DICOD: Distributed Convolutional Coordinate Descent for Convolutional Sparse Coding

Moreau Thomas [1] **Oudre Laurent** [2] **Vayatis Nicolas** [1]

Abstract

In this paper, we introduce DICOD, a convolutional sparse coding algorithm which builds shift invariant representations for long signals. This algorithm is designed to run in a distributed setting, with local message passing, making it communication efficient. It is based on coordinate descent and uses locally greedy updates which accelerate the resolution compared to greedy coordinate selection. We prove the convergence of this algorithm and highlight its computational speed-up which is super-linear in the number of cores used. We also provide empirical evidence for the acceleration properties of our algorithm compared to state-of-the-art methods.

1. Convolutional Representation for Long Signals

Sparse coding aims at building sparse linear representations of a data set based on a dictionary of basic elements called atoms. It has proven to be useful in many applications, ranging from EEG analysis to images and audio processing (Adler et al., 2013; Kavukcuoglu et al., 2010; Mairal et al., 2010; Grosse et al., 2007). Convolutional sparse coding is a specialization of this approach, focused on building sparse, shift-invariant representations of signals. Such representations present a major interest for applications like segmentation or classification as they separate the shape and the localization of patterns in a signal. This is typically the case for physiological signals which can be composed of recurrent patterns linked to specific behavior in the human body such as the characteristic heartbeat pattern in ECG recordings. Depending on the context, the dictionary can either be fixed analytically (*e.g.* wavelets, see Mallat 2008), or learned from the data (Bristow et al., 2013; Mairal et al., 2010).

Several algorithms have been proposed to solve the convolutional sparse coding. The Fast Iterative Soft-Thresholding Algorithm (FISTA) was adapted for convolutional problems in Chalasani et al. (2013) and uses proximal gradient descent to compute the representation. The Feature Sign Search (FSS), introduced in Grosse et al. (2007), solves at each step a quadratic subproblem for an active set of the estimated nonzero coordinates and the Fast Convolutional Sparse Coding (FCSC) of Bristow et al. (2013) is based on Alternating Direction Method of Multipliers (ADMM). Finally, the coordinate descent (CD) has been extended by Kavukcuoglu et al. (2010) to solve the convolutional sparse coding. This method greedily optimizes one coordinate at each iteration using fast local updates. We refer the reader to Wohlberg (2016) for a detailed presentation of these algorithms.

To our knowledge, there is no scalable version of these algorithms for long signals. This is a typical situation, for instance, in physiological signal processing where sensor information can be collected for a few hours with sampling frequencies ranging from 100 to 1000Hz. The existing algorithms for generic ℓ_1-regularized optimization can be accelerated by improving the computational complexity of each iteration. A first approach to improve the complexity of these algorithms is to estimate the non-zero coordinates of the optimal solution to reduce the dimension of the optimization space, using either screening (El Ghaoui et al., 2012; Fercoq et al., 2015) or active-set algorithms (Johnson & Guestrin, 2015). Another possibility is to develop parallel algorithms which compute multiple updates simultaneously. Recent studies have considered distributing coordinate descent algorithms for general ℓ_1-regularized minimization (Scherrer et al., 2012a;b; Bradley et al., 2011; Yu et al., 2012). These papers propose synchronous algorithms using either locks or synchronizing steps to ensure the convergence in general cases. You et al. (2016) derive an asynchronous distributed algorithm for the projected coordinate descent which uses centralized communication and finely tuned step size to ensure the convergence of their method.

In the present paper, we design a novel distributed algo-

[1]CMLA, ENS Cachan, CNRS, Université Paris-Saclay, 94235 Cachan, France [2]L2TI, Université Paris 13, 93430 Villetaneuse, France. Correspondence to: Moreau Thomas <thomas.moreau@cmla.ens-cachan.fr>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

rithm tailored for the convolutional problem which is based on coordinate descent, named Distributed Convolution Coordinate Descent (DICOD). DICOD is asynchronous and each process can run independently without locks or synchronization steps. This algorithm uses a local communication scheme to reduce the number messages between the processes and does not rely on external learning rates. We also prove that this algorithm scales super-linearly with the number of cores compared to the sequential CD, up to certain limitations.

In Section 2, we introduce the DICOD algorithm for the resolution of convolutional sparse coding. Then, we prove in Section 3 that DICOD converges to the optimal solution for a wide range of settings and we analyze its complexity. Finally, Section 4 presents numerical experiments that illustrate the benefits of the DICOD algorithm with respect to other state-of-the-art algorithms and validate our theoretical analysis.

2. Distributed Convolutional Coordinate Descent (DICOD)

Notations. The space of multivariate signals of length T in $\mathbb{R}^P$ is denoted by $\mathcal{X}_T^P$. For these signals, their value at time $t \in [\![0, T-1]\!]$ is denoted by $X[t] \in \mathbb{R}^P$ and for all $t \notin [\![0, T-1]\!]$, $X[t] = \mathbf{0}_P$. The indicator function of t_0 is denoted $\mathbf{1}_{t_0}$. For any signal $X \in \mathcal{X}_T^P$, the reversed signal is defined as $X^\intercal[t] = X[T-t]^\intercal$ and the d-norm is defined as $\|X\|_d = \left(\sum_{t=0}^{T-1} \|X[t]\|_d^d\right)^{1/d}$. Finally, for $L, W \in \mathbb{N}^*$, the convolution between $Z \in \mathcal{X}_L^1$ and $\boldsymbol{D} \in \mathcal{X}_W^P$ is a multivariate signal $Z * \boldsymbol{D} \in \mathcal{X}_T^P$ with $T = L+W-1$ such that for $t \in [\![0, T-1]\!]$,

$$(Z * \boldsymbol{D})[t] \triangleq \sum_{\tau=0}^{W-1} \langle Z[t-\tau], \boldsymbol{D}[\tau] \rangle \, .$$

This section reviews in Subsection 2.1 the convolutional sparse coding as an ℓ_1-regularized optimization problem and the coordinate descent algorithm to solve it. Then, Subsection 2.2 and Subsection 2.3 respectively introduce the Distributed Convolutional Coordinate Descent (DICOD) and the Locally Greedy Coordinate Descent (LGCD) algorithms to efficiently solve convolutional sparse coding for long signals. Finally, Subsection 2.4 discusses related work on ℓ_1-regularized coordinate descent algorithms.

2.1. Coordinate Descent for Convolutional Sparse Coding

Convolutional Sparse Coding. Consider the multivariate signal $X \in \mathcal{X}_T^P$. Let $\boldsymbol{D} = \left\{\boldsymbol{D}_k\right\}_{k=1}^K \subset \mathcal{X}_W^P$ be a set of K patterns with $W \ll T$ and $Z = \{Z_k\}_{k=1}^K \subset \mathcal{X}_L^1$ be a set of K activation signals with $L = T-W+1$. The convolutional sparse representation models a multivariate signal X as the sum of K convolutions between a local pattern $\boldsymbol{D}_k$ and an activation signal Z_k such that:

$$X[t] = \sum_{k=1}^K (Z_k * \boldsymbol{D}_k)[t] + \mathcal{E}[t], \qquad \forall t \in [\![0, T-1]\!], \quad (1)$$

with $\mathcal{E} \in \mathcal{X}_T^P$ representing an additive noise term. This model also assumes that the coding signals Z_k are sparse, in the sense that only few entries are nonzero in each signal. The sparsity property forces the representation to display localized patterns in the signal. Note that this model can be extended to higher order signals such as images by using the proper convolution operator. In this study, we focus on 1D-convolution for the sake of simplicity.

Given a dictionary of patterns $\boldsymbol{D}$, convolutional sparse coding aims to retrieve the sparse decomposition Z^* associated to the signal X by solving the following ℓ_1-regularized optimization problem

$$Z^* = \underset{Z=(Z_1,\dots Z_K)}{\operatorname{argmin}} \; E(Z) \, , \qquad \text{where} \quad (2)$$

$$E(Z) \triangleq \frac{1}{2} \left\| X - \sum_{k=1}^K Z_k * \boldsymbol{D}_k \right\|_2^2 + \lambda \sum_{k=1}^K \left\| Z_k \right\|_1 , \quad (3)$$

for a given regularization parameter $\lambda > 0$. The problem formulation (2) can be interpreted as a special case of the LASSO problem with a band circulant matrix. Therefore, classical optimization techniques designed for LASSO can easily be applied to solve it with the same convergence guarantees. Kavukcuoglu et al. (2010) adapted the coordinate descent to efficiently solve the convolutional sparse coding.

Convolutional Coordinate Descent. The coordinate descent is a method which updates one coordinate at each iteration. This type of optimization algorithms is efficient for sparse optimization problem since few coordinates need to be updated to find the optimal solution and the greedy selection of updated coordinates is a good strategy to achieve fast convergence to the optimal point. Algorithm 1 summarizes the greedy convolutional coordinate descent.

The method proposed by Kavukcuoglu et al. (2010) iteratively updates at each iteration one coordinate (k_0, t_0) of the coding signal Z to its optimal value $Z'_{k_0}[t_0]$ when all other coordinates are fixed. A closed form solution exists to compute the value $Z'_{k_0}[t_0]$ for the update,

$$Z'_{k_0}[t_0] = \frac{1}{\|\boldsymbol{D}_{k_0}\|_2^2} \text{Sh}(\beta_{k_0}[t_0], \lambda), \quad (4)$$

with the soft thresholding operator defined as

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

Algorithm 1 Greedy Coordinate Descent

Input: $\boldsymbol{D}, X$, parameter $\epsilon > 0$
$\mathcal{C} = [\![1, K]\!] \times [\![0, L-1]\!]$
Initialization: $\forall (k,t) \in \mathcal{C}$,
$Z_k[t] = 0, \;\; \beta_k[t] = \left(\boldsymbol{D}_k^{\scriptscriptstyle\lnot} * X\right)[t]$
repeat
  $\forall (k,t) \in \mathcal{C}, \; Z'_k[t] = \frac{1}{\|\boldsymbol{D}_k\|_2^2} \text{Sh}(\beta_k[t], \lambda)$,
  Choose $(k_0, t_0) = \arg\max_{(k,t)\in\mathcal{C}} |\Delta Z_k[t]|$
  Update β using (5) and $Z_{k_0}[t_0] \leftarrow Z'_{k_0}[t_0]$
until $|\Delta Z_{k_0}[t_0]| < \epsilon$

Algorithm 2 DICOD$_M$

Input: $\boldsymbol{D}, X$, parameter $\epsilon > 0$
In parallel for $m = 1 \cdots M$
For all (k,t) in $\mathcal{C}_m$, initialize $\beta_k[t]$ and $Z_k[t]$
repeat
  Receive messages and update β with (5)
  $\forall (k,t) \in \mathcal{C}_m$, compute $Z'_k[t]$ with (4)
  Choose $(k_0, t_0) = \arg\max_{(k,t)\in\mathcal{C}_m} |\Delta Z_k[t]|$
  Update β with (5) and $Z_{k_0}[t_0] \leftarrow Z'_{k_0}[t_0]$
  if $t_0 - mL_M < W$ **then**
    Send $(k_0, t_0, \Delta Z_{k_0}[t_0])$ to core $m-1$
  if $(m+1)L_M - t_0 < W$ **then**
    Send $(k_0, t_0, \Delta Z_{k_0}[t_0])$ to core $m+1$
until global convergence $\|\Delta Z\|_\infty < \epsilon$

and an auxiliary variable $\beta \in \mathcal{X}_L^K$ defined as

$$\beta_k[t] = \left(\boldsymbol{D}_k^{\scriptscriptstyle\lnot} * \left(X - \sum_{k'=1}^{K} Z_{k'} * \boldsymbol{D}_{k'} + Z_k[t] e_t * \boldsymbol{D}_k\right)\right)[t] \;,$$

where e_t is a dirac with value 1 in t and 0 elsewhere. Note that $\beta_k[t]$ is simply the residual when $Z_k[t]$ is equal to 0.

The success of this algorithm highly depends on the efficiency in computing this coordinate update. For problem (2), Kavukcuoglu et al. (2010) show that if at iteration q, the coordinate (k_0, t_0) of $Z^{(q)}$ is updated to the value $Z'_{k_0}[t_0]$, then it is possible to compute $\beta^{(q+1)}$ from $\beta^{(q)}$ using

$$\beta_k^{(q+1)}[t] = \beta_k^{(q)}[t] - (\boldsymbol{D}_k^{\scriptscriptstyle\lnot} * \boldsymbol{D}_{k_0})[t - t_0] \Delta Z_{k_0}^{(q)}[t_0], \quad (5)$$

for all $(k,t) \neq (k_0, t_0)$. For all $t \notin [\![-W+1, W-1]\!]$, $(\boldsymbol{D}_k^{\scriptscriptstyle\lnot} * \boldsymbol{D}_{k_0})[t]$ is zero. Thus, only $\mathcal{O}(KW)$ operations are needed to maintain β up-to-date with the current estimate Z. In the following,

$$\Delta E_{k_0}[t_0] = E(Z^{(q)}) - E(Z^{(q+1)})$$

denotes the cost variation obtained when the coordinate (k_0, t_0) is replaced by its optimal value $Z'_{k_0}[t_0]$.

The selection of the updated coordinate (k_0, t_0) can follow different strategies. Cyclic updates (Friedman et al., 2007) and random updates (Shalev-Shwartz & Tewari, 2009) are efficient strategies as they have a $\mathcal{O}(1)$ computational complexity. Osher & Li (2009) propose to select the coordinate greedily to maximize the cost reduction of the update. In this case, the coordinate is chosen as the one with the largest difference $\max_{(k,t)} \left|\Delta Z_k[t]\right|$ between its current value $Z_k[t]$ and the value $Z'_k[t]$ with

$$\Delta Z_k[t] = Z_k[t] - Z'_k[t] \quad (6)$$

This strategy is computationally more expensive, with a cost of $\mathcal{O}(KT)$ but it has a better convergence rate (Nutini et al., 2015). In this paper, we focus on the greedy approach as it aims to get the largest gain from each update. Moreover, as the updates in the greedy scheme are more complex to compute, distributing them provides a larger speedup compared to other strategies.

The procedure is run until $\max_{k,t} |\Delta Z_k[t]|$ becomes smaller than a specified tolerance parameter ϵ.

2.2. Distributed Convolutional Coordinate Descent (DICOD)

For convolutional sparse coding, the coordinate descent updates are only weakly dependent as it is shown in (5). It is thus natural to parallelize it for this problem.

DICOD. Algorithm 2 describes the steps of DICOD with M workers. Each worker $m \in [\![1, M]\!]$ is in charge of updating the coordinates of a segment $\mathcal{C}_m$ of length $L_M = L/M$ defined by:

$$\mathcal{C}_m = \left\{ (k,t) \;;\; k \in [\![1,K]\!],\; t \in \left[\!\left[(m-1)L_M,\; mL_M - 1 \right]\!\right] \right\} .$$

The local updates are performed in parallel for all the cores using the greedy coordinate descent introduced in Subsection 2.1. When a core m updates the coeffient (k_0, t_0) such that $t_0 \in [\![(m-1)L_M + W, mL_M - W]\!]$, the updated coordinates of β are all contained in $\mathcal{C}_m$ and there is no need to update β on the other cores. In these cases, the update is equivalent to a sequential update. When $t_0 \in [\![mL_M - W, mL_M]\!]$ $\left(\text{resp. } t_0 \in [\![(m-1)L_M, (m-1)L_M + W]\!]\right)$, some of the coordinates of β in core $m+1$ (resp. $m-1$) need to be updated and the update is not local anymore. This can be done

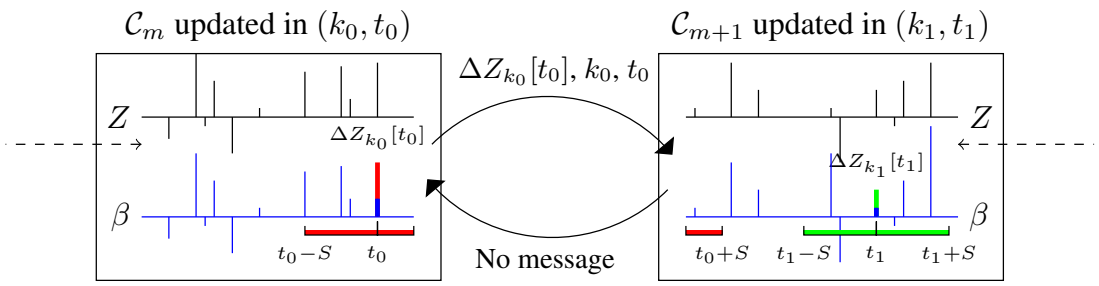

Figure 1. Communication process in DICOD for two cores $\mathcal{C}_m$ and $\mathcal{C}_{m+1}$. (*red*) The process needs to send a message to its neighbor as it updates a coordinate with t_0 located near the border of the core's segment, in the interference zone. (*green*) The update in t_1 is independent of other cores.

by sending the position of updated coordinate (k_0, t_0), and the value of the update $\Delta Z_{k_0}[t_0]$ to the neighboring core. Figure 1 illustrates this communication process. Inter-processes communications are very limited in DICOD. One node communicates with its neighbors only when it updates coordinates close to the extremity of its segment. When the size of the segment is reasonably large compared to the size of the patterns, only a small part of the iterations needs to send messages. We cannot apply the stopping criterion of CD in each worker of DICOD, as this criterion might not be reached globally. The updates in the neighbor cores can break this criterion. To avoid this issue, the convergence is considered to be reached once all the cores achieve this criterion simultaneously. Workers that reach this state locally are paused, waiting for incoming communication or for the global convergence to be reached.

The key point that allows distributing the convolutional coordinate descent algorithm is that the solutions on time segments that are not overlapping are only weakly dependent. Equation (5) shows that a local change has impact on a segment of length $2W - 1$ centered around the updated coordinate. Thus, if two coordinates which are far enough were updated simultaneously, the resulting point Z is the same as if these two coordinates had been updated sequentially. By splitting the signal into continuous segments over multiple cores, coordinates can be updated independently on each core up to certain limits.

Interferences. When two coordinates (k_0, t_0) and (k_1, t_1) are updated by two neighboring cores simultaneously, the updates might not be independent and cannot be considered to be sequential. The local version of β used for the second update does not account for the first update. We say that the updates are *interfering*. The cost reduction resulting from these two updates is denoted $\Delta E_{k_0,k_1}[t_0, t_1]$ and simple computations, detailed in Proposition A.2, show that

$$\Delta E_{k_0,k_1}[t_0, t_1] = \overbrace{\Delta E_{k_0}[t_0] + \Delta E_{k_1}[t_1]}^{\text{iterative steps}} \underbrace{- (\boldsymbol{D}_{k_1}^{\lnot} * \boldsymbol{D}_{k_0})[t_1 - t_0] \Delta Z_{k_0}[t_0] \Delta Z_{k_1}[t_1]}_{\text{interference}}, \tag{7}$$

If $|t_1 - t_0| \geq W$, then $(\boldsymbol{D}_{k_1}^{\lnot} * \boldsymbol{D}_{k_0})[t_1 - t_0] = 0$ and the updates can be considered to be sequential as the interference term is zero. When $|t_1 - t_0| < W$, the interference term does not vanish but Section 3 shows that under mild assumption, this term can be controlled and it does not make the algorithm diverge.

2.3. Locally Greedy Coordinate Descent (LGCD)

The theoretical analysis in Theorem 3 shows that DICOD provides a super-linear acceleration compared to the greedy coordinate descent. This result is supported with the numerical experiment presented in Figure 4. The super-linear speed up results from a double acceleration, provided by the parallelization of the updates – we update M coordinates at each iteration – and also by the reduction of the complexity of each iteration. Indeed, each core computes greedy updates with linear in complexity on $1/M$-th of the signal. This super-linear speed-up means that running DICOD sequentially will still provide a speed-up compared to the greedy coordinate descent algorithm.

Algorithm 3 presents LGCD. This algorithm is similar to a sequential version of DICOD. At each step, one segment $\mathcal{C}_m$ is selected uniformly at random between the M segments. The greedy coordinate descent algorithm is applied locally on this segment. This update is only locally greedy and maximizes

$$(k_0, t_0) = \underset{(k,t) \in \mathcal{C}_m}{\operatorname{argmax}} |\Delta Z_k[t]|$$

This coordinate is then updated to its optimal value $Z'_{k_0}[t_0]$. In this case, there is no interference as the segments are not updated simultaneously.

Algorithm 3 Locally greedy coordinate descent LGCD_M

Input: $\boldsymbol{D}, X$, parameter $\epsilon > 0$, number of segments M
Initialize $\beta_k[t]$ and $Z_k[t]$ for all (k, t) in $\mathcal{C}$
Initialize $dZ_m = +\infty$ for $m \in [\![1, M]\!]$
repeat
  Randomly select $m \in [\![1, M]\!]$
  $\forall (k, t) \in \mathcal{C}_m$, compute $Z'_k[t]$ with (4)
  Choose $(k_0, t_0) = \underset{(k,t)\in\mathcal{C}_m}{\text{argmax}} |\Delta Z_k[t]|$
  Update β with (5)
  Update estimate $Z_{k_0}[t_0]^{(q+1)} \leftarrow Z'_{k_0}[t_0]$
  Update $dZ_m = \left|Z_{k_0}[t_0]^{(q+1)} - Z'_{k_0}[t_0]\right|$
until $\|dZ\|_\infty < \epsilon$ and $\|\Delta Z\|_\infty < \epsilon$

Note that if $M = T$, this algorithm becomes very close to the randomized coordinate descent. The coordinate is selected greedily only between the K different channels of the signal Z at the selected time. So the selection of M depends on a tradeoff between the randomized coordinate descent and the greedy coordinate descent.

2.4. Discussion

This algorithm differs from the existing paradigm to distribute CD (Scherrer et al., 2012a;b; Bradley et al., 2011; Yu et al., 2012; You et al., 2016) as it does not rely on centralized communication. Indeed, other parallel coordinate descent algorithms rely on a parameter server, which is an extra worker that holds the current value of Z. As the size of the problem and the number of nodes grow, the communication cost can rapidly become an issue with this kind of centralized communication. The natural workload split proposed with DICOD allows for more efficient interactions between the workers and reduces the need for inter-node communications. Moreover, to prevent the interferences breaking the convergence, existing algorithms rely either on synchronous updates (Bradley et al., 2011; Yu et al., 2012) or on reduced step size in the updates (You et al., 2016; Scherrer et al., 2012a). In both case, they are less efficient than our asynchronous greedy algorithm that can leverage the convolutional structure of the problem to use both large updates and independent processes without external parameters.

As seen in the introduction, another way to improve the computational complexity of sparse coding algorithms is to estimate the non-zero coordinates of the optimal solution in order to reduce the dimension of the optimization space. As this research direction is orthogonal to the parallelization of the coordinate descent, it would be possible to combine our algorithm with either screening (El Ghaoui et al., 2012; Fercoq et al., 2015) or active-set methods (Johnson & Guestrin, 2015). The evaluation of the performances of our algorithm with these strategies is left for future work.

3. Properties of DICOD

Convergence of DICOD. The magnitude of the interference is related to the value of the cross-correlation between dictionary elements, as shown in Proposition 1. Thus, when the interferences have low probability and small magnitude, the distributed algorithm behaves as if the updates were applied sequentially, resulting in a large acceleration compared to the sequential CD algorithm.

Proposition 1. *For concurrent updates for coordinates (k_0, t_0) and (k_1, t_1) of a sparse code Z, the cost update $\Delta E_{k_0 k_1}[t_0, t_1]$ is lower bounded by*

$$\Delta E_{k_0,k_1}[t_0, t_1] \geq \Delta E_{k_0}[t_0] + \Delta E_{k_1}[t_1] - 2\frac{(\boldsymbol{D}^{\tilde{}}_{k_0} * \boldsymbol{D}_{k_1})[t_0 - t_1]}{\|\boldsymbol{D}_{k_0}\|_2 \|\boldsymbol{D}_{k_1}\|_2} \sqrt{\Delta E_{k_0}[t_0] \Delta E_{k_1}[t_1]}. \quad (8)$$

The proof of this proposition is given in Appendix C.1. It relies on the $\|\boldsymbol{D}_k\|_2^2$-strong convexity of (4), which gives $|\Delta Z_k[t]| \leq \frac{\sqrt{2\Delta E_k[t](Z)}}{\|\boldsymbol{D}_k\|_2}$ for all Z. Using this inequality with (7) yields the result.

This proposition controls the magnitude of the interference using the cost reduction associated to a single update. When the correlations between the different elements of the dictionary are small enough, the interfering update does not increase the cost function. The updates are less efficient but do not worsen the current estimate. Using this control on the interferences, we can prove the convergence of DICOD.

Theorem 2. *Consider the following hypotheses,*

H1. *For all (k_0,t_0),(k_1,t_1) such that for all $t_0 \neq t_1$,*

$$\left|\frac{(\boldsymbol{D}^{\tilde{}}_{k_0} * \boldsymbol{D}_{k_1})[t_0 - t_1]}{\|\boldsymbol{D}_{k_0}\|_2 \|\boldsymbol{D}_{k_1}\|_2}\right| < 1 .$$

H2. *There exists $A \in \mathbb{N}^*$ such that all cores $m \in [\![1, M]\!]$ are updated at least once between iteration i and $i + A$ if the solution is not locally optimal.*

H3. *The delay in communication between the processes is inferior to the update time.*

Under (H1)-(H2)-(H3), the DICOD algorithm converges to the optimal solution Z^* of (2).

Assumption (H1) is satisfied as long as the dictionary elements are not replicated in shifted positions in the dictionary. It ensures that the cost is updated in the right direction

at each step. This assumption can be linked to the shifted mutual coherence introduced in Papyan et al. (2016).

Hypothesis **(H2)** ensures that all coordinates are updated regularly if they are not already optimal. This analysis is not valid when one of the cores fails. As only one core is responsible for the update of a local segment, if a worker fails, this segment cannot be updated anymore and thus the algorithm will not converge to the optimal solution.

Finally, under **(H3)**, an interference only results from one update on each core. Multiple interferences occur when a core updates multiple coordinates in the border of its segment before receiving the communication from other processes border updates. When $T \gg W$, the probability of multiple interference is low and this hypothesis can be relaxed if the updates are not concentrated on the borders.

Proof sketch for Theorem 2. The full proof can be found in Appendix C.2. The main argument in proving the convergence is to show that most of the updates can be considered sequentially and that the remaining updates do not increase the cost of the current point. By **(H3)**, for a given iteration, a core can interfere with at most one other core. Thus, without loss of generality, we can consider that at each step q, the variation of the cost E is either $\Delta E_{k_0}[t_0](Z^{(q)})$ or $\Delta E_{k_0 k_1}[t_0, t_1](Z^{(q)})$, for some $(k_0, t_0), (k_1, t_1) \in [\![1, K]\!] \times [\![0, T-1]\!]$. Proposition 1 and **(H1)** proves that $\Delta E_{k_0 k_1}[t_0, t_1](Z^{(q)}) \geq 0$. For a single update $\Delta E_{k_0}[t_0](Z^{(q)})$, the update is equivalent to a sequential update in CD, with the coordinate chosen randomly between the best in each segments. Thus, $\Delta E_{k_0}[t_0](Z^{(q)}) > 0$ and the convergence is eventually proved using results from Osher & Li (2009). □

Speedup of DICOD. We denote $S_{cd}(M)$ the speedup of DICOD compared to the sequential greedy CD. This quantifies the number of iterations that can be run by DICOD during one iteration of CD.

Theorem 3. *Let $\alpha = \frac{W}{T}$ and $M \in \mathbb{N}^*$. If $\alpha M < \frac{1}{4}$ and if the non-zero coordinates of the sparse code are distributed uniformly in time, the expected speedup $\mathbb{E}[S_{cd}(M)]$ is lower bounded by*

$$\mathbb{E}[S_{cd}(M)] \geq M^2(1 - 2\alpha^2 M^2 \left(1 + 2\alpha^2 M^2\right)^{\frac{M}{2}-1}) .$$

This result can be simplified when the interference probability $(\alpha M)^2$ is small.

Corollary 4. *The expected speedup $\mathbb{E}[S_{cd}(M)]$ when $M\alpha \to 0$ is such that*

$$\mathbb{E}[S_{cd}(M)] \underset{\alpha \to 0}{\gtrsim} M^2(1 - 2\alpha^2 M^2 + \mathcal{O}(\alpha^4 M^4)) .$$

Proof sketch for Theorem 3. The full proof can be found in Appendix D. There are two aspects involved in DICOD speedup: the computational complexity and the acceleration due to the parallel updates. As stated in Subsection 2.1, the complexity of each iteration for CD is linear with the length of the input signal T. In DICOD, each core runs on a segment of size $\frac{T}{M}$. This accelerates the execution of individual updates by a factor M. Moreover, all the cores compute their update simultaneously. The updates without interference are equivalent to sequential updates. Interfering updates happen with probability $\left(M\alpha\right)^2$ and do not increase the cost. Thus, one iteration of DICOD with N_i interferences provides a cost variation equivalent to $M - 2N_i$ iterations using sequential CD and, in expectation, it is equivalent to $M - 2\mathbb{E}[N_i]$ iterations of DICOD. The probability of interference depends on the ratio between the length of the segments used for each core and the size of the dictionary. If all the updates are spread uniformly on each segment, the probability of interference between 2 neighboring cores is $\left(\frac{MW}{T}\right)^2$. The expected number of interference $\mathbb{E}[N_i]$ can be upper bounded using this probability and this yields the desired result. □

The overall speedup of DICOD is super-linear compared to sequential greedy CD for the regime where $\alpha M \ll 1$. It is almost quadratic for small M but as M grows, there is a sharp transition that significantly deteriorates the acceleration provided by DICOD. Section 4 empirically highlights this behavior. For a given α, it is possible to approximate the optimal number of cores M to solve convolutional sparse coding problems.

Note that this super-linear speed up is due to the fact that CD is inefficient for long signals, as its iterations are computationally too expensive to be competitive with the other methods. The fact that we have a super-linear speed-up means that running DICOD sequentially will provide an acceleration compared to CD (see Subsection 2.3). For the sequential run of DICOD, called LGCD, we have a linear speed-up in comparison to CD, when M is small enough. Indeed, the iteration cost is divided by M as we only need to find the maximal update on a local segment of size $\frac{T}{M}$. When increasing M over $\frac{T}{W}$, the iteration cost does not decrease anymore as updating β costs $\mathcal{O}\left(KW\right)$ and finding the best coordinate has the same complexity.

4. Numerical Results

All the numerical experiments are run on five Linux machines with 16 to 24 Intel Xeon 2.70 GHz processors and at least 64 GB of RAM on local network. We use a combination of Python, C++ and the OpenMPI 1.6 for the algorithms implementation. The code to reproduce the figures

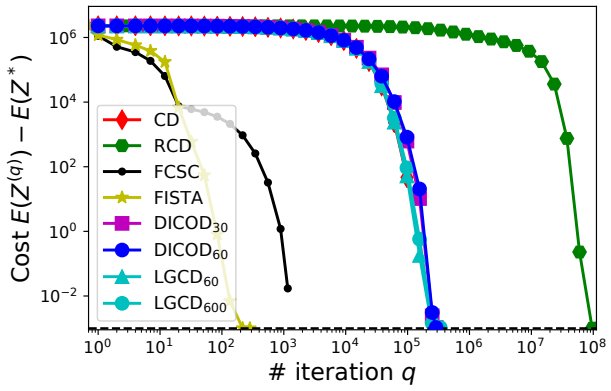

Figure 2. Evolution of the loss function for DICOD, LGCD, CD, FCSC and FISTA while solving sparse coding for a signal generated with default parameters relatively to the number of iterations.

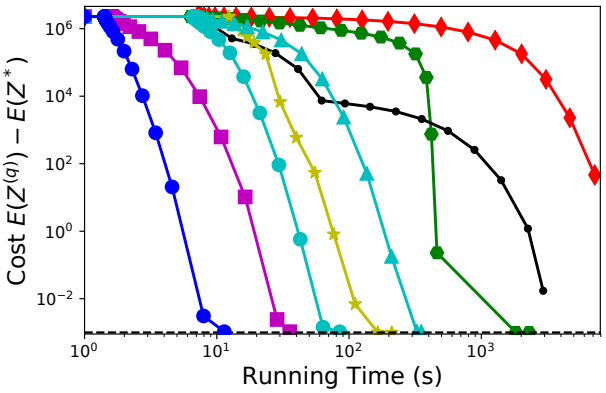

Figure 3. Evolution of the loss function for DICOD, LGCD, CD, FCSC and FISTA while solving sparse coding for a signal generated with default parameters, relatively to time. This highlights the speed of the algorithm on the given problem.

is available online[1]. The run time denotes the time for the system to run the full algorithm pipeline, from cold start and includes for instance the time to start the sub-processes. The convergence refers to the variation of the cost with the number of iterations and the speed to the variation of the cost relative to time.

Long convolutional Sparse Coding Signals. To further validate our algorithm, we generate signals and test the performances of DICOD compared to state-of-the-art methods proposed to solve convolutional sparse coding. We generate a signal X of length T in $\mathbb{R}^P$ following the model described in (1). The K dictionary atoms $\boldsymbol{D}_k$ of length W are drawn as a generic dictionary. First, each entry is sampled from a Gaussian distribution. Then, each pattern is normalized such that $\|\boldsymbol{D}_k\|_2 = 1$. The sparse code entries are drawn from a Bernoulli-Gaussian distribution with Bernoulli parameter $\rho = 0.007$, mean 0 and standard variation $\sigma = 10$. The noise term $\mathcal{E}$ is chosen as a Gaussian white noise with variance 1. The default values for the dimensions are set to $W = 200$, $K = 25$, $P = 7$, $T = 600 \times W$ and we used $\lambda = 1$.

Algorithms Comparison. DICOD is compared to the main state-of-the-art optimization algorithms for convolutional sparse coding: Fast Convolutional Sparse Coding (FCSC) from Bristow et al. (2013), Fast Iterative Soft Thresholding Algorithm (FISTA) using Fourier domain computation as described in Wohlberg (2016), the greedy convolutional coordinate descent (CD, Kavukcuoglu et al. 2010) and the randomized coordinate descent (RCD, Nesterov 2010). All the specific parameters for these algorithms are fixed based on the authors' recommendations.

[1]Code available at github.com/tomMoral/dicod

DICOD$_M$ denotes the DICOD algorithm run using M cores. We also include LGCD$_M$, for $M \in \{60, 600\}$, the sequential run of the DICOD algorithm using M segments, as described in Algorithm 3.

Figure 2 shows that the evolution of the performances of LGCD relatively to the iterations are very close to the performances of CD. The difference between these two algorithms is that the updates are only locally greedy in LGCD. As there is little difference visible between the two curves, this means that in this case, the computed updates are essentially the same. The differences are larger for LGCD$_{600}$, as the choice of coordinates are more localized in this case. The performance of DICOD$_{60}$ and DICOD$_{30}$ are also close to the iteration-wise performances of CD and LGCD. The small differences between DICOD and LGCD result from the iterations where there are interferences. Indeed, if two iterations interfere, the cost does not decrease as much as if the iterations were done sequentially. Thus, it requires more steps to reach the same accuracy with DICOD$_{60}$ than with LGCD and with DICOD$_{30}$, as there are more interferences when the number of cores M increases. This explains the discrepancy in the decrease of the cost around the iteration 10^5. However, the number of extra steps required is quite low compared to the total number of steps and the performances are mostly not affected by the interferences. The performances of RCD in terms of iterations are much slower than the greedy methods. Indeed, as only a few coordinates are useful, it takes many iterations to draw them randomly. In comparison, the greedy methods are focused on the coordinates which largely divert from their optimal value, and are thus most likely to be important. Another observation is that the performance in term of number of iterations of the global methods FCSC and FISTA are much better than the methods based on local updates. As

each iteration can update all the coordinates for FISTA, the number of iterations needed to reach the optimal solution is indeed smaller than for CD, where only one coordinate is updated at a time.

In Figure 3, the speed of theses algorithms can be observed. Even though it needs many more iterations to converge, the randomized coordinate descent is faster than the greedy coordinate descent. Indeed, for very long signals, the iteration complexity of greedy CD is prohibitive. However, using the locally greedy updates, with LGCD_{60} and LGCD_{600}, the greedy algorithm can be made more efficient. LGCD_{600} is also faster than the other state-of-the-art algorithms FISTA and FCSC. The choice of $M = 600$ is a good tradeoff for LGCD as it means that the segments are of the size of the dictionary W. With this choice for $M = \frac{T}{W}$, the computational complexity of choosing a coordinate is $\mathcal{O}\left(KW\right)$ and the complexity of maintaining β is also $\mathcal{O}\left(KW\right)$. Thus, the iterations of this algorithm have the same complexity as RCD but are more efficient.

The distributed algorithm DICOD is faster compared to all the other sequential algorithms and the speed up increases with the number of cores. Also, DICOD has a shorter initialization time compared to the other algorithms. The first point in each curve indicates the time taken by the initialization. For all the other methods, the computations for constants – necessary to accelerate the iterations – have a computational cost equivalent to the on of the gradient evaluation. As the segments of signal in DICOD are smaller, the initialization time is also reduced. This shows that the overhead of starting the cores is balanced by the reduction of the initial computation for long signals. For shorter signals, we have observed that the initialization time is of the same order as the other methods. The spawning overhead is indeed constant whereas the constants are cheaper to compute for small signals.

Speedup Evaluation. Figure 4 displays the speedup of DICOD as a function of the number of cores. We used 10 generated problems for 2 signal lengths $T = 150 \cdot W$ and $T = 750 \cdot W$ with $W = 200$ and we solved them using DICOD_M with a number of cores M ranging from 1 to 75. The blue dots display the average running time for a given number of workers. For both setups, the speedup is super-linear up to the point where $M\alpha = \frac{1}{2}$. For small M the speedup is very close to quadratic and a sharp transition occurs as the number of cores grows. The vertical solid green line indicates the approximate position of the maximal speedup given in Corollary 4 and the dashed lined is the expected theoretical run time derived from the same expression. The transition after the maximum is very sharp. This approximation of the speedup for small values of $M\alpha$ is close to the experimental speedup observed with

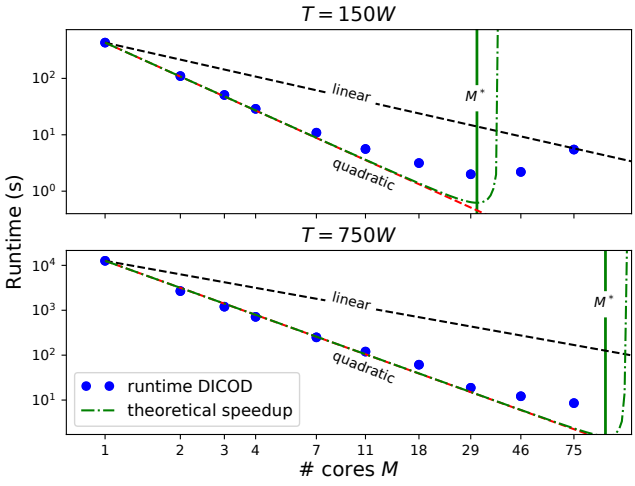

Figure 4. Speedup of DICOD as a function of the number of processes used, average over 10 run on different generated signals. This highlights a sharp transition between a regime of quadratic speedups and the regime where the interference are slowing down drastically the convergence.

DICOD. The computed optimal value of M^* is close to the optimal number of cores in these two examples.

5. Conclusion

In this work, we introduced an asynchronous distributed algorithm that is able to speed up the resolution of the Convolutional Sparse Coding problem for long signals. This algorithm is guaranteed to converge to the optimal solution of (2) and scales superlinearly with the number of cores used to distribute it. These claims are supported by numerical experiments highlighting the performances of DICOD compared to other state-of-the-art methods. Our proofs rely extensively on the use of one dimensional convolutions. In this setting, a process m only has two neighbors $m-1$ and $m+1$. This ensures that there is no high order interferences between the updates. Our analysis does not apply straightforwardly to distributed computation using square patches of images as the higher order interferences are more complicated to handle. A way to apply our algorithm with these guarantees to images is to split the signals along only one direction, to avoid higher order interferences. The extension of our results to the grid splitting of images is an interesting direction for future work.

References

Adler, A., Elad, M., Hel-Or, Y., and Rivlin, E. Sparse Coding with Anomaly Detection. In *IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 22 – 25, Southampton, United Kingdom, 2013.

Bradley, J. K., Kyrola, A., Bickson, D., and Guestrin, C. Parallel Coordinate Descent for ℓ_1-Regularized Loss Minimization. In *International Conference on Machine Learning (ICML)*, pp. 321–328, Bellevue, WA, USA, 2011.

Bristow, H., Eriksson, A., and Lucey, S. Fast convolutional sparse coding. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 391–398, Portland, OR, USA, 2013.

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

El Ghaoui, L., Viallon, V., and Rabbani, T. Safe feature elimination for the LASSO and sparse supervised learning problems. *Journal of Machine Learning Research (JMLR)*, 8(4):667–698, 2012.

Fercoq, O., Gramfort, A., and Salmon, J. Mind the duality gap : safer rules for the Lasso. In *International Conference on Machine Learning (ICML)*, pp. 333–342, Lille, France, 2015.

Friedman, J., Hastie, T., Höfling, H., and Tibshirani, R. Pathwise coordinate optimization. *The Annals of Applied Statistics*, 1(2):302–332, 2007.

Grosse, R., Raina, R., Kwong, H., and Ng, A. Y. Shift-Invariant Sparse Coding for Audio Classification. *Cortex*, 8:9, 2007.

Johnson, T. and Guestrin, C. Blitz: A Principled Meta-Algorithm for Scaling Sparse Optimization. In *International Conference on Machine Learning (ICML)*, pp. 1171–1179, Lille, France, 2015.

Kavukcuoglu, K., Sermanet, P., Boureau, Y.-l., Gregor, K., and Lecun, Y. Learning Convolutional Feature Hierarchies for Visual Recognition. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1090–1098, Vancouver, Canada, 2010.

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

Mallat, S. *A Wavelet Tour of Signal Processing*. Academic press, 2008.

Nesterov, Y. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2010.

Nutini, J., Schmidt, M., Laradji, I. H., Friedlander, M. P., and Koepke, H. Coordinate Descent Converges Faster with the Gauss-Southwell Rule Than Random Selection. In *International Conference on Machine Learning (ICML)*, pp. 1632–1641, Lille, France, 2015.

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

Papyan, V., Sulam, J., and Elad, M. Working Locally Thinking Globally - Part II: Theoretical Guarantees for Convolutional Sparse Coding. *arXiv preprint*, arXiv:1607(02009), 2016.

Scherrer, C., Halappanavar, M., Tewari, A., and Haglin, D. Scaling Up Coordinate Descent Algorithms for Large ℓ_1 Regularization Problems. Technical report, Pacific Northwest National Laboratory (PNNL), 2012a.

Scherrer, C., Tewari, A., Halappanavar, M., and Haglin, D. J. Feature Clustering for Accelerating Parallel Coordinate Descent. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 28–36, South Lake Tahoe, United States, 2012b.

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

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

You, Y., Lian, X., Liu, J., Yu, H.-F., Dhillon, I. S., Demmel, J., and Hsieh, C.-J. Asynchronous Parallel Greedy Coordinate Descent. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 4682–4690, Barcelona, Spain, 2016.

Yu, H. F., Hsieh, C. J., Si, S., and Dhillon, I. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In *IEEE International Conference on Data Mining (ICDM)*, pp. 765–774, Brussels, Belgium, 2012.