{k_0}[\omega_0]$, the update reads

$$Z'_{k_0}[\omega_0] = \frac{1}{\|\boldsymbol{D}_{k_0}\|_2^2} \mathrm{ST}(\beta^{(q)}_{k_0}[\omega_0], \lambda) \ , \tag{7}$$

where $\beta^{(q)}$ is an auxiliary variable in $\mathcal{X}^K_\Omega$ defined for $(k, \omega) \in [1, K] \times \Omega$ as

$$\beta^{(q)}_k[\omega] = \left(\left(X - Z^{(q)} * \boldsymbol{D} + Z^{(q)}_k[\omega] e_\omega * \boldsymbol{D}_k \right) * \boldsymbol{D}^{\tilde{}}_k \right)[\omega] \ ,$$

and where e_ω is a dirac (canonical vector) with value 1 in ω and 0 elsewhere. To make this update efficient it was noticed by Kavukcuoglu et al. (2010) that if the coordinate $Z_{k_0}[\omega_0]$ is updated with an additive update $\Delta Z_{k_0}[\omega_0] = Z'_{k_0}[\omega_0] - Z^{(q)}_{k_0}[\omega_0]$, then it is possible to cheaply obtain $\beta^{(q+1)}$ from $\beta^{(q)}$ using the relation

$$\beta^{(q+1)}_k[\omega] = \beta^{(q)}_k[\omega] - (\boldsymbol{D}_{k_0} * \boldsymbol{D}^{\tilde{}}_k)[\omega - \omega_0] \Delta Z^{(q)}_{k_0}[\omega_0] \ , \tag{8}$$

for all $(k, \omega) \neq (k_0, \omega_0)$. As the support of $(\boldsymbol{D}_{k_0} * \boldsymbol{D}^{\tilde{}}_k)[\omega]$ is $\prod_{i=1}^d [-L_i + 1, L_i[$, equation (8) implies that, after an update in ω_0, the value $\beta^{(q+1)}_k$ only changes in the neighborhood $\mathcal{V}(\omega_0)$ of ω_0, where

$$\mathcal{V}(\omega_0) = \prod_{i=1}^d [(\omega_0)_i - L_i + 1, (\omega_0)_i + L_i[\ . \tag{9}$$

We recall that L_i is the size of an atom in the direction i. This means that only $\mathcal{O}(2^d K |\Theta|)$ operations are needed to maintain β up-to-date after each Z update. The procedure is run until $\max_{k,t} |\Delta Z_k[t]|$ becomes smaller than a specified tolerance parameter $\epsilon \geq 0$.

The selection of the updated coordinate (k_0, ω_0) can follow different strategies. Cyclic updates (Friedman et al., 2007) and random updates (Shalev-Shwartz and Tewari, 2009) are efficient strategies which only require to access one value of (7). They have a $\mathcal{O}(1)$ computational complexity. Osher and Li (2009) propose to select greedily the coordinate which is the farther from its optimal value. In this case, the coordinate is chosen as the one with the largest additive update $\max_{(k,\omega)} |\Delta Z_k[\omega]|$. The quantity $\Delta Z_k[\omega]$ acts as a proxy for the cost reduction obtained with this update.

This strategy is computationally more expensive, with a cost of $\mathcal{O}(K|\Omega|)$. Yet, it has a better convergence rate (Nutini et al., 2015; Karimireddy et al., 2018) as it updates in priority the important coordinates. To reduce the iteration complexity of greedy coordinate selection while still selecting more relevant coordinates, Moreau et al. (2018) proposed to select the coordinate in a locally greedy fashion. The domain Ω is partitioned in M disjoint sub-domains $\mathcal{C}_m$: $\Omega = \cup_m \mathcal{C}_m$ and $\mathcal{C}_m \cap \mathcal{C}_{m'} = \emptyset$ for $m \neq m'$. Then, at each iteration q, the coordinate to update is selected greedily on the m-th sub-domain $\mathcal{C}_m$

$$(k_0, \omega_0) = \underset{(k,\omega) \in [1,K] \times \mathcal{C}_m}{\operatorname{argmax}} |\Delta Z_k[\omega]| \ ,$$

with $m = q \mod M$ (cyclic rule).

Algorithm 2 DiCoDiLe with W workers

Input: $\boldsymbol{D}^{(0)}, X$, stopping criterion ν.
repeat
  Compute $Z^{(q+1)}$ with DiCoDiLe-$Z(X, \boldsymbol{D}^{(q)}, W)$.
  Compute ϕ and ψ with (17) and W workers.
  Update $\boldsymbol{D}^{(q+1)}$ with PGD and armijo backtracking line-search.
until Cost variation is smaller than ν

This selection differs from the greedy selection as it is only locally selecting the best coordinate to update. The iteration complexity of this strategy is therefore linear with the size of the sub-segments $\mathcal{O}(K|\mathcal{C}_m|)$. If the size of the sub-segments is 1, this algorithm is equivalent to a cyclic coordinate descent, while with $M = 1$ and $\mathcal{C}_1 = \Omega$ it boils down to greedy updates. The selection of the sub-segment size is therefore a trade-off between the cyclic and greedy coordinate selection. By using sub-segments of size $2^d|\Theta|$, the computational complexity of selecting the coordinate to update and the complexity of the update of β are matched. The global iteration complexity of LGCD becomes $\mathcal{O}(K|\Theta|)$. Algorithm 1 summarizes the computations for LGCD.

4 Distributed Convolutional Dictionary Learning (DiCoDiLe)

In this section, we introduce DiCoDiLe, a distributed algorithm for convolutional sparse coding. Its step are summarized in Algorithm 2. The CSC part of this algorithm will be referred to as DiCoDiLe-Z and extend the DICOD algorithm to handle multidimensional data such as images, which were not covered by the original algorithm. The key to ensure the convergence with multidimensional data is the use of asynchronous soft-locks between neighboring workers. We also propose to employ a locally greedy CD strategy which boosts running time, as demonstrated in the experiments. The dictionary update relies on smart pre-computation which make the computation of the gradient of (6) independent of the size of Ω.

4.1 Distributed sparse coding with LGCD

For convolutional sparse coding, the coordinates that are far enough – compared to the size of the dictionary – are only weakly dependent. It is thus natural to parallelize CSC resolution by splitting the data in continuous sub-domains. It is the idea of the DICOD algorithm proposed by Moreau et al. (2018) for one-dimensional signals.

Given W workers, DICOD partitions the domain Ω in W disjoint contiguous time intervals $\mathcal{S}_w$.

Then each worker w runs asynchronously a greedy coordinate descent algorithm on $\mathcal{S}_w$. To ensure the convergence, Moreau et al. (2018) show that it is sufficient to notify the neighboring workers if a local update changes the value of β outside of $\mathcal{S}_w$.

Let us consider a sub-domain, $\mathcal{S}_w = \prod_{i=1}^d [l_i, u_i[$. The Θ-border is

$$\mathcal{B}_L(\mathcal{S}_w) = \prod_{i=1}^d [l_i, l_i + L_i[\cup [u_i - L_i, u_i[\,. \tag{10}$$

Algorithm 3 DiCoDiLe-Z with W workers

Input: $\boldsymbol{D}, X$, parameter $\epsilon > 0$, sub-domains $\mathcal{S}_w$,
In parallel for $w = 1 \cdots W$
Compute a partition $\{\mathcal{C}_m^{(w)}\}_{m=1}^M$ of the worker domain $\mathcal{S}_w$ with sub-domains of size $2^d|\Theta|$.
Initialize $\beta_k[t]$ and $Z_k[t]$ $\forall (k,t) \in [1,K] \times \mathcal{S}_w$,
repeat
  for m=1...M **do**
    Receive messages and update Z and β with (8)
    Choose $(k_0, \omega_0) = \underset{(k,\omega) \in [1,K] \times \mathcal{C}_m^{(w)}}{\operatorname{argmax}} |\Delta Z_k[\omega]|$
    if $|\Delta Z_{k_0}[\omega_0]| < \underset{(k,\omega) \in [1,K] \times \mathcal{V}(\omega_0)}{\max} |\Delta Z_k[\omega]$ **then**
      The coordinate is soft-locked, **goto 6**
    end if
    Update β with (8) and $Z_{k_0}[\omega_0] \leftarrow Z'_{k_0}[\omega_0]$
    if $\omega_0 \in \mathcal{B}_{2L}(\mathcal{S}_w)$ **then**
      Send $(k_0, \omega_0, \Delta Z_{k_0}[\omega_0])$ to neighbors
    end if
  end for
until global convergence $\|\Delta Z\|_\infty < \epsilon$

In case that a Z update affects β in the Θ-border $\mathcal{B}_L(\mathcal{S}_w)$ of $\mathcal{S}_w$, then one needs to notify other workers w' whose domain overlap with the update of β *i.e.*$\mathcal{V}(\omega_0) \cap \mathcal{S}_{w'} \neq \emptyset$. It is done by sending the triplet $(k_0, \omega_0, \Delta Z_{k_0}[\omega_0])$ to these workers, which can then update β using formula (8). Figure 2 illustrates this communication process on a 2D example. There is few inter-processes communications in this distributed algorithm as it does not rely on centralized communication, and a worker only communicates with its neighbors. Moreover, as the communication only occurs when $\omega_0 \in \mathcal{B}_L(\mathcal{S}_w)$, a small number of iterations need to send messages if $|\mathcal{B}_L(\mathcal{S}_w)| \ll |\mathcal{S}_w|$.

As a worker can be affected by its neighboring workers, the stopping criterion of CD cannot be applied independently in each worker. To reach a consensus, the convergence is considered to be reached once no worker can offer a change on a $Z_k[\omega]$ that is higher than ϵ. Workers that reach this state locally are paused, waiting for incoming communication or for the global convergence to be reached.

DiCoDiLe-Z This algorithm, described in Algorithm 3, is distributed over W workers which update asynchronously the coordinates of Z. The domain Ω is once also partitioned with sub-domain $\mathcal{S}_w$ but unlike DICOD, the partitioning is not restricted to be chosen along one direction. Each worker $w \in [1, W]$ is in charge of updating the coordinates of one sub-domain $\mathcal{S}_w$.

The worker w computes a sub-partition of its local domain $\mathcal{S}_w$ in M disjoint sub-domains $\mathcal{C}_m^{(w)}$ of size $2^d|\Theta|$. Then, at each iteration q, the worker w chooses an update candidate

$$(k_0, \omega_0) = \underset{(k,\omega) \in [1,K] \times \mathcal{C}_m^{(w)}}{\operatorname{argmax}} |\Delta Z_k[\omega]| \ , \tag{11}$$

with $m = q \mod M$. The process to accept or reject the update candidate uses a soft-lock mechanism described in the following paragraphs. If the candidate is accepted, the value of the coordinate $Z_{k_0}[\omega_0]$ is updated to $Z'_{k_0}[\omega_0]$, beta is updated with (8) and the neighboring workers w' are notified if $\mathcal{V}(\omega_0) \cap \mathcal{S}_{w'} \neq \emptyset$. If the candidate is rejected, the worker moves on to the next sub-domain $\mathcal{C}_{m+1}^w$ without updating a coordinate.

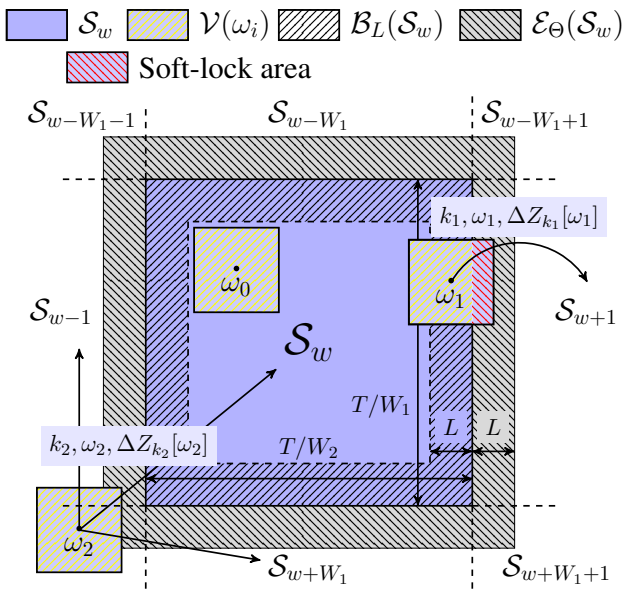

Figure 2: Communication process in DiCoDiLe-Z with $d = 2$ and 9 workers centered around worker w. The update in ω_0 is independent of the other workers. The update in ω_1 is performed if no better coordinate update is possible in the soft-lock area $\mathcal{V}(\omega_1) \cap \mathcal{S}_{w+1}$ (*red hatched area*). If accepted, the worker $w + 1$ needs to be notified. The update in ω_2 changes the value of the optimal updates in the Θ-extension of the other workers sub-domains. Thus it needs to notify all the neighboring workers.

Interferences. When W coordinates $(k_w, \omega_w)_{w=1}^W$ of Z are updated simultaneously by respectively $\Delta Z_{k_w}[\omega_w]$, the updates might not be independent. The local version of β used for the update does not account for the other updates. The cost difference resulting from these updates is denoted ΔE, and the cost reduction induced by only one update w is denoted $\Delta E_{k_w}[\omega_w]$. Simple computations, detailed in Proposition A.2, show that

$$\Delta E = \sum_{i=1}^{W} \Delta E_{k_w}[\omega_w] \\ - \sum_{w \neq w'} (\boldsymbol{D}_{k_w} * \boldsymbol{D}^{\curvearrowleft}_{k_{w'}})[\omega_{w'} - \omega_w] \Delta Z_{k_w}[\omega_w] \Delta Z_{k_{w'}}[\omega_{w'}] \ , \tag{12}$$

If for all w, all other updates w' are such that if $\omega_{w'} \notin \mathcal{V}(\omega_w)$, then $(\boldsymbol{D}_{k_w} * \boldsymbol{D}^{\curvearrowleft}_{k_{w'}})[\omega_{w'} - \omega_w] = 0$ and the updates can be considered to be sequential as the interference term is zero. When $\left| \{\omega_w\}_w \cap \mathcal{V}(\omega_0) \right| = I_0 > 1$, the interference term does not vanish. Moreau et al. (2018) show that if $I_0 < 3$, then, under mild assumption, the interference term can be controlled and DICOD converges. This is sufficient for 1D partitioning as this is always verified in this case. However, their analysis cannot be extended to $I_0 \geq 3$, limiting the partitioning of Ω.

Soft Locks. To avoid this limitation, we propose the soft-lock, a novel mechanism to avoid interfering updates of higher order. The idea is to avoid updating concurrent coordinates simultaneously. A classical tool in computer science to address such concurrency issue is to rely on a synchronization primitive, called *lock*. This primitive can ensure that one worker enters a given block of code. In our case, it would be possible to use this primitive to only allow one worker to make an update on $\mathcal{B}_L(\mathcal{S}_w)$. This would avoid any interfering update. The drawback of this method is however that it prevents the algorithm to be asynchronous, as typical lock implementations rely on centralized communications. Instead, we propose to use an implicit locking mechanism which keeps our algorithm asynchronous. This idea is the following.

Additionally to the sub-domain $\mathcal{S}_w$, each worker also maintains the value of Z and β on the Θ-extension $\mathcal{E}_L(\mathcal{S}_w)$ of its sub-domain, defined for $\mathcal{S}_w = \prod_{i=1}^d [l_i, u_i[$ as

$$\mathcal{E}_L(\mathcal{S}_w) = \prod_{i=1}^d [l_i - L_i, l_i[\cup [u_i, u_i + L_i[\, . \tag{13}$$

These two quantities can be maintained asynchronously by sending the same notifications on an extended border $\mathcal{B}_{2L}(\mathcal{S}_w)$ with twice the size of Θ in each direction. Then, at each iteration, the worker w gets a candidate coordinate (k_0, ω_0) to update from (11). If $\omega_0 \notin \mathcal{B}_L(\mathcal{S}_w)$, the coordinate is updated like in LGCD. When $\omega_0 \in \mathcal{B}_L(\mathcal{S}_w)$, the candidate is accepted if there is no better coordinate update in $\mathcal{V}(\omega_0) \cap \mathcal{E}_L(\mathcal{S}_w)$, *i.e.* if

$$|\Delta Z_{k_0}[\omega_0]| > \max_{k,\omega \in [1,K] \times \mathcal{V}(\omega_0) \cap \mathcal{E}_L(\mathcal{S}_w)} |\Delta Z_k[\omega]| \ . \tag{14}$$

In case of equality, the preferred update is the one included in the sub-domain $\mathcal{S}_{w'}$ with the minimal index. Using this mechanism effectively prevents any interfering update. If two workers $w < w'$ have update candidates ω_0 and ω_0' such that $\omega_0' \in \mathcal{V}(\omega_0)$, then with (14), only one of the two updates will be accepted – either the largest one if $|\Delta Z_{k_0}[\omega_0]| \neq |\Delta Z_{k_0'}[\omega_0']|$ or the one from w if both updates have the same magnitude – and the other will be dropped. This way the Z updates can be done in an asynchronous way.

Speed-up analysis of DiCoDiLe-Z Our algorithm DiCoDiLe-Z has a sub-linear speed-up with the number of workers W. As each worker runs LGCD locally, the computational complexity of a local iteration in a worker w is $\mathcal{O}(2^d K|\Theta|)$. This complexity is independent of the number of workers used to run the algorithm. Thus, the speed-up obtained is equal to the number of updates that are performed in parallel. If we consider W update candidates (k_w, ω_w) chosen by the workers, the number of updates that are performed corresponds to the number of updates that are not soft locked. When the updates are uniformly distributed on $\mathcal{S}_w$, the probability that an update candidate located in $\omega \in \mathcal{S}_w$ is not soft locked can be lower bounded by

$$P(\omega \notin \textbf{SL}) \geq \prod_{i=1}^d (1 - \frac{W_i L_i}{T_i}) \tag{15}$$

Computations are detailed in Proposition B.1. When the ratios $\frac{T_i}{W_i L_i}$ are large for all i, *i.e.* when the size of the workers sub-domains $|\mathcal{S}_w|$ are large compared to the dictionary size $|\Theta|$, then $P(\omega \notin \textbf{SL}) \simeq 1$. In this case, the expected number of accepted update candidates is close to W and DiCoDiLe-Z scales almost linearly. When W_i reaches $\frac{T_i}{L_i}(\frac{2^{1/d}}{2^{1/d}-1})$, then $P(\omega \notin \textbf{SL}) \gtrsim \frac{1}{2}$, and the acceleration is reduced to $\frac{W}{2}$.

In comparison, Moreau et al. (2018) reports a super-linear speed-up for DICOD on 1D signals. This is due to the fact that they are using GCD locally in each worker, which has a very high iteration complexity when W is small, as it scales linearly with the size of the considered sub domain. Thus, when sub-dividing the signal between more workers, the iteration complexity is decreasing and more updates are performed in parallel, which explains why the speed-up is almost quadratic for low W. As the complexity of iterative GCD are worst than LGCD, DiCoDiLe-Z has better performance than DICOD in low W regime. Furthermore, in the 1D setting, DICOD becomes similar to DiCoDiLe-Z once the size of the workers sub-domain becomes too small to be further partitioned with sub-segments $\mathcal{C}_m^{(w)}$ of size $2^d|\Theta|$. The local iteration of DiCoDiLe-Z are the same as the one performed by DICOD in this case. Thus, DICOD does not outperform DiCoDiLe-Z in the large W setting either.

4.2 Distributed dictionary updates

Dictionary updates need to minimize a quadratic objective under constraint (6). DiCoDiLe uses PGD to update its dictionary, with an Armijo backtracking line-search (Wright and Nocedal, 1999). Here, the bottle neck is that the computational complexity of the gradient for this objective is $\mathcal{O}(PK^2|\Omega|\log(|\Omega|))$. For very large signals or images, this cost is prohibitive. We propose to also use the distributed set of workers to scale the computations. The function to minimize in (6) can be factorized as

$$\nabla_{\boldsymbol{D}} F(Z, \boldsymbol{D}) = Z^{\tilde{}} * (X - Z * \boldsymbol{D}) = \psi - \phi * \boldsymbol{D} \ , \tag{16}$$

with $\phi \in \mathcal{X}_{\Phi}^{K\times K}$ being the restriction of the convolution $Z^{\tilde{}} * Z$ to $\Phi = \prod_{i=1}^{d}[-L_i+1, L_i[$, and $\psi \in \mathcal{X}_{\Theta}^{K\times P}$ the convolution $Z^{\tilde{}} * X$ restricted to Θ. These two quantities can be computed in parallel by the workers. For $\tau \in \Phi$,

$$\phi[\tau] = \sum_{\omega\in\Omega} Z[\omega]Z[\tau+\omega] = \sum_{w=1}^{W}\sum_{\omega\in\mathcal{S}_w} Z[\omega]Z[\tau+\omega] \tag{17}$$

and for all $\tau, \omega \in \Phi \times \mathcal{S}_w$, we have $\omega - \tau \in \mathcal{S}_w \cup \mathcal{E}_\Theta(\mathcal{S}_w)$. Thus, the computations of the local ϕ^w can be mapped to each worker, before being provided to the reduce operation $\phi = \sum_{w=1}^{W} \phi^w$. The same remark is valid for the computation of $Z^{\tilde{}} * X$. By making use of these distributed computation, ϕ and ψ can be computed with respectively $\mathcal{O}(K^2|\mathcal{S}_w|\log(|\mathcal{S}_w \cup \mathcal{E}_\Theta(\mathcal{S}_w)|))$ and $\mathcal{O}(KP|\mathcal{S}_w|\log(|\mathcal{S}_w \cup \mathcal{E}_\Theta(\mathcal{S}_w)|))$ operations. The values of ψ and ϕ are sufficient statistics to compute $\nabla_{\boldsymbol{D}} F$ and F, which can thus be computed with complexity $\mathcal{O}(K^2P|\Theta|\log(2^d|\Theta|))$, independently of the signal size. This allows to efficiently update the dictionary atoms.

5 Numerical Experiments

The numerical experiments are run on a SLURM cluster with 30 nodes. Each node has 20 physical cores, 40 logical cores and 250 GB of RAM. DiCoDiLe is implemented in Python (Python Software Fundation, 2017) using `mpi4py` (Dalcín et al., 2005)[1].

5.1 Performance of distributed sparse coding

The distributed sparse coding step in DiCoDiLe is impacted by the coordinate descent strategy, as well as the data partitioning and soft-lock mechanism. The following experiments show their impact on DiCoDiLe-Z.

Coordinate Selection Strategy The experimental data are generated following the sparse convolutional linear model (2) with $d = 1$ in $\mathbb{R}^P$ with $P = 7$. The dictionary is composed of $K = 25$ atoms $\boldsymbol{D}_k$ of length $L = 250$. Each atom is sampled from a standard Gaussian distribution and then normalized. The sparse code entries are drawn from a Bernoulli-Gaussian distribution, with Bernoulli parameter $\rho = 0.007$, mean 0 and standard variation 10. The noise term ξ is sampled from a standard Gaussian distribution with variance 1. The length of the signals X is denoted T. The regularization parameter was set to $0.1\lambda_{\max}$.

[1] Code available at github.com/tommoral/dicodile

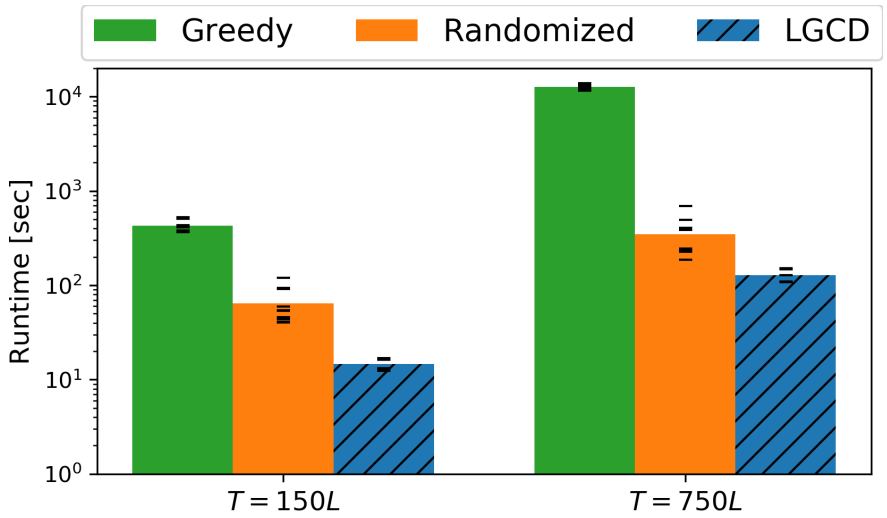

Figure 3: Average running time of CD with three coordinate selection schemes – (*blue*) Locally Greedy, (*orange*) Randomized and (*green*) Greedy – for two signal lengths. LGCD consistently outperforms the two other strategies.

Using one worker, Figure 3 compares the average running times of three CD strategies to solve (4): Greedy (GCD), Randomized (RCD) and LGCD. LGCD consistently outperforms the two other strategies for both $T = 150L$ and $T = 750L$. For both LGCD and RCD, the iteration complexity is constant compared to the size of the signal, whereas the complexity of each GCD iteration scales linearly with T. This makes running time of GCD explode when T grows. Besides, as LGCD also prioritizes the more important coordinates, it is more efficient than uniform RCD. This explains Figure 3 and justifies the use of LGCD is the following experiments.

Figure 4 investigates the role of CD strategies in the distributed setting. The average runtimes of DICOD (with GCD on each worker), and DiCoDiLe-Z (with LGCD) are reported in Figure 4. We stick to a small scale problem in 1D to better highlight the limitation of the algorithms but scaling results for a larger and 2D problems are included in Figure C.1 and Figure C.2. DiCoDiLe-Z with LGCD scales sub-linearly with the number of workers used, whereas DICOD scales almost quadratically. However, DiCoDiLe-Z outperforms DICOD consistently, as the performance of GCD is poor for low number of workers using large sub-domains. The two coordinate selection strategies become equivalent when W reaches $T/4L$, as in this case DiCoDiLe has only one sub-domain $\mathcal{C}_1^{(w)}$.

Impact of 2D grid partitioning on images The performance of DiCoDiLe-Z on images are evaluated using the standard colored image Mandrill at full resolution (512×512). A dictionary $\boldsymbol{D}$ consists of $K = 25$ patches of size 16×16 extracted from the original image. We used $\lambda = 0.1\lambda_{\max}$ as a regularization parameter.

The reconstruction of the image with no soft-locks is shown in Figure 5. To ensure the algorithm finishes, workers are stopped either if they reach convergence or if $\|Z\|_\infty$ becomes larger than $\min_k \frac{50}{\|\boldsymbol{D}_k\|_\infty}$ locally. When a coefficient reaches this value, at least one pixel in the encoded patch has a value 50 larger than the maximal value for a pixel, so we can safely assume algorithm is diverging. The reconstructed image shows

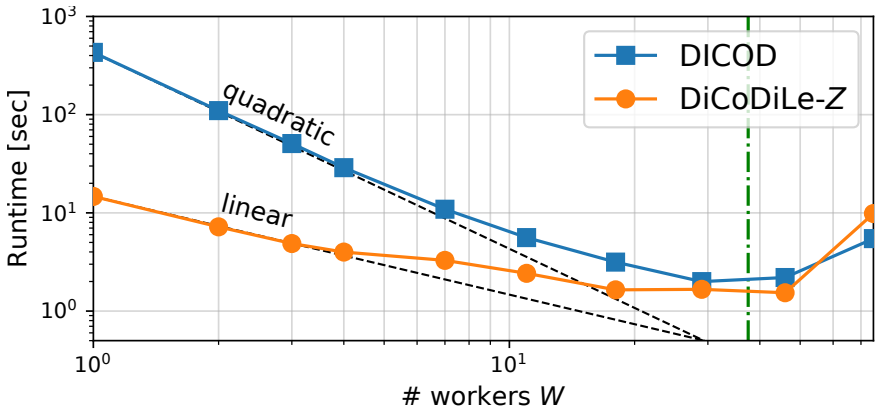

Figure 4: Scaling of DICOD (*orange*) and DiCoDiLe-Z (*blue*) with the number of workers W. DiCoDiLe-Z only scales sub-linearly while DICOD scales super-linearly. However DiCoDiLe-Z is more efficient that DICOD as the performance of the former with one worker is quite poor. When W reaches $T/4L$ (*dashed-green*), DiCoDiLe-Z and DICOD become equivalent as DiCoDiLe-Z only has one sub-domain in each worker.

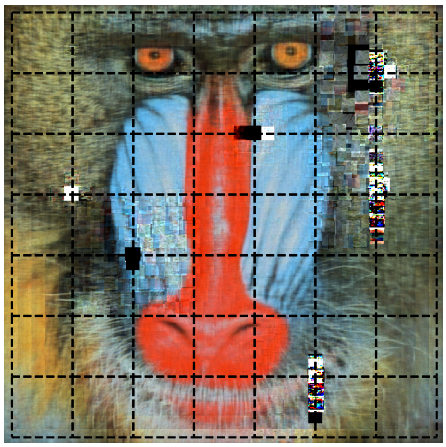

Figure 5: Reconstruction of the image Mandrill using DiCoDiLe with **no** soft-locks and 49 workers. The dashed lines show the partitions of the domain Ω. The algorithm diverges at the edges of some of the sub-domains due to interfering updates between more than two workers.

artifacts around the edges of some of the sub-domains $\mathcal{S}_w$, as the activation are being wrongly estimated in these locations. This visually demonstrates the need for controlling the interfering updates when more than two workers might interfere. When using DiCoDiLe-Z, the workers do not diverge anymore due to soft-locks.

To demonstrate the impact of the domain partitioning strategy on images, Figure 6 shows the average runtime for DiCoDiLe-Z using either a line of worker or a grid of workers. In this experiment $K = 5$ and the atom size is 8×8. Both strategies scale similarly in the regime with low number of workers. But when W reaches $T_1/3L_1$, the performances of DiCoDiLe-Z stops improving when using the unidirectional partitioning of Ω. Indeed in this case many update candidates are selected at the border of the sub-domains $\mathcal{S}_w$, therefore increasing the chance to reject an update. Moreover, the scaling of the linear split is limited to $W = T_1/2L_1 = 32$, whereas the 2D grid split can be used with W up to 1024 workers.

5.2 Application to real signals

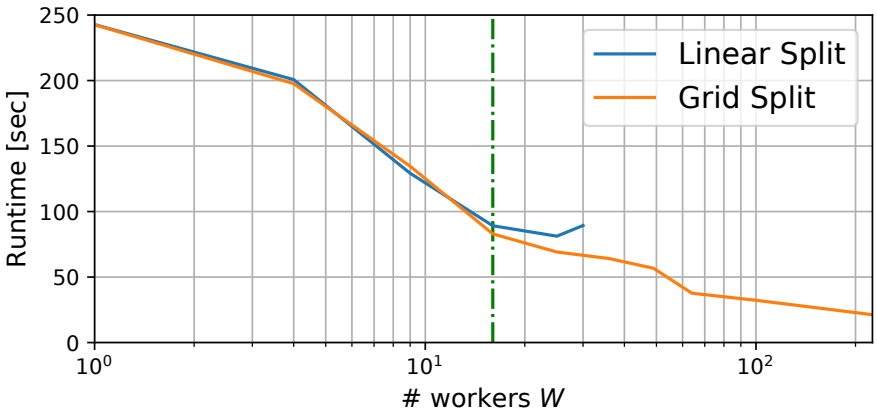

Figure 6: Scaling on 2D images of DiCoDiLe-Z with the number of workers for two partitioning strategies of Ω: (*blue*) Ω is split only along one direction, as in DICOD, (*orange*) Ω is split along both directions, on a grid. The running times are similar for low number of workers but the performances with the linear splits stop improving once W reaches the scaling limit $T_1/4L_1$ (*green*). With the grid of workers adapted to images, the performance improves further. The linear split stops when W reached T_1/L_1 because no more coordinate in $\mathcal{S}_w$ are independent from other neighbors.

Learning dictionary on Hubble Telescope images We present here results using CDL to learn patterns from an image of the Great Observatories Origins Deep Survey (GOODS) South field, acquired by the Hubble Space Telescope (Giavalisco and the GOODS Teams, 2004). We used the STScI-H-2016-39 image[2] with resolution 6000×3600, and used DiCoDiLe to learn 25 atoms of size 32×32 with regularization parameter $\lambda = 0.1\lambda_{\max}$. The learned patterns are presented in Figure 7. This unsupervised algorithm is able to highlight structured patterns in the image, such as small stars. Patterns 1 and 7 are very fuzzy. This is probably due to the presence of very large object in the foreground. As this algorithm is not scale invariant, the objects larger than our patterns are encoded using these low frequency atoms.

For comparison purposes, we run the parallel Consensus ADMM algorithm[3] proposed by Skau and Wohlberg (2018) on the same data with one node of our cluster but it failed to fit the computations in 250 GB of RAM. We restricted our problem to a smaller patch of size 512×512 extracted randomly in the Hubble image. Figure C.3 in supplementary material reports the evolution of the cost as a function of the time for both methods using $W = 36$. The other algorithm performs poorly in comparison to DiCoDiLe.

[2]The image is at www.hubblesite.org/image/3920/news/58-hubble-ultra-deep-field
[3]Code available at https://sporco.readthedocs.io/

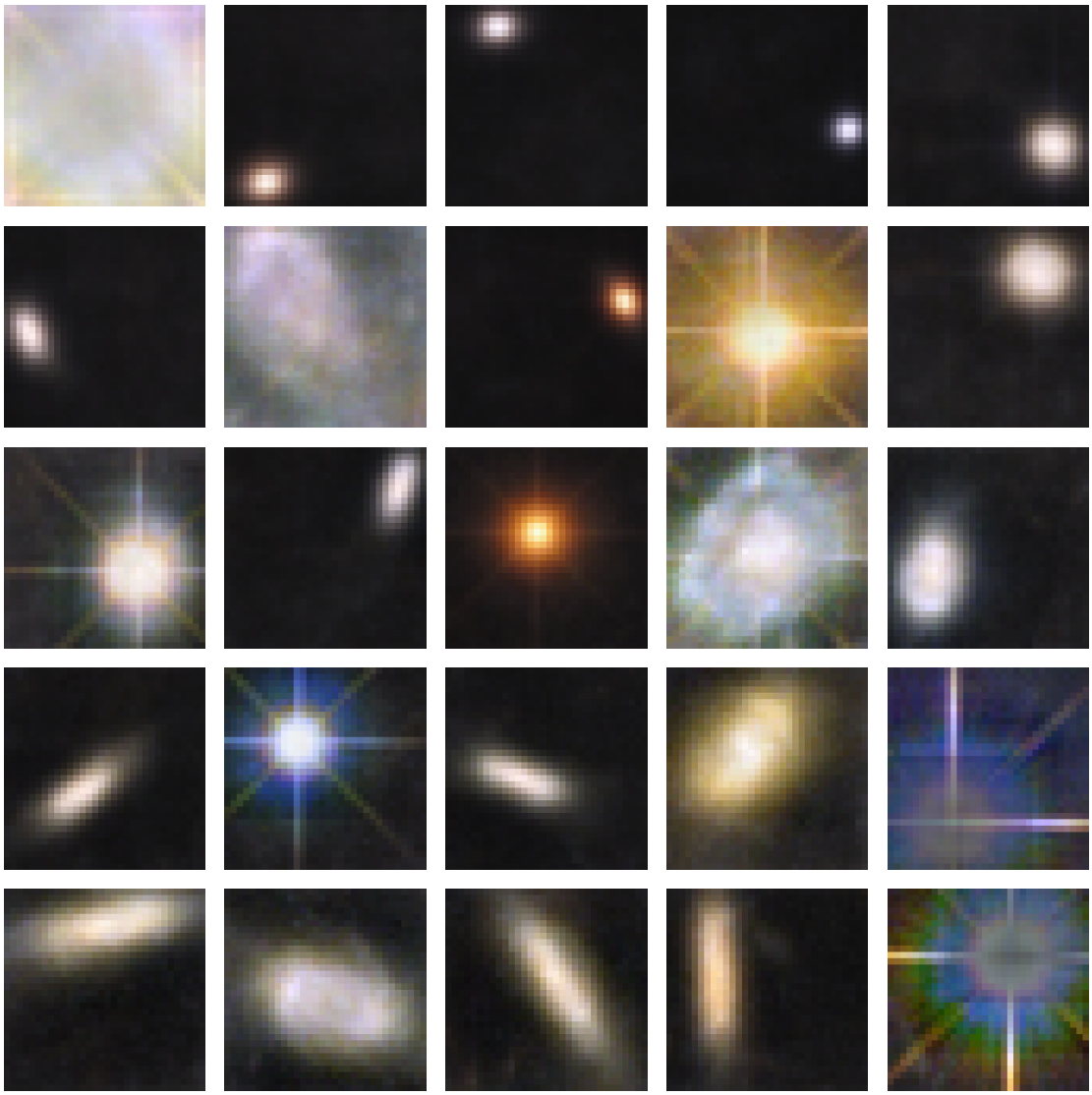

Figure 7: 25 atoms 32×32 learned from the Hubble Telescope images (`STScI-H-2016-39-a`) using DiCoDiLe with 400 workers and a regularization parameter $\lambda = 0.1\lambda_{\max}$. The atoms are sorted based on $\|Z_k\|_1$ from the top-left to the bottom-right. Most atoms displays a structure which corresponds to spatial objects. The atoms 1 and 7 have fuzzier structure because they are mainly used to encode larger scale objects.

6 Conclusion

This work introduces DiCoDiLe, an asynchronous distributed algorithm to solve the CDL for very large multidimensional data such as signals and images. The number of workers that can be used to solve the CSC scales linearly with the data size, allowing to adapt the computational resources to the problem dimensions. Using smart distributed pre-computation, the complexity of the dictionary update step is independent of the data size. Experiments show that it has good scaling properties and demonstrate that DiCoDiLe is able to learn patterns in astronomical images for large size which are not handled by other state-of-the-art parallel algorithms.

References

Michal Aharon, Michael Elad, and Alfred Bruckstein. K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation. *IEEE Transaction on Signal Processing*, 54(11):4311–4322, 2006.

Amir Beck and Marc Teboulle. A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

J. Bergstra, A. Courville, and Y. Bengio. The Statistical Inefficiency of Sparse Coding for Images (or, One Gabor to Rule them All). *arXiv e-prints*, September 2011.

Hilton Bristow, Anders Eriksson, and Simon Lucey. Fast convolutional sparse coding. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 391–398, Portland, OR, USA, 2013.

Rakesh Chalasani, Jose C. Principe, and Naveen Ramakrishnan. A fast proximal method for convolutional sparse coding. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1–5, Dallas, TX, USA, 2013.

Patrick L Combettes and Heinz H. Bauschke. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. Springer, 2011. ISBN 9788578110796. doi: 10.1017/CBO9781107415324.004.

Lisandro Dalcín, Rodrigo Paz, and Mario Storti. MPI for Python. *Journal of Parallel and Distributed Computing*, 65(9):1108–1115, 2005. ISSN 07437315. doi: 10.1016/j.jpdc.2005.03.010.

Pol del Aguila Pla and Joakim Jalden. Convolutional Group-Sparse Coding and Source Localization. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2776–2780, 2018. ISBN 978-1-5386-4658-8. doi: 10.1109/ICASSP.2018.8462235. URL https://ieeexplore.ieee.org/document/8462235/.

Tom Dupré la Tour, Thomas Moreau, Mainak Jas, and Alexandre Gramfort. Multivariate Convolutional Sparse Coding for Electromagnetic Brain Signals. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1–19, 2018.

Jerome Friedman, Trevor Hastie, Holger Höfling, and Robert Tibshirani. Pathwise coordinate optimization. *The Annals of Applied Statistics*, 1(2):302–332, 2007. doi: 10.1214/07-AOAS131.

Daniel Gabay and Bertrand Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & Mathematics with Applications*, 2(1):17–40, 1976. ISSN 08981221. doi: 10.1016/0898-1221(76)90003-1. URL http://www.sciencedirect.com/science/article/pii/0898122176900031.

M. Giavalisco and the GOODS Teams. The Great Observatories Origins Deep Survey: Initial Results From Optical and Near-Infrared Imaging. *The Astrophysical Journal Letters*, 600(2):L93, 2004. doi: 10.1086/379232. URL http://arxiv.org/abs/astro-ph/0309105%0Ahttp://dx.doi.org/10.1086/379232.

Roger Grosse, Rajat Raina, Helen Kwong, and Andrew Y Ng. Shift-Invariant Sparse Coding for Audio Classification. *Cortex*, 8:9, 2007.

Mainak Jas, Tom Dupré la Tour, Umut Şimşekli, and Alexandre Gramfort. Learning the Morphology of Brain Signals Using Alpha-Stable Convolutional Sparse Coding. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1–15, Long Beach, CA, USA, 2017. URL http://arxiv.org/abs/1705.08006.

Sai Praneeth Karimireddy, Anastasia Koloskova, Sebastian U. Stich, and Martin Jaggi. Efficient Greedy Coordinate Descent for Composite Problems. *preprint ArXiv*, 1810.06999, 2018. URL http://arxiv.org/abs/1810.06999.

Koray Kavukcuoglu, Pierre Sermanet, Y-lan Boureau, Karol Gregor, and Yann Lecun. Learning Convolutional Feature Hierarchies for Visual Recognition. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1090–1098, Vancouver, Canada, 2010.

Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online Learning for Matrix Factorization and Sparse Coding. *Journal of Machine Learning Research (JMLR)*, 11(1):19–60, 2010.

Thomas Moreau, Laurent Oudre, and Nicolas Vayatis. DICOD: Distributed Convolutional Sparse Coding. In *International Conference on Machine Learning (ICML)*, Stockohlm, Sweden, 2018.

Julie Nutini, Mark Schmidt, Issam H Laradji, Michael P. Friedlander, and Hoyt Koepke. Coordinate Descent Converges Faster with the Gauss-Southwell Rule Than Random Selection. In *International Conference on Machine Learning (ICML)*, pages 1632–1641, Lille, France, 2015.

Stanley Osher and Yingying Li. Coordinate descent optimization for ℓ_1 minimization with application to compressed sensing; a greedy algorithm. *Inverse Problems and Imaging*, 3(3):487–503, 2009.

Python Software Fundation. Python Language Reference, version 3.6, 2017.

Shai Shalev-Shwartz and A Tewari. Stochastic Methods for ℓ_1-regularized Loss Minimization. In *International Conference on Machine Learning (ICML)*, pages 929–936, Montreal, Canada, 2009.

Erik Skau and Brendt Wohlberg. A Fast Parallel Algorithm for Convolutional Sparse Coding. In *IEEE Image, Video, and Multidimensional Signal Processing Workshop (IVMSP)*, 2018. ISBN 9781538609514. doi: 10.1109/IVMSPW.2018.8448536.

Jean-Luc Starck, Michael Elad, and David L Donoho. Image Decomposition Via the Combination of Sparse Representation and a Variational Approach. *IEEE Trans.Image Process.*, 14(10):1570–1582, 2005. ISSN 1057-7149. doi: 10.1109/TIP.2005.852206.

Brendt Wohlberg. Efficient convolutional sparse coding. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7173–7177, Florence, Italy, 2014. ISBN 9781479928927. doi: 10.1109/ICASSP.2014.6854992.

Brendt Wohlberg. Efficient Algorithms for Convolutional Sparse Representations. *IEEE Transactions on Image Processing*, 25(1), 2016.

S. Wright and J. Nocedal. *Numerical optimization*. Science Springer, 1999. ISBN 0387987932. doi: 10.1007/BF01068601.

Florence Yellin, Benjamin D. Haeffele, and René Vidal. Blood cell detection and counting in holographic lens-free imaging by convolutional sparse dictionary learning and coding. In *IEEE International Symposium on Biomedical Imaging (ISBI)*, Melbourne, Australia, 2017.

A Computation for the cost updates

When a coordinate $Z_k[t]$ is updated to $Z'_k[t]$, the cost update is a simple function of $Z_k[t]$ and $Z'_k[t]$.

Proposition A.1. *If the coordinate (k_0, ω_0) is updated from value $Z_{k_0}[\omega_0]$ to value $Z'_{k_0}[\omega_0]$, then the change in the cost function $\Delta E_{k_0}[\omega_0]$ is given by:*

$$\Delta E_{k_0}[\omega_0] = \frac{\|\boldsymbol{D}_{k_0}\|_2^2}{2}(Z_{k_0}[\omega_0]^2 - Z'_{k_0}[\omega_0]^2) - \beta_{k_0}[\omega_0](Z_{k_0}[\omega_0] - Z'_{k_0}[\omega_0]) + \lambda(|Z_{k_0}[\omega_0]| - |Z'_{k_0}[\omega_0]|).$$

Proof. We denote $Z_k^{(1)}[\omega] = \begin{cases} Z'_{k_0}[\omega_0], & \text{if } (k,\omega) = (k_0,\omega_0) \\ Z_k[\omega], & \text{elsewhere} \end{cases}$. Let α_0 be the residual when coordinate (k_0, ω_0) of Z is set to 0. For $\omega \in \Omega$,

$$\alpha_{k_0}[\omega] = (X - Z * \boldsymbol{D})[\omega] + \boldsymbol{D}_{k_0}[\omega - \omega_0]Z_{k_0}[\omega_0] = (X - Z * \boldsymbol{D})[\omega] + Z_{k_0}[\omega_0](e_{\omega_0} * \boldsymbol{D}_{k_0})[\omega_0] \ .$$

We also have $\alpha_{k_0}[\omega] = (X - Z^{(1)} * \boldsymbol{D})[\omega] + \boldsymbol{D}_{k_0}[\omega - \omega_0]Z'_{k_0}[\omega_0]$. Then

$$\begin{aligned}
\Delta E_{k_0}[\omega_0] =& \frac{1}{2}\sum_{\omega\in\Omega}\left(X - Z * \boldsymbol{D}\right)^2[\omega] + \lambda\|Z\|_1 - \frac{1}{2}\sum_{\omega\in\Omega}\left(X - Z^{(1)} * \boldsymbol{D}\right)^2[\omega] + \lambda\|Z^{(1)}\|_1 \\
=& \frac{1}{2}\sum_{\omega\in\Omega}\left(\alpha_{k_0}[\omega] - \boldsymbol{D}_{k_0}[\omega - \omega_0]Z_{k_0}[\omega_0]\right)^2 - \frac{1}{2}\sum_{\omega\in\Omega}\left(\alpha_{k_0}[\omega] - \boldsymbol{D}_{k_0}[\omega - \omega_0]Z'_{k_0}[\omega_0]\right)^2 + \lambda(|Z_{k_0}[\omega_0]| - |Z'_{k_0}[\omega_0] \\
=& \frac{1}{2}\sum_{\omega\in\Omega}\boldsymbol{D}_{k_0}[\omega - \omega_0]^2(Z_{k_0}[\omega_0]^2 - Z'_{k_0}[\omega_0]^2) - \sum_{\omega\in\Omega}\alpha_{k_0}[\omega]\boldsymbol{D}_{k_0}[\omega - \omega_0](Z_{k_0}[\omega_0] - Z'_{k_0}[\omega_0]) + \lambda(|Z_{k_0}[\omega_0]| - |Z \\
=& \frac{\|\boldsymbol{D}_{k_0}\|_2^2}{2}(Z_{k_0}[\omega_0]^2 - Z'_{k_0}[\omega_0]^2) - \underbrace{(\alpha_{k_0} * \boldsymbol{D}^{\curvearrowleft}_{k_0})[\omega_0]}_{\beta_{k_0}[\omega_0]}(Z_{k_0}[\omega_0] - Z'_{k_0}[\omega_0]) + \lambda(|Z_{k_0}[\omega_0]| - |Z'_{k_0}[\omega_0]|)
\end{aligned}$$

This conclude our proof. □

Using this result, we can derive the optimal value $Z'_{k_0}[\omega_0]$ to update the coordinate (k_0, ω_0) as the solution of the following optimization problem:

$$Z'_{k_0}[\omega_0] = \arg\min_{y\in\mathbb{R}} e_{k_0,\omega_0}(u) \sim \arg\min_{u\in\mathbb{R}} \frac{\|\boldsymbol{D}_{k_0}\|_2^2}{2}\left(u - \beta_{k_0}[\omega_0]\right)^2 + \lambda|u| \ . \tag{A.1}$$

In the case where multiple coordinates (k_w, ω_w) are updated in the same iteration to values $Z'_{k_w}[\omega_w]$, we obtain the following cost variation.

Proposition A.2. *The update of the W coordinates $(k_w, \omega_w)_{w=1}^W$ with additive update $\Delta Z_{k_w}[\omega_w]$ changes the cost by:*

$$\Delta E = \overbrace{\sum_{i=1}^{W}}^{\textit{iterative steps}}\Delta E_w - \underbrace{\sum_{w\neq w'}(\boldsymbol{D}_{k_w} * \boldsymbol{D}^{\curvearrowleft}_{k_{w'}})[\omega_{w'} - \omega_w]\Delta Z_{k_w}[\omega_w]\Delta Z_{k_{w'}}[\omega_{w'}]}_{\textit{interference}},$$

Proof. We define $Z_k^{(1)}[t] = \begin{cases} Z_{k_w}[\omega_w], & \text{if } (k,\omega) = (k_w, \omega_w) \\ Z_k[t], & \text{otherwise} \end{cases}$.

Let

$$\alpha[\omega] = (X - Z * \boldsymbol{D})[t] + \sum_{w=1}^{K} \boldsymbol{D}_{k_w}[\omega - \omega_w] Z_{k_w}[\omega_w] = (X - Z^{(1)} * \boldsymbol{D})[t] + \sum_{w=1}^{K} \boldsymbol{D}_{k_w}[\omega - \omega_w] Z'_{k_w}[\omega_w] \ ,$$

and

$$\alpha_w[\omega] = (X - Z * \boldsymbol{D})[t] + \boldsymbol{D}_{k_w}[\omega - \omega_w] Z_{k_w}[\omega_w] = \alpha[\omega] - \sum_{w' \neq w} \boldsymbol{D}_{k_{w'}}[\omega - \omega_{w'}] Z_{k_{w'}}[\omega_{w'}] \ .$$

Then

$$
\begin{aligned}
\Delta E =& \frac{1}{2}\sum_{\omega\in\Omega}\Big(X[\omega]-Z*\boldsymbol{D}[\omega]\Big)^2+\lambda\|Z\|_1-\sum_{\omega\in\Omega}\Big(X[\omega]-Z^{(1)}*\boldsymbol{D}[\omega]\Big)^2+\lambda\|Z^{(1)}\|_1\\
=& \frac{1}{2}\sum_{\omega\in\Omega}\left(\alpha[\omega]-\sum_{w=1}^{W}\boldsymbol{D}_{k_w}[\omega-\omega_w]Z_{k_w}[\omega_w]\right)^2-\frac{1}{2}\sum_{\omega\in\Omega}\left(\alpha[\omega]-\sum_{w=1}^{W}\boldsymbol{D}_{k_w}[\omega-\omega_w]Z'_{k_w}[\omega_w]\right)^2\\
&+\sum_{w=1}^{W}\lambda(|Z_{k_w}[\omega_w]|-|Z'_{k_w}[\omega_w]|)\\
=& \frac{1}{2}\sum_{\omega\in\Omega}\sum_{w=1}^{W}\boldsymbol{D}_{k_w}[\omega-\omega_w]^2(Z_{k_w}[\omega_w]^2-Z'_{k_w}[\omega_w]^2)+\sum_{w=1}^{W}\lambda(|Z_{k_w}[\omega_w]|-|Z'_{k_w}[\omega_w]|)\\
&-\sum_{\omega\in\Omega}\Bigg[\sum_{w=1}^{W}\alpha[\omega]\boldsymbol{D}_{k_w}[\omega-\omega_w]\Delta Z_{k_w}[\omega_w]\\
&\qquad-\sum_{w\neq w'}\boldsymbol{D}_{k_w}[\omega-\omega_w]\boldsymbol{D}_{k_{w'}}[\omega-\omega_{w'}](Z_{k_w}[\omega_w]Z_{k_{w'}}[\omega_{w'}]-Z'_{k_w}[\omega_w]Z'_{k_{w'}}[\omega_{w'}])\Bigg]\\
=& \sum_{w=1}^{W}(\|\boldsymbol{D}_{k_w}\|_2^2(Z_{k_w}[\omega_w]^2-Z'_{k_w}[\omega_w]^2)-(\alpha_{k_w}*\boldsymbol{D}^{\curvearrowleft}_{k_w})[\omega_w]\Delta Z_{k_w}[\omega_w]+\lambda(|Z_{k_w}[\omega_w]|-|Z'_{k_w}[\omega_w]|)\\
&-\sum_{\omega\in\Omega}\Bigg[\sum_{w=1}^{W}\alpha_{k_w}[\omega]\boldsymbol{D}_{k_w}[\omega-\omega_w]\Delta Z_{k_w}[\omega_w]\\
&\qquad+\sum_{w\neq w'}\boldsymbol{D}_{k_w}[\omega-\omega_w]\boldsymbol{D}_{k_{w'}}[\omega-\omega_{w'}](\Delta Z_{k_w}[\omega_w]Z'_{k_{w'}}[\omega_{w'}]+\Delta Z_{k_{w'}}[\omega_{w'}]Z'_{k_w}[\omega_w])\\
&\qquad\quad-\boldsymbol{D}_{k_w}[\omega-\omega_w]\boldsymbol{D}_{k_{w'}}[\omega-\omega_{w'}](Z_{k_w}[\omega_w]Z_{k_{w'}}[\omega_{w'}]-Z'_{k_w}[\omega_w]Z'_{k_{w'}}[\omega_{w'}])\Bigg]\\
=& \sum_{w=1}^{W}\Delta E_{k_w}[\omega_w]-\sum_{w\neq w'}\Bigg[\left(\sum_{\omega\in\Omega}\boldsymbol{D}_{k_w}[\omega-\omega_w]\boldsymbol{D}_{k_{w'}}[\omega-\omega_{w'}]\right)\times\\
&\qquad\Big[Z_{k_w}[\omega_w]Z_{k_{w'}}[\omega_{w'}]-Z'_{k_w}[\omega_w]Z_{k_{w'}}[\omega_{w'}]-Z_{k_w}[\omega_w]Z'_{k_{w'}}[\omega_{w'}]+Z'_{k_{w'}}[\omega_{w'}]Z'_{k_w}[\omega_w]\Big]\Bigg]\\
=& \sum_{w=1}^{W}\Delta E_{k_w}[\omega_w]-\sum_{w\neq w'}\left(\sum_{\omega\in\Omega}\boldsymbol{D}_{k_w}[\omega]\boldsymbol{D}_{k_{w'}}[\omega+\omega_w-\omega_{w'}]\right)(Z_{k_w}[\omega_w]-Z'_{k_w}[\omega_w])(Z_{k_{w'}}[\omega_1]-Z'_{k_{w'}}[\omega_{w'}])\\
=& \sum_{w=1}^{W}\Delta E_{k_w}[\omega_w]-\sum_{w\neq w'}(\boldsymbol{D}_{k_{w'}}*\boldsymbol{D}^{\curvearrowleft}_{k_w})[\omega_w-\omega_{w'}]\Delta Z_{k_w}[\omega_w]\Delta Z_{k_{w'}}[\omega_{w'}]
\end{aligned}
$$

□

B Probability of a coordinate to be soft-locked

Proposition B.1. *The probability of an update candidate being accepted, if the updates are spread uniformly on Ω, is lower bounded by*

$$P(\omega \notin \boldsymbol{SL}) \geq \prod_{i=1}^{d}(1 - \frac{W_i L_i}{T_i})$$

Proof. Let $w \in [1, W]$ be a worker indice and $m \in [1, M]$ be the indice of the current sub-domain being considerd by the worker w. We denote $\omega = \operatorname{argmax}_{\omega \in \mathcal{C}_m^{(w)}} \max_k |\Delta Z_k[\omega]|$ the position of the update candidate chosen by w and $j = |\{w \ \textit{s.t.} \ \mathcal{S}_w \cap \mathcal{V}(\omega) = \emptyset\}|$ the number of workers that are impacted by this update. With the definition of the soft-lock, there exists at least one worker in $\{w \ \textit{s.t.} \ \mathcal{S}_w \cap \mathcal{V}(\omega) = \emptyset\}$ has a coordinate which is not soft-locked. As the updates are uniformly spread across the workers, we get the probability of ω being locked is $P(\omega \notin \mathbf{SL} \mid \omega) \geq \frac{1}{j}$.

Thus, the probability of an update candidate $(k, \omega) \in [1, K] \times \mathcal{S}_w$ to be accepted is

$$P(\omega \notin \mathbf{SL}) = \sum_{\omega \in \mathcal{S}_w} P(\omega \notin \mathbf{SL} \mid \omega) P(\omega) \tag{B.1}$$

$$\geq \frac{1}{|\mathcal{S}_w|} \sum_{\omega \in \mathcal{S}_w} \frac{1}{j_\omega} \tag{B.2}$$

$$\geq \frac{1}{|\mathcal{S}_w|} \prod_{i=1}^{d} (T_i - W_i L_i) \tag{B.3}$$

$$\geq \prod_{i=1}^{d} (1 - \frac{W_i L_i}{T_i}) \tag{B.4}$$

where (B.3) results from the summation on Ω. This step is clearer by looking at Figure B.1.

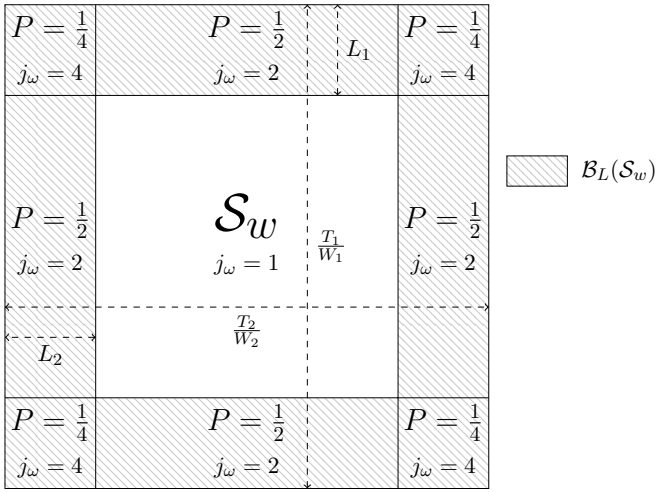

Figure B.1: Summary of the value of j_ω on a 2D example. We can easily see that the sum along each direction is $\frac{T_i}{W_i} - L_i$ on each workers. When multiplied by the number of workers W_i along each direction, we obtain (B.3).

□

C Extra experiments

Scaling of DiCoDiLe-Z for larger 1D signals Figure C.1 scaling properties of DICOD and DiCoDiLe-Z are consistent for larger signals. Indeed, in the case where $T = 750$, DiCoDiLe-Z scales linearly with the number of workers. We do not see the drop in performance, as in this experiments W does not reach the scaling limit $T/3L$. Also, DiCoDiLe-Z still outperforms DICOD for all regimes.

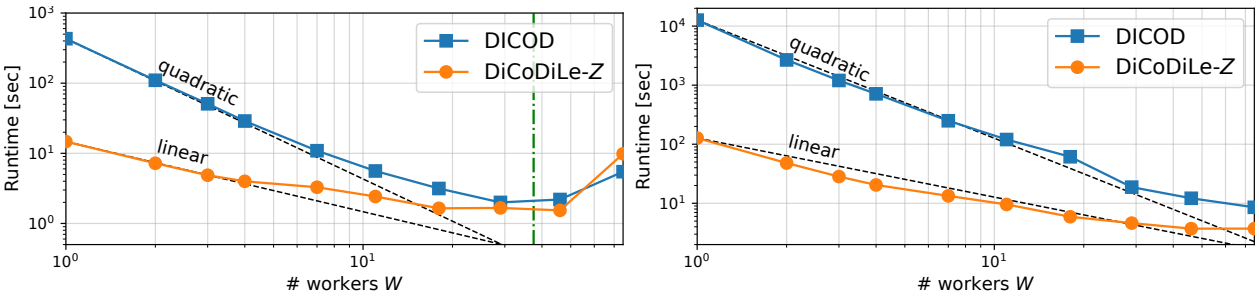

Figure C.1: Scaling of DICOD (*orange*) and DiCoDiLe-Z(*blue*) with the number of workers W. DiCoDiLe-Zonly scales sub-linearly while DICOD scales super-linearly. However DiCoDiLe-$Z$is more efficient that DICOD as the performance of the former with one worker are really bad. The green line denotes the number of cores where DiCoDiLe-$Z$and DICOD become the same as DiCoDiLe-Zonly has one sub-domain in each worker. The results are consistent for both small (*left*, $T = 150L$) and large (*right* $T = 750L$) signals sizes.

Scaling of DiCoDiLe-Z for 2D signals Figure C.2 illustrates the scaling performance of DiCoDiLe-Z on images. The average running times of DiCoDiLe-Z are reported as a function of the number of workers W for different regularization parameters λ and for greedy and locally greedy selection. As for the 1D CSC resolution, the locally greedy selection consistently outperforms the greedy selection. This is particularly the case when using low number of workers W. Once the size of the sub-domains becomes smaller, the performances of both coordinate selection become closer as both algorithms become equivalent. Also, the convergence of DiCoDiLe-Z is faster for large λ. This is expected, as in this case, the solution Z^* is sparser and less coordinates need to be updated.

C.1 Comparison with Skau and Wohlberg (2018)

Figure C.3 displays the evolution of the objective function (3) as a function of time for DiCoDiLe and the Consensus ADMM algorithm proposed by Skau and Wohlberg (2018). Both algorithms were run on a single node with 36 workers. We used the Hubble Space Telescope GOODS South image as input data. As the algorithm by Skau and Wohlberg (2018) raised a memory error with the full image, we used a random-patch of size 512×512 as input data. Due to the non-convexity of the problem, the algorithms converge to different local minima, even when initialized with the same dictionary. The Figure C.3 shows 5 runs of both algorithm, as well as the median curve, obtained by interpolation. We can see that DiCoDiLe outperforms the Consensus ADMM as it converges faster and to a better local minima. To ensure a fair comparison, the objective function is computed after projecting back the atoms of the dictionary $\boldsymbol{D}_k$ to the ℓ_2-ball by scaling it with $\|\boldsymbol{D}_k\|_2^2$. This implies that we also needed to scale Z by the inverse of the dictionary norm *i.e.* $\frac{1}{\|\boldsymbol{d}_k\|_2^2}$. This projection and scaling step is necessary for the solver as ADMM iterations do not guarantee that all iterates are feasible (*i.e.*, satisfy the constraints). This also explains the presence of several bumps in the evolution curve of the objective function.

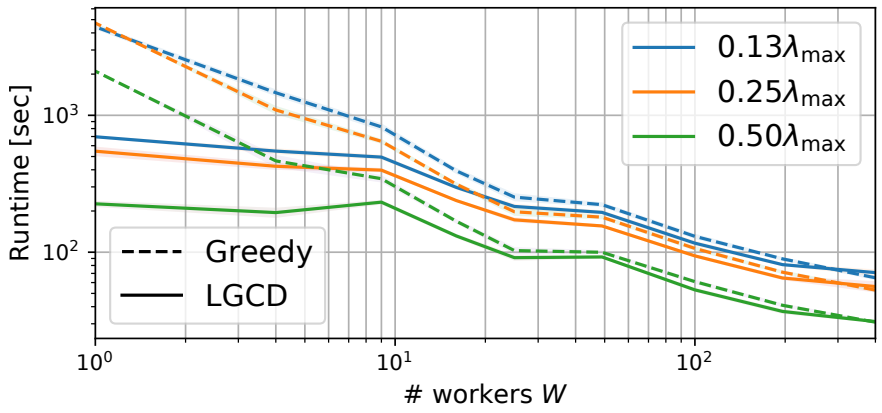

Figure C.2: Scaling of DiCoDiLe-Z with the number of workers W for different value of λ and for two strategies of coordinate selection: Greedy and Locally Greedy. The convergence is faster when λ is large because the activation becomes sparser and less coordinates need to be updated. Also, the locally greedy coordinate selection outperforms the greedy selection up to a certain point where the performances become similar as the sub-domain $\mathcal{S}_w$ becomes too small to be further partitioned with sub-domains of size Θ.

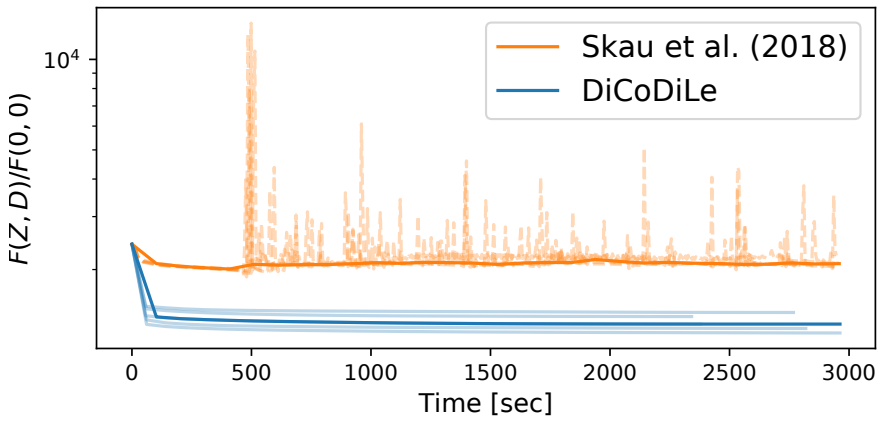

Figure C.3: Comparison of DiCoDiLe algorithm with Consensus ADMM dictionary learning proposed by Skau and Wohlberg (2018). Algorithms were run 5 times with 36 workers on a random-patch of size 512×512 taken from the Hubble Space Telescope GOODS South image. The solid lines denote the median curves obtained through interpolation.