Multilayer Sparse Matrix Factorization

Quoc-Tung Le

Multilayer Sparse Matrix Factorization

LE QUOC TUNG

Ecole Normale Superieure Paris-Saclay
INTERNSHIP MASTER 2 MVA 2019-2020

Supervised by Prof. **REMI GRIBONVAL**

Abstract

Matrix factorization plays an important role in many machine learning and data mining problems such as dictionary learning, data visualization, dimension reduction, to name but a few. In most scenarios, additional constraints are posed to enforce certain properties of the factorization such as: low-rank, weighted low rank, non-negative. Sparsity is one of such desired characteristics and has been at the heart of a plethora of signal processing and data analysis. Since sparsity is usually enforced with regularization (l_1 norm, nuclear norm), current techniques lack control over the sparse pattern of the solution, especially in non-convex optimization. This report is devoted to address this issue. On the one hand, it will describe a new projection operator to increase the variety of proximal algorithm, a promising method to tackle sparse structured factorization. On the other hand, it will discuss the incorporation of Hard Thresholding Pursuit (HTP), a mechanism in Compressive Sensing to burst the robustness of current algorithms. Research on fixed support factorization is also introduced. Experiments carried out on classical linear operators such as the Discrete Fourier Transform, the Hadamard Transform and other self-crafted matrices will demonstrate the effect of the proposals.

Contents

1 Introduction

Sparsity is inspired by the fact that sparse objects are easier to manipulate (sparse matrix, sparse graphs) than denser ones, more prone to interpretation and they do appear ubiquitously in nature. Techniques involving sparsity either consist in posing constraints to enforce the sparse solutions explicitly or in promoting sparsity implicitly through regularization. Both approaches attract great attention in the research community and eventually result into many powerful techniques such as Best Subset Selection ([2], [15]), Ridge Regression ([16]), the LASSO ([27], [8]), Matching Pursuit Family ([7], [24], [20],[12]).

Indeed, important linear transforms such as the Discrete Fourier Transform, the Discrete Cosine Transform and the Hadamard Transform, ... have effective implementations such the Fast Fourier Transformation and many other variants (more details in [10]). Instead of a plain computational complexity of $O(n^2)$, the complexity of computing those operators is improved to $O(n\log(n))$. The achievement of such speedup is largely due to the structured factorization of those transforms. In fact, although all of the mentioned linear operators are dense matrices, they admit the following formula:

$$A = \prod_{j=1}^{d} S_j \tag{1}$$

where $d = \log n$, n is the size of the transform and S_j are sparse factors ($\forall j = 1, \ldots, \log n$).

The ultimate objective of the internship is to construct a well-grounded method to reliably reverse engineer such matrix factorizations. The problem is challenging in many aspects: None of the methods listed above has a direct link with the problem of interest since they mostly deal with linear settings. The Discrete Fourier Transform, the Discrete Cosine Transform and the Hadamard Transform have $\log n$ factors, which means we have to face a problem of multi-linear layers. It could be seen as training a neural network without activation function and having additional sparse constraints. Therefore, our problem becomes highly non-convex and resists many existing methods.

The starting point of the internship is paper [19] and two algorithms introduced in the paper: PALM and hierarchical factorization. The main contributions of the paper are to formalize the problem of sparse matrix factorization and to introduce proximal algorithms to construct a general algorithmic framework. We will spend a dedicated Section 2 to review proximal algorithms in general and their applications in [19], [6], [3], [12], [4] and [21] in particular. The report then will discuss a new proximal (projection) operator to enhance the performance of these algorithms. In addition, based on Hard Thresholding Pursuit (HTP) [12] (a method belonging to the Matching Pursuit Family), we propose an algorithm to improve the robustness of factorization algorithm, especially with randomized initialization.

Fixed support factorization is another topic of interest in this report. The support of a matrix is the set of indices whose coefficients can be nonzero. The smaller is the support, the sparser is the matrix. Normal sparse matrix factorization comprises the search for both support and its corresponding coefficients. Fixed support factorization engages only to the latter. It could serve as a subprocedure of the problem of sparse matrix factorization. Our main result of this topic is to show its difficulties in the viewpoint of optimization as well as to improve the performance of existing algorithms.

Last but not least, here is the structure of the report: After the introduction, related works in Section 2 will introduce many approaches in the literature with several in-depth introductions: Proximal Algorithms [23], Iterative Shrinkage Thresholding Algorithm [6], Hard Thresholding Pursuit [12] and sparse matrix factorization in [19]. They serve as the basis of this report and provides many important tools. Section 3 is devoted to the discussion of a new proximal operator estimation named Generalized Hungarian Method. This new operator as well as HTP will play the main roles in the algorithm proposed in Section 4. Fixed support factorization will be introduced in Section 5. All theoretical discussions will be followed by proper experiments at the end of each section. Section 6 compares all the experimental results and concludes the report.

2 Related Work

Fast transforms are at the core of many important algorithms in signal processing, feature extraction, dimensionality reduction, data visualization and compressing model. Their appearance is ubiquitous

and irreplaceable. The Discrete Cosine Transform and the Discrete Sine Transform are the basis of the mel-frequency cepstral coefficients (MFCCs), a fundamental concept in speech recognition. The Hadamard transform is part of the well-known image denoising algorithm BM3D. The Discrete Fourier Transform is indispensable for spectral analysis, filter bank, data compression. For some reason, they all admit a representation as a multiplication of sparse matrices.

Traditional matrix factorization tools are the truncated Singular Value Decomposition (truncated SVD), local approximations of operators by low rank patches, including the Fast Multipole Method (FMM) [26], H-matrices [13], to name but a few. While truncated SVD cannot approximate well matrices which are near orthogonal or could not be well-approximated with low rank, others are heavily based on hand-crafted rather than data-driven features. Hence, latest trend is to adopt signal processing and machine learning techniques to tackle these problems.

Sparse recovery is a research subject that has been extensively studied in the context of linear inverse problem. The problem statement is: given a measurement matrix A and a signal y, compute a sparse code x such that $Ax \approx y$. There are many proposed approaches, notably norm regularization (norm L_1) and greedy methods [7], [24], [20],[12]. Theoretical guarantee for greedy methods are extensively studied, especially under the assumption of Restricted Isometry Property (RIP) [11], [25], [29], [12] or Tropp's Exact Recovery Condition (ERC) [28]. Although they appear differently, those criteria require the columns of matrix A to be more or less near *orthogonal* in a certain level.

Recently, several results in [10] are developed around well-known linear transforms such as the Discrete Fourier Transform, the Discrete Cosine Transform, the Discrete Sine Transform and the Hadamard Transform. Authors in [10] reveal that all those linear transforms share the same sparse factorization pattern called Butterfly Parametrization (BP). Further analysis reveals that BP and its variants contain several classes of important linear transformations ranging from convolution, Toeplitz matrices to orthogonal polynomial matrices. Such structure is easily adopted into a complete end-to-end training framework such as Neural Networks and produces fairly good results comparing to similar approaches without sparsity constraints. This allows the author to prove the expressiveness of BP and to demonstrate effective performance during inference phase. Nevertheless, this approach heavily limits the sparse structure and the supports of the learned operator. In fact, the structures constructed by BP have to be equivalent to the sparse structure of the Discrete Fourier Transform (up to a permutation), which does not appear in many problems.

Deep learning and Neural Networks have a strong connection with matrix factorization, especially in the viewpoint of optimization. Indeed, factorizing a matrix into multiple factors could be seen as training a (deep) Neural Network without activation function (or with identity activation function). Interesting questions involving those approaches are how to promote (or enforce) the sparsity into the solution of an optimization algorithm and what does the landscape of the optimization problem look like. The latter is more or less addressed in [30], [17], [1]. So far, the conclusion is in shallow fully connected linear networks (up to than three layers) with square error loss, there is not any spurious local minimum and premier order critical point which is not the global minimum is a strictly saddle point given that the data is full rank. On the other hand, networks deeper than three layers always have "bad" saddle points (saddle points with Hessian having no strictly negative eigenvalue). Nevertheless, none of them engaged to the problem of sparsity in training neural network.

2.1 Proximal operators and proximal algorithms

Next, we will introduce the notion of proximal operator and of proximal algorithms in general. The discussion continues with other related algorithms and many applications of proximal algorithms to sparse matrix factorization in the literature.

Definition 2.1 (Proximal Operator, [23])**.** *Given $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$, the proximal operator $\mathbf{prox}_f : \mathbb{R}^n \to \mathbb{R}^n$ of f is defined as:*

$$\mathbf{prox}_f(v) = \arg\min_x (f(x) + \frac{1}{2}\|x - v\|_2^2) \tag{2}$$

We will also often encounter the proximal operator of the scaled function λf where $\lambda > 0$, which can be expressed as:

$$\mathbf{prox}_{\lambda f}(v) = \arg\min_x (f(x) + \frac{1}{2\lambda}\|x - v\|_2^2 \tag{3}$$

Example 2.1. *Let $f = I_{\mathcal{C}}(x)$ be the indicator of a closed set $\mathcal{C} \subset \mathbb{R}^n$ defined as:*

$$I_{\mathcal{C}}(x) = \begin{cases} 0 & \text{if } x \in \mathcal{C} \\ +\infty & \text{otherwise} \end{cases} \tag{4}$$

The proximal operator of f is $\mathbf{prox}_f(v) = \arg\min_x (f(x) + \frac{1}{2}\|x - v\|_2^2) = \arg\min_{x \in \mathcal{C}} \frac{\|x-v\|_2^2}{2}$, which is a projection operator to $\mathcal{C}$. The minimizer might not be necessarily unique. However, under certain condition of C, there exists a unique minimizer (for example, C is a convex set).

Example 2.1 shows an important relationship between the notions of a proximal operator and that of a projection operator: a proximal operator is a generalized version of a projection operator. As a result, many properties of projection operators will be expected for proximal operators. Below is a list of the main properties of proximal operators.

Main properties [23]

1) Separable sum. If f is separable across two variables, so $f(x, y) = g(x) + h(y)$, then

$$\mathbf{prox}_f(v, w) = (\mathbf{prox}_g(v), \mathbf{prox}_h(w)) \tag{5}$$

2) Postcomposition. If $f(x) = \alpha\varphi(x) + b$, with $\alpha > 0$, then

$$\mathbf{prox}_{\lambda f}(v) = \mathbf{prox}_{\alpha\lambda\varphi}(v) \tag{6}$$

3) Precomposition. If $f(x) = \varphi(\alpha x + b)$ with $\alpha \neq 0$, then

$$\mathbf{prox}_{\lambda f}(v) = \frac{1}{\alpha}(\mathbf{prox}_{\alpha^2\lambda\varphi}(\alpha v + b) - b) \tag{7}$$

4) Affine addition. If $f(x) = \varphi(x) + a^T x + b$, then

$$\mathbf{prox}_{\lambda f}(v) = \mathbf{prox}_{\lambda\Phi}(v - \lambda a) \tag{8}$$

5) Regularization. If $f(x) = \varphi(x) + \frac{\rho}{2}\|x - a\|_2^2$, then

$$\mathbf{prox}_f(v) = \mathbf{prox}_{\tilde{\lambda}\varphi}((\tilde{\lambda}/\lambda)v + (\rho\tilde{\lambda})a) \tag{9}$$

where $\tilde{\lambda} = \lambda/(1 + \lambda\rho)$.
6) Fixed point. The point x^* minimizes a convex function f if and only if: $x^* = \mathbf{prox}_f(x^*)$

We will only prove the last property, which is also the most important since it shows how an optimization algorithm is solved with fixed point iteration.

Proof. In this proof, we assume f is subdifferentiable on its domain for simplicity despite of the fact that the property still holds in more general setting. Recall that a function $f : \mathbb{R}^n \mapsto \mathbb{R}$ is subdifferentiable if $\forall x \in \mathbf{dom} f \subset \mathbb{R}^n, \exists g \in \mathbb{R}^n$ such that $\forall z \in \mathbf{dom} f$:

$$f(z) - f(x) \geq g^T(z - x), \tag{10}$$

The set of all vectors g satisfying this property at a point x is denoted as $\partial f(x)$.

Firstly, if x^* minimizes f (not necessarily convex), $f(x^*) < f(x) + \frac{1}{2}\|x - x^*\|^2, \forall x \neq x^*$. Therefore, $\mathbf{prox}_f(x^*) = x^*$.

To show the converse, we use the subdifferential characterization of the minimizer of a convex function: x is a minimizer of a convex function g if and only if $0 \in \partial g(x)$. The point $\tilde{x}$ is a minimizer of $g : x \mapsto f(x) + \frac{1}{2}\|x - x^*\|_2^2$ if and only if:

$$0 \in \partial g(\tilde{x}) = \partial f(\tilde{x}) + (\tilde{x} - x^*) \tag{11}$$

If $x^* = \text{prox}_f(x^*)$, then (11) holds with $\tilde{x} = x^*$. Hence, we have $0 \in \partial f(x^*)$. Therefore, $x^* \in \arg\min f$. □

Thanks to this property of proximal operator, the search for the minimum of a function could be solved by finding the fixed point of the proximal operator. Optimization algorithms based on this principle are called proximal algorithms. They contain a huge variety of algorithms and could be further categorized into subclasses. Introducing all of these methods would be out of the scope of this report. The focus is on the most important algorithms exploiting proximal operators. Their pseudo-code are displayed in Algorithm 1 and Algorithm 2.

Algorithm 1 Proximal Minimization algorithm [23]

procedure PROXIMAL_MINIMIZATION_ALGORITHM($f(.), \lambda$)
    Initialize a solution x_0.
    $k \leftarrow 0$
    while *terminating condition is not met* **do**
        $x_{k+1} \leftarrow \mathbf{prox}_{\lambda f}(x_k)$.
        $k \leftarrow k + 1$.
    end while
end procedure

Algorithm 2 Proximal Gradient algorithm [23]

procedure PROXIMAL_GRADIENT_ALGORITHM($f(.) = g(.) + h(.), \lambda$)
    Initialize a solution x_0.
    $k \leftarrow 0$
    while *terminating condition is not met* **do**
        $x_{k+1} \leftarrow \mathbf{prox}_{\lambda g}(x_k - \lambda \nabla h(x_k))$.
        $k \leftarrow k + 1$.
    end while
end procedure

There are two algorithms: the Proximal Minimization and the Proximal Gradient algorithms. While the former is just a direct application of the fixed point property, the latter combines some ingredient of gradient descent method. Both methods have their own strengths: Algorithm 1 is simple and more direct, especially when the proximal operator could be effectively calculated. On the other hand, Algorithm 2 is useful when one could split the objective function f into the sum of two functions g, h such that h is differentiable and the proximal operator of g is easy to manipulate.

Remark 2.1. *In Algorithm 2, if $g(x) = I_{\mathcal{C}}(x)$, $\mathbf{prox}_{\lambda g}(x)$ is in fact a projection operator. The algorithm will become projected gradient descent, a well-known method to solve optimization problems with constraints.*

2.2 Linear inverse problems with sparsity constraints

The basic of linear inverse problem leads us to study the problem of the following form:

$$Ax = y + e \tag{12}$$

where e is unobserved noise (pertubation), $A \in \mathbb{R}^{m \times n}, y \in \mathbb{R}^m$ is a known signal and x is the vector we would like to estimate.

A classical approach to linear inverse problem is to consider the least squares solution of (12):

$$\hat{x}_{LS} = \arg\min_x \|Ax - y\|^2 \tag{13}$$

However, the formula is rarely used in practice since the naive solution $A^{-1}b$ is unstable when A is ill-conditioned (for example, A is (or near) singular). Therefore, regularization is employed to overcome such difficulty. In general, regularization replaces the ill-conditioned original problem by a nearby well-conditioned one. Popular regularization techniques are Tikhonov (14) and norm L_1 (15).

$$\hat{x}_{RLS} = \arg\min_x \frac{1}{2}\|Ax - y\|^2 + \lambda \|Lx\|^2 \tag{14}$$

$$\hat{x}_{L1} = \arg\min_x \frac{1}{2}\|Ax - y\|^2 + \lambda\|x\|_1 \tag{15}$$

The latter received great attention since it allows the sparsity promotion for the solution.

From an optimization viewpoint, both problems are convex. Therefore, many convex optimization algorithms could be employed to deal with (14) and (15). However, real applications of (14) and (15) usually involve images, which makes the dimension of the problem inherently large. This hinders the effectiveness of popular convex optimization algorithms such as interior point methods. Relatively cheaper operators have been come up with to handle such challenges. The report will recall an algorithm named Iterative Shrinkage Thresholding Algorithm [6] and one of its variants Fast Iterative Shrinkage Thresholding Algorithm [3]. Both of them are straightforward applications of proximal operators.

Another version of linear inverse problem involving norm L_0 is:

$$\begin{aligned} \underset{x\in\mathbb{C}^m}{\text{minimize}} \quad & \|Ax - y\|^2 \\ \text{subject to:} \quad & \|x\|_0 \leq s \end{aligned} \tag{16}$$

where $\|(x)\|_0 = |\{x_i | x_i \neq 0, 1 \leq i \leq m\}|$ and $s \ll m$.

This problem is non-convex and an optimal solution is not guaranteed to be achievable with polynomial algorithms (except under certain conditions on A and y). Algorithms for (16) typically involve hard thresholding, another proximal operator in disguise. This family containing several popular algorithms such as Iterative Hard Thresholding [4], Compressive Sensing Matching Pursuit [21] and Hard Thresholding Pursuit [12]. They will be all introduced in this report.

2.2.1 Algorithms using proximity operators: algorithms for norm L_1.

Iterative Shrinkage Thresholding Algorithm (ISTA) [6] and its variant Fast Iterative Shrinkage Thresholding Algorithm (FISTA) [3] are algorithms solving problem (15). Both algorithms work around an iterative operator, for example with ISTA

$$x_{k+1} = \mathcal{T}_{\lambda t}(x_k - tA^T(Ax_k - b)) \tag{17}$$

where $\mathcal{T}_\alpha : \mathbb{R}^m \to \mathbb{R}^m$ is the shrinkage operator defined by:

$$\mathcal{T}_\alpha(x)_i = (|x_i| - \alpha)_+ \text{sgn}(x_i) \tag{18}$$

It is easier to see that the operator involves some gradient descent spirit since $A^T(Ax - b)$ is the gradient of $\frac{1}{2}\|Ax - b\|^2$. The scalar t could be seen as the learning rate of the algorithm.

In fact, ISTA and FISTA could deal with more general setting where the objective function has the form:

$$F(x) = f(x) + g(x) \tag{19}$$

where $g : \mathbb{R}^m \to \mathbb{R}$ is a (possibly nonsmooth) convex function and $f : \mathbb{R}^m \to \mathbb{R}$ is continuously differentiable and f has Lipschitz continuous gradient with Lipschitz constant $L(f)$, i.e:

$$\|\nabla f(x) - \nabla f(y)\| \leq L(f)\|x - y\| \tag{20}$$

for every $x, y \in \mathbb{R}^m$. In the instance of problem (15), $f(x) = \frac{1}{2}\|Ax - b\|^2$ and $g(x) = \|x\|_1$.

Let

$$Q_L(x, y) = f(y) + \langle x - y, \nabla f(y)\rangle + \frac{L}{2}\|x - y\|^2 + g(x) \tag{21}$$

be a quadratic approximation of $F(x)$ and

$$\begin{aligned} p_L(y) &= \arg\min_x \left(Q_L(x, y)\right) \\ &= \arg\min_x \left(f(y) + \langle x - y, \nabla f(y)\rangle + \frac{L}{2}\|x - y\|^2 + g(x)\right) \\ &= \arg\min_x \left(g(x) + \frac{L}{2}\left\|x - \left(y - \frac{1}{L}\nabla f(y)\right)\right\|^2\right). \end{aligned} \tag{22}$$

The function $y \mapsto p_L(y)$ could be seen as a proximal operator where the original function $F(x) = f(x) + g(x)$ is approximated by the first order approximation $\hat{F}(x) = f(y) + \langle x - y, \nabla f(y) \rangle + g(x)$. When $f(x) = \frac{1}{2}\|Ax - b\|^2$ and $g(x) = \lambda\|x\|_1$, we have:

$$\begin{aligned} p_L(y) &= \arg\min_x \left(\lambda\|x\|_1 + \frac{L}{2} \left\| x - \left(z - \frac{1}{L}\nabla f(y) \right) \right\|^2 \right) \\ &= \arg\min_x \left(\lambda\|x\|_1 + \frac{L}{2} \left\| x - \left(y - \frac{1}{L}A^T(Ay - b) \right) \right\|^2 \right) \\ &= \arg\min_{x_i, i=1,\ldots,m} \sum_{i=1}^{m} \lambda|x_i| + \frac{L}{2} \left(x_i - \left(y - \frac{1}{L}A^T(Ay - b) \right)_i \right)^2 \end{aligned} \tag{23}$$

Since coordinates are separated, we could solve them separately. Consider $h : \mathbb{R} \to \mathbb{R}, h(x) = |x|$. The result is:

$$\begin{aligned} p_L(y)_i &= \arg\min_{x_i} \left\{ \lambda|x_i| + \frac{L}{2} \left(x_i - \left(y - \frac{1}{L}A^T(Ay - b) \right)_i \right)^2 \right\} \\ &= \text{prox}_{\frac{\lambda}{L}h} \left((y - \frac{1}{L}A^T(Ay - b))_i \right) \\ &= \mathcal{T}_{\frac{\lambda}{L}} \left(y_i - \frac{1}{L} \left(A^T(Ay - b) \right)_i \right) \end{aligned} \tag{24}$$

which is identical with (17) with $t = \frac{1}{L}$.

ISTA and FISTA pseudocodes are depicted in Algorithm 3 and Algorithm 4.

Algorithm 3 Iterative Shrinkage Thresholding Algorithm [6]

procedure ITERATIVE_SHRINKAGE_THRESHOLDING_ALGORITHM(A, y, t)
    Initialize a solution $x_0 \in \mathbb{R}^n$.
    $k \leftarrow 0$.
    while *terminating condition is not met* **do**
        $x_{k+1} \leftarrow p_L(x_{k-1})$.
        $k \leftarrow k + 1$.
    end while
end procedure

Algorithm 4 Fast Iterative Shrinkage Thresholding Algorithm [3]

procedure FAST_ITERATIVE_SHRINKAGE_THRESHOLDING_ALGORITHM(A, y, L)
    Initialize a solution $y_1 = x_0 \in \mathbb{R}^n$.
    $t_1 \leftarrow 1$.
    $k \leftarrow 1$.
    while *terminating condition is not met* **do**
        $x_k \leftarrow p_L(y_k)$.
        $t_{k+1} \leftarrow \frac{1+\sqrt{1+4t_k^2}}{2}$.
        $y_{k+1} = x_k + \frac{t_k - 1}{t_{k+1}}(x_k - x_{k-1})$.
        $k \leftarrow k + 1$.
    end while
end procedure

The only difference between ISTA and FISTA is that: the shrinkage operator of FISTA is not applied directly to the previous point x_{k-1}, but to a linear combination between x_{k-1} and x_{k-2}. This idea

probably came from [22], where Nesterov proposed a gradient-based algorithm having optimal worst case convergence rate in oracle model. In [3], ISTA and FISTA are proved to have convergence rate $O(\frac{1}{k})$ and $O(\frac{1}{k^2})$ respectively.

2.2.2 Algorithms using hard thresholding: algorithms for norm L_0.

Iterative Hard Thresholding (IHT) [4], Compressive Sensing Matching Pursuit (CoSAMP) [21] and Hard Thresholding Pursuit (HTP) [12] are developed to solve problem (16). In this setting, the main objective is to convert a signal of high dimension into a linear combination of small number of predefined measurements. All the predefined measurements are stored in a matrix $A \in \mathbb{C}^{m\times N}$. There are two problems needing addressed: How to construct A and given a signal $y \in \mathbb{C}^m$ and a measurement matrix A, how can one construct a sparse representation of y from A. All of those ingredients are formalized in problem (16).

Both IHT and CoSAMP are greedy algorithms. Let $H_s : \mathbb{R}^m \to \mathbb{R}^m$ be the hard thresholding operator keeping s components with largest absolute values (and setting others to zero) of a vector $x \in \mathbb{R}^m$. Then H_s is the proximal operator of function $h(x) = I_C(x)$ where $C = \{x|\|x\|_0 \leq s\}$. Given a vector x, denote supp(x) as the set of indices of x whose coefficients is nonzero. The pseudocodes of IHT and CoSAMP are displayed in Algorithm 5 and Algorithm 6 respectively.

Algorithm 5 Iterative Hard Thresholding [4]

procedure ITERATIVE_HARD_THRESHOLDING(A, y, s)
    Initialize a solution $x_0 = 0$.
    $k \leftarrow 0$
    while *terminating condition is not met* **do**
        $x_{k+1} \leftarrow H_s(x_k + A^*(y - Ax_k))$.
        $k \leftarrow k + 1$.
    end while
end procedure

Algorithm 6 Compressive Sensing Matching Pursuit [21]

procedure COMPRESSIVE_SENSING_MATCHING_PURSUIT(A, y, s)
    Initialize a solution $x_0 = 0$.
    $k \leftarrow 0$
    while *terminating condition is not met* **do**
        $U_{k+1} = \text{supp}(x_k) \cup \{\text{indices of } 2s \text{ largest entries of } A^*(y - Ax_k)\}$.
        $u_{k+1} = \arg\min\{\|y - Az\|_2, \text{supp}(z) \subseteq U_{k+1}\}$.
        $x_{k+1} = H_s(u_{k+1})$.
        $k \leftarrow k + 1$.
    end while
end procedure

In word, IHT performs a gradient descent step with unit learning rate and keeps the s largest coefficients. On the other hand, CoSAMP considers a larger support (at most $3s$ coefficients). It also uses projection and thresholding to update the solution. In practice, IHT is also modified by an adaptive learning rate (instead of unit learning rate like Algorithm 5) [12]. That leads to a variant of HTP introduced later in this introduction.

HTP is a finer combination of IHT and CoSAMP. It executes two steps in one iteration: hard thresholding with $H_s(x_k + A^*(y - Ax_k))$ and projection based on the support resulting from the hard thresholding operator. It has one more step compared to IHT and one less step compared to CoSAMP. Its pseudocode is displayed in Algorithm 7.

Algorithm 7 Hard Thresholding Pursuit [12]

procedure HARD_THRESHOLDING_PURSUIT(A, y, s)
  Initialize a solution $x_0 = 0$.
  $k \leftarrow 0$
  while *terminating condition is not met* **do**
    $U_{k+1} = \{\text{indices of } s \text{ largest entries of } x_k + A^*(y - Ax_k)\}$.
    $x_{k+1} = \arg\min\{\|y - Az\|_2, \text{supp}(z) \subseteq U_{k+1}\}$.
    $k \leftarrow k + 1$.
  end while
end procedure

The terminating conditions might vary: termination after certain number of iterations or the amount of improvement of the objective function being smaller than a threshold ϵ. The algorithm also has many variants, which turns out to be very useful in different problems. We will introduce two most important variants: Hard Thresholding Pursuit with step size (HTP$^\mu$) and Fast Hard Thresholding Pursuit (FHTP). HTP$^\mu$ is a more general version in comparison to HTP since it adds the step size μ to the update of $x_{k+1} = H_s(x_k + \mu A^*(y - Ax_k))$. The step size μ can be constant or adaptive. As proved in [12], under certain conditions of step size, one can prove the convergence of HTP$^\mu$ (Simple HTP does not converge with certain measurement matrices A). On the other hand, FHTP gets rid of orthogonal projection and replaces it with several gradient descent steps. The orthogonal projection is expensive and unstable to execute since it involves the calculation of matrix inverse (or pseudo inverse). Gradient step provides us a cheaper numerical method to search for the orthogonal projection. HTP$^\mu$ and FHTP are displayed in Algorithm 8 and Algorithm 9 respectively.

Algorithm 8 Hard Thresholding Pursuit with step size μ [12]

procedure HARD_THRESHOLDING_PURSUIT_STEP(A, y, s, μ)
  Initialize a solution $x_0 = 0$.
  $k \leftarrow 0$
  while *terminating condition is not met* **do**
    $U_{k+1} = \{\text{indices of } s \text{ largest entries of } x_k + \mu A^*(y - Ax_k)\}$.
    $x_{k+1} = \arg\min\{\|y - Az\|_2, \text{supp}(z) \subseteq U_{k+1}\}$.
    $k \leftarrow k + 1$.
  end while
end procedure

Algorithm 9 Fast Hard Thresholding Pursuit [12]

procedure FAST_HARD_THRESHOLDING_PURSUIT(A, y, s, μ)
  Initialize a solution $x_0 = 0$.
  $k \leftarrow 0$
  while *terminating condition is not met* **do**
    $U_{k+1} = \{\text{indices of } s \text{ largest entries of } x_k + \mu A^*(y - Ax_k)\}$.
    $u_{0,k+1} = H_s(x_k + \mu A^*(y - Ax_k))$
    for $l \in \{1, \ldots, N\}$ **do**
      $u_{l,k+1} = u_{l-1,k+1} + t_{l,k+1}[A^*(y - Au_{l-1,k+1})]_{U_{n+1}}$.
    end for
    $x_{k+1} = u_{N,k+1}$
    $k \leftarrow k + 1$.
  end while
end procedure

IHT, CoSAMP and HTP have many interesting theoretical results, most of them are under Restricted Isometry Property (RIP) assumption [12]. However, discussing them would be out of this report scope. Therefore, we conclude the introduction of IHT, CoSAMP and HTP.

2.3 Multilayer sparse matrix factorization with proximal algorithms

In the context of sparse matrix factorization, consider a linear operator corresponding to matrix $A \in \mathbb{R}^{m\times n}$. The objective is to find sparse factors $S_j \in \mathbb{R}^{a_j \times a_{j+1}}, j \in \{1,\dots,J\}$ with $a_1 = m, a_{J+1} = n$ such that $A = \prod_{i=1}^{J} S_j$. This leads to the following problem:

$$\underset{S_1,\dots,S_j}{\text{Minimize}} \underbrace{\|A - \prod_{j=1}^{J} S_j\|_F^2}_{\text{Data fidelity}} + \underbrace{\sum_{j=1}^{J} g_j(S_j)}_{\text{Sparsity-inducing penalty}} \tag{25}$$

where the sparsity-inducing penalty g_j is chosen as the indicator function $\delta_{\mathcal{E}_j}$ of constraint sets of interest $\mathcal{E}_j$. To further avoid the scaling ambiguities when constraint sets are (positively) homogeneous, one could also put a scalar factor λ before $\prod_{j=1}^{J} S_j$ and normalize all S_j such that $\|S_j\|_F = 1$. Therefore, $\mathcal{E}_j = \mathcal{N}_j \cap \mathcal{S}_j$ where $\mathcal{N}_j = \{S \in \mathbb{R}^{a_{j+1}\times a_j} : \|S\|_F = 1\}$ and $\mathcal{S}_j$ is an explicit or implicit sparsity constraint. Problem (25) becomes:

$$\underset{\lambda,S_1,\dots,S_j}{\text{Minimize}} \|A - \lambda \prod_{j=1}^{J} S_j\|_F^2 + \sum_{j=1}^{J} \delta_{\mathcal{E}_j}(S_j) \tag{26}$$

In [19], a proximal gradient algorithm was employed to solve problem (26). The idea is pretty straightforward since the objective function has already two components: $\|A - \lambda \prod_{j=1}^{J} S_j\|_F^2$ which is differentiable and $\sum_{j=1}^{J} \delta_{\mathcal{E}_j}(S_j)$ which could allow fast proximal operator (or projection operator) computation under certain choices of $\mathcal{E}_j$.

Example 2.2. *Let $\mathcal{E} = \{S, \|S\|_0 \le s, \|S\|_F = 1\}$ and $\mathcal{I}$ be the set of indices of the s largest absolute value of S. Then proximal operator of a matrix U w.r.t $\delta_{\mathcal{E}}$ is*

$$P_{\mathcal{E}}(U) = \frac{U_{\mathcal{I}}}{\|U_{\mathcal{I}}\|_F} \tag{27}$$

where $U_{\mathcal{I}}$ is the matrix whose entries match those of U on $\mathcal{I}$ and are set zero elsewhere.

Example 2.3. *let $\mathcal{E} = \{S \in \mathbb{R}^{p\times q} : \|S_i\|_0 \le s_i, \forall i \in \{1,\dots,p\}, \|S\|_F = 1\}$ where S_i is the i^{th} row of the matrix S. The proximal operator of a matrix U w.r.t $\delta_{\mathcal{E}}$ is*

$$P_{\mathcal{E}}(U) = \frac{U_{\mathcal{I}}}{\|U_{\mathcal{I}}\|_F} \tag{28}$$

where $\mathcal{I} = \cup_i \mathcal{I}_i$, $\mathcal{I}_i$ contains s_i entries of i^{th} row with largest absolute value, $\forall i \in \{1,\dots,p\}$.

Example 2.4. *More generally, let $\mathcal{E} = \{S \in \mathbb{R}^{p\times q} : \|S_{\mathcal{H}_i}\|_0 \le s_i, \forall i \in \{1,\dots,K\}, \|S\|_F = 1\}$ where $\{\mathcal{H}_1,\dots,\mathcal{H}_K\}$ forms a partition of the index set.*

The projection operator formula is

$$P_{\mathcal{E}}(U) = \frac{U_{\mathcal{I}}}{\|U_{\mathcal{I}}\|_F} \tag{29}$$

where $\mathcal{I} = \cup_i \mathcal{I}_i$, $\mathcal{I}_i \subseteq \mathcal{H}_i$ contains s_i entries of $U_{\mathcal{H}_i}$ with largest absolute value, $\forall i \in \{1,\dots,K\}$.

Taking $\{\mathcal{H}_1,\dots,\mathcal{H}_K\}$ as set of columns (or rows) will be equivalent to restrict the number of nonzero entries of columns (or rows) of S.

Remark 2.2. *All previous examples correspond to disjoint index sets $\mathcal{H}_i$. If $\mathcal{H}_i$ might overlap, the proximal operator becomes more difficult to calculate. The first contribution in this report is to deal with an instance of such a problem in Section 3.*

The proximal gradient algorithm for (26) is called Proximal Alternating Linearized Minimization (PALM) in [19] and displayed in Algorithm 10.

Algorithm 10 PALM for Multi-layer Sparse Approximation [19]

procedure PALM($A, J, \mathcal{E}_j, \gamma, N$)
  Initialize $\{S_i^0\}_{j=1}^J, \lambda_0$.
  for $i \in \{0, \dots, N-1\}$ **do**
    for $j \in \{1, \dots, J\}$ **do**
      $S_j^{i+1} \leftarrow P_{\mathcal{E}_j}(S_j^i - \frac{1}{c_j^i}\nabla_{S_j}\|A - \lambda(\prod_{l=1}^{j-1} S_l^{i+1})(\prod_{l=j}^{J} S_l^i)\|_F^2)$
    end for
    Update λ_{i+1}
  end for
  Return $\lambda, S_1, \dots, S_J$.
end procedure

PALM is not totally a proximal gradient algorithm. Each time, it optimizes the parameters in only one matrix while fixing the others. This strategy is similar to coordinate descent optimization.

In fact, PALM performs poorly in practice with various initializations. Factorization of well-known linear operators such as the Discrete Fourier Transform or the Hadamard Transform is impossible with PALM. The author in [19] proposed a more refined method called hierarchical factorization. More precisely, instead of trying to factorize A into a product of multiple matrices S_j, we restrict the number of factors to two at a time. Consecutively factorizing newly-obtained matrices will eventually lead to the factorization of multiple matrices. PALM is employed as a subprocedure in each iteration. The pseudo-code for hierarchical factorization is displayed in Algorithm 11

Algorithm 11 Hierarchical factorization for Multi-layer Sparse Approximation

procedure HIERARCHICAL_FACTORIZATION($A, J, \mathcal{E}_j, \tilde{\mathcal{E}}_j, \gamma, N$)
  $T_0 \leftarrow A$.
  for $l \in \{1, \dots, J-1\}$ **do**
    Factorize the residual T_{l-1} into 2 factors:
    $\lambda', \{F_2, F_1\} = \text{PALM}(T_{l-1}, 2, \{\mathcal{E}_j, \tilde{\mathcal{E}}_j\}, \gamma, N)$
    $T_l = \lambda' F_2, S_l = F_1$
    Use PALM to globally optimize $\lambda, S_1, \dots, S_l, T_l$
  end for
  $S_J = T_{J-1}$
  Return $\lambda, S_1, \dots, S_J$.
end procedure

Hierarchical factorization does not successfully reconstruct the Hadamard Transform with the following setting: A is the Hadamard Transform of size $n = 2^N, \tilde{\mathcal{E}}_i = \{T \in \mathbb{R}^{n\times n}, \|T\|_0 \leq \frac{n^2}{2^i}, \|T\|_F = 1\}$ and $\mathcal{E}_i = \{S \in \mathbb{R}^{n\times n}, \|S\|_0 \leq 2n, \|S\|_F = 1\}$ [19]. The proximal operator corresponding to $\mathcal{E}_i$ ($\tilde{\mathcal{E}}_i$) is to keep the $2k$ ($\frac{n^2}{2^i}$) coefficients with largest absolute value of the matrix while setting others to zero (as explained in the Example 2.2). However, this proximal operator could easily return a zero row or column in the factorization, which makes the product of two matrices singular (meanwhile the Hadamard Transform is a full rank matrix). It is thus preferable to make sure that each row and column after proximal operator is nonzero. This strategy is adopted in [19] and named SPLINCOL. The support found by SPLINCOL(k) is $S_C \cup S_R$ where S_C and S_R are respectively unions of k coefficients with largest absolute value per column and per row. The procedure of SPLINCOL is shown in algorithm Algorithm 12.

Algorithm 12 Proximal operator SPLINCOL

procedure SPLINCOL(A, k)
  $\mathcal{I} = \emptyset$.
  for Each row of A **do**
    $\mathcal{I} = \mathcal{I} \cup \{k$ coefficients with largest absolute value of the current row$\}$
  end for
  for Each column of A **do**
    $\mathcal{I} = \mathcal{I} \cup \{k$ coefficients with largest absolute value of the current column$\}$
  end for
  Return $\frac{U_{\mathcal{I}}}{\|U_{\mathcal{I}}\|_F}$
end procedure

Hierarchical factorization with SPLINCOL as proximal operators successfully recover the Hadamard Transform [19]. However, it did not succeed with the Discrete Fourier transform yet [19]. The main challenge is the combinatorial explosion of the set of supports of a sparse structure. Each fixed support has its own (local) minimum that is not easy to escape with traditional first order optimization method. In addition, the current algorithm is very sensitive to initialization. This will be partly addressed in Section 4.

2.4 Summary and main challenges

Proximal operators and proximal algorithms found their utilities in many important questions such as linear inverse problem or sparse matrix factorization as we have shown. In certain cases such as linear inverse problem with Tikhonov and L_1 norm regularization, the setting is convex. Thus, algorithms based on proximal operators such as ISTA and FISTA will deliver optimal solutions. On the other hand, when a linear inverse problem involves L_0 regularization or matrix factorization, the problems become nonconvex. Proximal algorithms are theoretically proved to find optimal solutions under certain assumptions for linear inverse problems with L_0 regularization. Sparse matrix factorization, yet, is still under construction. Not only is the problem of matrix factorization nonconvex, but the sparsity constraint also imposes inherently many combinatoric challenges leading to many NP-hard problems in our conjecture. Therefore, one could say that matrix factorization with sparsity constraints possesses a lot of characteristics to resist an effective algorithm. This report will serve as our first step to try to utilize proximal operators to construct such a method.

3 Generalized Hungarian method - A projection for proximal algorithms

This section introduces a new proximal operator: projection onto the space of k-regular sparse matrices. As described later, this sparse pattern is popular among handmade linear operators and it will come in handy for many algorithms involving proximal operators.

3.1 Problem formulation

The discussion continues with the question: How to enhance PALM and hierarchical factorization (Algorithm 10 and Algorithm 11) so that they could reliably factorize familiar (yet important) transforms such as the Discrete Fourier Transform and the Hadamard Transforms. A natural idea is to make the conditions $\mathcal{E}_l, \tilde{\mathcal{E}}_l$ (Section 2.3) more informative since the ones used in [19] are quite loose in the sense that the space of matrix covered by $\mathcal{E}_l, \tilde{\mathcal{E}}_l$ are still large. Moreover, they can make projected matrix have low rank, which potentially drive algorithms into bad stationary points. Luckily, those transforms do have some specific structure that would be taken advantage of. We first introduce some important definitions.

Definition 3.1 (Valid support of a matrix)**.** *A valid support of a matrix $U \in \mathbb{R}^{n\times n}$ is the set of the matrix indices V satisfying $\forall (i,j) \notin V$, $U[i,j] = 0$.*

Remark 3.1. *A matrix U might admit many valid supports. The coefficient corresponding to an index in the support can still be zero.*

It is usually easier to visualize the support of a matrix by a binary matrix. Consider $U \in \mathbb{R}^{m\times n}$, the support of U could be represented by a matrix supp$(U) \in \{0,1\}^{m\times n}$ such that supp$(U)_{ij} = 1$ if and only if (i,j) is contained in the support of U.

Example 3.1. *Here is an example of U and an valid support supp(U)*

$$U = \begin{pmatrix} 0.2 & 0.1 & 0 \\ 0 & 0.5 & 0.5 \\ 0 & 0 & 0.5 \end{pmatrix}, supp(U) = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \tag{30}$$

and an example that does not satisfy Definition 3.1

$$U = \begin{pmatrix} 0.2 & 0.1 & 0 \\ 0 & 0.5 & 0.5 \\ \mathbf{0.5} & 0 & 0.5 \end{pmatrix}, supp(U) = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ \mathbf{0} & 1 & 1 \end{pmatrix} \tag{31}$$

Definition 3.2 (k-regular sparse matrix)**.** *A matrix $U \in \mathbb{R}^{n\times n}$ (or $\mathbb{C}^{n\times n}$) admits a valid support supp(U) whose rows and columns have exactly k coefficients of value 1 is called a k-regular sparse matrix.*

Example 3.2. *Examples of a k-regular sparse matrix:*

$$A = \begin{pmatrix} 0.5 & 0.5 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0.5 & 0.5 \\ 0.5 & 0 & 0 & 0.5 \end{pmatrix}, supp(A) = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} \tag{32}$$

In general, for all $k \le n, k, n \in \mathbb{N}$, there always exists a k-regular sparse matrix of size $n \times n$. Indeed, consider binary matrix B satisfying $B_{ij} = 1$ if and only if $j = i + l \mod n, \forall 0 \le l \le k-1$. $B \in \mathbb{R}^{n\times n}$ is a k-regular sparse matrix.

$$B = \begin{pmatrix} 1 & 1 & 1 & \dots & 0 & 0 \\ 0 & 1 & 1 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & 1 & \dots & 1 & 1 \\ 1 & 1 & 1 & \dots & 0 & 1 \end{pmatrix} \tag{33}$$

We introduce an useful lemma:

Lemma 3.1 (Decomposition lemma)**.** *Any k-regular sparse matrix admits a support that can be decomposed into sum of k permutation matrices.*

Remark 3.2. *In Example 3.2, the support of A could be written as:*

$$supp(A) = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \tag{34}$$

The proof of Lemma 3.1 is based on graph matching theory. We will shortly state the definition of graph matching and Hall theorem, which will play crucial roles in the proof.

Definition 3.3 (Matching and perfect matching)**.** *Let $G = (V, E)$ be a bipartite graph where $V = S \cup T, S \cap T = \emptyset, E \subset U \times V \cup V \times U$. A matching is a set $M \subseteq E$ such that no endpoint of any edge $e \in M$ is shared with another edge $e' \in M$. A perfect matching (or complete matching) M is a matching in which $|S| = |T| = |M|$.*

Theorem 3.2 (Hall marriage theorem, [14])**.** *Let $G = (V = S \cup T, E)$ be a bipartite graph where $|S| = |T|$. Suppose that for all subsets $U \subseteq T$, we have $|\Gamma(U)| \ge |U|$ ($\Gamma(U) = \{y \in S | (x,y) \in E$ for some $x \in U\}$). Then G admits a perfect matching (or complete matching).*

Proof of Lemma 3.1. Let $A \in \mathbb{C}^{n\times n}$ be a k-regular sparse matrix with a support A satisfying the property in Definition 3.2. A representing graph of supp(A) is $G = (V, E)$ which is a bipartite graph

where $V = S \cup T, |S| = |T| = n$, i^{th} element of S (resp. T) corresponds to i^{th} row (resp. column) of A. $(u, v) \in E \iff \text{supp}(A)[u, v] = 1$. Since each permutation matrix is equivalent to a perfect matching in G, it is sufficient to prove that the set of edges of G could be partitioned into k perfect matchings.

The proof is by induction. Given G a bipartite graph representing A a k-regular sparse matrix, we will prove G satisfies the condition of Hall theorem (Theorem 3.2) by contradiction using the Dirichlet principle. Assume that there is a subset U of T such that $|\Gamma(U)| < |U|$. Since the matrix is k-regular, each vertex of V has degree k and thus, the number of edges between U and $\Gamma(U)$ is exactly $k|U|$. By the Dirichlet principle, there exists one vertex in $\Gamma(U)$ having degree at least $k|U|/\Gamma(U) > k$. This absurdity proves that our assumption is false and the condition of Hall Theorem is satisfied. Thus, there exists one perfect matching in G. Remove this perfect matching from E. The degree of each vertex of the newly-obtained graph decreases by 1, which is $k - 1$. Therefore, each row (column) of the represented matrix has $k - 1$ coefficients in the support, which means the matrix is $(k - 1)$-regular. By induction, we could further decompose to partition the support of the original matrix into sum of k permutation matrices. □

Let $\mathcal{E}_l, \tilde{\mathcal{E}}_l$ be the set of 2-regular and $n/2^l$-regular sparse matrices of unit norm respectively. The Discrete Fourier Transform and the Hadamard Transform did allow such a factorization (more detail in [10]). The constraint is much more restrictive than the ones in [19] (where $\mathcal{E}_l, \tilde{\mathcal{E}}_l$ are simply the set of unit norm matrices having at most $2n$ and $n^2/2^l$ non-zero entries respectively). This part is devoted to answer the question: how can we project a given matrix U onto the set of k-regular sparse matrix of unit norm effectively?

The following part works with the set of complex number ($\mathbb{C}$) to make the statements and proofs as general as possible. We define the projection of a matrix $U \in \mathbb{C}^{n\times n}$ into a closed set $\mathcal{E} \subset \mathbb{C}^{n\times n}$ as:

$$P_{\mathcal{E}}(U) = \arg\min_{S\in\mathcal{E}} \|S - U\|_F^2 \tag{35}$$

It is noticeable that the result of the projection might not be unique.

Lemma 3.3. *Let I be the set of all tuples of $n \times k$ pairs of indices in which each row and column contains exactly k elements. Let $\mathcal{I} \in I$ be a tuple whose $\sum_{(i,j)\in\mathcal{I}} |U_{ij}|^2$ is maximized. The projection of U to the closest k-regular sparse matrix of unit Frobenius norm is:*

$$P_{\mathcal{E}}(U) = \frac{U_{\mathcal{I}}}{\|U_{\mathcal{I}}\|_F} \tag{36}$$

Proof. Let $\text{vec}(U)$ and $\text{vec}(S)$ be the vectorization of the matrices $U, S \in \mathbb{C}^{n\times m}$ where S is of unit Frobenius norm. Then we have:

$$\begin{aligned} \|S - U\|_F^2 &= (\text{vec}(S) - \text{vec}(U))^*(\text{vec}(S) - \text{vec}(U)) \\ &= 1 + \|U\|_F^2 - \text{vec}(S)^*\text{vec}(U) - \text{vec}(U)^*\text{vec}(S) \end{aligned} \tag{37}$$

Since $1 + \|U\|_F^2$ is fixed, it is sufficient to maximize $\text{vec}(S)^*\text{vec}(U) + \text{vec}(U)^*\text{vec}(S)$. From this point, we will use S to indicate $\text{vec}(S)$ to improve the readability. Given a candidate support $\mathcal{J}$ of S, we have:

$$
\begin{aligned}
\text{vec}(S)^*\text{vec}(U) + \text{vec}(U)^*\text{vec}(S) &= 2\sum_{i\in\mathcal{J}} (\Re(U_i)\Re(S_i) + \Im(U_i)\Im(S_i)) \\
&\leq 2\sqrt{\left(\sum_{i\in\mathcal{J}} (\Re(U_i)^2 + \Im(U_i)^2)\right)\left(\sum_{i\in\mathcal{J}} (\Re(S_i)^2 + \Im(S_i)^2)\right)} \\
&= 2\sqrt{\left(\sum_{i\in\mathcal{J}} |U_i|^2\right)\left(\sum_{i\in\mathcal{J}} |S_i|^2\right)} \\
&= 2\sqrt{\sum_{i\in\mathcal{J}} |U_i|^2}
\end{aligned}
\tag{38}
$$

The equality holds when $S_i = \frac{U_i}{\sqrt{\sum_{i\in\mathcal{J}} |U_i|^2}}$. To maximize $\text{vec}(S)^*\text{vec}(U) + \text{vec}(U)^*\text{vec}(S)$, it is sufficient to find support $\mathcal{J} \in I$ such that $\sum_{i\in\mathcal{J}} |U_i|^2$ is maximized. Thus, the proof is concluded. □

Remark 3.3. *Without the condition of unit norm, the projection of U to the set of k-regular sparse matrices is simply $U_{\mathcal{I}}$ with $\mathcal{I}$ defined in Lemma 3.3.*

Suppose that we want to find a projection of matrix $U \in \mathbb{C}^{n\times n}$ onto the space of k-regular sparse unit norm matrices. Thanks to Lemma 3.3, the problem of projection is reducible to another discrete (combinatorial) problem. Let $c \in \mathbb{R}_+^{n\times n}$ where $c_{ij} = |U_{ij}|^2$. Then it is sufficient to solve the following problem:

$$
\begin{aligned}
\underset{x}{\text{maximize}} \quad & \sum_{1\leq i,j\leq n} c_{ij}x_{ij} \\
\text{subject to:} \quad & \sum_{j=1}^{n} x_{ij} = k \qquad \forall i = 1, n \\
& \sum_{i=1}^{n} x_{ij} = k \qquad \forall j = 1, n \\
& x_{ij} \in \{0, 1\} \qquad \forall 1 \leq i, j \leq n
\end{aligned}
\tag{39}
$$

This is an integer programming problem, thus, NP-hard in general. If we remove the integer constraint, this problem becomes an instance of capacitated transportation problem [9, Chapter 3]. It could be also formalized as min-cost flow [9, Chapter 4] or more generally, a linear programming problem. Therefore, it is quite well-researched.

A familiar technique for integer programming problems is to relax the integer constraints. Problem (40) is the relaxed version of problem (39). We attempt to firstly prove that there exists an optimal integer solution for the relaxed problem (40). This is easy if we view the problem as a min-cost flow instance and apply [9, Theorem 4.5 ,Chapter 4]). On the other hand, there is also a direct proof for this specific problem.

$$
\begin{aligned}
\underset{x}{\text{maximize}} \quad & \sum_{1\leq i,j\leq n} c_{ij}x_{ij} \\
\text{subject to:} \quad & \sum_{j=1}^{n} x_{ij} = k \qquad \forall i = 1, n \\
& \sum_{i=1}^{n} x_{ij} = k \qquad \forall j = 1, n \\
& 0 \leq x_{ij} \leq 1 \qquad \forall 1 \leq i, j \leq n
\end{aligned}
\tag{40}
$$

Lemma 3.4 (Existence of integer solution)**.** *Problem* (40) *has an integer maximizer, which is also a maximizer of problem* (39)*.*

Proof. If problem (40) has an integer maximizer, the integer solution clearly maximizes problem (39). It is sufficient to prove the existence of an optimal integer solution for problem (40).

Suppose that x is an optimal solution of problem (40) and that there exist (i_1, j_1) such that $x_{i_1 j_1} \notin \mathbb{Z}$. Consequently, there exists $i_2 \neq i_1$ such that $x_{i_2 j_1} \notin \mathbb{Z}$ (since the sum of each column is an integer). Similarly, there must exist $j_2 \neq j_1$ such that $x_{i_2 j_2} \notin \mathbb{Z}$ (since the sum of each row is integer). Applying the same argument, one could travel through indices by alternating between rows and columns. Since the number of entries is finite, the existence of a cycle is clear. Moreover, if we define G a complete bipartite graph which has $2n$ vertices in total (and n vertices for each part), our way of traversing described above is equivalent to traverse through the edges of G. Since G is bipartite, the cycle must has even length $2\ell, \ell \in \mathbb{N}$.

Indexing those pairs of indices from 1 to 2ℓ, one could increase x_{ij} by a small enough amount of ϵ for the $(2i+1)$th indices and decrease it by ϵ for the $(2i)$th indices without violating any condition. Since x is optimal, this could not change the objective function (or otherwise we could change ϵ to $-\epsilon$ and obtain a better solution). Choosing the smallest ϵ so that at least one of indices in the cycle becomes integer. we now have a new valid optimal solution x' which has at least one more integer element. By repeating this process, one could finally obtain an optimal integer solution. □

3.2 Generalized Hungarian Method

In the statement of problem (39), when $k = 1$, the problem becomes the problem of maximum weighted matching or assignment problem [18]. The famous algorithm introduced in [18] was named the Hungarian method. That is why in this more general setting, our proposed algorithm is named the Generalized Hungarian method. After deriving the Lagrange dual problem [5, Chapter 5], this report will process into the algorithm and its analysis.

3.2.1 Duality

In this part, we will try to construct an algorithm to deal with the problem (39). Firstly, we derive the duality of problem (40). Here, we will MINIMIZE the objective function instead of maximizing. The new problem of minimization is equivalent with the maximization one because we could change c_{ij} into $-c_{ij}$ and use an algorithm for the minimization problem to solve the problem (39)). We start by reformulating problem (40)

$$\begin{aligned}
\underset{x}{\text{minimize}} \quad & \sum_{1 \leq i,j \leq n} c_{ij} x_{ij} \\
\text{subject to:} \quad & \sum_{j=1}^{n} x_{ij} = k \qquad \forall i = 1, n \\
& \sum_{i=1}^{n} x_{ij} = k \qquad \forall j = 1, n \\
& 0 \leq x_{ij} \leq 1 \qquad \forall 1 \leq i, j \leq n
\end{aligned} \tag{41}$$

Remark 3.4. *This problem is totally feasible given that the matrix is square. Indeed, a simple feasible solution is matrix B in Example 3.2.*

Starting from Lagrange form

$$
\begin{aligned}
&\mathcal{L}(x, \lambda, \alpha, \beta, \gamma) \\
&= \sum_{i,j} c_{ij} x_{ij} - \sum_{i=1}^{n} \lambda \left(\sum_{j=1}^{n} x_{ij} - k \right) - \sum_{j=1}^{n} \alpha_j \left(\sum_{i=1}^{n} x_{ij} - k \right) - \beta_{ij} x_{ij} + \sum_{i,j} \gamma_{ij} (x_{ij} - 1) \\
&= \sum_{i,j} x_{ij} (c_{ij} - \lambda_i - \alpha_j - \beta_{ij} + \gamma_{ij}) + k(\sum_{i=1}^{n} \lambda_i + \sum_{j=1}^{n} \alpha_j) - \sum_{i,j} \gamma_{ij}
\end{aligned} \tag{42}
$$

the Lagrange dual problem [5, Chapter 5] is thus

$$
\begin{aligned}
\underset{\lambda,\alpha,\beta,\gamma}{\text{maximize}} \quad & k(\sum_{i=1}^{n} \lambda_i + \sum_{j=1}^{n} \alpha_j) - \sum_{i,j} \gamma_{ij} & \\
\text{subject to:} \quad & \gamma_{ij} \geq 0 & \forall 1 \leq i, j \leq n \\
& \beta_{ij} \geq 0 & \forall 1 \leq i, j \leq n \\
& c_{ij} - \lambda_i - \alpha_j - \beta_{ij} + \gamma_{ij} = 0 & \forall 1 \leq i, j \leq n.
\end{aligned} \tag{43}
$$

Since β_{ij} does not appear in the objective function, we can further simplify the dual problem as:

$$
\begin{aligned}
\underset{\lambda,\alpha,\beta,\gamma}{\text{maximize}} \quad & k(\sum_{i=1}^{n} \lambda_i + \sum_{j=1}^{n} \alpha_j) - \sum_{i,j} \gamma_{ij} & \\
\text{subject to:} \quad & \gamma_{ij} \geq 0 & \forall 1 \leq i, j \leq n \\
& c_{ij} - \lambda_i - \alpha_j + \gamma_{ij} \geq 0 & \forall 1 \leq i, j \leq n
\end{aligned} \tag{44}
$$

If we fix λ and α, the best value for γ_{ij} is $\gamma_{ij} = \max(0, \lambda_i + \alpha_j - c_{ij})$ and thus $-\gamma_{ij} = \min(0, c_{ij} - \lambda_i - \alpha_j)$. The most simplified version of the dual problem is:

$$
\underset{\lambda_i,\alpha_j}{\text{maximize}} \; k(\sum_{i=1}^{n} \lambda_i + \sum_{j=1}^{n} \alpha_j) + \sum_{i,j} \min(0, c_{ij} - \lambda_i - \alpha_j) \tag{45}
$$

Lemma 3.5 (Sufficient condition of optimal solution of problem (45))**.** *Let $\mathcal{J}_i = \{j | \lambda_i + \alpha_j > c_{ij}\}$ and $\mathcal{K}_j = \{i | \lambda_i + \alpha_j > c_{ij}\}$. Assume that λ, α is a maximizer of* (45)*, for each $1 \leq i \leq n$, $|\mathcal{J}_i| \leq k$ (and similarly, $|\mathcal{K}_j| \leq k$).*

Proof. We will only give the proof for $|\mathcal{J}_i| \leq k$ (the proof for $|\mathcal{K}_j| \leq k$ is similar). Indeed, consider only terms involving λ_i in the objective function, we have:

$$
\begin{aligned}
& k\lambda_i + \sum_{j} \min(0, c_{ij} - \lambda_i - \alpha_j) \\
&= k\lambda_i + \sum_{j \in \mathcal{J}_i} (c_{ij} - \lambda_i - \alpha_j) \\
&= (k - |\mathcal{J}_i|)\lambda_i + \sum_{j \in \mathcal{J}_i} (c_{ij} - \alpha_j)
\end{aligned} \tag{46}
$$

If $|\mathcal{J}_i| > k$, we prove that the solution λ, α is not optimal. Consider $\alpha' = \alpha, \lambda_k = \lambda'_k, \forall k \neq i$, $\lambda'_i = \lambda_i - \Delta$ where $\Delta = \min_{j \in \mathcal{J}_i} (\lambda_i + \alpha_j - c_{ij}) > 0$, we have:

$$
\begin{aligned}
&k(\sum_{i=1}^{n}\lambda_i+\sum_{j=1}^{n}\alpha_j)+\sum_{i,j}\min(0,c_{ij}-\lambda_i-\alpha_j)-k(\sum_{i=1}^{n}\lambda'_i+\sum_{j=1}^{n}\alpha'_j)-\sum_{i,j}\min(0,c_{ij}-\lambda'_i-\alpha'_j)\\
&=k\lambda_i+\sum_{j}\min(0,c_{ij}-\lambda_i-\alpha_j)-k\lambda'_i-\sum_{j}\min(0,c_{ij}-\lambda'_i-\alpha_j)\\
&=k(\lambda_i-\lambda'_i)+(\sum_{j}\min(0,c_{ij}-\lambda_i-\alpha_j)-\sum_{j}\min(0,c_{ij}-\lambda'_i-\alpha_j))\\
&=k\Delta-|\mathcal{J}_i|\Delta\\
&=(k-|\mathcal{J}_i|)\Delta\\
&<0
\end{aligned}
\tag{47}
$$

The third line is due to the fact that $\sum_j \min(0, c_{ij} - \lambda_i - \alpha_j)$ increases by Δ if and only if $j \in \mathcal{J}_i$. Otherwise, it remains unchanged and equal to zero. Therefore, (λ, α) is not an optimal solution of the Lagrange dual problem (45). □

Remark 3.5. *A stronger argument is that there exists an optimal solution of the dual problem with $|\mathcal{J}_i| \leq k-1, \forall i$. Applying this result into the case of $k = 1$, we rediscover the dual of the classical assignment problem [18] since it signifies that $\lambda_i + \alpha_j \leq c_{ij}, \forall 1 \leq i, j \leq n$.*

3.2.2 Detail of the proposed method

Firstly, we present several important definitions.

Definition 3.4 (Potential and its value)**.** *Let $G = (V, E)$ be a complete weighted bipartite graph where $V = S \cup T, |S| = |T|$ and weight function $c : S \times T \to \mathbb{R}$. A potential π is a function $f : V \to \mathbb{R}$. A potential π is feasible if and only if $\forall v \in V, |\{u \in V | c_{vu} < \pi(u) + \pi(v)\}| \leq k$.*

The value of a potential is: $P(\pi) = k(\sum_{v\in V}\pi(v)) + \sum_{(u,v)\in E}\min(0, c_{uv} - \pi(u) - \pi(v))$

Definition 3.5 (k-disjoint perfect matching and its value)**.** *Let $G = (V, E)$ be a complete weighted bipartite graph where $V = S \cup T, |S| = |T|$ and weight function $c : S \times T \to \mathbb{R}$. A k-disjoint perfect matching $M \subset E$ is a disjoint union of k perfect matchings (Definition 3.3).*

The value of a k-disjoint perfect matching M is: $V(M) = \sum_{e\in M} c(e)$.

Remark 3.6. *Thanks to Lemma 3.1, the problem* (39) *could be casted as finding a k-disjoint perfect matching that maximizes its value.*

Then we established a small lemma which will be helpful:

Lemma 3.6. *Let $G = (V, E)$ be a complete weighted bipartite graph where $V = S \cup T, |S| = |T|$ and weight function $c : S \times T \to \mathbb{R}$. Let $\mathcal{M}$ be the set of k-disjoint perfect matching, Π be the set of feasible potential. Then the following equality holds:*

$$
\min_{M\in\mathcal{M}} V(M) = \max_{\pi\in\Pi} P(\pi) \tag{48}
$$

Proof. Since the definition of potential value is identical to the formula of the dual problem (45), both problems are essentially the same if we view $\pi(u), u \in S$ and $\pi(v), v \in T$ as α and λ respectively. Thanks to Lemma 3.5, the maximum solution is achieved when $|\mathcal{J}_i|, |\mathcal{K}_j| \leq k$. This condition is equivalent with the definition of feasibility in Definition 3.4. Therefore, $\max_{\pi\in\Pi} P(\pi)$ is the maximum value of the dual problem (45).

On the other hand, given a feasible solution x of the problem (39), we can construct a matching M such that $(i, j) \in M \iff x_{ij} = 1$ in a complete weighted bipartite graph $G = (S \cup T, E)$ whose matrix weight is c (c is defined in the problem (39)). Due to the constraint, x is a k-regular sparse matrix and it is identical to $\text{supp}(x)$. Thanks to Lemma 3.1, x can be decomposed into the sum of k permutation matrices. Since each permutation matrix is equivalent to a perfect matching in M, M is a k-disjoint perfect matching. Moreover, the definition of matching value over M is identical to the

formula of the problem (39). Since the relaxed problem (41) has an integer solution (Lemma 3.4), it means that $\min_{M \in \mathcal{M}} V(M)$ is equal to the minimum of the relaxed problem (41).

Lastly, the relaxed problem (41) is an instance of linear programming with a feasible solution (Remark 3.4). Therefore, the strong duality holds [5, Section 5.2.3]. That concludes the proof. □

Definition 3.6 (Graph built from a potential)**.** *Let $G = (V, E)$ be a complete weighted bipartite graph where $V = S \cup T, |S| = |T|$ and weight function $c : S \times T \to \mathbb{R}$ and π be a feasible potential. Define:*

1. $E_< = \{u, v \in V | c(u, v) < \pi(u) + \pi(v)\}$
2. $E_= = \{u, v \in V | c(u, v) = \pi(u) + \pi(v)\}$

Let $E_\pi = E_< \cup E_=$. The graph $G_\pi = (V, E_\pi)$ built from π is a subgraph of G having edges set E_π.

Definition 3.7 (k-saturated vertex)**.** *Let $G = (V, E)$ be a complete weighted bipartite graph where $V = S \cup T, |S| = |T|$ and weight function $c : S \times T \to \mathbb{R}$ and a set of edges M. A vertex v in V is k-saturated w.r.t M (or k-saturated in short) if there are exactly k edges in M having v as their endpoints. Otherwise, the vertex is k-unsaturated.*

Remark 3.7. *When all the vertices are k-saturated w.r.t M, the degree of each vertex is exactly k in M. Therefore, M is k-disjoint perfect matching.*

Definition 3.8 (Alternating and augmenting path)**.** *Let $G = (V, E)$ be a bipartite graph where $V = S \cup T, |S| = |T|$. Let $M \subset E$ be a subset of graph edges. An alternating path $(e_1, \ldots, e_{2n+1})$ satisfies that $e_i \in M \iff i \mod 2 = 0$. An augmenting path $(e_1, \ldots, e_{2n+1})$ of M is an alternating path starting and ending with k-unsaturated vertices.*

An augmentation of M with respect to an augmenting path $(e_1, \ldots, e_{2n+1})$ is $M \setminus \{e_2, e_4, \ldots, e_{2n}\} \cup \{e_1, e_3, \ldots, e_{2n+1}\}$.

Remark 3.8. *The definition of augmenting path and augmentation is taken from the concepts in maximum matching and max flow - min cut problem [9, Chapter 3]. Verifying whether a bipartite graph admits a k-disjoint perfect matching is reducible to solving the max flow - min cut problem illustrated in Figure 1.*

To be more specific, given an unweighted bipartite graph $G = (S \cup T, E)$, we can add two more vertices s and t as source and target vertices. s only has connection to all vertices in S with capacity k (and t only has connection to vertices in T with capacity k). Edges in E have unit capacity. If the max flow between s and t is nk, G has a k disjoint perfect matching.

The proof for this result is based on the observation that since all the capacity is integer, the max flow problem admits an integer solution x [9, Chapter 3]. Consider $M = \{(u, v) | (u, v) \in E, x(u, v) = 1\}$. Since the max flow is nk, each vertex $v \in S \cup T$ must be the endpoints of exactly k edges in M. Consequently, all vertices of G are k-saturated w.r.t M. Therefore, M is a k-disjoint perfect matching.

Lemma 3.7. *Let $G = (V, E)$ be a complete weighted bipartite graph where $V = S \cup T, |S| = |T|$ and weight function $c : S \times T \to \mathbb{R}$, π be a feasible potential and consider $G_\pi = (V, E_< \cup E_=)$ defined in Definition 3.6. If G_π admits a k-disjoint perfect matching M containing all edges in $E_<$, then π maximizes its potential (and M minimizes its value).*

Proof. In fact, the value of any k-disjoint perfect matching M is an upper bound of the value of the potential π. Indeed,

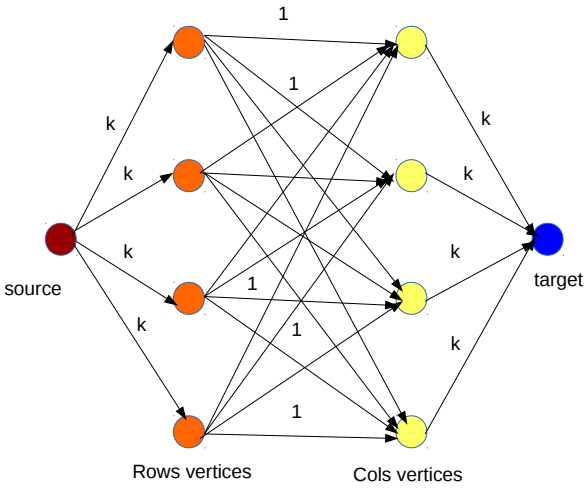


Figure 1: A max flow instance solving k-disjoint perfect matching.

$$
\begin{aligned}
V(M) &= \sum_{(u,v)\in M} c(u,v) \\
&\geq \sum_{(u,v)\in M} (\pi(u) + \pi(v) + \min(0, c(u,v) - \pi(u) - \pi(v))) \\
&\geq \sum_{(u,v)\in M} (\pi(u) + \pi(v)) + \sum_{(u,v)\in E} \min(0, c(u,v) - \pi(u) - \pi(v)) \\
&= k(\sum_{v\in V} \pi(v)) + \sum_{(u,v)\in E} \min(0, c(u,v) - \pi(u) - \pi(v)) \\
&= P(\pi)
\end{aligned}
\tag{49}
$$

The third line is derived from the fact that $M \subset E$ and $\min(0, c(u,v) - \pi(u) - \pi(v)) \leq 0, \forall u, v$. Then since M is a k-disjoint perfect matching, each $\pi(v)$ is repeated k times, which explains the fourth line.

On the other hand, if there exists a k-disjoint perfect matching $M \subset E$ such that $E_< \subset M \subset E_< \cup E_=$ (since M is a matching of E_π), we have:

$$
\begin{aligned}
V(M) &= \sum_{(u,v)\in M} c(u,v) \\
&= \sum_{(u,v)\in M} (\pi(u) + \pi(v) + \min(0, c(u,v) - \pi(u) - \pi(v))) \\
&= \sum_{(u,v)\in M} (\pi(u) + \pi(v)) + \sum_{(u,v)\in E} \min(0, c(u,v) - \pi(u) - \pi(v)) \\
&= k(\sum_{v\in V} \pi(v)) + \sum_{(u,v)\in E} \min(0, c(u,v) - \pi(u) - \pi(v)) \\
&= P(\pi)
\end{aligned} \tag{50}
$$

The equality in second line is due to the fact that if $(u,v) \in M$ then $(u,v) \in E_\pi$ hence $c(u,v) = \pi(u) + \pi(v) + \min(0, c(u,v) - \pi(u) - \pi(v))$. Since $E_< \subset M$, $\min(0, c(u,v) - \pi(u) - \pi(v)) = 0, \forall (u,v) \notin M$, we have the third line.

Finally, applying Lemma 3.6 concludes the proof. □

Remark 3.9. *This is the reason why the feasible potential is defined as in Definition 3.4. If a vertex u violates the constraint $\pi(u) + \pi(v) > c_{u,v}$ for more than k vertices v, there is no k-disjoint perfect matching satisfying the condition of Lemma 3.7 since the degree of u in M is bigger than k (whereas each k-disjoint perfect matching requires all vertices to have degree equal to k).*

Another way to understand the definition of feasibility in Definition 3.4 is the Lemma 3.5, which proves that π could not be optimal if it is not feasible.

Now we can describe the proposed algorithm:

Algorithm 13 A proposed algorithm

procedure GENERALIZED_HUNGARIAN_METHOD($G = (S \cup T, E), c, k$)
  Initialize $\pi(u) = \min_{i,j} c_{ij}, \forall u \in S$.
  Initialize $\pi(v) = 0, \forall v \in T$ ▷ Initialize of a feasible potential
  Initialize a matching $M = \emptyset$.
  Build bipartite graph $G_\pi = (S \cup T, E_< \cup E_=)$ w.r.t π.
  $M \leftarrow E_<$.
  while M is not a k-disjoint perfect matching **do**
    Perform Breath First Search algorithm (BFS) on graph $G'_\pi = (S \cup T, E_=)$.
    Let Z be the set of reachable vertices from the set of k-unsaturated vertices in S by an alternating path.
    if Z has a k-unsaturated vertices $t \in T$ reachable from $s \in S$ **then**
      Perform augmentation of M w.r.t augmenting path connecting s and t.
    else
      Let $S_1 = S \cap Z, S_2 = S \setminus Z, T_1 = T \cap Z, T_2 = T \setminus Z$.
      Let $\sigma_1 = \min_{(u,v)\in S_1\times T_2, c(u,v)>\pi(u)+\pi(v)} c(u,v) - \pi(u) - \pi(v)$.
      Let $\sigma_2 = \min_{(u,v)\in S_2\times T_1, c(u,v)<\pi(u)+\pi(v)} \pi(u) + \pi(v) - c(u,v)$.
      Let $\Delta = \min(\sigma_1, \sigma_2)$.
      Increase $\pi(u)$ by Δ for $u \in S_1$, decrease $\pi(v)$ by Δ for $v \in T_1$.
      Rebuild G_π from the new potential.
    end if
  end while
  Return M.
end procedure

3.2.3 Proof of correctness of Algorithm 13

Lemma 3.8. *If three following statements are true, then Algorithm 13 is correct.*

1. *The change of potential (line* 17*, Algorithm 13) does not violate its feasibility.*
2. *After the change of potential (line* 17*, Algorithm 13) and the rebuilding of graph G_π (line* 18*, Algorithm 13), edges in $E_<$ is in M.*
3. *After $O(n)$ changes of potential, at least one augmentation will be performed.*

Proof. If the third statement is true, then after at most $O(n)$ changes of potential, $|M|$ increases by at least 1. Since the augmentation is performed using an augmenting path which connects k-unsaturated vertex to another k-unsaturated vertex, the degree of any vertex is at most k. Therefore, after kn augmentation, all vertices must become the endpoint of exactly k edges in M, which means M is k-disjoint perfect matching. Algorithm 13 thus finitely terminates.

Moreover, the second statement ensures that $E_< \subset M$ after the change of potential. Moreover, augmentation is perform only with edges in $E_=$. Therefore, $E_< \subset M$ throughout Algorithm 13.

Finally, the potential remains feasible throughout all changes of potential due to the first statement. Applying Lemma 3.7, we have M is the optimal solution. □

Theorem 3.9. *Algorithm 13 is correct.*

Proof. It is sufficient to prove the correctness of three statements in Lemma 3.8.

We process with the first two statements. The proof is by induction. Before the WHILE loop (line 7, 13), it is clear that both statements are true.

Let $C_M(v)$ be the number of edges of M whose endpoint is v. We have that $|\{u \in V | \pi(v) + \pi(u) > c(u,v)\}| \leq C_M(v)$ since $E_< \subset M$ and $C_M(v) \leq k$ since any vertex is at most k-saturated w.r.t M. Therefore, $|\{u \in V | \pi(v) + \pi(u) > c(u,v)\}| \leq C_M(v) \leq k, \forall v \in V$.

Inside the WHILE loop, we perform the change of potential if there is no augmentation in $G' = (V, E_=)$. Thus, the following statements are true $\forall e \in E_\pi$:

1. If $e \in S_1 \times T_2$, $e \in E_<$ or $e \in E_= \cap M$. Thus, $e \in M$.
2. If $e \in S_2 \times T_1$, $e \in E_<$ or $e \in E_= \cap \overline{M}$. Thus, $e \in M$ if and only if $e \in E_<$.

because some vertices in $S_2 \cup T_2$ must be in Z otherwise, which violates the definition of S_2 and T_2.

Denote π' as the new potential after the change of potential π (line 17), we have:

$$\pi'(u) = \begin{cases} \pi(u) + \Delta & \text{if } u \in S_1 \\ \pi(v) - \Delta & \text{if } u \in T_1 \\ \pi(u) & \text{otherwise} \end{cases} \tag{51}$$

where Δ is defined in line 16, Algorithm 13. It is clear that $\Delta > 0$.

Thus, $\pi(u) + \pi(v) = \pi'(u) + \pi'(v), \forall (u,v) \in S_1 \times T_1 \cup S_2 \times T_2$. It is sufficient to investigate the change in $(u,v) \in S_1 \times T_2 \cup S_2 \times T_1$.

1. $(u,v) \in S_1 \times T_2$: we have:

$$c(u,v) - (\pi'(u) + \pi'(v)) = c(u,v) - (\pi(u) + \pi(v)) - \Delta \tag{52}$$

   Since $\Delta \leq \sigma_1$, the only edges having $c(u,v) < \pi'(u) + \pi'(v)$ after the change of potential are (u,v) satisfying $c(u,v) \leq \pi(u) + \pi(v)$ before the change of potential. That implies $(u,v) \in E_\pi$ before the change of potential. Therefore, all of them are in M due to the first statement we made about $e \in E_\pi$.

2. $e \in S_2 \times T_1$: we have:

$$c(u,v) - (\pi'(u) + \pi'(v)) = c(u,v) - (\pi(u) + \pi(v)) + \Delta \tag{53}$$

Since $\Delta \leq \sigma_2$, the only edges having $c(u,v) < \pi'(u) + \pi'(v)$ after the change of potential are (u,v) satisfying $c(u,v) < \pi(u) + \pi(v)$ before the change of potential. That implies $(u,v) \in E_<$ before the change of potential. Therefore, they are all in M thanks to the second statement we made about $e \in E_\pi$.

In summary, after the change of potential, all of edges having $c(u,v) < \pi'(u) + \pi'(v)$ must be still in M, which is the second statement of Lemma 3.8. That also justifies why the potential is still feasible since $|\{u \in V | \pi(v) + \pi(u) > c(u,v)\}| \leq C_M(v) \leq k, \forall v \in V$. That concludes the proof of the first statement of Lemma 3.8.

Lastly, each time the potential is changed, Z size must increase at least one since there will be at least one more edge $e \in S_1 \times T_2 \cup S_2 \times T_1$. Therefore, after $O(n)$ changes of potential, an augmentation must be executed. □

Remark 3.10. *The idea of the proposed algorithm is similar to that of the Hungarian method: while keeping the potential always feasible, we try to gradually build up a k-disjoint perfect matching. Once achieving such matching, we find the optimal solution. The change of potential raises its value. That is an intuition come from the Hungarian method. A sketch for the proof is as follow: Firstly S_2 and T_1 contain only k-saturated vertices IF there is no augmentation. Let π' be the new potential (Equation* (51)*), $A = |M \cap (S_1 \times T_2)|$ and $B = |M \cap (S_2 \times T_1)|$ and $C = |M \cap (S_2 \times T_2)|$. Recall that $C_M(v)$ is the number of edges of M whose endpoint is v. $C_M(S)$ is the number of edges of M whose endpoints are in $S \subset V$. The change of dual value is:*

$$\begin{aligned}
P(\pi) - P(\pi') &= k\left(\sum_{v \in V}(\pi(u) - \pi'(v))\right) \\
&\quad + \sum_{u,v} \min(c(u,v) - \pi(v) - \pi(u)) - \min(c(u,v) - \pi'(u) - \pi'(v)) \\
&= \Delta(k|S_1| - k|T_1|) - \Delta A + \Delta B \\
&= \Delta(k|S_1| - k|T_1|) - \Delta(C_M(T_2) - C) + \Delta(C_M(S_2) - C) \\
&= \Delta(k|S_1| - k|T_1|) + \Delta C_M(S_2) - \Delta C_M(T_2) \\
&= \Delta(k|S_1| - k|T_1|) + \Delta(n - |S_1|)k - \Delta C_M(T_2) \\
&> \Delta(k|S_1| - k|T_1|) + \Delta(n - |S_1|)k - \Delta(n - |T_1|)k \\
&= 0
\end{aligned} \tag{54}$$

The sign $>$ is because all vertices are not yet k-saturated in T_2. Therefore, the dual value increases.

3.2.4 Worst case complexity and a $O(kn^3)$ algorithm

Using a similar argument with the analysis of Hungarian method, we have the following: there are kn augmentations, each augmentation has at most $O(n)$ changes of potential, each change of potential has complexity $O(n^2)$ (since we perform a BFS and loop through all edges $(u,v) \in S_1 \times T_2 \cup S_2 \times T_1$). Overall complexity is $O(kn^4)$.

To improve the worst case complexity, we adopt a technique from the Hungarian method to achieve $O(kn^3)$ performance. The main bottleneck is to calculate Δ since updating the potential only takes $O(n)$ in each iteration. In order to reduce complexity, we define:

$$\sigma(u) = \begin{cases} \min\limits_{v \in S_1, c(u,v) - \pi(v) - \pi(u) > 0} c(u,v) - \pi(v) - \pi(u) & \text{if } u \in T \\ \min\limits_{v \in T_1, c(u,v) - \pi(v) - \pi(u) < 0} \pi(u) + \pi(v) - c(u,v) & \text{if } u \in S \end{cases} \tag{55}$$

Therefore, the calculation of Δ is reduced to $O(n)$ by simply considering $\min\limits_{u \in S_2 \cup T_2} \sigma(u)$. We need to find a way to maintain σ effectively after the change of potential. The main trick is to observe that while there is no augmentation, the set Z grows. It means that if $x \in Z \subset V$ before the change of potential, x remains in Z until the augmentation. Instead of looping through all vertices in S_1' and

T_1' (which are the new S_1 and T_1 after the change of potential) to calculate new Δ' in each iteration, one could notice that $\sigma(u) = \min(\sigma(u) - \Delta, \min_{v \in S_1' \setminus S_1} c(u,v) - \pi(u) - \pi(v)), u \in T_2$. The update of $\sigma(v), v \in S_2$ also has the same form.

Finally, employing algorithm like BFS could reduce the complexity of the augmenting to $O(n^2)$ in total. We do not have to perform BFS from the beginning in every iteration. Remembering which vertices are already in Z can avoid useless BFS exploration. The following pseudo code summarizes the whole strategy.

Algorithm 14 An optimized augmentation

procedure AUGMENTATION_GENERALIZED_HUNGARIAN_METHOD(G, c)
    Initialize $inZ[v] =$ **FALSE**, $\forall v \in V$.
    $inZ[u] =$ **TRUE** for $u \in S$ k-unsaturated.
    Initialize an empty queue Q. Add all $u \in S$ k-unsaturated to Q.
    Initialize $\sigma(u) =$ **INF**, $u \in V$.
    Initialize $\Delta = 0$
    while $G' = (V, E_=)$ does not have augmenting path **do**
        Perform BFS using vertices in Q for graph $G' = (V, E_=)$ with inZ memorizing discovered vertices.
        Update $\sigma(u) = \min(\sigma(u) - \Delta, c(u,v) - \pi(u) - \pi(v)), u \in T_2$ with newly added $v \in S_1$.
        Update $\sigma(u) = \min(\sigma(u) - \Delta, \pi(u) + \pi(v) - c(u,v)), u \in S_2$ with newly added $v \in T_1$.
        Calculate $\Delta = \min_{u \in T_2 \cup S_2} \sigma(u)$.
        Update potential for $u \in S_1 \cup T_1$ (similar to Algorithm 13).
        Add the set $U = \{u | \sigma(u) = \Delta\}$ to Q.
    end while
    Perform augmenting path to increase the size of the matching.
end procedure

With all those combined tricks, each edge is used $O(1)$ between two augmentations (edge is used in BFS and in updating σ). Updating potential and Δ is $O(n)$ between two changes of potential. Therefore, between two augmentations, the computational complexity of Algorithm 14 becomes $O(n^2)$ (since there are at most $O(n)$ changes of potential). The complexity of Algorithm 13 becomes $O(kn^3)$.

3.3 Experiments

The experiment in this report utilizes the same one from [19]. The matrix to be factorized is the Hadamard Transform of size $2^n \times 2^n, n \in \mathbb{N}$. The Hadamard Transform is a linear operator of size $2^n \times 2^n, n \in \mathbb{N}$. They are recursively define as follow:

$$\begin{aligned} H_0 &= 1 \\ H_{n+1} &= \frac{1}{\sqrt{2}} \begin{pmatrix} H_n & H_n \\ H_n & -H_n \end{pmatrix} \\ &= \frac{1}{\sqrt{2}} \begin{pmatrix} I_n & I_n \\ I_n & -I_n \end{pmatrix} \begin{pmatrix} H_n & 0_n \\ O_n & H_n \end{pmatrix} \end{aligned}$$

Applying this recursion, one can prove the Hadamard Transform of size $2^n \times 2^n$ can be factorized into n 2-regular sparse matrices. Similarly, the Fourier Discrete Transform admits the same structure. This is why we will use Generalized Hungarian Method with $k = 2$ (GHM(2)) in the experiment.

The objective of the experiment is to compare the performance of proximal operators SPLINCOL (Algorithm 12) with GHM (Algorithm 13) under the effect of PALM (Algorithm 10) and hierarchical factorization (Algorithm 11).

To be more specific, we attempt to factorize the Hadamard Transform and the Fourier Discrete Transform by solving the optimization problem (26) in this experiment. With PALM, the projection operator are GHM(2) and SPLINCOL(2) for all factors. With hierarchical factorization, at the k^{th}

factorization, the projection operator of the first factor is either GHM(2) or SPLINCOL(2) and that of the second factor is SPLINCOL(2^{n-k}).

Initialization is important. Default initialization is $\lambda_0 = 1, S_1^0 = 0, S_j^0 = \mathbf{Id}, \forall j > 1$. Except when specified otherwise, both PALM and hierarchical factorization employs *default initialization*. The step size for S_j at i^{th} iteration is calculated by the formula:

$$c_j^i = (1+\alpha)(\lambda_i)^2 \|R\|_2^2 \|L\|_2^2 \tag{56}$$

where $R = \prod_{k=1}^{j-1} S_k^{i+1}$, $L = \prod_{k=j+1}^{J} S_k^i$ and $\alpha = 10^{-3}$. c_j^i is proportional to the Lipschitz modulus with respect to S_j at the i^{th} iteration (more detail in [19]).

The performance of an algorithm is measured by the error, which is calculated as:

$$RE = \frac{\|A - \lambda \prod_{j=1}^{l} S_j\|_2}{\|A\|_2} \tag{57}$$

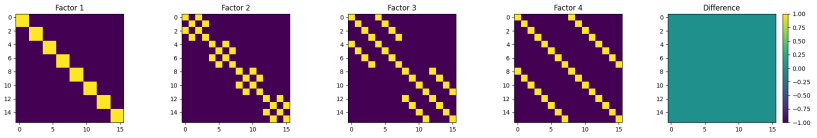
A factorization is said to be *Success* or *Exact* if its error is smaller than 10^{-4}. This small error makes sure that there is not much difference between the original linear operator with the factorized ones. Otherwise, the factorization is *Unsuccess*

Algorithm / Proximal	HGM	SPLINCOL
PALM	*Success*	*Unsuccess*
Hierarchical factorization	*Success*	*Success*

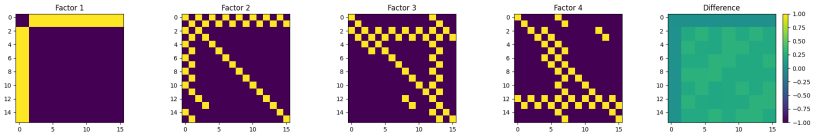
Table 1: Factorization of the Hadamard Transform up to $2^8 \times 2^8$

.

Table 1 shows the result of the factorization of the Hadamard Transform with HGM and SPLINCOL as proximal operators of PALM and hierarchical factorization. The result clearly favors GHM over SPLINCOL. In detail, HGM succeeds exactly factorizing the Hadamard Transform (meaning the error $RE < 10^{-4}$) with both PALM and hierarchical factorization. On the other hand, SPLINCOL only manages to factorize successfully with hierarchical factorization as reported in [19].

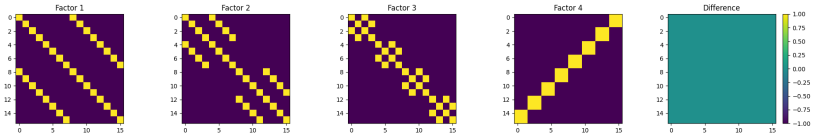


(a) Supports of four factors of the Hadamard Transform size 16×16 with GHM and difference matrix.

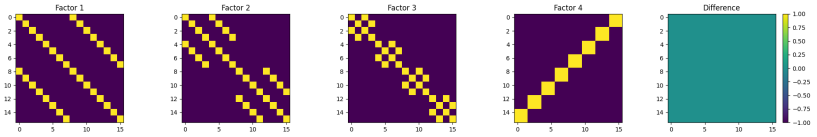


(b) Supports of four factors of the Hadamard Transform size 16×16 with SPLINCOL and difference matrix.

Figure 2: The supports of the output of PALM for the Hadamard Transform.



(a) Supports of four factors of the Hadamard Transform size 16×16 with GHM and difference matrix.



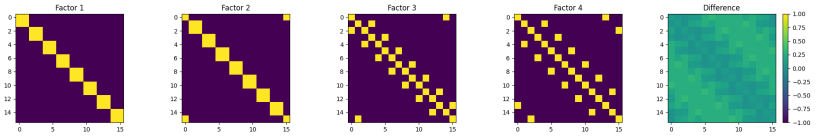
(b) Supports of four factors of the Hadamard Transform size 16×16 with SPLINCOL and difference matrix.

Figure 3: The supports of the output of hierarchical factorization for the Hadamard Transform and difference between the factorized matrix and outputs product. Yellow indicates the indices of the support.

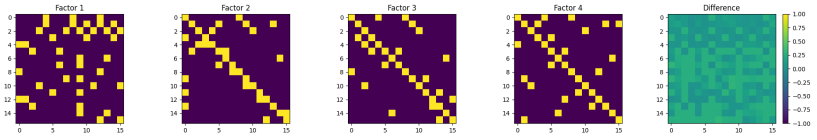
Figure 2 and Figure 3 further illustrate the solutions found by PALM and hierarchical factorization. Since the definition of k-regular sparse matrix describes well the support of the Hadamard Transform, GHM is more informative than SPLINCOL. That is why GHM still succeeds with PALM.

Except default initialization, random initializations such as Gaussian random matrices do not allow PALM and hierarchical factorization to achieve exact factorization. Explanation for this behavior will be found in the next section.

The experiment with the Discrete Fourier Transform is also carried out. However, the factorizations carried out by PALM and hierarchical factorization are *Unsuccess* with the default setting. Figure 4 and Figure 5 show the support of the factors found by PALM and hierarchical factorization. Both HGM and SPLINCOL are unable to locate the correct support of the Discrete Fourier Transform.

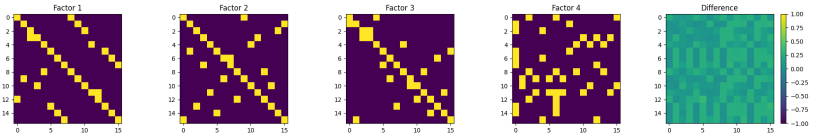


(a) Supports of four factors of the Discrete Fourier transform size 16×16 with GHM and difference matrix.

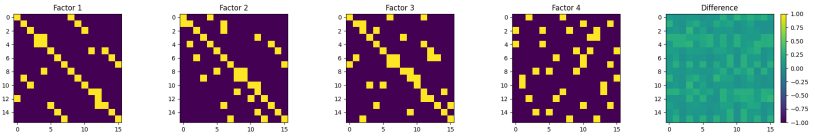


(b) Supports of four factors of the Discrete Fourier transform size 16×16 with SPLINCOL and difference matrix.

Figure 4: The supports of the output of PALM for Discrete Fourier Transform and difference between the factorized matrix and outputs product. Yellow indicates the indices of the support.



(a) Supports of four factors of the Discrete Fourier transform size 16×16 with GHM and difference matrix.



(b) Supports of four factors of the Discrete Fourier transform size 16×16 with SPLINCOL and difference matrix.

Figure 5: The supports of the output of hierarchical factorization for Discrete Fourier Transform and difference between the factorized matrix and outputs product. Yellow indicates the indices of the support.

4 Hard Thresholding Pursuit and a sparse matrix factorization algorithm

Hard Thresholding Pursuit (HTP) [12] and its application to sparse matrix factorization are the main topics of interest in this section. Since HTP and its variants are introduced in Section 2.2.2, this section is devoted to discuss its adaptation to our problem. This section will be concluded by the experimental results of the proposed algorithm.

4.1 Hard Thresholding Pursuit for sparse matrix factorization

We first discuss an inconvenience with current factorization algorithm such as PALM and hierarchical factorization. The following part will be devoted to introduce several remedies to tackle this difficulty and improve the performance of the algorithms.

4.1.1 A difficulty of PALM and hierarchical factorization

As discussed in Section 2.1, PALM does not do well in practice. Although hierarchical factorization works better than PALM, it is dependent to initialization (and so far, random initialization such as Gaussian random matrices or uniform random matrices do not make hierarchical factorization exactly factorize the Hadamard Transform). From practical point of view, as we will soon demonstrate, the reason why those algorithms fail to deliver good factorization is that they are *lazy* to change the factor support. In short, the support of factors hardly change after the first iteration (illustrated by Figure 6 and Figure 7). However, since the support of the first iteration of Figure 6 is very different to a support of an exact factorization, the error of factorization is high. In contrast, the support in Figure 7 already allows an exact factorization, thus, allows the algorithms to factorize perfectly.

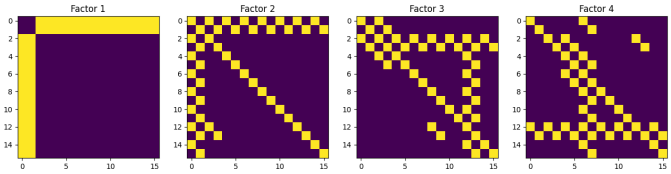
To explain this behavior, it is worth reviewing the update rule of PALM in Algorithm 10

$$S_j^{i+1} \leftarrow P_{\mathcal{E}_j}(S_j^i - \gamma \nabla_{S_j} \|A - (\prod_{l=1}^{j-1} S_l^{i+1})(\prod_{l=j}^{J} S_l^i)\|_F^2) \tag{58}$$

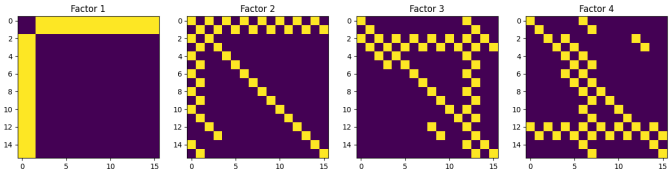
where $\mathcal{E}_j = \{S \in \mathbb{R}^{n \times n}, \|S\|_0 \leq 2n, \|S\|_F = 1\}$. The projection into $\mathcal{E}_j$ is to simply choose coefficients having the largest absolute values. Since coefficients outside the current support are zero, one step gradient descent cannot change much (or even maintain) the order of absolute values. Therefore, the support hardly changes. This argument is also applied for our Generalized Hungarian Method operator.

Therefore, to be able to tackle this problem, the change of coefficients in each step needs to be large enough so that the projection operator could explore different supports. However, in the spirit of an iterative method, if the change between two iterations is too high, there is no guarantee that the objective function $\|A - \lambda \prod_{i=1}^{J} S_j\|_F^2$ decreases. The algorithm, thus, is unstable and does not guarantee to converge in general.

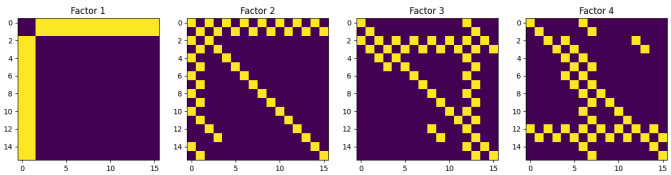
The introduced difficulty greatly hinders the effectiveness of proximal algorithm. Most of the time, the algorithm finds itself trapped in a bad support and fails to explore better support. That is why the initialization of hierarchical factorization in [19] has to be quite specific to make it work. In general, the difficulty is also applied to any algorithm attempting to enforce a sparse structure to the solution by proximal operator.



(a) Supports of four factors of the Hadamard Transform size 16×16 in the first iteration.

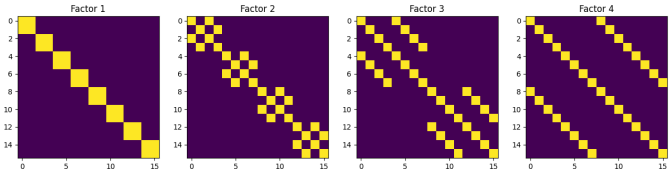


(b) Supports of four factors of the Hadamard Transform size 16×16 in the 50^{th} iteration.

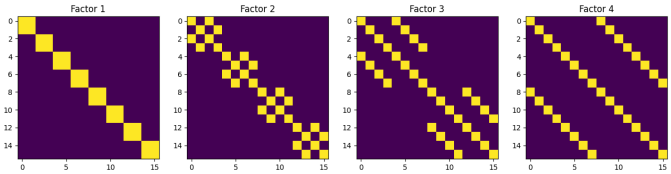


(c) Supports of four factors of the Hadamard Transform size 16×16 in the 100^{th} iteration.

Figure 6: The evolution of support of factors of PALM with SPLINCOL(2) projection during the Hadamard Transform factorization. Yellow indicates the support.



(a) Supports of four factors of the Hadamard Transform size 16×16 in the first iteration.



(b) Supports of four factors of the Hadamard Transform size 16×16 in the 50^{th} iteration.

Figure 7: The evolution of support of factors of PALM with Generalized Hungarian Method projection during the Hadamard Transform factorization. Yellow indicates the support.

4.1.2 Hard Thresholding Pursuit and support exploration

In this section, we propose a method to overcome the difficulty introduced in Section 4.1.1. To simplify the problem as well as the algorithm, we will focus on the problem of factorization a matrix Y into $Y = AB, A \in \mathcal{E}_A, B \in \mathcal{E}_B$ where $\mathcal{E}_A, \mathcal{E}_B$ have some certain sparsity structures.

Unlike the problem of linear regression which consists in finding a sparse x such that $Ax \approx y$ where A, y are known, here both A and B are parameters to optimize. The problem is biconvex (and not convex). That means for every fixed value of A (B), the optimization problem $\min_B f(A, B)$ ($\min_A f(A, B)$) is convex. One natural idea is to alternately optimize one factor while fixing the other. To further enforce the constraint of $\mathcal{E}_A, \mathcal{E}_B$ in the optimization process, we employ the projection operator from proximal algorithms. This could be viewed as a more general operator compared to H_s. In fact, H_s is also a projection operator related to the constraint set $\mathcal{E} = \{x \in \mathbb{R}^n, \|x\|_0 \leq s\}$.

Assuming that the value of A is fixed, the procedure to optimize B is displayed in Algorithm 15.

Algorithm 15 MODIFIED HARD THRESHOLDING PURSUIT

1: **procedure** MODIFIED_HTP($Y, A, \mathcal{E}_B, \mu$)
2: Initialize a solution $B_0 \leftarrow 0$.
3: $k \leftarrow 0$
4: **while** *terminating condition is not met* **do**
5: $U_{k+1} \leftarrow \mathbf{prox}_{\mathcal{E}_B}(B_k + \mu A^*(Y - AB_k))$
6: $S_{k+1} \leftarrow \text{supp}(U_{k+1})$.
7: $B_{k+1} \leftarrow \arg\min_B \{\|Y - AB\|_2, \text{supp}(B) \subseteq S_{k+1}\}$.
8: $k \leftarrow k + 1$.
9: **end while**
10: Return B
11: **end procedure**

In the line 7 of Algorithm 15, we need to calculate $\arg\min\{\|Y - AB\|_2, \text{supp}(B_{k+1}) \subseteq S_{k+1}\}$. This can be calculated by orthogonal projection. Indeed, let $\text{supp}(B^i)$ be the support of i^{th} column of B, $A_{\text{supp}(B^i)}$ be matrix A restricted to the columns corresponding to $\text{supp}(B^i)$. To have $Y = AB$, we need:

$$Y^i = A_{\text{supp}(B^i)} B^i_{\text{supp}(B^i)} \tag{59}$$

In words, the column i^{th} of Y is a linear combination of several columns of A. The calculation of coefficients in the support of the i^{th} column of B is an orthogonal projection from Y^i to the subspace spanned by $A_{\text{supp}(B^i)}$. Its formulation is:

$$B^i_{\text{supp}(B^i)} = (A_{\text{supp}(B^i)})^\dagger Y^i \tag{60}$$

where $A^\dagger$ is the pseudo inverse of matrix A.

Another way to perform this task is to adopt the trick in FHTP. Instead of making use of orthogonal projection, the algorithm employs several gradient descent steps. Although it is unable to compute the exact orthogonal projection, it does offer other benefits. Firstly, it saves us the running time since we avoid computing the pseudo inverse of a MULTIPLE matrices. The stress here is that the support of B^k might be different from column to column. Therefore, we cannot reuse the pseudo-inverse to reduce the complexity. Without further clarification, when referring to Algorithm 15, we means the version with gradient descent.

Aside from Algorithm 15, our factorization algorithm also employs proximal operators similar to PALM to polish the current solution (A, B). The detailed calculation of the update of A and B at i^{th} iteration is as follow:

$$A_{i+1} = P_{\mathcal{E}_A}(A_i + \mu((Y - A_i B_i)B_i^*))$$
$$B_{i+1} = P_{\mathcal{E}_B}(B_i + \mu(A_{i+1}^*(Y - A_{i+1} B_i)))$$

where $\mathcal{E}_A, \mathcal{E}_B$ are respectively constraint sets of A and B, P_C is the proximal operator of the indicator function of the constraint set (or equivalently a projection operator to the constraint set).

The algorithm description is shown in Algorithm 16.

Algorithm 16 HARD THRESHOLDING PURSUIT FACTORIZATION

1: **procedure** HARD_THRESHOLDING_PURSUIT_FACTORIZATION($Y, \mathcal{E}_A, \mathcal{E}_B, \mu_H, \mu_P$)
2: Initialize a solution A_0, B_0.
3: $k \leftarrow 0$
4: **while** *terminating condition is not met* **do**
5: **if** The supports of A and B need changing **then**
6: $B' \leftarrow$ MODIFIED_HTP$(Y, A_k, \mathcal{E}_B, \mu_H)$.
7: $A' \leftarrow$ (MODIFIED_HTP$(Y^T, B'^T, \mathcal{E}_A^T, \mu_H)).T$.
8: $A_k = A', B_k = B'$.
9: **end if**
10: $A_{k+1} \leftarrow P_{\mathcal{E}_A}(A_k + \mu_P((Y - A_k B_k)B_k^*))$ ▷ PALM update for A
11: $B_{k+1} \leftarrow P_{\mathcal{E}_B}(B_k + \mu_P(A_{k+1}^*(Y - A_{k+1}B_k)))$ ▷ PALM update for B
12: $k \leftarrow k + 1$.
13: **end while**
14: Return A_k, B_k
15: **end procedure**

In line 5 of Algorithm 16, A and B use Algorithm 15 to explore new support every fixed number of iterations or the objective function improvement is less than $\epsilon > 0$ after applying the proximal operators. It is worth emphasizing that the initialization of Algorithm 15 is zero matrix but not the current value of A or B. Since we initialize with the current value of A and B, the analysis in Section 4.1.1 is valid and the support of A and B are unlikely to change. The effect of support exploration of Algorithm 15 will be shown in the experiment section.

To have a multi layer factorization algorithm, we apply the idea of hierarchical factorization as shown in Algorithm 17.

Algorithm 17 Hierarchical factorization with Hard Thresholding Pursuit

1: **procedure** HIERARCHICAL_FACTORIZATION($Y, J, \mathcal{E}_j, \tilde{\mathcal{E}}_j, \mu_H, \mu_P$)
2: $T_0 \leftarrow Y$.
3: **for** $l \in \{1, \ldots, J-1\}$ **do**
4: $S_l, T_l \leftarrow$ HARD_THRESHOLDING_PURSUIT_FACTORIZATION$(T_{l-1}, \mathcal{E}_l, \tilde{\mathcal{E}}_l, \mu_H, \mu_P)$
5: Using PALM to globally optimize $S_1, \ldots, S_l, T_l$.
6: **end for**
7: Return $S_1, \ldots, S_{J-1}, T_{J-1}$.
8: **end procedure**

4.1.3 A hybrid method between L_0 relaxation and norm L_1 regularization for sparse matrix factorization

To further improve the robustness of Algorithm 16 and Algorithm 17, we propose a strategy combining the relaxation of L_0 constraints and norm L_1 regularization in this report. This strategy is applicable when the sparsity is enforced by L_0 constraints. Starting from the original problem:

$$\begin{aligned} \underset{X}{\text{minimize}} \quad & \|Y - AB\|_F^2 \\ \text{subject to} \quad & A \in \mathcal{E}_A, \\ & B \in \mathcal{E}_B \end{aligned} \tag{61}$$

where $\mathcal{E}_A, \mathcal{E}_B$ are sets induced by L_0 constraints to promote the sparsity, we first consider another problem:

$$\begin{aligned} \underset{X}{\text{minimize}} \quad & \|Y - AB\|_F^2 + \gamma_A\|A\|_1 + \gamma_B\|B\|_1 \\ \text{subject to} \quad & A \in \mathcal{E}'_A, \\ & B \in \mathcal{E}'_B \end{aligned} \tag{62}$$

where γ_A, γ_B are hyper-parameters controlling the L_1 regularization, $\mathcal{E}'_A$ and $\mathcal{E}'_B$ are sets induced by L_0 constraints and $\mathcal{E}_A \subset \mathcal{E}'_A$ and $\mathcal{E}_B \subset \mathcal{E}'_B$.

The solution of the problem (62) is then used as the initialization of the problem (61), where we already have the available Algorithm 16.

Example 4.1. *Let Y be the matrix of the Hadamard Transform of size $2^n \times 2^n$. As shown in Section 3.3, there exist A, B such that A is a 2-regular sparse matrix, B is a 2^{n-1}-regular sparse matrix and $Y = AB$. To find A, B, one can solve the problem* (61) *with $\mathcal{E}_A$ and $\mathcal{E}_B$ are the sets of 2-regular sparse matrix and 2^{n-1} sparse matrix respectively. Both $\mathcal{E}_A$ and $\mathcal{E}_B$ can be described by L_0 constraints in each row and column.*

With our proposed strategy, we will first turn our attention towards problem (62) *with $\mathcal{E}'_A$ and $\mathcal{E}'_B$ less restrictive than $\mathcal{E}_A$ and $\mathcal{E}_B$. For example, one can set $\mathcal{E}'_A$ as the set of 3-regular sparse matrix and $\mathcal{E}'_B = \mathcal{E}_B$.*

The main advantage of this strategy is that one can choose $\mathcal{E}'_A$ and $\mathcal{E}'_B$ such that the structure imposed by those sets are close to $\mathcal{E}_A$ and $\mathcal{E}_B$. Usually, $\mathcal{E}'_A$ and $\mathcal{E}'_B$ allow denser feasible solution than $\mathcal{E}_A$ and $\mathcal{E}_B$. Therefore, we propose the usage of L_1 regularization to make up for the difference between constraint sets. Therefore, once projected by $P_{\mathcal{E}_A}$ and $P_{\mathcal{E}_B}$, the solution of problem (62) does not change much. In addition, the relaxation (62) will allow its solution to score lower objective function than the problem (61) (since the constraints are less restrictive). Hence, the solution obtained from problem (62) can serve well as an initialization for the original problem.

Algorithm 18 Hierarchical factorization with relaxation

1: **procedure** RELAXATION_FACTORIZATION($Y, \mathcal{E}_A, \mathcal{E}_B, \mathcal{E}'_A, \mathcal{E}'_B, \gamma_A, \gamma_B, \mu_H, \mu_P$)
2: Initialize a solution A_0^r, B_0^r.
3: $k \leftarrow 0$
4: **while** *terminating condition is not met* **do**
5: **if** The supports of A and B need changing **then**
6: $B'^r \leftarrow$ MODIFIED_HTP$(Y, A_k^r, \mathcal{E}'_B, \mu_H)$.
7: $A'^r \leftarrow$ (MODIFIED_HTP$(Y^T, (B'^r)^T, \mathcal{E}'^T_A, \mu_H)).T$.
8: $A_k^r = A'^r, B_k^r = B'^r$.
9: **end if**
10: $A_{k+1}^r \leftarrow P_{\mathcal{E}'_A}(A_k^r + \mu_P((Y - A_k^r B_k^r)(B_k^r)^* - \gamma_A \text{sgn}(A^r)))$
11: $B_{k+1}^r \leftarrow P_{\mathcal{E}'_B}(B_k^r + \mu_P((A_{k+1}^r)^*(Y - A_{k+1}^r B_k^r) - \gamma_B \text{sgn}(B^r)))$
12: $k \leftarrow k + 1$.
13: **end while**
14: Initialize a solution $A_0 = A_k^r, B_0 = B_k^r$.
15: $k \leftarrow 0$
16: **while** *terminating condition is not met* **do**
17: **if** The supports of A and B need changing **then**
18: $B' \leftarrow$ MODIFIED_HTP$(Y, A_k, \mathcal{E}_B, \mu_H)$.
19: $A' \leftarrow$ (MODIFIED_HTP$(Y^T, B'^T, \mathcal{E}_A^T, \mu_H)).T$.
20: $A_k = A', B_k = B'$.
21: **end if**
22: $A_{k+1} \leftarrow P_{\mathcal{E}_A}(A_k + \mu_P((Y - A_k B_k)B_k^*))$ ▷ PALM update for A
23: $B_{k+1} \leftarrow P_{\mathcal{E}_B}(B_k + \mu_P(A_{k+1}^*(Y - A_{k+1} B_k)))$ ▷ PALM update for B
24: $k \leftarrow k + 1$.
25: **end while**
26: Return A_k, B_k
27: **end procedure**

The relaxed problems can be solved by Algorithm 16 with slight modification of the calculation of the proximal steps in line 10 and 11. They are shown in the following formulas:

$$A \leftarrow P_{\mathcal{E}'_A}(A - \mu((AB - Y)B^* + \gamma_A \text{sgn}(A)))$$
$$B \leftarrow P_{\mathcal{E}'_B}(A - \mu(A^*(AB - Y) + \gamma_B \text{sgn}(B)))$$

where $\mu > 0$ is the step size, $\text{sgn}(A)$ returns the matrix of $\{-1, 0, 1\}$ to indicate the sign at each coefficient of A.

Finally, the strategy is displayed in Algorithm 18

4.2 Experiments

There are three conducted experiments. The first experiment is to use Hierarchical Factorization Hard Thresholding Pursuit (HFHTP) (Algorithm 16) and Relaxation Factorization (RF) (Algorithm 18) to factorize the Discrete Fourier Transform and the Hadamard transform. The second experiment is to use HFHTP to factorize MEG matrix [19]. Finally, HFHTP and PALM are tested with randomly generated matrix to compare their performance.

1) *Factorization the Discrete Fourier Transform and the Hadamard Transform:* An important improvement with HFHTP is the robustness of the algorithm against different initialization. In Section 3, there is only default initialization [19] making PALM and hierarchical factorization work. However, with HTP, it allows random initialization to work. We consider two types of initializations:

   a) *Block initialization*: Block initialization is to take a block diagonal matrix as initialization.A block diagonal matrix is a square diagonal matrix in which the diagonal elements are square matrices of any size (possibly even 1×1), and the off-diagonal elements are 0. Below are examples of block diagonal matrices.

$$A_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}, A_2 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}, A_3 = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{63}$$

   Since the constraint set of the Hadamard (Discrete Fourier) Transform factor is 2^i-regular. All the initialized matrices are block diagonal matrix with diagonal square matrices of size $2^i \times 2^i$. Nonzero coefficients are randomly generated following standard Gaussian distribution.

   b) *Random initialization*: Similar to block initialization, random initialization has all nonzero coefficients following standard Gaussian distribution. However, the initialized matrix only needs to be 2^i-regular. Below are examples of this initialization (with matrix of size 4×4).

$$A_1 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}, A_2 = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}, A_3 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} \tag{64}$$

   HFHTP and RF is run with the proximal step size $\mu_P = \mu_H = 0.1$. $\mathcal{E}_l$ and $\tilde{\mathcal{E}}_l$ are defined similarly as experiments in Section 3. For RF, the relaxed L_0 constraint $\mathcal{E}'_l$ is the set of 4-sparse regular matrices, $\gamma_A = \gamma_B = 10^{-4}$.

   The evaluation of a factorization is either *Success* or *Unsuccess*, similar to experiments in Section 3.

   The experiments with the Hadamard Transform is summarized by Table 2

Algorithm / Initialization	Default init	Block init	Random init
PALM	*Success*	*Unsuccess*	*Unsuccess*
Hierarchical factorization	*Success*	*Unsuccess*	*Unsuccess*
RF	*Success*	*Unsuccess*	*Unsuccess*
HFHTP	*Success*	*Success*	*Unsuccess*

Table 2: Factorization of the Hadamard Transform up to $2^8 \times 2^8$

.

As we can see, HFHTP succeeds with both default and block initialization while others only succeed with default one. Although RF does not succeed with more initialization than PALM and hierarchical factorization, it is successful with the first factorization (recall that RF is similar

to hierarchical factorization, it factorizes a matrix into 2 matrices at a time). Meanwhile, other methods fail the first factorization with random initialization. It turns out that the Hadamard Transform has an undesired property: it could be factorized into matrix A (2-regular matrix) and B (2^{n-1}-regular matrix) such that B could not be further factorized. Here is the example of the Hadamard Transform factorization of size 8×8 found by HFHTP.

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, B = \begin{pmatrix} 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 1 & -1 & 0 & 0 & -1 & 1 & 0 \\ 1 & 0 & -1 & 0 & 1 & 0. & -1 & 0 \\ 1 & 0 & 0 & -1 & -1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & -1 & 1 & -1 & 1 & 0 & 0 \end{pmatrix} \tag{65}$$

The proof of B could not be factorized into two 2-regular matrices could be found in Appendix 6. Therefore, HFHTP is still very robust even the initialization being totally random. On the other hand, other initializations lead to better factorization since the initialized matrices's sparse structure are close to the standard factorization.Therefore, HFHTP could take this advantage to perform better.

Nevertheless, all algorithm fails to achieve exact factorization of the Discrete Fourier Transform, even HFHTP. Our explanation is: The optimization problem involving the Discrete Fourier Transform has some difficult stationary points that even HTP could not help the algorithm to escape. HTP stops changing support after few iterations, unlike its usual behavior (which will be elaborated in detail later). This issue will be left for future work.

2) Factorization of MEG matrix.

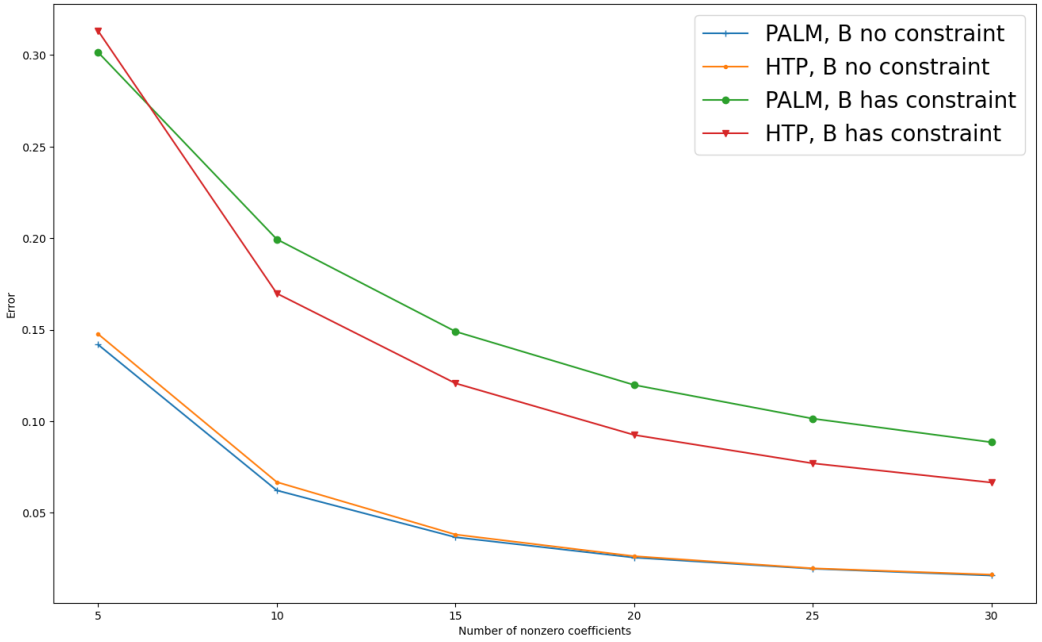


Figure 8: Result of MEG factorization with PALM and HFHTP

The objective of this experiment is to show the performance of HFHTP by comparing the performance of HFHTP with PALM in the factorization of MEG matrix.

Magnetoencephalography (MEG) operator is a matrix of size 8193×204 used in functional brain imaging. Finding a sparse factorization of the matrix MEG (denoted as $Y_{MEG} \in \mathbb{R}^{8193 \times 204}$) is

reducible to the following problem:

$$\begin{aligned} \underset{A\in\mathbb{R}^{8193\times 204}, B\in\mathbb{R}^{204\times 204}}{\text{minimize}} \quad & \|Y_{MEG} - AB\|_F^2 \\ \text{subject to} \quad & \|(A^T)^i\|_0 \leq k \qquad \forall i = 1, \dots 8193 \\ & \|B^j\|_0 \leq h \qquad \forall j = 1, \dots 204 \end{aligned} \tag{66}$$

where $(A^T)^i$ (resp. B^j) is the i^{th} row (resp. j^{th} column) of matrix A (resp. B).

We calculate the approximation error of factorizing a matrix Y into two matrices A, B as:

$$RE = \frac{\|Y - AB\|_2}{\|Y\|_2} \tag{67}$$

to measure the effectiveness of our algorithms.

There are two tests with MEG: one similar to [19] where MEG is factorized into $A \in \mathbb{R}^{8193\times 204}, B \in \mathbb{R}^{204\times 204}$. A has at most $k \in \{5, 10, 15, 20, 25, 30\}$ nonzero coefficients per row. B does not have any constraints. We add another test where B does have constraints: each column has at most 50 nonzero coefficients (thus $h = 50$).

HFHTP is run with the proximal step size μ_P defined in Equation (56). The stepsize of HTP is $\mu_H = 10^{-3}$.

The result is illustrated in Figure 8. PALM and HFHTP do equally well in the first test. They reach very small error (less than 2%) with a large reduction of computation (around 5 times less number of arithmetic operators) when $k = 30$. In the second test, HFHTP outperforms PALM in five over six cases. In our intuition, since B has no constraint in the first test, HTP does not have much impact. On the other hand, when B does have strict constraint, the choice of support of B greatly affects the performance of the algorithm. HFHTP with HTP as subprocedure allows us to explore more support of A and B, thus leads to a better performance. The impact of this mechanism will be demonstrated in detail in the next experiment.

3) Factorization of random Gaussian matrix:

Generated random matrices are also a subject of interest. They will further verify the impact of HTP, which helps the algorithm escape bad stationary points. This is the main objective of this experiment.

There are three tests in experiment with randomly generated Gaussian matrices. The matrix we would like to factorize is $Y_{random} = AB$ where $A \in \mathbb{R}^{500\times 32}, B \in \mathbb{R}^{32\times 32}$. A has 16 nonzero coefficients per row while B has $8, 12, 16$ nonzero coefficient per column respectively. Nonzero coefficients are randomly generated from standard Gaussian distribution.

The way we random matrix is to firstly randomly choose matrices A, B such that each row (or column) has limited number of nonzero coefficients. Then we return matrix $Y_{random} = AB$ and try to reverse engineer Y. Thus, the generated random matrices in our experiments do have underlying structure and the objective is to recover such structure by factorizing Y_{random} into A, B with sparse constraints.

The constraint set for the first factor and second factor are $\{A|A \in \mathbb{R}^{500\times 32}$ has at most 16 nonzero coefficients per row$\}$ and $\{B|B \in \mathbb{R}^{32\times 32}$ has at most k nonzero coefficients per column$\}, k \in \{5, 10, 15, 20, 25\}$. We use the same error as the second experiment.

There are two types of initializations:

(a) A and B are identity matrices (or contains 1 on the main diagonal in case it is not square matrix) (denoted as default in figures).

(b) A and B are randomly generated by Gaussian distribution (denoted as Gaussian in figures).

HFHTP is run with the proximal step size μ_P defined in Equation (56). The stepsize of HTP is $\mu_H = 10^{-3}$.

The illustrations of PALM and HFHTP performance are Figure 9, Figure 10, Figure 11.

The result shows clearly that HFHTP outperforms PALM consistently on all tests and with both initializations. Figure 12 helps us to get the insights why HFHTP is able to perform better. While PALM converges prematurely, HTP helps HFHTP to continuously search for better support. PALM does not have this mechanism as we already explain in Section 4.1.1 and consequently, it gets trapped in a "bad" support. The exploration of HTP does not guarantee to decrease the error

as Figure 12 showed. Nevertheless, we could eventually traverse seemingly better supports to further improve the performance.

In addition, observation from Figure 9, Figure 10 and Figure 11 shows that there is a huge difference in performance of PALM between two types of initializations. To be more specific, random initialization leads to worse results comparison to the other. This is not the case with HFHTP since both initializations work equally well. Combining with the results of the Hadamard Transform factorization, it suggests that HFHTP is less sensitive to initialization.

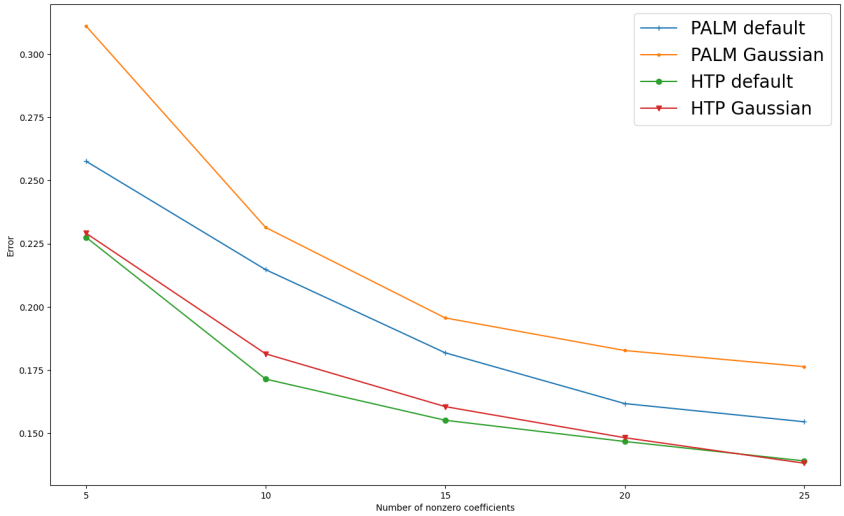


Figure 9: Results of PALM and HFHTP with $Y_{random} = AB$, $A \in \mathbb{R}^{500\times 32}$ has 16 nonzero coefficients per row, $B \in \mathbb{R}^{32\times 32}$ has 8 nonzero coefficients per column.

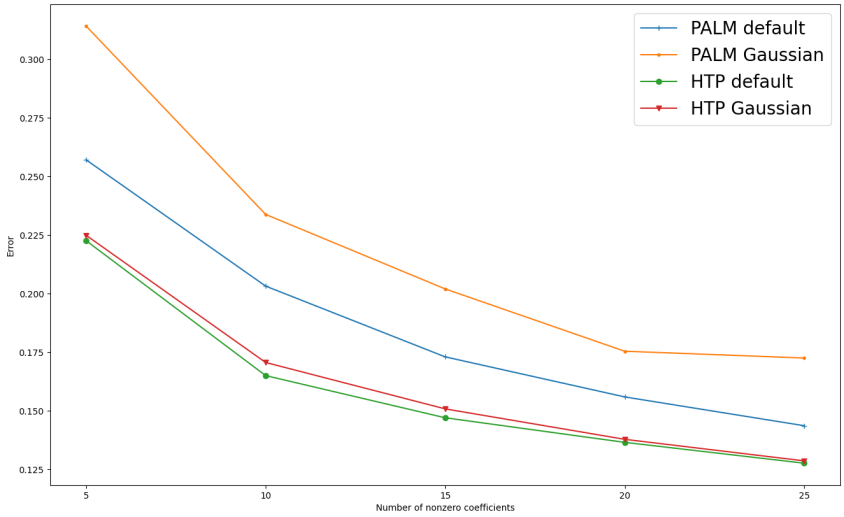


Figure 10: Results of PALM and HFHTP with $Y_{random} = AB$, $A \in \mathbb{R}^{500\times 32}$ has 16 nonzero coefficients per row, $B \in \mathbb{R}^{32\times 32}$ has 12 nonzero coefficients per column.

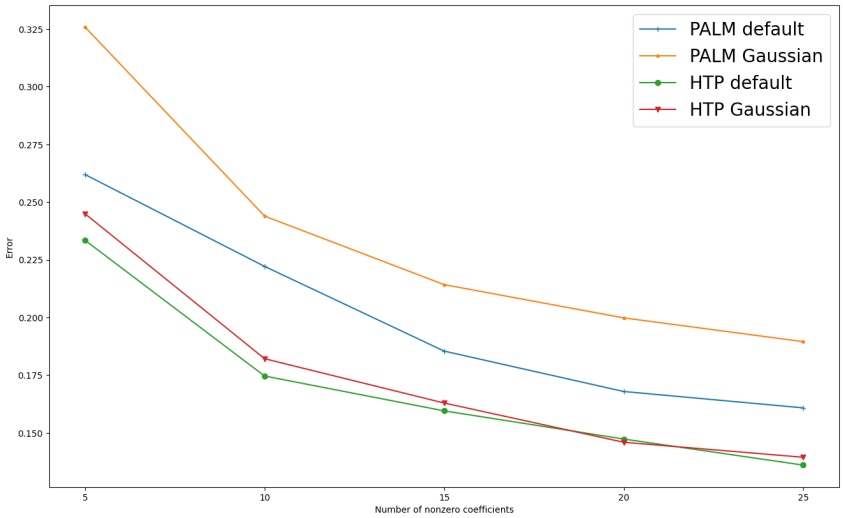


Figure 11: Results of PALM and HFHTP with $Y_{random} = AB$, $A \in \mathbb{R}^{500\times 32}$ has 16 nonzero coefficients per row, $B \in \mathbb{R}^{32\times 32}$ has 16 nonzero coefficients per column.

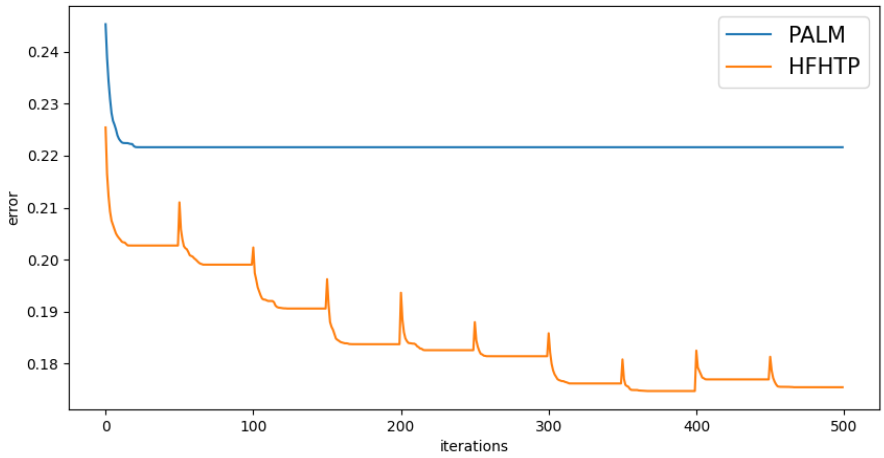


Figure 12: The error evolution during training of HFHTP and PALM

5 Matrix factorization with fixed support

This last technical section is concerned with the problem of factorization of a matrix $Y = AB, Y \in \mathbb{R}^{n\times m}, A \in \mathbb{R}^{n\times r}, B \in \mathbb{R}^{r\times m}$ (the field could be $\mathbb{C}$). This is a classical problem in literature, which contains many well-studied research. However, instead of dealing with matrices A, B having all degree of freedom (or equivalently, the support of A and B are the whole matrices), we look at a more general version:

$$\underset{A,B}{\text{minimize}} \quad \|Y - AB\|_F^2 \tag{68}$$

$$\text{subject to:} \quad \text{supp}(A) \subseteq C_A \tag{69}$$

$$\text{supp}(B) \subseteq C_B \tag{70}$$

where C_A (C_B) is the subset of the set of indices of A (B).

This section will introduce several explorations related to the problem (68). Although the result is less conclusive comparing to the ones in previous sections, we can provide interesting ideas and

conjectures about the landscape of the problem (68). In the classical setting where A and B are allowed to have full support, many important results are discovered and proved. Before introducing those results, we re-introduce some important definitions.

Definition 5.1 (Spurious local minimum [30])**.** *We say a critical point x (i.e the gradient $\nabla f(x) = 0$) is a spurious local minimum if x is a local minimum but it is not global minimum.*

Definition 5.2 (Strict saddle property [30])**.** *A twice differentiable function satisfies the strict saddle property if each critical point x is a local minimum or the Hessian evaluated at x has at least one strictly negative eigenvalue.*

The following theorems provides some insightful results of the landscape of the optimization problem (68) when C_A and C_B contains all indices of matrices A and B respectively.

Theorem 5.1. *[30] Consider the following problem:*

$$\underset{A\in\mathbb{R}^{d_2\times d_1}, B\in\mathbb{R}^{d_1\times d_0}}{\textbf{\textit{minimize}}} f(A,B) = \|Y - ABX\|_F^2 \tag{71}$$

where $X \in \mathbb{R}^{d_0\times N}$ and $Y \in \mathbb{R}^{d_2\times N}$ are the input and output training examples. Assume X is full row rank. Then the objective function f appearing in (71) *has no spurious local minima and obeys the strict saddle property.*

Theorem 5.2. *[17] Consider the following problem:*

$$\underset{W_1,\ldots,W_H}{\textbf{\textit{minimize}}} f(A,B) = \|Y - W_H\ldots W_2W_1X\|_F^2 \tag{72}$$

where $X \in \mathbb{R}^{d_x\times N}$ and $Y \in \mathbb{R}^{d_y\times N}$ are the input and output training examples. Assume XX^T and XY^T are of full rank with $d_y \leq d_x$ and $\Sigma = YX^T(XX^T)^{-1}XY^T$ has d_y distinct eigenvalues. Then for all H, the objective function f appearing in (72) *has no spurious local minima and obeys the strict saddle property if $(W_H\ldots W_2)$ has max rank.*

In Theorem 5.1 and Theorem 5.2, if we take $X = I$ the identity matrix, we have the matrix factorization problem, which is our problem of interest. In summary, theorems Theorem 5.1, Theorem 5.2 and many others in the literature want to convey a simple message: in classical setting of full support, the problem of training linear neural network (or matrix factorization in particular) with square loss has nice properties which allow simple algorithm like gradient descent to converge to globally optimal solution. These properties are the non-existence of spurious local minimum and the strict saddle property. Indeed, gradient descent is likely not to be stuck at any saddle point but a local minimum and when the algorithm converges to a local minimum, it converges to a global minimum.

At first sight, it seems Theorem 5.1 and Theorem 5.2 completely explain the surprisingly good behavior of first order method. However, it does not really. The function in Figure 13 has both properties but gradient descent will be doomed if it is not initialized properly. The function has both properties: it has only one local (and global) minimum and all other first order critical points are saddle points. Nevertheless, if the initialized solution is on the right side of the peak, the gradient descent will push the solution to infinity, which is not the global minimum. You might wonder what is the function described by Figure 13. It is derived from the problem Figure 68. We consider the problem (68) where $Y \in \mathbb{R}^{3\times 3}$, $A \in \mathbb{R}^{3\times 2}$, $B \in \mathbb{R}^{2\times 3}$. C_A and C_B are defined by the following support.

$$suppA = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 0 \end{pmatrix}, suppB = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

where the index having value 1 means it is in the support.

With the given $suppA, suppB$, we want to factorize Y as:

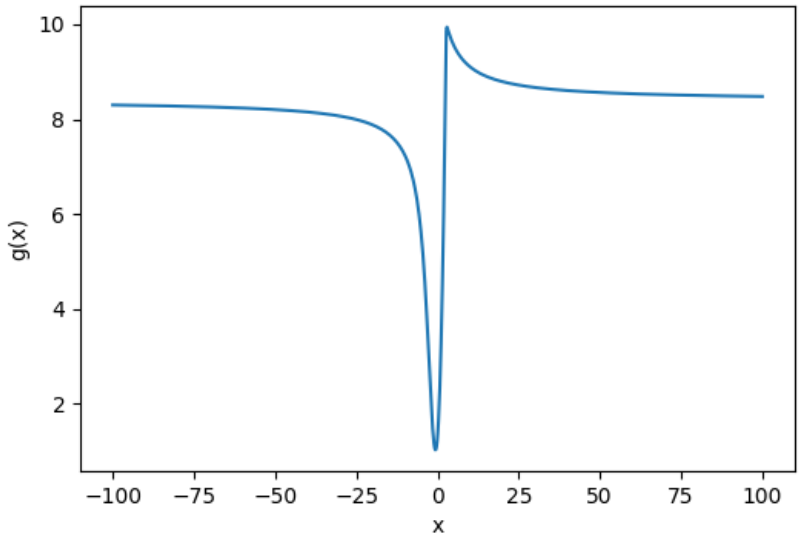


Figure 13: A function resisting first order method

$$
\begin{aligned}
Y &= \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & 0 \\ a_{31} & 0 \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & 0 & 0 \end{pmatrix} \\
&= \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} & a_{11}b_{13} \\ a_{21}b_{11} & a_{21}b_{12} & a_{21}b_{13} \\ a_{31}b_{11} & a_{31}b_{12} & a_{31}b_{13} \end{pmatrix} \\
&= \begin{pmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{11}b_{13} \\ a_{21}b_{11} & a_{21}b_{12} & a_{21}b_{13} \\ a_{31}b_{11} & a_{31}b_{12} & a_{31}b_{13} \end{pmatrix} + \begin{pmatrix} a_{12}b_{21} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \\
&= \begin{pmatrix} a_{11} \\ a_{21} \\ a_{31} \end{pmatrix} \otimes \begin{pmatrix} b_{11} \\ b_{12} \\ b_{13} \end{pmatrix} + \begin{pmatrix} x & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}
\end{aligned} \tag{73}
$$

where $x = a_{12}b_{21}$.

Thus, Y is the sum between a matrix of rank one and a matrix whose support contains only index $(1, 1)$. Illustration of the landscape of $f(A, B) = \|AB - Y\|_F^2$ is infeasible because A, B has 8 variables. However, we could get a grasp of the landscape of f by considering another function $g(x) = \min_{A,B,a_{12}b_{21}=x} f$. In words, we fixed x and try to minimize f. This is much easier since all we need to do is calculate the best rank 1 approximation of a matrix. Figure 13 is actually graph of g where Y is a Gaussian random matrix. Although it is not exactly the landscape of f, we can still deduce that gradient descent has to be properly initialized in order to work well (otherwise, the solution converge to infinity).

Based on this graph as well as the Theorem 5.1 Theorem 5.2, we propose several conjectures:

Conjecture 5.1. *The objective function of problem* (68) *has no spurious local minima and obeys the strict saddle property regardless of C_A, C_B.*

Conjecture 5.2. *The objective function of problem* (71) *does not have the "bad" shape like $g(x)$.*

They are problems of interest in my upcoming PhD.

6 Summary and conclusion

In the internship, I have worked on the problem of (multilayer) sparse matrix factorization. The main tools that we extensively utilized is proximal operators and proximal algorithms. My main contribution during the internship is the proposal of a new proximal operator: Generalized Hungarian Method. That new proximal operator allows proximal algorithms such as PALM and hierarchical factorization to successfully reverse engineer the Hadamard Transform.

Another highlight in my internship is the proposal of hierarchical factorization hard thresholding pursuit (HFHTP). HFHTP with Hard Thresholding Pursuit as a mechanism to escape bad support outperforms PALM under many sparse constraints of factors. The robustness of HFHTP was shown by the factorization of the Hadamard Transforms and randomly generated Gaussian matrices. Aside default initialization, which is the only mode allowing PALM to function well, other types of initializations become more competitive with HFHTP. The performance is also improved with HFHTP in MEG matrix and randomly generated Gaussian matrices factorization. With constraints on both factors, HFHTP becomes more competitive since it has HTP as a mechanism to explore the set of support. In contrast, PALM does not have this mechanism and it is easily trapped in local minimum, which explains its sensitive behavior towards initialization.

Last but not least, although the fixed support factorization was fully understood in the scope of the internship, the topic sparks many interesting untouched realm and we will likely to carry on its research during my PhD. Finally, our algorithms are not able to perfectly factorize Discrete Fourier Transform yet. It is also a subject that I would try to address in the future.

Acknowledgement

I would like to express my deepest gratitude towards my supervisor, professor Gribonval Rémi. His guidance during the internship was great despite our geographical distance and many other inconveniences caused by the Coronavirus pandemic. His supervision opened up many horizons that I would not have had.

A special thanks to Mr. Hakim Hadj Djilani and Léon Zheng, my colleagues. Our discussions during the internship were proved to be important. I have learned many things from those two, from theoretical aspects to practical implementation.

Finally, my words of thanks are towards Laboratoire de l'informatique du parallélisme (LIP), ENS de Lyon. The professors, colleagues and administrative staffs are very friendly. The research environment in ENS de Lyon is amazing, which is a great encouragement to me to continuously work harder. Without their helpfulness and friendliness, I was not able to finish my internship.

Appendix A

$$B = \begin{pmatrix} 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 1 & -1 & 0 & 0 & -1 & 1 & 0 \\ 1 & 0 & -1 & 0 & 1 & 0. & -1 & 0 \\ 1 & 0 & 0 & -1 & -1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & -1 & 1 & -1 & 1 & 0 & 0 \end{pmatrix} \tag{74}$$

Lemma 6.1. *The matrix B in Equation* (74) *could not be factorized into* 2*-regular sparse matrices.*

Proof. Firstly, B is a 4-regular sparse matrix with $\text{supp}(B)[i,j] = 1$ if and only if $B[i,j] \neq 0$. It is clear that it is the unique valid support of B satisfying Definition 3.2.

Suppose that $B = CD$ where C and D are 2-regular sparse matrices with a valid support $\text{supp}(C)$ and $\text{supp}(D)$ satisfying Definition 3.2. Each row of B is thus a linear combination of rows of D. Since C has at most two non-zero entries per row, each row of B is a linear combination of at most two rows of D.

Let B_k be the k^{th} row of B. B_k has 4 nonzero entries. Moreover, $B_k = \alpha D_{k_1} + \beta D_{k_2}$ (D_{k_1}, D_{k_2} are rows of matrix D) and D_{k_1}, D_{k_2} has at most 2 nonzero entries. That results into D_{k_1} and D_{k_2} have exactly 2 nonzero entries each and they have disjoint support. In addition, $\alpha, \beta \neq 0$.

On the other hand, C are 2-regular sparse matrices, thus each column has at most 2 nonzero entries. Therefore for all l, there exists i, j, i_2, j_2 such that $B_i = \alpha_1 D_l + \beta_1 D_{i_2}, B_j = \alpha_2 D_l + \beta_2 D_{j_2}$ (in fact, that is equivalent to C having support at entries $(i, l), (j, l), (i, i_2), (j, j_2)$). Again, since all B_i, B_j have 4 nonzero entries, D_l must have exactly 2 nonzero entries and $\alpha_1, \alpha_2 \neq 0$.

Therefore, we could conclude that there exists two row B_i, B_j whose supports share a linearly dependent segment of size two. That is a contradiction (by simply checking all the rows of B). □

References

[1] P. F. Baldi and K. Hornik. Learning in linear neural networks: a survey. *IEEE Transactions on Neural Networks*, 6(4):837–858, 1995.

[2] E. M. L. Beale, M. G. Kendall, and D. W. Mann. The discarding of variables in multivariate analysis. *Biometrika*, 54(3/4):357–366, 1967.

[3] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Imaging Sciences*, 2:183–202, 01 2009.

[4] Thomas Blumensath and Mike Davies. Iterative thresholding for sparse approximations. *Journal of Fourier Analysis and Applications*, 14:629–654, 12 2008.

[5] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, USA, 2004.

[6] Antonin Chambolle, R.A. Vore, Nam-Yong Lee, and Bradley Lucier. Nonlinear wavelet image processing: Variational problems, compression, and noise removal through wavelet shrinkage. *Image Processing, IEEE Transactions on*, 7:319 – 335, 04 1998.

[7] S. CHEN, S. A. BILLINGS, and W. LUO. Orthogonal least squares methods and their application to non-linear system identification. *International Journal of Control*, 50(5):1873–1896, 1989.

[8] Scott Shaobing Chen, David L. Donoho, and Michael A. Saunders. Atomic decomposition by basis pursuit. *SIAM Review*, 43(1):129–159, 2001.

[9] William J. Cook, William H. Cunningham, William R. Pulleyblank, and Alexander Schrijver. Combinatorial optimization, 1997.

[10] Tri Dao, Albert Gu, Matthew Eichhorn, Atri Rudra, and Christopher Ré. Learning fast algorithms for linear transforms using butterfly factorizations. *CoRR*, abs/1903.05895, 2019.

[11] M. A. Davenport and M. B. Wakin. Analysis of orthogonal matching pursuit using the restricted isometry property. *IEEE Transactions on Information Theory*, 56(9):4395–4401, 2010.

[12] Simon Foucart. Hard thresholding pursuit: An algorithm for compressive sensing. *SIAM J. Numerical Analysis*, 49:2543–2563, 01 2011.

[13] Wolfgang Hackbusch. A sparse matrix arithmetic based on h-matrices. *I. Introduction to Hmatrices, Computing*, pages 89–108, 1999.

[14] P. Hall. On representatives of subsets. *Journal of the London Mathematical Society*, s1-10(1):26–30, 1935.

[15] R. R. Hocking and R. N. Leslie. Selection of the best subset in regression analysis. *Technometrics*, 9(4):531–540, 1967.

[16] Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 42(1):80–86, 2000.

[17] Kenji Kawaguchi. Deep learning without poor local minima. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 586–594. Curran Associates, Inc., 2016.

[18] H. W. Kuhn and Bryn Yaw. The hungarian method for the assignment problem. *Naval Res. Logist. Quart*, pages 83–97, 1955.

[19] Luc Le Magoarou and Rémi Gribonval. Multi-layer Sparse Matrix Factorization. SPARS 2015 Signal Processing with Adaptive Sparse Structured Representations, July 2015.

[20] Stéphane Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Trans. Signal Process.*, 41(12):3397–3415, 1993.

[21] D. Needell and J.A. Tropp. Cosamp: Iterative signal recovery from incomplete and inaccurate samples. *Applied and Computational Harmonic Analysis*, 26(3):301 – 321, 2009.

[22] Yurii Nesterov. A method for solving the convex programming problem with convergence rate $o(1/k^2)$. 1983.

[23] Neal Parikh and Stephen Boyd. Proximal algorithms. *Found. Trends Optim.*, 1(3):127–239, January 2014.

[24] Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad. Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. In *Proceedings of 27th Asilomar Conference on Signals, Systems and Computers*, pages 40–44 vol.1, 1993.

[25] M. D. Plumbley. On polar polytopes and the recovery of sparse representations. *IEEE Transactions on Information Theory*, 53(9):3188–3195, 2007.

[26] V Rokhlin. Rapid solution of integral equations of classical potential theory. *Journal of Computational Physics*, 60(2):187 – 207, 1985.

[27] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[28] J. A. Tropp. Greed is good: algorithmic results for sparse approximation. *IEEE Transactions on Information Theory*, 50(10):2231–2242, 2004.

[29] J. A. Tropp and A. C. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on Information Theory*, 53(12):4655–4666, 2007.

[30] Zhihui Zhu, Daniel Soudry, Yonina C. Eldar, and Michael B. Wakin. The global optimization geometry of shallow linear neural networks. *CoRR*, abs/1805.04938, 2018.