ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) | 979-8-3503-4485-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICASSP48485.2024.10447412

DECENTRALIZED LOW RANK MATRIX RECOVERY FROM COLUMN-WISE PROJECTIONS BY ALTERNATING GD AND MINIMIZATION

Shana Moothedath and Namrata Vaswani

Iowa State University, Ames, IA, USA

ABSTRACT

This work studies our recently developed algorithm, decentralized alternating projected gradient descent algorithm (Dec-AltGDmin), for recovering a low rank (LR) matrix from independent column-wise linear projections in a decentralized setting. This means that the observed data is spread across L agents and there is no central coordinating node. Since this problem is non-convex and since it involves a subspace recovery step, most existing literature from decentralized optimization is not useful. We demonstrate using extensive numerical simulations and communication, time, and sample complexity comparisons that (i) existing decentralized gradient descent (GD) approaches fail, and (ii) other common solution approaches on LR recovery literature – projected GD, alternating GD and alternating minimization (AltMin) – either have a higher communication (and time) complexity or a higher sample complexity. Communication complexity is often the most important concern in decentralized learning.

Index Terms— Low Rank matrix recovery, compressed sensing, decentralized algorithms

1. INTRODUCTION

In this work we perform an extensive comparison of our recently developed decentralized algorithm, decentralized alternating projected gradient descent algorithm (Dec-AltGDmin) [7], for solving the following LR matrix recovery problem: recover an LR matrix from independent column-wise linear projections (LR column-wise Compressive Sensing (LRcCS)) [8, 5, 1]. We consider a decentralized setting, where the signals are spread across L agents and there is no central coordinating node; instead, agents exchange information through a communication network depicted by an undirected graph $\mathscr{G}$. Our setting finds application in federated sketching, where the objective is to restore data such as images or videos from their compressed signals collected at various geographically dispersed nodes, including mobile phones or IoT devices. Often, in such scenarios, a central coordinating node cannot communicate directly with all nodes due to spatial constraints, necessitating the adoption of a decentralized approach. Another application is in dynamic MRI or MRI, where a set of similar images from multiple patients across multiple decentralized sites with each site being a node [9].

In recent works [6, 10], we proposed a decentralized algorithm, Dec-AltGDmin, for solving the LRcCS problem. We presented convergence guarantee for Dec-AltGDmin under simple assumptions. In this paper, our goal is to compare the performance of Dec-AltGDmin with the standard LR matrix recovery algorithms in the literature. Since all of the existing LR algorithms are centralized, in principle a potential method to compare the decentralization side of our algorithm is by modifying the Decentralized Gradient Descent algorithms (DGD) introduced in [11] and [12]. DGD was proposed for decentralized convex optimization, hence a direct comparison with ours is not feasible since our problem is non-convex. To this end, we adapt DGD [11, 12] as discussed in Section 3.

1.1. Problem setting and notation

LRcCS problem aims to recover a set of q n-dimensional vectors/signals $\boldsymbol{X}^\star := [\boldsymbol{x}_1^\star, \boldsymbol{x}_2^\star, \ldots, \boldsymbol{x}_q^\star]$ such that the $n \times q$ matrix $\boldsymbol{X}^\star$ has rank $r \ll \min(n,q)$, from m-length projections $\boldsymbol{y}_k$ given by

$$\boldsymbol{y}_k := \boldsymbol{A}_k \, \boldsymbol{x}_k^\star, \; k = 1, 2, \ldots, q. \tag{1}$$

The $m \times n$ matrices $\boldsymbol{A}_k$ are known and mutually independent for different k. We consider $\boldsymbol{y}_k$'s are low-dimensional signals and hence $m < n$ and the goal is to have to use as few number of samples m as possible. The total sample complexity is given by mq.

We consider a decentralized setting where there is no central coordinating node, each node of the network can only communicate with its neighboring nodes. We assume that there is a set of L distributed nodes/sensors, each of which obtains sketches (linear projections) of a disjoint subset of columns of $\boldsymbol{X}^\star$. We denote the set of columns sketched at node g by $\mathscr{S}_g$. The sets $\mathscr{S}_g$ form a partition of $[q] := \{1, 2, \ldots, q\}$, i.e., they are mutually disjoint and $\cup_{g=1}^{L} \mathscr{S}_g = [q]$. The communication network is specified by an undirected graph $\mathscr{G} = (V, E)$, where V denotes the set of nodes, with $|V| = L$, and E denotes the set of undirected edges. The neighbor set of the g^{th} node (sensor) is defined as by $\mathcal{N}_g := \{j : (g, j) \in E\}$.

Let us denote the reduced (rank r) Singular Value Decomposition (SVD) of the rank-r matrix $\boldsymbol{X}^\star$ as $\boldsymbol{X}^\star \overset{\text{SVD}}{=} \boldsymbol{U}^\star \, \Sigma^\star \, \boldsymbol{V}^{\star\top}$. Let κ be the condition number of $\Sigma^\star$. We define $\boldsymbol{B} := \boldsymbol{V}^\top$ and $\tilde{\boldsymbol{B}} := \Sigma \boldsymbol{V}^\top$. Thus $\boldsymbol{X}^\star \overset{\text{SVD}}{=} \boldsymbol{U}^\star \, \Sigma^\star \, \boldsymbol{B}^\star = \boldsymbol{U}^\star \, \tilde{\boldsymbol{B}}^\star$. For a matrix $\tilde{Z}$, we use Z to denote orthonormal basis. When computed using QR decomposition, $\tilde{Z} \overset{\text{QR}}{=} Z\boldsymbol{R}$. We denote the Frobenius norm as $\|\cdot\|_F$ and the induced ℓ_2 norm as $\|\cdot\|$. We use $\mathbf{e}_k$ to denote the k^{th} canonical basis vector and $h \in [d]$ for $h \in \{1, 2, \ldots, d\}$ for some integer d. We define the Subspace Distance (SD) measure between two matrices $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$ as $SD(\boldsymbol{U}_1, \boldsymbol{U}_2) := \left\| (I - \boldsymbol{U}_1 \boldsymbol{U}_1^\top) \boldsymbol{U}_2 \right\|_F$, where I is the identity matrix. Further, $^\top$ denotes matrix or vector transpose and $|\boldsymbol{z}|$ for a vector $\boldsymbol{z}$ denotes element-wise absolute values. We use $\mathbb{1}_{\text{statement}}$ to denote an indicator function that takes the value 1 if statement is true and zero otherwise. We use $\circ$ to denote component-wise multiplication (Hadamard product). We reuse c, C to denote different numerical constants in each use with $c < 1$ and $C \geqslant 1$. We make the following simple and commonly used assumptions.

Assumption 1 (Right singular vectors' incoherence)**.** *Assume that* $\max_k \|\boldsymbol{x}_k^\star\| = \max_k \|\boldsymbol{b}_k^\star\| \leq \sigma_{\max}^\star \mu \sqrt{r/q}$ *for a constant* $\mu \geq 1$ *(μ does not grow with n, q, r). This further implies that* $\max_k \|\boldsymbol{x}_k^\star\| \leq \kappa \mu \|\boldsymbol{X}^\star\|_F / \sqrt{q}$.

Assumption 2. *The matrices $\boldsymbol{A}_k$ are independent and identically distributed (i.i.d.) random Gaussian (each matrix entry is i.i.d. standard Gaussian). The graph, $\mathscr{G}$, of the network topology is connected.*

1.2. Related work

The LRcCS problem has been studied in the *centralized* setting in three recent works. The first is an Alternating Minimization solution

	Sample Comp. $mq \gtrsim$	Time Comp. per iter	Comm. Comp. per iter per node	Private
Altmin [1, 2]	$nr^2 \log(\frac{1}{\varepsilon})$	$mqnr \cdot \log(\frac{1}{\varepsilon})$	$nr \log(\frac{1}{\varepsilon})$	Yes
AltGD [3]	unknown	$mqnr$	nr	No
ProjGD [4]	unknown	$mqnr \cdot \log(\frac{1}{\varepsilon})$	nq	No
AltGDmin [5]	$(n+q)r^2 \log(\frac{1}{\varepsilon})$	$mnqr$	nr	Yes
Dec-Altmin	$nr^2 \log(\frac{1}{\varepsilon})$	$mqnr \cdot L^3 \log^2(\frac{1}{\varepsilon})$	$nr \log(\frac{1}{\varepsilon})$	Yes
Dec-AltGDmin (this paper, proposed in [6])	$(n+q)r^2 \log(\frac{1}{\varepsilon})$	$mqnr \cdot L^3 \log \frac{1}{\varepsilon}$	nr	Yes

Table 1: The time costs are calculated assuming zero cost for addition/subtraction operations.

(Altmin) that solves the harder magnitude only generalization of LRcCS, referred to as LRPR (LR Phase Retrieval) [1, 2]. The second, parallel work, studies a convex relaxation called mixed norm min [8]. The third [5] is a gradient descent (GD) based provable solution to LRcCS, that we called AltGDmin. The convex solution exhibits very slow performance, with notably poor experimental results and an even higher sample complexity compared to AltGDmin [5] in scenarios demanding precise recovery. The AltMin approach [1, 2] also demonstrates significantly slower performance than AltGDmin. Furthermore, due to its focus towards a more challenging problem (LRPR), its sample complexity guarantee for LRcCS falls short in comparison to that of AltGDmin, resulting in inferior recovery performance with fewer samples [5]. Other existing algorithms in LR literature that can have been theoretically studied only for LR matrix completion include projected GD (ProjGD) introduced in [4, 13] and alternating GD (AltGD) introduced in [3]. Both of these works were for LR matrix completion problem, however, is extendable to the LRcCS problem. The major bottleneck of these algorithms is that both of them are not private. Recently, LRcCS has been studied in the decentralized setting in our prior works [6, 7, 10, 14]. In [7] we presented the Dec-AltGDmin algorithm and an empirical validation using generated data. In [6, 10], we presented the theoretical guarantee and and its proof. In [14], we compared Dec-AltGDmin with the DGD approach in [11] and [12], both theorems and experiments.

Projected GD is a GD-based solution approach for solving constrained optimization problems, involving the projection of each GD step's output onto a given constraint set. Over the last decade, the formulation of decentralized GD and projected GD algorithms has garnered significant interest [11, 15, 16, 17, 18, 19, 20, 12], starting with the seminal work of Nedić *et al.* [11]. Recent advancements also explore the application of projected GD in constrained optimization problems [15, 18, 19] or for establishing the consensus constraints [18]. However, all existing approaches that come with guarantees assume convex cost functions and either no constraints or convex constraint sets. While there exist some works that considered non-convex functions [21], they impose additional assumptions.

1.3. Contributions

There are two broad classes of approaches for obtaining a decentralized solution for LRcCS. The first involves considering the AltGDmin solution [5], which is known to be sample, communication and time efficient for the centralized federated setting, and using decentralized GD approaches to modify its GD step. The two most well-known decentralized GD algorithms are those of Nedic *et al.*[11] and Yuan *et al.* [12].

The second approach is to consider modifications of other fast approaches from LR matrix recovery literature. Convex relaxation solutions, studied in [8] for LRcCS, are known to be very slow and hence we do not consider these. Direct iterative approaches involve three types of solutions - AltMin, AltGD, and ProjGD. Factorize the unknown matrix $\boldsymbol{X}$ as $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{B}$ with $\boldsymbol{U}, \boldsymbol{B}$ being matrices with r columns and rows respectively, and consider $f(\boldsymbol{X}) = f(\boldsymbol{U}\boldsymbol{B}) = \sum_k ||\boldsymbol{y}_k - \boldsymbol{A}_k \boldsymbol{x}_k||^2 = ||\boldsymbol{y}_k - \boldsymbol{A}_k \boldsymbol{U} \boldsymbol{b}_k||^2$. After initializing $\boldsymbol{U}$ using spectral initialization, AltMin alternatively updates $\boldsymbol{B}$ by minimizing $f(\boldsymbol{U}\boldsymbol{B})$ over it keeping $\boldsymbol{U}$ fixed, and vice versa. After each update, it orthonormalizes the columns of $\boldsymbol{U}$. This is one way to prevent the norm of one of $\boldsymbol{U}$ or $\boldsymbol{B}$ from growing in an unbounded fashion. AltGD, which was studied in [3] for solving the (robust) LR matrix completion problem, replaces minimization in AltMin by a GD step for both, and it does not orthonormalize $\boldsymbol{U}$: the reason is once we orthonormalize $\boldsymbol{U}$, the previous estimate of $\boldsymbol{B}$ becomes useless. Instead, it implements AltGD for $f(\boldsymbol{U}\boldsymbol{B}) + ||\boldsymbol{U}^\top \boldsymbol{U} - \boldsymbol{B}\boldsymbol{B}^\top||_F$. This second term helps balance the norms of $\boldsymbol{U}$ and $\boldsymbol{B}$. ProjGD, which was studied in [4, 13] for solving the (robust) LR matrix completion problem, uses projected GD on $\boldsymbol{X}$: one GD step for $\boldsymbol{X}$ followed by projection onto the space of rank r matrices.

In this work, we demonstrate the following using simulations:

1. DGD applied to the GD step of AltGDmin fail to converge. The reason is the GD step updates $\boldsymbol{U}$. DGD applied to this involves one local GD step at each node followed by taking the numerical average of the updated $\boldsymbol{U}$'s. However, for subspaces, numerical averaging does not improve estimation accuracy. For example, given $SD(\boldsymbol{U}_1, \boldsymbol{U}^\star) \leq \varepsilon$ and $SD(\boldsymbol{U}_2, \boldsymbol{U}^\star) \leq \varepsilon$, we can only argue that $SD((\boldsymbol{U}_1 + \boldsymbol{U}_2)/2, \boldsymbol{U}^\star) \leq \varepsilon$, we cannot argue that this is less than $\varepsilon/2$. We considered two versions of DGD, (i) DGD with a single consensus loop (average aggregation of neighbors' estimates) as in [11, 12] and (ii) DGD with multiple consensus iterations (Figure 1).
2. Even centralized (easier) version of AltGD and projected GD do not converge for small values of m. In past work we explained why it is not clear if it is possible to obtain convergence guarantees for these approaches under the small sample complexity that works for AltGDmin. In this work, we demonstrate this fact numerically (Figure 2).

Algorithm 1 AvgCons: Equal-neighbor average consensus

Input: $Z_{\text{in}}^{(g)}$, for all $g \in [L]$ ($Z_{\text{in}}^{(g)}$ is a matrix)
Parameters: T_{con}, $\mathscr{G}$ (graph connectivity)
Initialize $Z_0^{(g)} \leftarrow Z_{\text{in}}^{(g)}$, for all $g \in [L]$
for $t = 1$ to T_{con} **do**
  $Z_{t+1}^{(g)} \leftarrow Z_t^{(g)} + \sum_{j \in \mathcal{N}_g} \frac{1}{d_g}\left(Z_t^{(j)} - Z_t^{(g)}\right)$, for $g \in [L]$
end for
Output: $Z_{\text{out}}^{(g)} \leftarrow L \cdot Z_{T_{\text{con}}}^{(g)}$

3. Moreover even when AltGD and projected GD do converge, we demonstrate numerically that both take much longer (ProjGD is slow per iteration while AltGD needs many more iterations to converge). We also demonstrate by simulations that Altmin is much slower (per iteration cost of Altmin is very high (Figure 2).

In addition, we also provide a detailed communication and time per iteration, sample complexity, and privacy comparison of all approaches in Table 1. ProjGD is very inefficient because the partial gradients from the different nodes will be of size $n \times q$. AltGD uses factorization, but because its cost function contains a norm balancing term, the update of a column of $\boldsymbol{B}$ depends on entries of all columns of $\boldsymbol{B}$. For this reason it requires two data exchanges per iteration. Also, due to this, AltGD is not private. Projected GD is also not private because the center estimates $\boldsymbol{X}$. Only AltGDmin and Altmin (with GD used for updating $\boldsymbol{U}$) are private: the center cannot estimate the entire $\boldsymbol{X}$, it can only estimate $\boldsymbol{U}$.

2. THE PROPOSED ALGORITHM AND GUARANTEE

We present the Dec-AltGDmin algorithm in Algorithm 3. There are two key steps. We decompose $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{B}$ and define

$$f(\boldsymbol{U},\boldsymbol{B}) = \sum_{g=1}^{L} f_g(\boldsymbol{U},\boldsymbol{B}), \text{ where } f_g(\boldsymbol{U},\boldsymbol{B}) = \sum_{k \in \mathscr{S}_g} \|\boldsymbol{y}_k - \boldsymbol{A}_k \boldsymbol{U} \boldsymbol{b}_k\|^2. \quad (2)$$

At each GD iteration, for each new estimate of $\boldsymbol{U}$, we first solve for $\boldsymbol{B}$ by minimizing $f(\boldsymbol{U},\boldsymbol{B})$ over it while keeping $\boldsymbol{U}$ fixed at its current value. Then we compute $\tilde{\boldsymbol{U}} = \boldsymbol{U} - \eta \nabla_U f(\boldsymbol{U},\boldsymbol{B})$ and orthonormalize it using QR decomposition as $\boldsymbol{U} = \text{QR}(\tilde{\boldsymbol{U}})$. Here $\nabla_U f(\boldsymbol{U},\boldsymbol{B}) = \sum_{k=1}^{q} \boldsymbol{A}_k^\top (\boldsymbol{A}_k \boldsymbol{U} \boldsymbol{b}_k - \boldsymbol{y}_k)\boldsymbol{b}_k^\top$.

To initialize, we compute $\boldsymbol{U}_0$ as the top r left singular vectors of $\boldsymbol{X}_0 = (1/m)\sum_{k \in [q]} \boldsymbol{A}_k^\top \boldsymbol{y}_{k,\text{trunc}}(\alpha) e_k^\top$ with $\alpha := \tilde{C}\frac{\sum_{ki}(\boldsymbol{y}_{ki})^2}{mq}$ and $\boldsymbol{y}_{k,\text{trunc}}(\alpha) := \boldsymbol{y}_k \circ \mathbb{1}_{\{\boldsymbol{y}_{ki}^2 \leqslant \alpha\}}$. This is $\boldsymbol{y}_k$ with large magnitude entries zeroed out. To compute $\boldsymbol{X}_0$, we use an average consensus algorithm given in Algorithm 1. [22]. Let T_{con} denote the number of iterations of the consensus algorithm and $Z_{\text{in}}^{(g)}$, for $g \in [L]$, be the input. For $Z_0^{(g)} := Z_{\text{in}}^{(g)}$ average consensus updates as

$$Z_{t+1}^{(g)} = Z_t^{(g)} + \sum_{j \in \mathcal{N}_g} \mathbf{W}_{gj}\left(Z_t^{(j)} - Z_t^{(g)}\right), \text{ for } g \in [L] \quad (3)$$

and outputs $\text{AvgCon}_{(g)}(\{Z_{\text{in}}^{(g)}\}_{g=1}^{L}, \mathscr{G}, T_{\text{con}}) := Z_{\text{out}}^{(g)} = L \cdot Z_{T_{\text{con}}}^{(g)}$, for all $g \in [L]$. Here the mixing matrix $\mathbf{W}$ is symmetric and doubly stochastic. During initialization and the main algorithm, we use average consensus at three places. In the initialization step, to approximate the threshold α and the top r singular vectors of $\boldsymbol{X}_0$ (which are equal to those of $\boldsymbol{X}_0\boldsymbol{X}_0^\top$) computed using the PM, and in the ProjGD iterations to approximate the sum of the individual gradients at all the nodes, i.e. compute $\nabla_U f(\boldsymbol{U},\boldsymbol{B}) = \sum_{k \in [q]} \boldsymbol{A}_k^\top (\boldsymbol{A}_k \boldsymbol{U}(\boldsymbol{b}_k) - \boldsymbol{y}_k)(\boldsymbol{b}_k)^\top$.

Theorem 2.1. *Assume that Assumptions 1 and 2 hold. Consider*

Algorithm 2 Initialization

Let $\boldsymbol{y}_k \equiv \boldsymbol{y}_k^{(00)}, \boldsymbol{A}_k \equiv \boldsymbol{A}_k^{(00)}$ for all $k \in [q]$.
$\alpha_{\text{in}}^{(g)} \leftarrow 9\kappa^2\mu^2 \frac{1}{mq} \sum_{k \in \mathscr{S}_g} \sum_{i=1}^{m} \boldsymbol{y}_{ki}^2 \; \forall g \in [L]$
$\alpha^{(g)} \leftarrow \text{AvgCons}_g(\alpha_{\text{in}}^{(g')}, g' \in [L], \mathscr{G}, T_{\text{con}})$
Let $\boldsymbol{y}_k \equiv \boldsymbol{y}_k^{(0)}, \boldsymbol{A}_k \equiv \boldsymbol{A}_k^{(0)}$
$\boldsymbol{y}_{k,\text{trunc}} := \boldsymbol{y}_k \circ \mathbb{1}\{|\boldsymbol{y}_{ki}| \leq \sqrt{\alpha^{(g)}}\} \; \forall k \in \mathscr{S}_g, g \in [L]$
Compute $\boldsymbol{X}_0^{(g)} = \left[\frac{1}{m}\boldsymbol{A}_k^\top \boldsymbol{y}_{k,\text{trunc}}, \; k \in \mathscr{S}_g\right], \forall g \in [L]$
Generate $\tilde{\boldsymbol{U}}_{\text{init}}$ with each entry i.i.d. standard Gaussian (use the same random seed at all nodes) for all $g \in [L]$
Orthonormalize $\tilde{\boldsymbol{U}}_{\text{init}}$ using QR to get $\boldsymbol{U}_{\text{init}}$ for all $g \in [L]$
$\boldsymbol{U}^{(g)} \leftarrow \boldsymbol{U}_{\text{init}}$
for $\tau = 1$ **to** T_{pm} **do**
  $\tilde{\boldsymbol{U}}_{\text{in}}^{(g)} \leftarrow \boldsymbol{X}_0^{(g)} \boldsymbol{X}_0^{(g)\top} \boldsymbol{U}^{(g)}$, for all $g \in [L]$
  $\tilde{\boldsymbol{U}}^{(g)} \leftarrow \text{AvgCons}_g(\tilde{\boldsymbol{U}}_{\text{in}}^{(g')}, g' \in [L], \mathscr{G}, T_{\text{con}})$
  QR on $\tilde{\boldsymbol{U}}^{(1)} \overset{\text{QR}}{=} \boldsymbol{U}^{(1)}\boldsymbol{R}^{(1)}$ to get $\boldsymbol{U}^{(1)}$
  $\boldsymbol{U}_{\text{in}}^{(1)} = \boldsymbol{U}^{(1)}, \boldsymbol{U}_{\text{in}}^{(g)} = \boldsymbol{0}, g \neq 1$
  $\boldsymbol{U}^{(g)} \leftarrow \text{AvgCons}_g(\boldsymbol{U}_{\text{in}}^{(g')}, g' \in [L], \mathscr{G}, T_{\text{con}})$ for $g \neq 1$
end for
Output $\boldsymbol{U}_0^{(g)} \leftarrow \boldsymbol{U}^{(g)}$

Algorithm 3 initialized using Algorithm 2. Let $\eta = 0.4/m\sigma_{\max}^{\star\,2}$, $T_{pm} = C\kappa^2(\log n + \log\kappa)$, $T = C\kappa^2\log(1/\varepsilon)$, $T_{\text{con}} = CL^3(T + \log(1/\varepsilon) + \log n + \log L + \log\kappa)$. *Assume that* $mq \geq C\kappa^6\mu^2(n+q)r\left(\kappa^2 r + \log(1/\varepsilon)\right)$. *Then, w.p. at least* $1 - 1/n$,

$$\text{SD}_2(\boldsymbol{U}_T^{(g)}, \boldsymbol{U}^\star) \leq \varepsilon, \text{ and } \|\boldsymbol{x}_k - \boldsymbol{x}_k^\star\| \leq \varepsilon\|\boldsymbol{x}_k^\star\| \text{ for all } k \in \mathscr{S}_g, g \in [L].$$

3. SIMULATIONS

The data was generated as follows. We note that, $\boldsymbol{X}^\star = \boldsymbol{U}^\star\boldsymbol{B}^\star$, where $\boldsymbol{U}^\star$ is an $n \times r$ orthonormal matrix. We generate the entries of $\boldsymbol{U}^\star$ by orthonormalizing an i.i.d standard Gaussian matrix. The entries of $\boldsymbol{B}^\star \in \mathbb{R}^{r \times q}$ are generated from an i.i.d Gaussian distribution. The matrices $\boldsymbol{A}_k$s were i.i.d. standard Gaussian.

Experiment 1: Comparison with existing decentralized approach. In this experiment, we compared the performance of our Dec-AltGDmin algorithm with the AltGDmin algorithm in [5] for a specific node, node g, utilizing the information available *only* at node g. We note that Dec-AltGDmin is the only algorithm with performance guarantees that addresses the non-convex LRcCS problem using a decentralized approach. A primary method for comparing our algorithm with the existing algorithms in the literature involves adapting the existing Decentralized Gradient Descent algorithms (DGD) introduced in references [11] and [12], initially designed for convex problems. For this, we modified the centralized AltGDmin algorithm using the DGD [11, 12] algorithm as follows. We replaced the centralized projected GD step for $\boldsymbol{U}$ by the DGD approach: $\boldsymbol{U}_+^{(g)} \leftarrow QR(\frac{1}{d_g}\sum_{g' \in \mathcal{N}_g} \boldsymbol{U}^{(g')} - \eta \nabla_{\boldsymbol{U}} f_g(\boldsymbol{U}^{(g)}, \boldsymbol{B}^{(g)}))$. While references [11] and [12] primarily focused on convex functions, the need for a specially designed initialization was not as crucial in those cases. However, for our non-convex problem, a meticulous initialization, known as spectral initialization, is essential. For our comparative study, we ensured that all three algorithms were initialized in a consistent manner using the spectral initialization. All data points in the figures are averaged over are 100 Monte-Carlo trials.

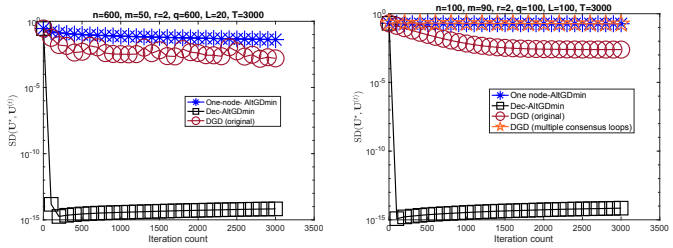
We evaluated and compared the three algorithms by considering networks of varying sizes and datasets with different dimensions.

Algorithm 3 The complete Dec-AltGDmin algorithm. It calls the AvgCons function summarized in Algorithm 1.

Input: $\boldsymbol{A}_k, \boldsymbol{y}_k$, $k \in \mathscr{S}_g$, $g \in [L]$ graph $\mathscr{G}$
Output: $\boldsymbol{U}^{(g)}$, $\boldsymbol{B}^{(g)}$ and $\boldsymbol{X}^{(g)} = \boldsymbol{U}^{(g)}\boldsymbol{B}^{(g)}$.
Parameters: η, T_{con}, T
Sample-split: Partition $\boldsymbol{A}_k, \boldsymbol{y}_k$ into $2T+2$ equal-sized disjoint sets: $\boldsymbol{A}_k^{(\ell)}, \boldsymbol{y}_k^{(\ell)}, \ell = 00, 0, 1, 2, \ldots 2T$
Initialization: Run Algorithm 2 to get $\boldsymbol{U}_0^{(g)}$.
AltGDmin iterations:
for $t = 1$ **to** T **do**
  $\boldsymbol{U}^{(g)} \leftarrow \boldsymbol{U}_{t-1}^{(g)}$
  Let $\boldsymbol{y}_k = \boldsymbol{y}_k^{(t)}, \boldsymbol{A}_k = \boldsymbol{A}_k^{(t)}$
  $\boldsymbol{b}_k \leftarrow (\boldsymbol{A}_k \boldsymbol{U}^{(g)})^{\dagger} \boldsymbol{y}_k \; \forall k \in \mathscr{S}_g$
  $\boldsymbol{x}_k \leftarrow \boldsymbol{U}^{(g)} \boldsymbol{b}_k \; \forall k \in \mathscr{S}_g$
  Let $\boldsymbol{y}_k = \boldsymbol{y}_k^{(T+t)}, \boldsymbol{A}_k = \boldsymbol{A}_k^{(T+t)}$.
  $\nabla f_g \leftarrow \sum_{k \in \mathscr{S}_g} \boldsymbol{A}_k^{\top} (\boldsymbol{A}_k \boldsymbol{U}^{(g)} \boldsymbol{b}_k - \boldsymbol{y}_k) \boldsymbol{b}_k^{\top}$
  $\widehat{\text{gradU}}^{(g)} \leftarrow \text{AvgCons}_g(\nabla f_{g'}, g' \in [L], \mathscr{G}, T_{\text{con}})$
  $\widetilde{\boldsymbol{U}}^{(g)} \leftarrow \boldsymbol{U}^{(g)} - \eta \widehat{\text{gradU}}^{(g)}$
  QR decompose $\widetilde{\boldsymbol{U}}^{(g)} \overset{\text{QR}}{=} \boldsymbol{U}^{(g)} \boldsymbol{R}^{(g)}$ to get $\boldsymbol{U}^{(g)}$
  $\boldsymbol{U}_t^{(g)} \leftarrow \boldsymbol{U}^{(g)}$
end for
Output $\boldsymbol{U}^{(g)}$, $\boldsymbol{B}^{(g)}$ and $\boldsymbol{X}^{(g)} = \boldsymbol{U}^{(g)}\boldsymbol{B}^{(g)}$

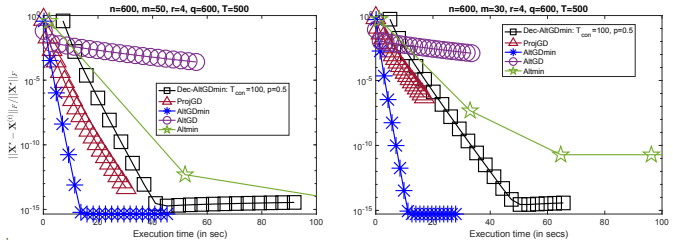
This comparative analysis allowed us to assess the performance of the different algorithms across a range of scenarios, providing a more robust understanding of their capabilities and limitations. In Figure 1 DGD (original) refers to the setting above with DGD [11, 12], DGD (multiple consensus loops) refers to the DGD idea in [11, 12] with multiple consensus loops, and One node: AltGDmin refers to AltGDmin running for a single node. Note that, the averaging using the neighbors' information in DGD [11, 12] only does one consensus iteration, i.e., $T_{\text{con}} = 1$. In DGD (multiple consensus loops), we modify DGD [11, 12] by performing multiple consensus iterations, i.e., $T_{\text{con}} = 100$. In Figure 1a we set $n = 600, m = 50, r = 2, q = 600$ and $p = 0.5$. In Figure 1a $L = 20$, i.e., the data is distributed across 20 agents, with each agent having access to only 5% of the data, both DGD (original) and One-node: AltGDmin fail to accurately estimate the subspace. In contrast, our proposed algorithm demonstrates accurate subspace estimation.

In Figure 1b we set $n = 100, m = 90, r = 2, q = 100, p = 0.5$, and $L = 100$. For a system with $n = 100$, $m = 90$ results in a compressed data size closely resembling the original data size, leading to minimal loss of information in the compressed signal. This characteristic is anticipated to facilitate an easier recovery process. We conducted 3000 GD iterations and averaged the results over 100 Monte-Carlo trials. In Figure 1b, it is evident that DGD (original) [11, 12], DGD (multiple consensus loops), and One node: AltGDmin fail to converge across both scenarios, whereas our proposed algorithm demonstrates convergence. The primary factor behind this contrast lies in the context of a larger network with 100 nodes with $q/L = 1$, each node has access to only one data point. This limitation renders it practically impossible for the One-node-AltGDmin algorithm to accurately approximate the true data. In the case of the DGD [11, 12] algorithm, the act of averaging subspace matrices, as performed in the GD step of [11] and [12], fails to enhance subspace error reduction due to the optimization variables being matrices instead of vectors. Thus when q/L is small, the DGD [11, 12] algorithm fails to reconstruct the actual data as the data samples available at a single node is not enough to learn the complete data, and also the averaging of subspace matrices of the neighboring nodes does not help to reduce the subspace error. From Figure 1b we notice that even after running multiple consensus iteration, the DGD [11, 12] algorithm does not converge. This is because averaging subspace matrices does not improve the subspace error.



(a) Small network: $L = 20$ (b) Large network: $L = 100$

Fig. 1: *Small network:* $L = 2, 20$, $n = q = 600$, $r = 2$, $m = 50$. *Large network:* $n = q = 100$, $r = 2$, $m = 40$. We compare performance of our algorithm Dec-AltGDmin with the centralized one from [5] for a single node and with the AltGDmin modified using the DGD approach of [11, 12]. In all figures, the network was generated as an ER graph with probability $p = 0.5$. In Figure 1b DGD (multiple consensus loops) refers to the setting with multiple consensus iterations for the DGD [11, 12] algorithm. The averaging using the neighbors' information in DGD [11, 12] only performs one consensus iteration. As shown even with this approach also DGD [11, 12] fails to converge.



(a) Error vs. Execution time: $m = 50$ (b) Error vs. Execution time: $m = 30$

Fig. 2: *In all cases* $n = 600$, $r = 4$, $q = 600$, $p = 0.5$, *and* $L = 20$. We compared performance of our proposed algorithm (Dec-AltGDmin) with the centralized counterpart from [5] and with the LR matrix recovery algorithms, Altmin [1, 2], ProjGD [4], and AltGD [3] algorithms. In all figures, the network was generated as an ER graph with probability $p = 0.5$.

Experiment 2: Comparison with existing centralized algorithms. In this experiment we compared the performance of the Dec-AltGDmin algorithm with that of three other existing algorithms that are widely studied in the LR literature, AltGDmin [5], AltMin [1, 2], AltGD [23], and ProjGD [4]. In this comparison, Dec-AltGDmin is the only decentralized algorithm, while the rest fall under the category of centralized algorithms. Dec-AltGDmin represents the decentralized version of AltGDmin, whereas Altmin, AltGD, and ProjGD [4] cannot be adapted for decentralized implementations. We set $n = 600$, $q = 600$, and $r = 4$. We chose different values of m in our experiments to evaluate the performance of the different algorithms based on the extend of compression the data undergoes. The number of agents is set as $L = 20$ and the network connectivity probability of the communication graph is set as $p = 0.5$ for all the cases. In Figs. 2a, 2b we varied the value of m as $50, 30$, respectively. From Figs. 2a and 2b we see that as the rate of compression increases (i.e., m/n decreases), the ProjGD, AltGD algorithms no longer converge. The AltGDmin and Dec-AltGDmin still converges for these highly compressed cases thereby validating the sample efficiency of these approaches over the other three.

4. REFERENCES

[1] S. Nayer and N. Vaswani, "Sample-efficient low rank phase retrieval," *IEEE Transactions on Information Theory*, 2021.

[2] S. Nayer, P. Narayanamurthy, and N. Vaswani, "Provable low rank phase retrieval," *IEEE Transactions on Information Theory*, March 2020.

[3] X. Yi, D. Park, Y. Chen, and C. Caramanis, "Fast algorithms for robust PCA via gradient descent," in *Neural Information Processing Systems (NeurIPS)*, 2016.

[4] P. Jain and P. Netrapalli, "Fast exact matrix completion with finite samples," in *Conference on Learning Theory*, 2015, pp. 1007–1034.

[5] S. Nayer and N. Vaswani, "Fast low rank column-wise compressive sensing," *IEEE Transactions on Information Theory*, 2022, to appear. Also at arXiv:2102.10217.

[6] Shana Moothedath and Namrata Vaswani, "Fast, communication-efficient, and provable decentralized low rank matrix recovery," *Arxiv: 2204.08117*, 2023.

[7] Shana Moothedath and Namrata Vaswani, "Fully decentralized and federated low rank compressive sensing," *in American Control Conference (ACC)*, 2022.

[8] Rakshith Sharma Srinivasa, Kiryung Lee, Marius Junge, and Justin Romberg, "Decentralized sketching of low rank matrices," in *Neural Information Processing Systems (NeurIPS)*, 2019, pp. 10101–10110.

[9] Silpa Babu, Sajan Goud Lingala, and Namrata Vaswani, "Fast low rank column-wise compressive sensing for accelerated dynamic mri," *IEEE transactions on computational imaging*, 2023.

[10] Shana Moothedath and Namrata Vaswani, "Dec-AltProjGD: Fully-decentralized alternating projected gradient descent for low rank column-wise compressive sensing," *Conference on Decision and Control (CDC)*, 2022.

[11] Angelia Nedic and Asuman Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, 2009.

[12] Kun Yuan, Qing Ling, and Wotao Yin, "On the convergence of decentralized gradient descent," *SIAM Journal on Optimization*, vol. 26, no. 3, pp. 1835–1854, 2016.

[13] Y. Cherapanamjeri, K. Gupta, and P. Jain, "Nearly-optimal robust matrix completion," *International Conference on Machine Learning*, 2016.

[14] Shana Moothedath and Namrata Vaswani, "Comparing decentralized gradient descent approaches and guarantees," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023, pp. 1–5.

[15] Angelia Nedić, Asuman Ozdaglar, and Pablo A Parrilo, "Constrained consensus and optimization in multi-agent networks," *IEEE Transactions on Automatic Control*, vol. 55, no. 4, pp. 922–938, 2010.

[16] Angelia Nedić, "Convergence rate of distributed averaging dynamics and optimization in networks," *Foundations and Trends® in Systems and Control*, vol. 2, no. 1, pp. 1–100, 2015.

[17] Soomin Lee and Angelia Nedić, "Distributed random projection algorithm for convex optimization," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 2, pp. 221–229, 2013.

[18] Alexander Rogozin and Alexander Gasnikov, "Projected gradient method for decentralized optimization over time-varying networks," *ArXiv preprint arXiv:1911.08527*, 2019.

[19] Firooz Shahriari-Mehr, David Bosch, and Ashkan Panahi, "Decentralized constrained optimization: Double averaging and gradient projection," *arXiv preprint arXiv:2106.11408*, 2021.

[20] Ilan Lobel and Asuman Ozdaglar, "Distributed subgradient methods for convex optimization over random networks," *IEEE Transactions on Automatic Control*, vol. 56, no. 6, pp. 1291–1306, 2010.

[21] Ran Xin, Usman A Khan, and Soummya Kar, "A fast randomized incremental gradient method for decentralized non-convex optimization," *IEEE Transactions on Automatic Control*, 2021.

[22] Alex Olshevsky and John N Tsitsiklis, "Convergence speed in distributed consensus and averaging," *SIAM journal on control and optimization*, vol. 48, no. 1, pp. 33–55, 2009.

[23] X. Yi, D. Park, Y. Chen, and C. Caramanis, "Fast algorithms for robust pca via gradient descent," in *Advances in neural information processing systems*, 2016, pp. 4152–4160.