Learning Fast Sparsifying Transforms

Citation for published version:

Digital Object Identifier (DOI):
10.1109/TSP.2017.2712120

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
IEEE Transactions on Signal Processing

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
Abstract—Given a dataset, the task of learning a transform that allows sparse representations of the data bears the name of dictionary learning. In many applications, these learned dictionaries represent the data much better than the static well-known transforms (Fourier, Hadamard etc.). The main downside of learned transforms is that they lack structure and therefore they are not computationally efficient, unlike their classical counterparts. These pose several difficulties especially when using power limited hardware such as mobile devices, therefore discouraging the application of sparsity techniques in such scenarios. In this paper we construct orthogonal and non-orthogonal dictionaries that are factorized as a product of a few basic transformations. In the orthogonal case, we solve exactly the dictionary update problem for one basic transformation, which can be viewed as a generalized Givens rotation, and then propose to construct orthogonal dictionaries that are a product of these transformations, guaranteeing their fast manipulation. We also propose a method to construct fast square but non-orthogonal dictionaries that are factorized as a product of few transforms that can be viewed as a further generalization of Givens rotations to the non-orthogonal setting. We show how the proposed transforms can balance very well data representation performance and computational complexity. We also compare with classical fast and learned general and orthogonal transforms.

I. INTRODUCTION

Dictionary learning methods [1] represent a well-known class of algorithms that have seen many applications in signal processing [2], image processing [3], wireless communications [4] and machine learning [5]. The key idea of this approach is not to use an off-the-shelf transform like the Fourier, Hadamard or wavelet but to learn a new transform, often called an overcomplete dictionary, for a particular task (like coding and classification) from the data itself. While the dictionary learning problem is NP-hard [6] in general, it has been extensively studied and several good algorithms to tackle it exist. Alternating minimization methods like the method of optimal directions (MOD) [7], K–SVD [8], [9] and direct optimization [10] have been shown to work well in practice and also enjoy some theoretical performance guarantees. While learning a dictionary we need to construct two objects: the dictionary and the representation of the data in the dictionary.

One problem that arises in general when using learned dictionaries is the fact that they lack any structure. This is to be compared with the previously mentioned off-the-shelf transforms that have a rich structure. This is reflected in their low computational complexity, i.e., they can be applied directly using $O(n \log n)$ computations for example [11]. Our goal in this paper is to provide a solution to the problem of constructing fast transforms, based upon the structure of Givens rotations, learned from training data.

We first choose to study orthogonal structures since sparse reconstruction is computationally cheaper in such a dictionary: we project the data onto the column space of the dictionary and keep the largest $s$ coefficients in magnitude to obtain the provable best $s$-term approximation. Working in an $n$ dimensional feature space, this operation has complexity $O(n^2)$. In a general non-orthogonal (and even overcomplete) dictionary, special non-linear reconstruction methods such as $\ell_1$ minimization [12], greedy approaches like orthogonal matching pursuit (OMP) [13] or variational Bayesian algorithms like approximate message passing (AMP) [14] need to be applied. Aside from the fact that in general these methods cannot guarantee to produce best $s$-term approximations they are also computationally expensive. For example, the classical OMP has complexity $O(sn^2)$ [15] and, assuming that we are looking for sparse approximations with $s \ll n$, it is in general computationally cheaper than $\ell_1$ optimization. Therefore, considering a square orthogonal dictionary is a first step in the direction of constructing a fast transform. For the analysis dictionary, recent work based on transform learning [16], [17] has been proposed. Still, notice that computing sparse representations in such a dictionary has complexity $O(n^2)$ and therefore, our goal of constructing a fast transform cannot be reached with just a general orthogonal dictionary. We make the case that our fundamental goal is to actually build a structured orthogonal dictionary such that matrix-vector multiplications with this dictionary can be achieved with less than $O(n^2)$ operations, preferably $O(n \log n)$. This connects our paper to previous work on approximating orthogonal (and symmetric) matrices [18] such that matrix-vector multiplications are computationally efficient.

When we talk about “learning fast sparsifying transforms” we do not refer to the efficient learning procedures (although the proposed learning methods have polynomial complexity) but we refer to the transforms themselves, i.e., once we have the transform, the computational complexity of using it is low, preferably $O(n \log n)$ to perform matrix-vector multiplication.

Previous work [19], [20], [21], [22], [23], [24], [25] in the literature has already proposed various structured dictionaries to cope with the high computational complexity of learned transforms. Previous work also dealt with the construction of structured orthogonal dictionaries. Specifically, [26] proposed to build an orthogonal dictionary composed of a product of a few Householder reflectors. In this fashion, the computational complexity of the dictionary is controlled and a trade-off between representation performance and computational complexity is shown.
Learned dictionaries with low computational complexity can bridge the gap between the classical transforms that are preferred especially in power limited hardware (or battery operated devices) and the overcomplete, computationally cumbersome, learned dictionaries that provide state-of-the-art performance in many machine learning tasks. The contribution of this paper is two fold.

First, we consider the problem of constructing an orthogonal dictionary as a product of a given number of generalized Givens rotations. We start by showing the optimum solution to the dictionary learning problem when the dictionary is a single generalized Givens rotation and then move to expand on this result and propose an algorithm that sequentially builds a product of generalized Givens rotations to act as a dictionary for sparse representations. Each step of the algorithm solves exactly the proposed optimization problem and therefore we can guarantee that it monotonically converges to a local minimum. We show numerically that the fast dictionaries proposed in this paper outperform those based on Householder reflectors [26] in terms of representation error, for the same computational complexity.

Second, based on a structure similar to the generalized Givens rotation we then propose a learning method that constructs square, non-orthogonal, computationally efficient dictionaries. In order to construct the dictionary we again solve exactly a series of optimization problems. Unfortunately we cannot prove the monotonic convergence of the algorithm since the sparse reconstruction step, based in this paper on OMP, cannot guarantee in general a monotonic reduction in the objective function. Still, we are able to show that these fast non-orthogonal transforms perform very well, better than their orthogonal counterparts.

In the results section we compare the proposed methods among each other and to previously proposed dictionary learning methods in the literature. We show that the methods proposed in this paper provide a clear trade-off between representation performance and computational complexity. Interestingly, we are able to provide numerical examples where the proposed fast orthogonal dictionaries have higher computational efficiency and provide better representation performance than the well-known discrete cosine transform (DCT), the transform at the heart of the jpeg compression standard [27].

II. A BRIEF DESCRIPTION OF DICTIONARY LEARNING OPTIMIZATION PROBLEMS

Given a real dataset \( Y \in \mathbb{R}^{n \times N} \) and sparsity level \( s \), the general dictionary learning problem is to produce the factorization \( Y \approx DX \) given by the optimization problem:

\[
\begin{align*}
\text{minimize}_{D, X} & \quad \|Y - DX\|_F^2 \\
\text{subject to} & \quad \|x_i\|_0 \leq s, \quad 1 \leq i \leq N \\
& \quad \|d_j\|_2 = 1, \quad 1 \leq j \leq n,
\end{align*}
\]  

where the objective function describes the Frobenius norm representation error achieved by the square dictionary \( D \in \mathbb{R}^{n \times n} \) with the sparse representations \( X \in \mathbb{R}^{n \times N} \) whose columns are subject to the \( \ell_0 \) pseudo-norm \( \|x_i\|_0 \) (the number of non-zero elements of columns \( x_i \)). To avoid trivial solutions, the dimensions obey \( s \ll n \ll N \). Several algorithms that work very well in practice exist [7] [8] [15] to solve this factorization problem. Their approach, and the one we also adopt in this paper, is to keep the dictionary fixed and update the representations and then reverse the roles by updating the dictionary with the representations fixed. This alternating minimization approach proves to work very well experimentally [7], [8] and allows some theoretical insights [28].

In this paper we also consider the dictionary learning problem (1) with an orthogonal dictionary \( Q \in \mathbb{R}^{n \times n} \) [29] [30] [31] [32]. The orthogonal dictionary learning problem (which we call in this paper Q-DLA) [33] is formulated as:

\[
\begin{align*}
\text{minimize}_{Q, X} & \quad \|Y - QX\|_F^2 \\
\text{subject to} & \quad \|x_i\|_0 \leq s, \quad 1 \leq i \leq N.
\end{align*}
\]  

Since the dictionary \( Q \) is orthogonal, the construction of \( X \) no longer involves \( \ell_1 \) [12], OMP [13] or AMP [14] approaches as in (1), but reduces to \( X = \mathcal{T}_s(Q^T Y) \), where \( \mathcal{T}_s() \) is an operator that given an input vector zeroes all entries except the largest \( s \) in magnitude and given an input matrix applies the same operation on each column in turn. To solve (2) for variable \( Q \) and fixed \( X \), a problem also known as the orthogonal Procrustes problem [34], a closed form solution \( Q = UV^T \) is given by the singular value decomposition of \( YX^T = U \Sigma V^T \).

III. A BUILDING BLOCK FOR FAST TRANSFORMS

For indices \( (i, j) \), \( j > i \) and variables \( p, q, r, t \in \mathbb{R} \) let us define the basic transform, which we call an R-transform:

\[
\begin{bmatrix}
I_{i-1} & p & r \\
p & I_{j-i-1} & q \\
r & q & I_{n-j}
\end{bmatrix} \in \mathbb{R}^{n \times n},
\]  

where we have denoted \( I_i \) as the identity matrix of size \( i \). For simplicity, we denote the non-zero part of \( R_{ij} \) as

\[
\tilde{R}_{ij} = \begin{bmatrix} p & r \\ q & t \end{bmatrix} \in \mathbb{R}^{2 \times 2}.
\]  

A right side multiplication between a R-transform and a matrix \( X \in \mathbb{R}^{n \times N} \) operates only rows \( i \) and \( j \) as

\[
R_{ij}X = [x_1 \ldots px_i + rx_j \ldots \ldots qx_i + tx_j \ldots x_n]^T,
\]  

where \( x_i^T \) is the \( i^{th} \) row of \( X \). The number of operations needed for this task is only \( 6N \). Left and right multiplications with a R-transform (or its transpose) are therefore computationally efficient. We use this matrix structure as a basic building block for the transforms learned in this paper.

Remark 1. Every matrix \( D \in \mathbb{R}^{n \times n} \) can be written as a product of at most \( \left\lceil n^2 - \frac{n}{2} + 1 \right\rceil \) R-transforms. Therefore, we can consider the R-transforms as fundamental building blocks for all square transforms \( D \).

Proof. Consider the singular value decomposition \( D = USV^T \). Each \( U \) and \( V \) can be factored as a product of \( \binom{n}{2} \) Givens rotations [35] which are all in fact constrained.
R-transforms (with \( p = t = c \) and \( r = q = d \) for some given \( c \) and \( d \) such that \( c^2 + d^2 = 1 \) and a diagonal matrix containing only \( \{\pm 1\} \) entries. While the diagonal \( \Sigma \) can be factored as a product of \( \binom{d}{2} \) diagonal R-transforms.

In this paper we will be interested to use \( R_{ij} \) in least squares problems with the objective function as:
\[
\|Y - R_{ij}X\|_F^2 = \|Y\|_F^2 + \|X\|_F^2 - 2\text{tr}(Z) - \left\|\begin{bmatrix} y_I^T \\ y_J^T \end{bmatrix} \right\|_F^2 - 2\text{tr}(Z_{(i,j)}) + \left\|\begin{bmatrix} y_I^T \\ y_J^T \end{bmatrix} - R_{ij} \right\|_F^2. \tag{6}
\]
For simplicity of exposition we have defined
\[
Z = YX^T, Z_{(i,j)} = \begin{bmatrix} Z_{ii} & Z_{ij} \\ Z_{ji} & Z_{jj} \end{bmatrix} \in \mathbb{R}^{2 \times 2}, Z_{ij} = y_I^T x_j, \tag{7}
\]
where \( y_I^T \) and \( x_j^T \) are the \( i^{th} \) rows of \( Y \) and \( X \), respectively.

We now introduce learning methods to create computationally efficient orthogonal and non-orthogonal dictionaries.

IV. A METHOD FOR DESIGNING FAST ORTHOGONAL TRANSFORMS: \( G_m \)-DLA

In this section we propose a method called \( G_m \)-DLA to learn orthogonal dictionaries that are factorized as a product of \( m \) G-transforms (constrained R-transforms).

A. An overview of G-transforms

We call \( G_{ij} \) a G-transform, an orthogonal constrained R-transform (3) parameterized only by \( c, d \in \mathbb{R} \) with \( c^2 + d^2 = 1 \), and the indices \((i, j)\), \( i \neq j \) such that the non-zero part of \( G_{ij} \), corresponding to (4), is given by
\[
\tilde{G}_{ij} \in \left\{ \begin{bmatrix} c & d \\ -d & c \end{bmatrix}, \begin{bmatrix} c & -d \\ d & c \end{bmatrix} \right\}. \tag{8}
\]
Classically, a Givens rotation is a matrix as in (3) with \( \tilde{G}_{ij} = \begin{bmatrix} c & d \\ -d & c \end{bmatrix} \) such that \( \det(G_{ij}) = 1 \), i.e., proper rotation matrices are orthogonal matrices with determinant one. These rotations are important since any orthogonal dictionary of size \( n \times n \) can be factorized in a product of \( \binom{n}{2} \) Givens rotations [35]. In this paper, since we are interested in the computational complexity of these structures, we allow both options in (8) that fully characterize all \( 2 \times 2 \) real orthogonal matrices – these structures are discussed in [36, Chapter 2.1]. With \( \tilde{G}_{ij} = \begin{bmatrix} c & d \\ -d & c \end{bmatrix} \) the G-transform in (3) is in fact a Householder reflector \( G_{ij} = I - 2g_{ij}g_{ij}^T \) where \( g_{ij} \in \mathbb{R}^n, \|g_{ij}\|_2 = 1 \), has all entries equal to zero except for the \( i^{th} \) and \( j^{th} \) entries that are \( \sqrt{0.5(1-c)} \) and \( -\text{sign}(d)\sqrt{0.5(1+c)} \), respectively - one might call this a “Givens reflector” to respectively distinguish its sparse structure. Givens rotations have been previously used in matrix factorization applications [37, 38].

B. One G-transform as a dictionary

Consider now the dictionary learning problem in (2). Let us keep the sparse representations \( X \) fixed and consider a single G-transform as a dictionary. We reach the following
\[
\begin{align*}
\text{minimize} & \quad \|Y - G_{ij}X\|_F^2. \tag{9}
\end{align*}
\]
When indices \((i, j)\) are fixed, the problem reduces to constructing \( \tilde{G}_{ij} \), a constrained two dimensional optimization problem. To select the indices \((i, j)\), among the \( \binom{n}{2} \) possibilities, an appropriate strategy needs to be defined. We detail next how to deal with these two problems to provide an overall solution for (9).

To solve (9) for the fixed coordinates \((i, j)\) we reach the optimization problem
\[
\begin{align*}
\text{minimize} & \quad \tilde{G}_{ij}, \tilde{G}_{ij}^T \quad \tilde{G}_{ij} = G_{ij}, \tilde{G}_{ij}^T = 1 \\
& \quad \|y_I^T - \tilde{G}_{ij} x_j^T\|_F^2 = \|y_I^T\|_F^2 + \|x_j^T\|_F^2 - 2\text{tr}(Z_{(i,j)}) - 2\|Z_{(i,j)}\|_F^2. \tag{10}
\end{align*}
\]
This is a two dimensional Procrustes problem [34] whose optimum solution is \( \tilde{G}_{ij} = UV^T \) where \( Z_{(i,j)} = U\Sigma V^T \). It has been shown in [26] that the reduction in the objective function of (10) when considering an orthogonal dictionary \( G_{ij} \) given by the Procrustes solution is
\[
\begin{align*}
& \quad \|y_I^T - \tilde{G}_{ij} x_j^T\|_F^2 = \|y_I^T\|_F^2 + \|x_j^T\|_F^2 - 2\text{tr}(Z_{(i,j)}) - 2\|Z_{(i,j)}\|_F^2 \\
& = \|y_I^T\|_F^2 + \|x_j^T\|_F^2 - 2\|Z_{(i,j)}\|_F^2 - 2\|Z_{(i,j)}\|_F^2. \tag{11}
\end{align*}
\]
where \( \|Z_{(i,j)}\|_F \) is the nuclear norm of \( Z_{(i,j)} \), i.e., the sum of its singular values.

Choosing \((i, j)\) in (9) requires a closer look at its objective function (6) for \( R_{ij} = \tilde{G}_{ij} \), the constrained G-transform structure. Using (11) we can state a result in the special case of a G-transform. We need both because for any indices \((i, j)\) the reduction in the objective function invokes the nuclear norm, while for the other indices the reduction invokes the trace. We can analyze the two objective function values separately because the Frobenius norm is elementwise and as such also blockwise. Therefore, the objective of (9) is
\[
\|Y - G_{ij}X\|_F^2 = \|Y\|_F^2 + \|X\|_F^2 - 2\text{tr}(Z) - 2C_{ij}, \tag{12}
\]
where \( C_{ij} = \|Z_{(i,j)}\|_F - \text{tr}(Z_{(i,j)}) \).

Since we want to minimize this quantity, the choice of indices needs to be made as follows
\[
(i^*, j^*) = \arg\max_{(i,j), j>i} C_{ij}, \tag{13}
\]
and then solve a Procrustes problem [34] to construct \( \tilde{G}_{i^*j^*} \).

These \((i^*, j^*)\) values are the optimum indices that lead to the maximal reduction in the objective function of (9). The expression in (13) is computationally cheap given that \( Z_{(i,j)} \) is a \( 2 \times 2 \) real matrix. Its trace is trivial to compute \( \text{tr}(Z_{(i,j)}) = Z_{ii} + Z_{jj} \) (one addition operation) while the singular values of \( Z_{(i,j)} \) can be explicitly computed as
\[
\sigma_{1,2} = \frac{1}{2} \left[ \|Z_{(i,j)}\|_F^2 \pm \sqrt{\|Z_{(i,j)}\|_F^2 - 4\det(Z_{(i,j)})^2} \right]. \tag{14}
\]
Therefore, the full singular value decomposition can be avoided and the sum of the singular values from (14) can be computed in only 23 operations (three of which are taking square roots). The cost of computing \( C_{ij} \) for all indices
(i, j), j > i, is \(25\pi(n-1)^2 \) operations. The computational burden is still dominated by constructing \(Z = YY^T\) which takes \(2snN\) operations.

**Remark 2.** Notice that \(C_{ij} \geq 0\) always. In general, this is because the sum of the singular values of any matrix \(Z\) of size \(n \times n\) is always greater than the sum of its eigenvalues. To see this, use the singular value decomposition of \(Z = U\Sigma V^T\), \(\Sigma = \text{diag}(\sigma)\), and develop:

\[
\text{tr}(Z) = \text{tr}(\Sigma V^T U) = \sum_{k=1}^{n} \sigma_k \Delta_{kk} \leq \sum_{k=1}^{n} \sigma_k = \| \Sigma \|_2,
\]

where we have used the circular property of the trace and \(\Delta = V^TU\) where \(\Delta_{kk}\) are its diagonal entries which obey \(\Delta_{kk} \leq 1\) since both \(U\) and \(V\) are orthogonal and their entries are sub-unitary. Therefore, in particular our case, we have that \(C_{ij} = 0\) when \(Z_{(i,j)}\) is symmetric and positive semidefinite (we have that \(\Delta = I\) in (15) and therefore \(\text{tr}(Z_{(i,j)}) = \| Z_{(i,j)} \|_2\).). If we have that \(C_{ij} = 0\) for all \(i\) and \(j\) then no G-transform can reduce the objective function in (9) and therefore the solution is \(G_{ij} = I\).

**Remark 3.** We can extend the G-transform to multiple indices. For example, if we consider three coordinates then \(G_{ijk}\) has the following structure:

\[
G_{ijk} = \begin{bmatrix} c & d & -d \\ d & -c & -d \\ -d & -d & -c \end{bmatrix}
\]

\(G_{ijk}\) is the Householder reflector, i.e., \(G_{ijk} = I - 2g_{ijk}g_{ijk}^T\) where \(g_{ijk} \in \mathbb{R}^n\) is a 2-sparse vector. Following steps from [26] we can also write

\[
\| Y - G_{ijk}X \|^2_F = \| Y \|^2_F + \| X \|^2_F - 2\text{tr}(YY^T) \quad + 2g_{ijk}^T(YY^T + XY^T)g_{ijk},
\]

which, together with (12), leads to \(g_{ijk}^T(YY^T + XY^T)g_{ijk} = -C_{ijk}\). This means that choosing to maximize \(C_{ij}\) in (12) is equivalent to computing an eigenvector of \(YY^T + XY^T\) of sparsity two associated with a negative eigenvalue.

There are also some differences between the two approaches. For example, matrix-vector multiplication with a G-transform \(G_{ij}\) takes 6 operations but when using the Householder structure \(G_{ij}x = (I - 2g_{ij}g_{ij}^T)x = x - 2g_{ij}^T(x)g_{ij}\) takes 8 operations (4 operations to compute the constant \(C = 2g_{ij}^Tx\), 2 operations to compute the 2-sparse vector \(z = Cg_{ij}\) and 2 operations to compute the final result \(x - z\)). Therefor, the G-transform structure is computationally preferable to the Householder structure. Each Householder reflector has \(n - 1\) (because of the orthogonality constraint) degrees of freedom while each G-transform has only 1 (the angle \(\theta \in [0, 2\pi]\) for which \(c = \cos \theta\) and \(d = \sin \theta\)) plus 1 bit (the choice of the rotation or reflector in (8)).

This concludes our discussion for the single G-transform case. Notice that the solution outlined in this section solves (9) exactly, i.e., it finds the optimum G-transform.

**C. A method for designing fast orthogonal transforms: \(G_m – DLA\)**

In this paper we propose to construct an orthogonal transform \(U \in \mathbb{R}^{n \times n}\) with the following structure:

\[
U = G_{1m.j} \ldots G_{ij} G_{i1,n}.
\]

The value of \(m\) is a choice of the user. For example, if we choose \(m\) to be \(O(n \log n)\) the transform \(U\) can be computed in \(O(n \log n)\) computational complexity – similar to the classical fast transforms. The goal of this section is to propose a learning method that constructs such a transform.

We fix the representations \(X\) and all G-transforms in (17) except for the \(k^{th}\), denoted as \(G_{ik,jk}\). To optimize the dictionary \(U\) only for this transform we reach the objective function

\[
\| Y - UX \|^2_F = \| Y - G_{im,jm} \ldots G_{ij,jk} X \|^2_F
\]

\[
= \| G_{ik,jk+1}^T \ldots G_{im,jm}^T Y - G_{ik,jk} \ldots G_{ij,jk} X \|^2_F
\]

\[
= \| Y_k - G_{ik,jk} X_k \|^2_F,
\]

where we have used the fact that multiplication by any orthogonal transform preserves the Frobenius norm. For simplicity we have denoted \(Y_k\) and \(X_k\) the known quantities in (18) and therefore \(Z_k = Y_k X_k^T\).

Notice that we have reduced the problem to the formulation in (9) whose full solution is outlined in the previous section. We can apply this procedure for all G-transforms in the product of \(U\) and therefore a full update procedure presents itself: we will sequentially update each transform and then the sparse representations until convergence. The full procedure we propose, called \(G_m – DLA\), is detailed in Algorithm 1.

**The initialization of \(G_m – DLA\)** uses a known construction. It has been shown experimentally in the past [39], that a good initial orthogonal dictionary is to choose \(U\) from the singular value decomposition of the dataset \(Y = USV^T\). We can also provide a theoretical argument for this choice. Consider that

\[
X = T_n(UU^TY) = T_n(U^TUSV^T) = T_n(USV^T).
\]

A sub-optimal choice is to assume that the operator \(T_n\) keeps only the first \(s\) rows of \(\Sigma V^T\), i.e., \(X = \Sigma_s V^T\) where \(\Sigma_s\) is the \(\Sigma\) matrix where we keep only the leading principal submatrix of size \(s \times s\) and set to zero everything else. This is a good choice since the positive diagonal elements of \(\Sigma\) are sorted in decreasing order of their values and therefore we expect \(X\) to keep entries with large magnitude. In fact, \(\| X \|^2_F = \sum_{k=s+1}^{n} \sigma_k^2\), where the \(\sigma_k\)'s are the diagonal elements of \(\Sigma\), due to the fact that the rows of \(V^T\) have unit magnitude. Furthermore, with the same \(X = \Sigma_s V^T\) we have

\[
\| Y - UX \|^2_F = \| UX - \Sigma_s V^T \|^2_F = \sum_{k=s+1}^{n} \sigma_k^2.
\]

We expect this error term to be relatively small since we sum over the smallest squared singular values of \(Y\). Therefore, with this choice of \(U\) and the optimal \(X = T_n(UU^TY)\) we have that \(\| Y - UX \|^2_F \leq \sum_{k=s+1}^{n} \sigma_k^2\), i.e., the representation error is always smaller than the error given by the best \(s\)-rank approximation of \(Y\).

In \(G_m – DLA\), with the sparse representations \(X = T_n(UU^TY)\) we proceed to iteratively construct each G-transform. At step \(k\), the problem to be solved is similar to (18)
Algorithm 1 – Gₘ⁻DLA

Input: Dataset \( Y \), the target sparsity \( m \), and the number of G-transforms \( K \).

Output: The sparsifying square orthogonal transform \( U = G_1X_1 \) and sparse representations \( X \) such that the objective function of our problem, a threshold can be added to the factorization.

1) Perform the economy size singular value decomposition of the dataset \( Y = Y_1 \).
2) Compute sparse representations \( X = Y_1X_1 \).
3) For \( k = 1, \ldots, m \) with \( X \) and all previous \( k - 1 \) transforms fixed and \( X_1 \) by (13)
4) \( G_1 \). For \( k = 1, \ldots, m \) with \( X \) and all previous \( k - 1 \) G-transforms fixed and \( X_1 \) by (13)
5) \( G_1 \). For \( k = 1, \ldots, m \) with \( X \) and all previous \( k - 1 \) G-transforms fixed and \( X_1 \) by (13)

Remark 5. At each iteration of the proposed algorithm, we can process the G-transforms in the order of their indices by sequentially applying the G-transforms in its composition. The usual complexity of applying the G-transforms in its composition is \( O(m \log n) \) for sufficiently large \( m \) of a true in general.

Remark 6. After indices \( (i, j) \) are selected, we have that \( Z_{ij} \) is updated to 

\[
Z_{ij} = (1/\epsilon) - \infty \text{, therefore this pair cannot be selected again in the future.}
\]

Even so, there are several options regarding the ordering of the G transforms such as the global or local minimum points. Since \( Q \) is the global optimum we can process the G-transforms in the order of their indices without any reordering.

Remark 7. It is not always true that the solution \( G_0 \) is symmetric positive semidefinite. Since the G-transforms are computed in the order of their indices, the problem of partitioning the set \( \{1, 2, \ldots, n\} \) in pairs of two and construct the corresponding G-transforms \( G_0 \) can be factually difficult. However, the G-transforms computed in our algorithm are maximally reduced and therefore the number of transforms is relatively high.

Perspective since the number of transforms is relatively high.

The number of G-transforms \( m \) could be decided during the running time of \( G_1 \) –DLA. However, the number of G-transforms \( m \) could be decided during the running time of \( G_1 \) –DLA. However, the number of G-transforms \( m \) could be decided during the running time of \( G_1 \) –DLA. However, the number of G-transforms \( m \) could be decided during the running time of \( G_1 \) –DLA.
This means that even with an appropriately large $m \sim O(n^2)$, $G_m$–DLA might not always be able to match the performance of Q–DLA. This is not a major concern since in this paper we explore fast transforms and therefore $m \ll n^2$.

This concludes the presentation of the proposed $G_m$–DLA method. Based on similar principles next we provide a learning method for fast square but non-orthogonal dictionaries.

V. A METHOD FOR DESIGNING FAST, GENERAL, NON-ORTHOGONAL TRANSFORMS: $R_m$–DLA

In the case of orthogonal dictionaries, the fundamental building blocks like Householder reflectors and Givens rotations are readily available. This is not the case for general dictionaries. In this section we propose a building block for non-orthogonal structures in subsection A and then show how this can be used in a similar fashion to the G-transform to learn computationally efficient square non-orthogonal dictionaries by deriving the $R_m$–DLA method in subsection B.

A. A building block for fast non-orthogonal transforms

We assume no constraints on the variables $p, q, r, t$ (these are four degrees of freedom) and therefore $R_{ij}$ from (3) is no longer orthogonal in general. We propose to solve the following optimization problem

$$\min_{(i, j), R_{ij}} \|Y - R_{ij}X\|_F^2. \quad (20)$$

As in the G-transform case, we proceed with analyzing how indices $(i, j)$ are selected and then how to solve the optimization problem (20), with the indices fixed. We define

$$Z = YX^T, \quad W = XX^T,$$

with entries $Z_{ij}$ and $W_{ii}$ respectively. Solving (20) for fixed $(i, j)$ leads to a least squares optimization problem as

$$\min_{R_{ij}} \left\| \begin{bmatrix} y^T_j \\ y^T_i \end{bmatrix} - \tilde{R}_{ij} \begin{bmatrix} x^T_j \\ x^T_i \end{bmatrix} \right\|_F^2, \quad (21)$$

where $y^T_i, x^T_i$ are $i$th rows of $Y$ and $X$ respectively and whose solution is

$$\tilde{R}_{ij} = \begin{bmatrix} Z_{ii} & Z_{ij} \\ Z_{ji} & Z_{jj} \end{bmatrix} \begin{bmatrix} W_{ii} & W_{ij} \\ W_{ji} & W_{jj} \end{bmatrix}^{-1}.$$

Choosing $(i, j)$ in (20) depends on the objective function value in (22) given by the least squares solution from above:

$$\left\| \begin{bmatrix} y^T_j \\ y^T_i \end{bmatrix} - \tilde{R}_{ij} \begin{bmatrix} x^T_j \\ x^T_i \end{bmatrix} \right\|_F^2 = \left\| \begin{bmatrix} y^T_j \\ y^T_i \end{bmatrix} \right\|_F^2 - \text{tr} \left( \begin{bmatrix} Z_{ii} & Z_{ij} \\ Z_{ji} & Z_{jj} \end{bmatrix} \begin{bmatrix} W_{ii} & W_{ij} \\ W_{ji} & W_{jj} \end{bmatrix}^{-1} \right). \quad (23)$$

This, together with the development in (6), leads to

$$\|Y - R_{ij}X\|_F^2 = \|Y\|_F^2 + \|X\|_F^2 - 2\text{tr}(Z) - C_{ij},$$

with $C_{ij} = \left\| \begin{bmatrix} x^T_i \\ x^T_j \end{bmatrix} \right\|_F^2 - 2\text{tr} \left( \begin{bmatrix} Z_{ii} & Z_{ij} \\ Z_{ji} & Z_{jj} \end{bmatrix} \right)$. \quad (24)

Since the matrices involved in the computation of $C_{ij}$ are $2 \times 2$ we can use the trace formula and the inversion of a $2 \times 2$ matrix formula to explicitly calculate

$$C_{ij} = W_{ii} + W_{jj} - 2(Z_{ii} + Z_{jj}) + \frac{W_{ii}(Z_{ii}^2 + Z_{jj}^2)}{W_{ij}W_{ji}} + \frac{(Z_{ii}Z_{ij} + Z_{ji}Z_{jj})(W_{ij} + W_{ji})}{W_{ij}W_{ji}} \quad (25)$$

Finally, to solve (20) we select the indices as

$$(i^*, j^*) = \arg\max_{j > i} C_{ij}, \quad (26)$$

and then solve a least square problem to construct $\tilde{R}_{i^*, j^*}$. The $C_{ij}$ are computed only when $W_{ij}W_{ji} - W_{ij}W_{ji} \neq 0$, otherwise $C_{ij} = -\infty$. To compute each $C_{ij}$ in (25) we need 24 operations and there are $\frac{m(m-1)}{2}$ such $C_{ij}$. The computational burden is dominated by constructing $Z = YX^T, W = XX^T$ which take $2snN$ and $snN$ operations, respectively.

**Remark 8.** A necessary condition for a dictionary $D \in \mathbb{R}^{n \times n}$ to be a local minimum point for the dictionary learning problem is that all $C_{ij} = 0$ for $Z = YX^T \Delta, W = DXX^T \Delta$.

This concludes our discussion for one transform $R_{ij}$. Notice that just like in the case of one G-transform, the solution given here finds the optimum $R_{ij}$ to minimize (20).

B. A method for designing fast general transforms: $R_m$–DLA

Similarly to $G_m$–DLA, we now propose to construct a general dictionary $D \in \mathbb{R}^{n \times n}$ with the following structure:

$$D = R_{i_1,j_1} \cdots R_{i_{2k},j_{2k}} R_{i_{2k+1},j_{2k+1}} \Delta. \quad (27)$$

The value of $m$ is a choice of the user. For example, if we choose $m$ to be $O(n \log n)$ the dictionary $D$ can be applied in $O(n \log n)$ computational complexity – similar to the classical fast transforms. The goal of this section is to propose a learning method that constructs such a general dictionary. As the transformations $R_{ij}$ are general, the diagonal matrix $\Delta \in \mathbb{R}^{n \times n}$ is there to ensure that all columns $d_i$ of $D$ are normalized $\|d_i\|_2 = 1$ (as in the formulation (1)). This normalization does not affect the performance of the method since $DX$ is equivalent to $(D \Delta)(\Delta^{-1}X)$.

We fix the representations $X$ and all transforms in (27) except for the $k^{th}$ transform $R_{i_k,j_k}$. Moreover, all transforms $R_{i_{k+1},j_{k+1}}, \ldots, R_{i_m,j_m}$ are set to $I$. Because the transforms $R_{ij}$ are not orthogonal we cannot access directly any transform $R_{i_k,j_k}$ in (27), but only the last most one $R_{i_{m,j_m}}$. In this case, to optimize the dictionary $D$ only for this $R_{i_k,j_k}$ transform we reach the objective

$$\|Y - R_{i_k,j_k} \cdots R_{i_{2k},j_{2k}} R_{i_{2k+1},j_{2k+1}} X\|_F^2 = \|Y - R_{i_k,j_k} X_k\|_F^2. \quad (28)$$

Therefore, our goal is to solve

$$\min_{R_{i_k,j_k}} \|Y - R_{i_k,j_k} X_k\|_F^2. \quad (29)$$

Notice that we have reduced the problem to the formulation in (20) whose full solution is outlined in the previous section. We can apply this procedure for all G-transforms in the
Algorithm 2 – $R_m$–DLA.
Fast Non-orthogonal Transform Learning.

Input: The dataset $Y \in \mathbb{R}^{n \times N}$, the number of $R_{ij}$ transforms $m$, the target sparsity $s$ and the number of iterations $K$.

Output: The sparsifying square non-orthogonal transform $D = R_{m,j_1} \ldots R_{m,j_k} X_{i_1,j_1} X_{i_2,j_2} \ldots X_{i_1,j_1} X_{i_2,j_2} \Delta$ and sparse representations $X$ such that $\|Y - DX\|_F^2$ is reduced.

Initialization:

1) Perform the economy size singular value decomposition of the dataset $Y = U\Sigma V^T$.
2) Compute sparse representations $X = T_j(U^T Y)$.

Iterations $1, \ldots, K$:

1) For $k = 1, \ldots, m$: with $X$ and all previous $k - 1$ R-transforms fixed and $R_{i_k,j_k}$, construct the new $R_{i_k,j_k}$, where indices $(i_k, j_k)$ are given by (26) and $R_{i_k,j_k}$ is given by the least squares solution that minimizes (28).
2) Compute $\Delta$ in (27) such that $\|d_j\|_2 = 1$.
3) Compute sparse representations $X = \text{OMP}(D, Y, s)$ where $D$ is given in (27).

Iterations $1, \ldots, K$:

1) For $k = 1, \ldots, m$: with $X$, indices $(i_k, j_k)$ and all transforms except the $k^{th}$ fixed, update only the non-zero part of $R_{i_k,j_k}$, denoted $R_{i_k,j_k}$, such that (30) is minimized.
2) Compute $\Delta$ in (27) such that $\|d_j\|_2 = 1$.
3) Compute sparse representations $X = \text{OMP}(D, Y, s)$ where $D$ is given in (27).

The product of $D$ and therefore a full update procedure presents itself: we will sequentially update each transform in (27), from the right to the left, and then the sparse representations until convergence or for a total number of iterations $K$.

Once these iterations terminate we can refine the result. As previously mentioned, we cannot arbitrarily update a transform $R_{i_k,j_k}$ because this transform is not orthogonal. But we can update its non-zero part $R_{i_k,j_k}$. Consider the development:

$$
\|Y - R_{m,j_1} \ldots R_{m,j_k} X_{i_1,j_1} X_{i_2,j_2} \ldots X_{i_1,j_1} X_{i_2,j_2} \Delta\|_F^2
= \|Y - B_k R_{i_k,j_k} X_k\|_F^2
= \|\text{vec}(Y) - (X_k^T \otimes B_k)\text{vec}(R_{i_k,j_k})\|_F^2
= \|\text{vec}(Y) - \sum_{t \in \{1, \ldots, n\} \setminus \{i_k, j_k\}} ((X_k^T)_t \otimes (B_k)_t) - Cx\|_F^2
= \|w - Cx\|_F^2,
$$

where $x = \text{vec}(\tilde{R}_{i_k,j_k}) \in \mathbb{R}^{s}$ and $C = \{(X_k^T)_i, (B_k)_j\} \in \mathbb{R}^{s \times nN \times s}$. We have denoted by $(B_k)_i$ the $i^{th}$ column of $B_k$ and $\otimes$ is the Kronecker product. To develop (30) we have used the fact that the Frobenius norm is an elementwise operator, the structure of $R_{i_k,j_k}$ and the fact that

$$
\text{vec}(B_k R_{i_k,j_k} X_k) = (X_k^T \otimes B_k)\text{vec}(R_{i_k,j_k}).
$$

The $x$ that minimizes (30) is given by the least squares solution $x = (C^T C)^{-1} C^T w$. Therefore, once the product of the $m$ transforms is constructed we can update the non-zero part of any transform to further reduce the objective function. What we cannot do is update the indices $(i_k, j_k)$ on which the calculation takes place, these stay the same.

Therefore, we propose a learning procedure that has two sets of iterations: the first constructs the transforms $R_{i_k,j_k}$ in a rigid manner, ordered from right to left most, and the second only updates the non-zero parts $R_{i_k,j_k}$ of all the transforms without changing the coordinates $(i_k, j_k)$. The full procedure we propose, called $R_m$–DLA, is detailed in Algorithm 2. This algorithm has two main parts which we will now describe.

The initialization of $R_m$–DLA has the goal to construct the sparse representation matrix $X \in \mathbb{R}^{s \times N}$. We have several options in this step. We can construct $X$ in the same way as for $G_m$–DLA from the singular value decomposition of the dataset or by running another dictionary learning algorithm (like the K–SVD [8] for example) and use the $X$ it constructs.

The iterations of $R_m$–DLA are divided into two sets. The goal of the first set of iterations is to decide upon all the indices $(i_k, j_k)$ while the second set of iterations optimizes over the non-zero components of all the transforms in the factorization with the fixed indices previously decided.

The proposed $R_m$–DLA can be itself efficiently implemented. When iteratively solving problems as (28) we have that $X_{k+1} = R_{i_k,j_k} X_k$ with $X_1 = X$ while when iteratively solving problems as (30) we have that $X_{k+1} = R_{i_k,j_k} X_k$ and $B_{k+1} = B_k R_{i_k,j_k}^{-1}$ with $X_1 = X$ and $B_1 = B_{m,j_1} \ldots B_{i_1,j_1}$. The explicit inverse $R_{i_k,j_k}^{-1}$ is not computed, instead the equivalent linear system for 2 variables can be efficiently solved.

The updates of all the transforms $R_{i_k,j_k}$ monotonically decrease our objective function since we solve exactly the optimization problems in these variables. Unfortunately, normalizing to unit $\ell_2$ norm the columns of the transform and constructing the sparse approximations via OMP, which is not an exact optimization step, may cause increases in the objective function. For these reasons, monotonic convergence of $R_m$–DLA to a local minimum point cannot be guaranteed. For this reason, at all times we keep track of the best solution pair $(D, X)$ and return it at the end of each iterative process.

This concludes our discussion of $R_m$–DLA. We now move to discuss the computational complexity of the transforms created by the proposed methods and to show experimentally their representation capabilities.

VI. THE COMPUTATIONAL COMPLEXITY OF USING LEARNED TRANSFORMS

In this section we look at the computational complexity of using the learned dictionaries to create the sparse representations on a dataset $Y$ of size $n \times N$. We are in a computational regime where we assume dimensions obey

$$
s \ll n \ll N.
$$

The computational complexity of using a general non-orthogonal dictionary $A$ of size $n \times n$ in sparse recovery problems with Batch–OMP [15] is

$$
N_A \approx (2n^2 + s^2 n + 3sn + s^3)N + n^3.
$$
n^3 is associated with the construction of the Gram matrix of the dictionary and it does not depend on the number of samples $N$ in the data. The total number of operations is dominated by constructing the projections in the dictionary column space which takes $2n^2$ operations per sample. The other operations depend on the sparsity $s$ and express the cost of iteratively finding the support of the sparse approximation.

The computational complexity of using an orthogonal dictionary $Q$ designed via Q–DLA is

$$N_Q \approx (2n^2 + ns)N. \quad (34)$$

As in the general case, the cost is dominated by constructing the projections $Q^T Y$ which takes $2n^2$ operations for each of the $N$ columns in $Y$. The cost of $ns$ expresses the approximate work done to identify the largest $s$ entries in magnitude in the representation of each data sample. This can be performed in an efficient manner by keeping the $s$ largest components in magnitude while the projections are computed for each data sample. Compared with (33), the iterative steps of constructing the support of the OMP solution for each sample and the construction of the Gram matrix (which is the identity matrix in this case) is no longer needed.

The same operation with a dictionary $U$ as (17) computed via $G_{m_1}$–DLA takes

$$N_U \approx (6m_1 + ns)N. \quad (35)$$

The result is similar to (34) but now the cost of constructing the projections $U^T Y$ takes now only $6m_1$ operations per data sample. Here is where the G-transform factorization is used explicitly to reduce the computational complexity.

Finally, with a dictionary $D$ as (27) computed via $R_{m_2}$–DLA the sparse approximation step via Batch–OMP [15] takes

$$N_D \approx (6m_2 + n + s^2n + 3sn + s^3)N + 6m_2n. \quad (36)$$

In this case, the cost of building the projections $D^T Y$ takes $6m_2$ operations to apply each $R_{ij}$ transform and then $n$ operations to apply the scaling of the diagonal $\Delta$. Simplifications occur also for the construction of the symmetric Gram matrix $D^T D$ which now takes $6m_2n$ operations, instead of the regular $n^3$ operations. This later simplification might not be significant since it is not dependent on the size of the dataset $N$.

A dictionary $U$ designed via $G_{m_1}$–DLA has approximately the same computationally complexity as a general orthogonal dictionary $Q$ designed via Q–DLA when

$$m_1 = \left\lceil \frac{n^2}{3} \right\rceil \quad. \quad (37)$$

Because any orthogonal matrix can be factorized as a product of $\frac{n(n-1)}{2}$ G-transforms and because of the upper limit imposed in (37) it is clear that we cannot express any orthogonal dictionary as an efficient transform for sparse representations. In some cases, the full orthogonal dictionary $Q$ might be more efficient than its factorization with G-transforms. In general, the representation error achieved by general orthogonal dictionaries designed via Q–DLA is a performance limit for G-transform based dictionaries.

A similar comparison can be made between the computational complexity of a general dictionary $A$ and that of a dictionary $D$ composed of $m_2$ transformations $R_{ij}$. Their complexities approximately match when

$$m_2 = \left\lceil \frac{(2n^2 + s^2n + 3sn + s^3)N + n^3}{6(N + n)} \right\rceil \approx \frac{n^2}{3}. \quad (38)$$

This shows that for both $G_{m}$–DLA and $R_{m}$–DLA the computationally efficient regimes are when $m \sim O(n)$ or in general $m \ll n^2$.

A last comment regards the comparison between dictionaries created with $G_{m}$–DLA and $R_{m}$–DLA. When $m_1 = m_2$ we expect $R_{m_1}$–DLA to perform better but at a higher computational cost. Assuming large datasets $N \to \infty$ and low sparsity $s \ll n$, computational complexities approximately match when

$$m_1 \approx \left\lceil m_2 + \frac{(s^2 + 3s + 1)n}{6} \right\rceil. \quad (39)$$

Due to the use of the OMP procedure for non-orthogonal dictionaries to create the sparse approximations, dictionaries designed via $R_m$–DLA are much more computationally complex than the orthogonal dictionaries designed via $G_m$–DLA. Otherwise, as depicted in (39), for the same representation performance the orthogonal dictionaries may contain many more G-transforms in their factorization than $R_{ij}$ transforms contained in the factorization of a non-orthogonal dictionary. As a consequence, it may be that orthogonal dictionaries are always more computationally efficient than general dictionaries for approximately equal representation capabilities. A definite advantage of $R_m$–DLA is that it has the potential to create dictionaries that go below representation errors given by orthogonal dictionaries designed via Q–DLA, the performance limit of $G_m$–DLA.

Using these approximate complexities, we discuss in the results section the representation performance versus the computational complexity trade-off that the dictionaries constructed via the proposed methods display.

VII. EXPERIMENTAL RESULTS

In this section we provide experimental results that show how transforms designed via the proposed methods $G_{m}$–DLA and $R_{m}$–DLA behave on image data.

The input data that we consider are taken from popular test images from the image processing literature (pirate, peppers, boat etc.). The test dataset $Y \in \mathbb{R}^{n \times N}$ consists of $8 \times 8$ non-overlapping patches with their means removed and normalized $Y \leftarrow Y/255$. We choose to compare the proposed methods on image data since in this setting fast transforms that perform very well, like the Discrete Cosine Transform (DCT) [41] for example, are available. Our goal is to provide dictionaries based on factorizations like (17) and (27) that perform well in terms of representation error with a small number $m$ of basic transforms in their composition. All algorithms run for $K = 150$ iterations and there are $N = 12288$ image patches in the dataset $Y$ each of size $n = 64$. 
Fig. 1. For the proposed $G_{256}$–DLA we show the relative representation error (40) in the initialization steps for the dataset $Y$ created from the patches of the images couple, peppers and boat with sparsity $s \in \{4, 8, 12\}$. Notice that in general the representation error can surpass 100%, for example, for orthogonal dictionaries, the maximum value $\epsilon = 4$ is achieved when $X = -Y$ and $D = I$ in (40).

Fig. 2. For the same experimental setup as in Figure 1, we show the representation error for the $K = 150$ regular iterations of $G_{256}$–DLA.

To measure the quality of a dictionary $D$ we choose to evaluate the relative representation error

$$\epsilon = \frac{||Y - DX||_F^2}{||Y||_F^2} \times 100 \% \quad (40)$$

Figures 1 and 2 show the evolution of $G_{256}$–DLA for $K = 150$ iterations (including the initialization procedure, i.e., the first 256 steps of the algorithm). The figures show how effective the initialization is in reducing the representation error for any sparsity level. Notice that the initialization procedure provides similar results regardless of the sparsity level $s$. The $K = 150$ iterations of $G_{256}$–DLA further lower the representation error providing better results with larger sparsity level. As previously discussed, each step of the algorithm monotonically decreases the objective function of the dictionary learning problem until convergence.

Figure 3 shows how $G_m$–DLA evolves with the number of transforms $m$ and the sparsity $s$. As expected, increasing the number of transforms $G_{ij}$ and $R_{ij}$ in the factorization always lowers the representation error but with diminishing returns as $m$ increases. This figure helps choose the number of transforms $m$ while balancing between the computational complexity and representation performance. Large decreases in the representation error are seen up to $m = 96$ or $m = 128$ while thereafter increasing $m$ brings smaller benefits. Also, with higher sparsity levels the number of transforms $m$ becomes less relevant. We notice that with $s = 12$ the representation performance hits a plateau after $m \geq 128$ transforms.

An interesting point of comparison is between the dictionaries constructed via $G_m$–DLA and $H_p$–DLA [26]. Figure 4 provides a detailed comparison between the two. A matrix-vector multiplication takes $4n$ operations for a reflector and only 6 operations for a G-transform. If we compare the computational complexities of the dictionaries constructed by the two methods we find approximate equality between $H_p$–DLA and $G_{1/2}$–DLA. Notice from this figure that for a low $m$ the G-transform approach provides better results than the Householder approach while also enjoying lower computational complexity. As the complexity of the dictionaries increases (larger number of G-transforms or reflectors) the gap between the two approaches decreases. The most complex dictionaries are designed via $H_{16}$–DLA and $G_{642}$–DLA and they closely match the performance of the general orthogonal dictionary learning approach Q–DLA while still keeping a computational advantage. In this case, the Householder approach keeps a slight edge in representation performance. Since the proposed approach updates the G-transforms sequentially the probability of getting stuck in local minimum points is more likely with large $m$. The difficulties that $G_m$–DLA encounters for large $m$ are also discussed in Remark 6.

Fig. 3. Performance of $G_m$–DLA and $R_m$–DLA in terms of the relative representation error (40) for different sparsity levels $s \in \{4, 8, 12\}$.

Fig. 4. Comparisons, in terms of relative representation errors (40), of $G_m$–DLA against the DCT, Q–DLA [33], SK–SVD [40] and Householder based orthogonal dictionaries [26] denoted here $H_p$–DLA where $p$ is the number of reflectors in the factorization of the dictionary. The number of transforms $m$ is chosen so that computational complexity comparisons against $H_p$–DLA is possible. Computational complexity approximately match between: $H_1$–DLA and $G_{42}$–DLA, $H_2$–DLA and $G_{85}$–DLA, $H_3$–DLA and $G_{128}$–DLA, $H_4$–DLA and $G_{256}$–DLA, $H_6$–DLA and $G_{512}$–DLA, $H_8$–DLA and $G_{1024}$–DLA, $H_{16}$–DLA and $G_{642}$–DLA. The sparsity level is set to $s = 4$ for all methods. We use the SK–SVD to build a square, non-orthogonal, dictionary.
<table>
<thead>
<tr>
<th>m</th>
<th>DCT</th>
<th>Gₘ−DLA</th>
<th>Rₘ−DLA</th>
<th>Q−DLA</th>
<th>SK−SVD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Representation error (%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>m</th>
<th>Rₘ−DLA</th>
<th>Sparse K−SVD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Representation error (%)</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>10</td>
<td>15</td>
</tr>
</tbody>
</table>

It is also interesting to see that the representation performance of the DCT is matched by H₃−DLA and Gₘ−DLA. The computational complexity of H₃−DLA approximately matches that of the DCT [41] (based on the FFT) while Gₘ−DLA is actually computationally simpler than the DCT. In fact, any dictionary constructed by Gₘ−DLA for 85 ≤ m ≤ 128 is faster and provides better representations than the DCT.

Fig. 5. For the same experimental setup as in Figure 4 we compare Gₘ−DLA against Rₘ−DLA.

Fig. 6. The evolution of Rₘ−DLA for m large enough to outperform any orthogonal dictionary.

Fig. 7. Pareto curves for Rₘ−DLA and the Sparse K−SVD approach [42]. We consider the representation error in (40) and the number of operations necessary to perform Dᵀy given a target vector y ∈ ℝⁿ. We train five square dictionaries D = ΦS with the Sparse K−SVD approach, each with a different sparsity parameter p ∈ {2, 3, 4, 6, 8} in the matrix S. For a transform created with Rₘ−DLA matrix-vector multiplication takes 6m operations. The experimental setup for training the transforms is the same as in Figure 4.

In the last experimental setup we compare our Rₘ−DLA with the previously proposed Sparse K−SVD approach [42]. We use the Sparse K−SVD to build a square dictionary D = ΦS ∈ ℝⁿ×ⁿ where Φ is a well-known classic transform (in our case the DCT) and S ∈ ℝⁿ×ⁿ is matrix with only p non-zero entries per column. In this fashion, matrix-vector multiplication like Dᵀy = SᵀΦᵀy takes 2pm + C operations, where C is the cost of applying the DCT (in our case, this is the same as using a transform designed via G₁₂₈−DLA or R₁₂₈−DLA). Rₘ−DLA performs consistently better than the Sparse K−SVD for very fast transforms while the performance gap closes for very low representation errors. The Sparse K−SVD suffers from the fact that the fast transform Φ is fixed and therefore the optimization takes place over only pn degree of freedom. We restrict ourselves to square transforms and avoid the comparison with overcomplete dictionaries designed via the K−SVD or the Sparse K−SVD. Experimental insights into how the representation performance scales with the number of atoms in the dictionary are given in [40], [43].

When designing a very fast orthogonal transform (whose complexity let us say is order n or n log n) then Gₘ−DLA provides very good results while achieving the lowest computational complexity. For improved performance, more complex orthogonal transforms perform better when designed via Hₙ−DLA. If representation capabilities is the only performance metric then the non-orthogonal transforms designed by Rₘ−DLA are the weapon of choice. For large m both Gₘ−DLA and Rₘ−DLA can suffer from long running times. For example, G₁₂₈−DLA takes several minutes to terminate while R₁₂₈−DLA’s running time is close to ten minutes on a modern Intel i7 computer system. We note that the algorithms are...
implemented in Matlab®. A careful implementation in a lower level compiled programming language will drive these running times much lower and reduce the memory footprint.

VIII. CONCLUSIONS

In this paper we present practical procedures to learn square orthogonal and non-orthogonal dictionaries already factored into a fixed number of computationally efficient blocks. We show how effective the dictionaries constructed via the proposed methods are on image data where we compare against the fast cosine transform on one side and general non-orthogonal and orthogonal dictionaries on the other. We also show comparisons with a recently proposed method that constructs Householder based orthogonal dictionaries. We show empirically that the proposed methods construct transforms that provide an improved trade-off between computational complexity and representation performance among the methods we consider. We are able to construct transforms that exhibit lower computational efficiency and lower representation error than the fast cosine transform for image data. We expect the current work to extend the use of learned transforms in time critical scenarios and to devices where, due to power limitations, only low complexity algorithms can be deployed.

REFERENCES


