Mathematical Formulations, Algorithms and Theory for Big Data Problems

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Vahan Huroyan

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Gilad Lerman

August, 2018

Acknowledgements

I would like to thank my adviser, Prof. Gilad Lerman for his help, instructions and support over the years. Without his help and support, this thesis work would not have been possible.

I am grateful to the members of my dissertation committee, Prof. Arnab Sen, Prof. William Leeb, and Prof. Ankur Mani, who were very kind to agree to be on the advisory board.

I have had the fortune of collaborating with a variety of professors and researchers during my time in graduate school. I want to thank my collaborators Prof. Hau-Tieng Wu (Duke University), Dr. Deepti Pachauri (3M). Hassan Mansour (Mitsubishi Electric Research Laboratories) and Mouhacine Benosman (Mitsubishi Electric Research Laboratories).

This work was supported by NSF awards DMS-09-56072 and DMS-14-18386 and The University of Minnesota External Stakeholder Award, which was pursued with 3M.

Dedication

To my parents, Hakob Huroyan and Ruzanna Shahbazyan.

Abstract

This is a collection of works that I have done during my PhD research at the University of Minnesota. There are three parts dedicated to different topics, of which abstracts are included below.

Abstract for Distributed Robust Subspace Recovery

We propose distributed solutions to the problem of Robust Subspace Recovery (RSR). Our setting assumes a huge dataset in an ad hoc network without a central processor, where each node has access only to one chunk of the dataset. Furthermore, part of the whole dataset lies around a low-dimensional subspace and the other part is composed of outliers that lie away from that subspace. The goal is to recover the underlying subspace for the whole dataset, without transferring the data itself between the nodes. We first apply the Consensus Based Gradient method to the Geometric Median Subspace algorithm for RSR. For this purpose, we propose an iterative solution for the local dual minimization problem and establish its r-linear convergence. We then explain how to distributedly implement the Reaper and Fast Median Subspace algorithms for RSR. The proposed algorithms display competitive performance on both synthetic and real data.

Abstract for Solving Jigsaw Puzzles By The Connection Graph Laplacian

We propose a novel mathematical framework to address the problem of automatically solving large jigsaw puzzles. The latter problem assumes a large image, which is cut into equal square pieces that are arbitrarily rotated and shuffled and asks to recover the original image given the rotated and shuffled pieces. We suggest a method for recovering the unknown orientations of the puzzle pieces by using the connection graph Laplacian associated with the puzzle. The connection graph Laplacian is also used to form a metric between puzzle pieces and this metric is more accurate than the commonly used metric. Numerical experiments demonstrate the competitive performance of the proposed method.

Abstract for Non-convex Analysis of Multi-Graph Matching

We propose an iterative algorithm together with its theoretical analysis for the Multi-Graph Matching (MGM) problem. The latter problem assumes a set of graphs, each of

which has the same number of vertices and further assumes that for each pair of graphs there exists a one-to-one correspondence map between their vertices. Given only noisy measurements of the mutual correspondences, the MGM problem asks to improve the correspondence maps between pairs of them. Our proposed algorithm iteratively solves the non-convex optimization problem associated with the MGM problem. We prove that for a specific noise model if the initial point of our proposed iterative algorithm is good enough, the algorithm linearly converges to the unique solution. Furthermore, we show how to find such an initial point. Numerical experiments demonstrate competitive speed and recovery results for our proposed algorithm with a state-of-the-art method.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In machine learning and data analysis, discrete optimization is a topic that consists of finding an optimal object within a finite set of objects. Due to the computational complexity, when the number of objects is too large, brute-force type algorithms are infeasible. Moreover, a small amount of error in measurements might drastically affect the result of such algorithms.

Recently, many successful algorithms suggest to find a larger set, usually convex but possibly non-convex, which contains the discrete set and solve the problem in this setting, rather than solving directly for the discrete set. After finding the solution, the next challenge is to find a way to project it onto the initial discrete set.

Due to the success of such algorithms in recent years, this has become one of the most active research areas in machine learning. In my dissertation I present my results for three problems in this context: Distributed Robust Subspace Recovery, Automatic Solution of Large Jigsaw Puzzles and Multi-Graph Matching.

Chapter 2

Part I: Distributed Robust Subspace Recovery

2.1 Introduction

Distributed computing is a central theme in modern computation. Its setting includes a system with multiple components, which communicate and coordinate in order to achieve their common computational goal. A special distributed setting assumes a central processor, which is connected to all other processors. This processor contains no data, but has enough memory to handle some computations, such as averaging communicated estimates. A more general distributed setting assumes an arbitrarily connected network of processors, among which the data is partitioned. Each processor computes a local estimate of the desired output based on its local data and on estimates passed by its neighbors. Then, it communicates its estimate to its neighbors. This procedure iterates until convergence.

Some common approaches for solving distributed computing problems are the diffusion method [3], the Consensus-Based Gradient Ascent (CBGA) [4, 5, 6, 7], the distributed subgradient method [8, 9] and the Consensus Alternating Direction Method of Multipliers (CADMM) [6, 9, 10, 11, 12]. Some of these algorithms have been successfully adapted to important applied problems of signal processing and wireless communications [12,13,14,15]. Various distributed algorithms have been proposed for the important problem of Principal Component Analysis (PCA) and related problems, such as the total

least squares. Most of them are for centrally-processed networks [16, 17, 18, 19, 20, 21], but some of them are for arbitrarily connected networks [4, 22]. To the best of our knowledge there are no distributed algorithms for robust versions of PCA.

This work discusses distributed algorithms for Robust Subspace Recovery (RSR) with arbitrarily connected networks. RSR is an alternative paradigm for PCA that is more robust to outliers. The underlying problem of RSR assumes data points, composed of inliers and outliers, where the inliers are well-explained by an affine low-dimensional subspace and the outliers come from a different model. The goal is to recover the underlying subspace in the presence of outliers. A careful review of the problem and its solutions appears in [23].

We first suggest a distributed implementation for the Geometric Median Subspace (GMS) [24] algorithm for RSR, which applies to arbitrarily connected networks. We propose an iterative algorithm for the local dual problem and establish its r-linear convergence (defined later in Definition 2). We also propose distributed implementations for two other RSR algorithms: Reaper [25] and FMS [26]. This is done by iterative application of distributed PCA. On the other hand, the GMS implementation does not iterate the distributed scheme and is thus more efficient in terms of the communication cost. We remark that the theorems for robustness of GMS, Reaper and FMS carry over to our distributed setting.

This chapter is organized as follows: §2.2 contains a short introduction to CBGA and its convergence analysis; §2.3 proposes the distributed CBGA algorithm for GMS and discusses its various properties; §2.4 proposes immediate distributed implementations for the Reaper and FMS algorithms; and §2.5 concludes with numerical experiments that test the proposed algorithms for distributed RSR. Sections 2.6.1 and 2.6.2 use ideas of §2.2 to solve the problems of distributed PCA and distributed geometric median. Section 2.6.3 explains how to apply CADMM instead of CBGA for a distributed version of GMS. Section 2.7 provides details of proofs of all theoretical statements.

2.2 Review of Consensus-Based Gradient Ascent (CBGA)

The setting of CBGA [7] assumes a connected network, with K nodes and M edges. It also assumes a convex set of matrices $S \subseteq \mathbb{R}^{D\times D}$ and convex functions $F_1, \ldots, F_K$ on S,

associated with the K nodes. The goal is to minimize $\sum_{k=1}^{K} F_k$ over S, where each node k has only access to F_k and may communicate to its neighbors. The consensus-based formulation of this problem uses local neighborhoods as follows. For $1 \leq k \leq K$, let $\mathcal{N}_k$ denote the set of all nodes connected (by an edge) to the node k. The desired problem, $\min_{\boldsymbol{Q} \in S} \sum_{k=1}^{K} F_k(\boldsymbol{Q})$, can be computed locally as follows:

$$\min_{\boldsymbol{Q}_1, \ldots \boldsymbol{Q}_K \in S} \sum_{k=1}^{K} F_k(\boldsymbol{Q}_k), \text{where } \boldsymbol{Q}_k = \boldsymbol{Q}_q, \forall 1 \leq k \leq K, q \in \mathcal{N}_k, q < k. \tag{2.1}$$

The constraints in the right side of (2.1) are called *consensus constraints*. The consensus constraints have the following formulation by a matrix equation. For $1 \leq m \leq M$, let e_m denote the edge indexed by m. We write $e_m = \{k, q\}$ whenever e_m connects the nodes indexed by k and q. For $1 \leq k \leq K$ and $1 \leq m \leq M$, $\boldsymbol{C}_{mk}$ is the following $D \times D$ matrix

$$\boldsymbol{C}_{mk} = c_{mk} \boldsymbol{I}, \text{ where } c_{mk} = \begin{cases} 1, & \text{if } e_m = \{k, q\} \text{ and } k < q; \\ -1, & \text{if } e_m = \{k, q\} \text{ and } q < k; \\ 0, & \text{otherwise.} \end{cases} \tag{2.2}$$

Let $\boldsymbol{C}$ denote the $DM \times DK$ block matrix with blocks $\{\boldsymbol{C}_{mk}\}_{m=1,k=1}^{M,K}$ and let $\bar{\boldsymbol{Q}} = [\boldsymbol{Q}_1^T, \ldots, \boldsymbol{Q}_K^T]^T$, then the consensus constraints can be formulated as $\boldsymbol{C}\bar{\boldsymbol{Q}} = \boldsymbol{0}$.

The minimization problem of (2.1) is inseparable and thus hard to compute in a distributed setting. That is, one cannot find the exact solution by just computing and adding results from each node. Instead, one needs to invoke the dual problem, which we describe next. The Lagrangian for problem (2.1) is

$$L(\boldsymbol{\Lambda}, \bar{\boldsymbol{Q}}) = \sum_{k=1}^{K} F_k(\boldsymbol{Q}_k) + \operatorname{tr}(\boldsymbol{\Lambda}^T \boldsymbol{C} \bar{\boldsymbol{Q}}),$$

where $\boldsymbol{\Lambda} = [\boldsymbol{\Lambda}_1^T, \ldots, \boldsymbol{\Lambda}_M^T]^T \in \mathbb{R}^{MD \times D}$, and the dual function is

$$d(\boldsymbol{\Lambda}) = \min_{\bar{\boldsymbol{Q}} \in S^K} L(\boldsymbol{\Lambda}, \bar{\boldsymbol{Q}}). \tag{2.3}$$

Finally, the dual problem of (2.1) is

$$\hat{\mathbf{\Lambda}} = \arg\max_{\mathbf{\Lambda}\in\mathbb{R}^{MD\times D}} d(\mathbf{\Lambda}). \tag{2.4}$$

Recall that strong duality means that the minimizer of (2.3) with $\hat{\mathbf{\Lambda}}$ found by the dual problem (2.4) coincides with the minimizer of (2.1). In order to solve (2.3), the CBGA procedure uses the following separability of the dual function: $d(\mathbf{\Lambda}) = \sum_{k=1}^{K} d_k(\mathbf{\Lambda})$, where

$$d_k(\mathbf{\Lambda}) = \min_{\boldsymbol{Q}_k\in S}(F_k(\boldsymbol{Q}_k) + \mathrm{tr}(\mathbf{\Lambda}_m^T \boldsymbol{A}_k)), \tag{2.5}$$

$$\boldsymbol{A}_k = \sum_{m\in\mathcal{E}_k} c_{mk}\mathbf{\Lambda}_m^T, \tag{2.6}$$

$\{c_{mk}\}_{m=1,k=1}^{M\ K}$ are defined in (2.2) and $\mathcal{E}_k$ denotes the set of all edges that contain the node k. Such separation gives rise to a distributed solution of (2.3). In order to solve (2.4), the CBGA procedure applies subgradient descent over $\mathbf{\Lambda}$. According to [27], one possible subgradient is $\boldsymbol{C}\bar{\boldsymbol{Q}}(\mathbf{\Lambda})$, where $\bar{\boldsymbol{Q}}(\mathbf{\Lambda})$ is the solution of (2.3) for the given $\mathbf{\Lambda}$. Moreover, if $d(\mathbf{\Lambda})$ is differentiable, then $\boldsymbol{C}\bar{\boldsymbol{Q}}(\mathbf{\Lambda})$ is the gradient. Therefore, the CBGA algorithm simultaneously solves problems (2.3) and (2.4). It starts with an initial guess of $\mathbf{\Lambda}$, then solves the separable problem of (2.3), next uses it for subgradient descent update of (2.4), which results in a new value of $\mathbf{\Lambda}$, and iterates the two main steps until convergence. The CBGA procedure converges if the following conditions are satisfied (see [27]): 1. the set H is convex and the functions F_k are convex; 2. strong duality holds for (2.1); 3. the subgradients of $d(\mathbf{\Lambda})$ are uniformly bounded for all values of $\mathbf{\Lambda}$. We emphasize that this procedure assumes a solution of the separable problem in (2.3) and without such a solution it is inapplicable.

2.3 Distributed GMS

We review the GMS problem in §2.3.1, propose a distributed solution in §2.3.2, establish convergence guarantees in §2.3.3 and discuss the time complexity and possible reduction of the communication cost in §2.3.4.

2.3.1 Review of GMS

In order to motivate the GMS algorithm for RSR, we first review the following convex formulation of PCA for full-rank data due to [24]. Assume that $\mathcal{X} = \{\boldsymbol{x}_i\}_{i=1}^N$ is a dataset of N points in $\mathbb{R}^D$, centered at $\mathbf{0}$ and recall that the PCA d-subspace is the d-dimensional linear subspace minimizing the sum of squared residuals. If the dataset $\mathcal{X}$ is full rank, then according to Theorem 10 of [24] the PCA d-subspace is spanned by the bottom d eigenvectors of the following matrix $\hat{\boldsymbol{Q}}$ (or equivalently, the top d eigenvectors of $-\hat{\boldsymbol{Q}}$):

$$\hat{\boldsymbol{Q}} = \arg\min_{\boldsymbol{Q}\in\mathbb{H}} \sum_{\boldsymbol{x}\in\mathcal{X}} \|\boldsymbol{Q}\boldsymbol{x}\|^2, \text{ where } \mathbb{H} = \{\boldsymbol{Q} \in \mathcal{S}^D, \operatorname{tr}(\boldsymbol{Q}) = 1\}. \tag{2.7}$$

Here and throughout the chapter $\mathcal{S}^D$ denotes the set of D-dimensional symmetric matrices, $\mathcal{S}_+^D$ denotes the set of D-dimensional positive semi-definite matrices and $\mathcal{S}_{++}^D$ denotes the set of D-dimensional positive definite matrices.

The GMS procedure modifies (2.7) by replacing the squared deviations $\|\boldsymbol{Q}\boldsymbol{x}\|^2$ in (2.7) with the more robust unsquared deviations $\|\boldsymbol{Q}\boldsymbol{x}\|$, while smoothing the resulted objective function around $\mathbf{0}$ with a parameter $\delta > 0$. The convex minimization problem of GMS [24] for the dataset $\mathcal{X} = \{\boldsymbol{x}_i\}_{i=1}^N \subset \mathbb{R}^D$ and the regularization parameter δ is

$$\tilde{\boldsymbol{Q}} = \arg\min_{\boldsymbol{Q}\in\mathbb{H}} F^\delta(\boldsymbol{Q}), \tag{2.8}$$

where $\mathbb{H}$ is defined in (2.7) and

$$F^\delta(\boldsymbol{Q}) = \sum_{\boldsymbol{x}\in\mathcal{X}, \|\boldsymbol{Q}\boldsymbol{x}\|\geq\delta} \|\boldsymbol{Q}\boldsymbol{x}\| + \sum_{\boldsymbol{x}\in\mathcal{X}, \|\boldsymbol{Q}\boldsymbol{x}\|<\delta} \left(\frac{\|\boldsymbol{Q}\boldsymbol{x}\|^2}{2\delta} + \frac{\delta}{2} \right). \tag{2.9}$$

Given a target dimension $1 \leq d \leq D-1$, the output of GMS is a d-dimensional subspace spanned by the bottom d eigenvectors of $\tilde{\boldsymbol{Q}}$ (or the top ones of $-\tilde{\boldsymbol{Q}}$).

Clearly, the objective function in (2.7) is strictly convex for full-rank data. The objective function in (2.9) is strictly convex under the following stronger condition, which is referred to as the two-subspaces criterion [24]:

Definition 1. *A dataset $\mathcal{Y}$ satisfies the two-subspaces criterion if*

$$(\mathcal{Y} \cap \boldsymbol{L}_1) \cup (\mathcal{Y} \cap \boldsymbol{L}_2) \neq \mathcal{Y} \text{ for all } D-1 \text{ dimensional subspaces } \boldsymbol{L}_1, \boldsymbol{L}_2 \in \mathbb{R}^D. \quad (2.10)$$

When this criterion is satisfied, the unique minimizer of (2.8) can be computed by a very simple IRLS procedure (see Algorithm 2 in [24]). If the dataset is not centered, one may appropriately center it at each iteration of the IRLS procedure. Alternatively and more commonly, one may initially center the original data by the geometric median.

Zhang and Lerman [24] discuss the conditions under which GMS recovers the underlying subspace and show that they hold with high probability under a certain probabilistic model describing inliers and outliers (see §1.3 and §2 of [24]). These conditions can be non-technically described as follows. First, the inliers need to spread throughout the whole underlying subspace, that is, they cannot concentrate on a lower dimensional subspace of the underlying subspace. Second, the outliers need to spread throughout the complement of the underlying subspace within the ambient space. Third, the magnitude of outliers needs to be restricted and they may not concentrate around lines. Zhang and Lerman [24] propose some ways of preprocessing the data to avoid some restrictions imposed by these conditions (see §5.2 of [24]).

The GMS solution to (2.9) can be interpreted as a robust inverse covariance estimator. Indeed, the solution to the least-squares problem (2.7) is a scaled version of the inverse sample covariance (see Theorem 10 of [24]). The IRLS procedure, which aims to solve (2.9), scales the cross products of the sample covariance at each iteration in a way which may avoid the effect of outliers, and then inverts the resulting matrix or a regularized version of it.

2.3.2 Consensus-Based Subgradient Algorithm for Distributed GMS

We assume a dataset $\mathcal{X}$ with $\{\mathcal{X}_k\}_{k=1}^K$ distributed at K nodes. We further assume that for $1 \leq k \leq K$, $\mathcal{X}_k$ satisfies the two-subspaces criterion (see Definition (1)), so they are full rank. For general $\mathcal{X}_1, \ldots, \mathcal{X}_K$ which may not satisfy this criterion, we suggest reducing their dimensions (see e.g., the discussion in §2.6.1) so that they are full-rank. In typical cases of noisy inliers concentrated around a subspace, the preprocessed $\mathcal{X}_1, \ldots, \mathcal{X}_K$ with full rank will also satisfy the two-subspaces criterion.

We follow §2.2 and solve the minimization problem for the dual function of GMS in each node, while communicating these solutions via CBGA. Following (2.5), (2.8) and (2.9), we need to solve at each node the following optimization problem:

$$d_k(\boldsymbol{\Lambda}) = \min_{\boldsymbol{Q}\in\mathbb{H}} G_k^\delta(\boldsymbol{Q}) \text{ for } G_k^\delta(\boldsymbol{Q}) = F_k^\delta(\boldsymbol{Q}) + \operatorname{tr}(\boldsymbol{Q}\boldsymbol{A}_k), \tag{2.11}$$

where

$$F_k^\delta(\boldsymbol{Q}) = \sum_{\boldsymbol{x}\in\mathcal{X}_k, \|\boldsymbol{Q}\boldsymbol{x}\|\geq\delta} \|\boldsymbol{Q}\boldsymbol{x}\| + \sum_{\boldsymbol{x}\in\mathcal{X}_k, \|\boldsymbol{Q}\boldsymbol{x}\|<\delta} \left(\frac{\|\boldsymbol{Q}\boldsymbol{x}\|^2}{2\delta} + \frac{\delta}{2}\right).$$

To find the minimizer of (2.11) sufficiently fast, we introduce an iterative algorithm similar to Algorithm 2 of [24] and guarantee its r-linear convergence. Let $\boldsymbol{Q}_k^0 = \boldsymbol{I}/D$ (or arbitrarily fix $\boldsymbol{Q}_k^0 \in \mathcal{S}_{++}^D \cap \mathbb{H}$) and for iteration $1 \leq t \leq T$, let $\boldsymbol{Q}_k^t$ be the solution of the following Lyapunov equation in $\boldsymbol{Q}$, where $c_k \in \mathbb{R}$ is chosen such that $\operatorname{tr}(\boldsymbol{Q}_k^t) = 1$:

$$\boldsymbol{Q}\left(\sum_{\boldsymbol{x}\in\mathcal{X}_k} \frac{\boldsymbol{x}\boldsymbol{x}^T}{2\max(\|\boldsymbol{Q}_k^{t-1}\boldsymbol{x}\|,\delta)}\right) + \left(\sum_{\boldsymbol{x}\in\mathcal{X}_k} \frac{\boldsymbol{x}\boldsymbol{x}^T}{2\max(\|\boldsymbol{Q}_k^{t-1}\boldsymbol{x}\|,\delta)}\right)\boldsymbol{Q} = c_k\boldsymbol{I} - \boldsymbol{A}_k. \tag{2.12}$$

The following lemma establishes the existence and uniqueness of $c_k \in \mathbb{R}$ and $\boldsymbol{Q}_k^t \in \mathcal{S}_{++}^D \cap \mathbb{H}$, which satisfy (2.12). It is proved in §2.7.5.

Lemma 1. *Let $\mathcal{X} = \{\boldsymbol{x}_i\}_{i=1}^N$ be a full rank dataset in $\mathbb{R}^D$, $\boldsymbol{Q} \in \mathcal{S}_{++}^D \cap \mathbb{H}$ and $\boldsymbol{A} \in \mathcal{S}^D$ with $\operatorname{tr}(\boldsymbol{A}) = 0$ and*

$$\|\boldsymbol{A}\|_2 \leq 1 \Big/ \operatorname{tr}\left(\left(\sum_{\boldsymbol{x}\in\mathcal{X}} \frac{\boldsymbol{x}\boldsymbol{x}^T}{2\max(\|\boldsymbol{x}\|,\delta)}\right)^{-1}\right). \tag{2.13}$$

There exists a unique $c' \in \mathbb{R}$ such that the following equation with $c = c'$

$$\boldsymbol{P}\left(\sum_{\boldsymbol{x}\in\mathcal{X}} \frac{\boldsymbol{x}\boldsymbol{x}^T}{2\max(\|\boldsymbol{Q}\boldsymbol{x}\|,\delta)}\right) + \left(\sum_{\boldsymbol{x}\in\mathcal{X}} \frac{\boldsymbol{x}\boldsymbol{x}^T}{2\max(\|\boldsymbol{Q}\boldsymbol{x}\|,\delta)}\right)\boldsymbol{P} + \boldsymbol{A} = c\boldsymbol{I} \tag{2.14}$$

has a unique solution $\boldsymbol{P} \in \mathcal{S}_{++}^D \cap \mathbb{H}$.

If $\boldsymbol{Q}_*$ *is the solution of* (2.14) *with* $c = 0$ *and* $\boldsymbol{A} = \boldsymbol{A}_k$*, then*

$$c' = -2(\mathrm{tr}(\boldsymbol{Q}_*) - 1) \Big/ \mathrm{tr}\left(\left(\sum_{\boldsymbol{x}\in\mathcal{X}} \frac{\boldsymbol{x}\boldsymbol{x}^T}{2\max(\|\boldsymbol{Q}_*\boldsymbol{x}\|, \delta)}\right)^{-1}\right). \tag{2.15}$$

Algorithm 1 summarizes the above procedure of solving (2.11). In §2.3.3 we establish the r-linear convergence of $\{\boldsymbol{Q}_k^t\}_{t\in\mathbb{N}}$ to the minimizer of (2.11).

Given this solution of the local problem, the iterative CBGA algorithm for GMS is straightforward. As explained in §2.2, at each iteration $s \geq 1$ and edge $e_m = \{k, q\}$, indexed by $1 \leq m \leq M$, the CBGA algorithm needs to update the corresponding $\boldsymbol{\Lambda}_m^s$ by the following gradient descent procedure

$$\boldsymbol{\Lambda}_m^s = \boldsymbol{\Lambda}_m^{s-1} + \mu \cdot (c_{mk}\boldsymbol{Q}_k^{s-1} - c_{mk}\boldsymbol{Q}_q^{s-1}). \tag{2.16}$$

Note that the update of $\boldsymbol{\Lambda}_m^s$ in (2.16) uses $\boldsymbol{\Lambda}_m^{s-1}$ and the local solutions $\{\boldsymbol{Q}_k^{s-1}\}_{k=1}^K$ of the previous iteration $s-1$. The idea is to use $\boldsymbol{\Lambda}_m^s$ in solving the local problems. However, these problems only require the matrices $\boldsymbol{A}_k^s = \sum_{m\in\mathcal{E}_k} c_{mk}(\boldsymbol{\Lambda}_m^s)^T$ for $k = 1, \ldots, K$. The combination of (2.16), the latter expression for $\boldsymbol{A}_k^s$ (see also (2.6)), the fact that $c_{mk}^2 = 1$ whenever the mth edge is incident to the kth vertex and appropriate replacement of the set of edges $\mathcal{E}_k$ with the set of vertices $\mathcal{N}_k$ results in the following update formula

$$\boldsymbol{A}_k^s = \boldsymbol{A}_k^{s-1} + \rho \sum_{q\in\mathcal{N}_k} \left(\boldsymbol{Q}_k^{s-1} - \boldsymbol{Q}_q^{s-1}\right). \tag{2.17}$$

The CBGA procedure for GMS thus iteratively updates the matrices $\{\boldsymbol{A}_k^s\}_{k=1}^K$, by using the solutions of the local problems according to (2.17), and solves the local problems by using the matrices $\{\boldsymbol{A}_k^s\}_{k=1}^K$. This simple procedure, which we refer to as CBGA-GMS is summarized in Algorithm 2. In §2.7.6 we discuss how a sufficiently small step-size in Algorithm 2 ensures that the above condition (2.13), which is necessary for solving the local problems, is satisfied at each node for all iterations of Algorithm 1. We also explain in §2.7.6 why the required upper bound in (2.13) can be relaxed in practice and based on this observation we suggest a practical choice for the step-size in (2.36).

Algorithm 1 Algorithm for computing the minimizer of (2.11)

Input: $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\} \subseteq \mathbb{R}^D$: data, $\boldsymbol{A}_k \in \mathcal{S}^D$ with $\mathrm{tr}(\boldsymbol{A}_k) = 0$, T_{GMS}: stopping iteration number, δ : regularization parameter (default: 10^{-10})
Set: $\boldsymbol{Q}_k^0 = \boldsymbol{I}/D$ and $t = 0$
while $t \leq T_{GMS}$ or $G_k^\delta(\boldsymbol{Q}_k^{t+1}) > G_k^\delta(\boldsymbol{Q}_k^t)$ **do**

- Let $\boldsymbol{Q}_*$ be the solution of (2.14) with $\boldsymbol{Q} = \boldsymbol{Q}_k^t$, $c = 0$ and $\boldsymbol{A} = \boldsymbol{A}_k$
- Compute c' according to (2.15)
- Let $\hat{\boldsymbol{Q}}_k^{t+1}$ be the solution of (2.14) with $\boldsymbol{Q} = \boldsymbol{Q}_k^t$, $c = c'$ and $\boldsymbol{A} = \boldsymbol{A}_k$
- $t := t + 1$

end while
return $\hat{Q}_k := \boldsymbol{Q}_k^t$

Algorithm 2 Consensus-Based Subgradient Algorithm for GMS (CBGA-GMS)

Input: Network with K nodes and M edges, $\mathcal{X}_1, \ldots, \mathcal{X}_K$: datasets in the K nodes, T_{CBGA}, T_{GMS}: stopping iteration numbers, δ: regularization parameter (default: 10^{-10}) and μ: sufficiently small constant step-size
Set: For all $1 \leq m \leq M$, $\boldsymbol{\Lambda}_m^0 = \boldsymbol{0}$ and for all $1 \leq k \leq K$, $\boldsymbol{A}_k^0 = \boldsymbol{0}$ and $\boldsymbol{Q}_k^0$ is the solution of Algorithm 1 with input $\mathcal{X}_k$, $\boldsymbol{A}_k^0$, T_{GMS} and δ
for $s = 1 : T_{CBGA}$ **do**
for $k = 1 : K$ **do**

- Transmit $\boldsymbol{Q}_k^{s-1}$ to $\mathcal{N}_k$
- Compute $\boldsymbol{A}_k^s$ according to (2.17)
- $\boldsymbol{Q}_k^s$ is the output of Algorithm 1 with input $\mathcal{X}_k, \boldsymbol{A}_k^s, T_{GMS}$ and δ

end for
end for
return $L_k :=$ the span of the bottom d eigenvectors of $\boldsymbol{Q}_k^{T_{CBGA}}, 1 \leq k \leq K$

2.3.3 Properties of CBGA-GMS

We establish r-linear convergence of Algorithm 1 and briefly discuss the mere convergence of Algorithm 2 and its recovery guarantees. For completeness, we include the definition of r-linear convergence.

Definition 2. *A sequence $\{x_k\}_{k=1}^{\infty} \subset \mathbb{R}$ r-linearly converges to x if there exists a sequence $\{v_k\}_{k=1}^{\infty} \subset \mathbb{R}$, such that $|x_k - x| < v_k$ for all k and there exists $q \in (0,1)$ such that $v_{k+1} \leq q v_k$ for all k sufficiently large.*

The following theorem guarantees that $\{\boldsymbol{Q}_k^t\}_{t\in\mathbb{N}}$ of Algorithm 1 r-linearly converges to the unique minimizer of (2.11). This theorem is later proved in §2.8.

Theorem 1. *Assume $\mathcal{X}_k = \{\boldsymbol{x}_i\}_{i=1}^{N_k} \subset \mathbb{R}^D$ satisfies the two-subspaces criterion, $\boldsymbol{A}_k \in \mathcal{S}^D$ satisfies* (2.13) *and* $\operatorname{tr}(\boldsymbol{A}_k) = 0$. *If $\{\boldsymbol{Q}_k^t\}_{t\in\mathbb{N}}$ is obtained by Algorithm 1 at node k with $T_{GMS} = \infty$, then it r-linearly converges to the unique minimizer of* (2.11).

Note that CBGA-GMS is a gradient descent method. Indeed, Theorem 2 of [24] implies the strict convexity of F^δ. This and Theorems 26.1 and 26.3 of [28] imply the differentiability of its dual function $d(\boldsymbol{\Lambda}) = \sum_{k=1}^{K} d_k(\boldsymbol{\Lambda})$, where $d_k(\boldsymbol{\Lambda})$ is defined in (2.11).

The conditions for convergence of CBGA discussed in §2.2 are satisfied for CBGA-GMS. Indeed, the first condition is straightforward, since G_k^δ and $\mathbb{H}$ are convex. The strong duality of the problem is shown by easily verifying Slater's condition (see §5.2.3 of [29]). Finally, the gradient of $d(\boldsymbol{\Lambda})$ is $C\bar{\boldsymbol{Q}}$ and its norm is bounded by $K||C||$. Indeed, for each $1 \leq k \leq K$, the kth block of $\bar{\boldsymbol{Q}}$, $\boldsymbol{Q}_k$, is in $\mathcal{S}_{++}^D$ with $\operatorname{tr}(\boldsymbol{Q}_k) = 1$ and thus $||C\bar{\boldsymbol{Q}}|| \leq K||C||$.

Since the convex optimization problem for the total data of CBGA-GMS is the same as the convex optimization problem for regular GMS [24], the exact and near recovery theory of CBGA-GMS follow from [24].

2.3.4 Time Complexity

Algorithm 1 solves (2.12) twice. The computation of the coefficient of (2.12),

$$\sum_{i=1}^{N_k} \boldsymbol{x}_i \boldsymbol{x}_i^T / (2 \max(\|\hat{\boldsymbol{Q}}_k^{s-1} \boldsymbol{x}_i\|, \delta))$$

, requires $O\left(N_k \times D^2\right)$ operations. Solving (2.12) requires $O\left(D^3\right)$ operations (see [30]). Since $N_k \geq D$, the total complexity for each iteration of algorithm 1 at node k is $O\left(N_k \times D^2\right)$. Denoting $N_{\max} = \max_{1 \leq k \leq K} N_k$, we conclude that the complexities of Algorithms 1 and 2 are $O\left(T_{GMS} \times N_{\max} \times D^2\right)$ and $O\left(T_{CBGA} \times T_{GMS} \times N_{\max} \times D^2\right)$ respectively.

Algorithm 2 transfers $D \times D$ matrices between nodes in each iteration, which might not be cost efficient. In order to reduce the communication cost we suggest transferring only the top d eigenvectors of those matrices. Once a node receives the top d eigenvectors, it reconstructs the $D \times D$ matrix $\boldsymbol{U}^T\boldsymbol{U}/\operatorname{tr}(\boldsymbol{U}^T\boldsymbol{U})$, where $\boldsymbol{U} \in \mathbb{R}^{d \times D}$ contains the orthogonal top d eigenvectors as rows. We cannot guarantee the convergence of this modified procedure, but it seems to work well in practice.

2.4 Distributed Reaper and Distributed FMS

We present distributed versions of two other RSR algorithms: Reaper [25] and FMS [26]. These algorithms are reviewed in §2.4.1 and their straightforward distributed implementations are explained in §2.4.2.

2.4.1 Review of the Reaper and FMS Algorithms

Assume a dataset $\mathcal{X} \subset \mathbb{R}^D$, a target dimension $d \in \{1, 2, \ldots D-1\}$ and a regularization parameter $\delta > 0$.

The Reaper algorithm [25] solves the following convex optimization problem[1]:

$$\min_{\boldsymbol{P} \in \mathcal{S}_+^D,\ \operatorname{tr}(\boldsymbol{P})=d} \sum_{\substack{\boldsymbol{x} \in \mathcal{X} \\ \|\boldsymbol{x}-\boldsymbol{P}\boldsymbol{x}\| \geq \delta}} \|\boldsymbol{x} - \boldsymbol{P}\boldsymbol{x}\| + \sum_{\substack{\boldsymbol{x} \in \mathcal{X} \\ \|\boldsymbol{x}-\boldsymbol{P}\boldsymbol{x}\| < \delta}} \left(\frac{\|\boldsymbol{x} - \boldsymbol{P}\boldsymbol{x}\|^2}{2\delta} + \frac{\delta}{2} \right). \tag{2.18}$$

It uses an IRLS framework for minimizing (2.18). The robust d-subspace is spanned by the top d eigenvectors of this solution. A generic condition for subspace recovery by Reaper with an error bound is established in [25].[2] It requires similar restrictions as those described in the first and third non-technical conditions for GMS in §2.3.1.

[1] The formulation in [25] adds the additional optimization constraint $\boldsymbol{I} - \boldsymbol{P} \in \mathcal{S}_+^D$, but as is obvious from the proof of Lemma 14 in [24], it is not needed and thus omitted from (2.18)

[2] For simplicity, the analysis in [25] is restricted to the case where $\delta = 0$.

Note that plugging $\boldsymbol{Q} = \boldsymbol{I} - \boldsymbol{P}$ into (2.9) results in an objective function similar to (2.18). The main difference is that (2.18) further assumes that $\boldsymbol{P} \in \mathcal{S}_+^D$.

The FMS algorithm [26] tries to directly solve a regularized least unsquared deviations variant of PCA. Recall that the PCA subspace minimizes the least-squares function $\sum_{\boldsymbol{x}\in\mathcal{X}} \text{dist}^2(\boldsymbol{x}, L)$, where $\text{dist}(\boldsymbol{x}, L) = \min_{y\in L} \|\boldsymbol{x}-\boldsymbol{y}\|_2$, over the Grassmannian $G(D, d)$, which is the set of d-dimensional linear subspaces in $\mathbb{R}^D$. The least unsquared deviations cost function is $\sum_{\boldsymbol{x}\in\mathcal{X}} \text{dist}(x, L)$, where $L \in G(D, d)$. FMS aims to minimize the following smooth version of this function with the regularization parameter $\delta > 0$:

$$\min_{L\in G(d,D)} \sum_{\boldsymbol{x}\in\mathcal{X},\text{dist}(\boldsymbol{x},L)\geq\delta} \text{dist}(\boldsymbol{x}, L) + \sum_{\boldsymbol{x}\in\mathcal{X},\text{dist}(\boldsymbol{x},L)<\delta} \left(\frac{\text{dist}^2(\boldsymbol{x}, L)}{2\delta} + \frac{\delta}{2}\right). \tag{2.19}$$

This minimization is hard to solve in general (it was proved to be NP hard when $\delta = 0$ [31]). FMS is a straightforward IRLS heuristic for solving (2.19). At each iteration it scales the original data points by the square root of their distance to the subspace of the previous iteration and then computes the current subspace by applying PCA to the scaled data. Recovery and r-linear convergence of FMS were established only for data generated from very particular probabilistic models [26] . However, in practice FMS seems to obtain competitive accuracy and speed for many datasets.

We note that the target function in (2.19) is similar to that in (2.9), where $\text{dist}(x, L)$ replaces $\|\boldsymbol{Q}\boldsymbol{x}\|$. In fact, both GMS and Reaper are convex relaxations of the minimization in (2.19), where Reaper is the tightest possible one [25].

2.4.2 Distributed Implementations for Reaper and FMS

We assume a dataset $\mathcal{X}$ with $\{\mathcal{X}_k\}_{k=1}^K$ distributed at K nodes so that $\mathcal{X}_k$ has full rank for $1 \leq k \leq K$. If the data is not full rank, it is preprocessed according to the discussion in §2.6.1.

Distributed Reaper requires distributedly solving (2.18). This can be done by applying distributed full PCA at each IRLS iteration of Algorithm 4.1 of [25]. More precisely, this procedure first initializes the IRLS weights by $\beta_{\boldsymbol{x}}^0 = 1$ for all data points $\boldsymbol{x} \in \mathcal{X}$. Then, at each iteration $s \geq 1$ it applies distributed full PCA of the weighted dataset $\{\sqrt{\beta_{\boldsymbol{x}}^{s-1}}\boldsymbol{x}\}_{\boldsymbol{x}\in\mathcal{X}}$ to obtain $\boldsymbol{P}_k^s$ at each processor with index k. Then, it updates

the weights by $\beta_{\boldsymbol{x}} \leftarrow 1/\max(\delta, \|\boldsymbol{x} - \boldsymbol{P}_k^s \boldsymbol{x}\|)$, for all $\boldsymbol{x} \in \mathcal{X}$. This procedure is iterated until convergence and the local subspace is obtained by the top d eigenvectors of $\boldsymbol{P}_k^{s'}$, where s' corresponds to the final iteration.

The distributed FMS is obtained by distributed PCA at each iteration of FMS. Note that FMS uses randomized SVD to find only the top d principal components. For central processing and $D \gg d$, we recommend applying a distributed randomized SVD algorithm [32]. For an ad hoc network, we are not aware of effective implementation of a distributed algorithm that can find only the top d principal components.

2.5 Numerical Experiments

This section tests the distributed algorithms proposed in this chapter using both synthetic and real data. It is organized as follows: §2.5.1 describes the synthetic data model, §2.5.2 contains experiments on data generated from this model and §2.5.7 contains experiments on real datasets.

Throughout this section, Algorithm 1 uses $T_{GMS} = 30$ and Algorithm 2 uses $T_{CBGA} = 250$ and μ as in (2.36) or in a specified range of values. In all RSR algorithms the regularization parameter is $\delta = 10^{-10}$. CBGA-PCA of §2.6.1 is used as "distributed PCA" and is also implemented in the iterative schemes of distributed Reaper and FMS. All codes necessary to duplicate these results are available in `https://github.com/vahanhuroyan/Distributed-RSR`.

2.5.1 Synthetic Data Model for Distributed RSR

In §2.5.2 we use the following synthetic model to generate distributed RSR data. It depends on the following parameters: K, N^0, N^1, D, d and σ. We create a connected graph with K nodes as explained below, and we randomly fix $L \in G(D, d)$. For each node we sample N^1/K inliers from the d-dimensional Multivariate Normal distribution $N(\boldsymbol{0}, \boldsymbol{P}_L)$, where $\boldsymbol{P}_L$ denotes the orthoprojector onto L, with additive Gaussian noise $N(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$, where $0 \leq \sigma < 1$. Furthermore, for each node we sample N^0/K outliers from the uniform distribution on $[0, 1]^D$. Note that the outliers are asymmetric. Unless otherwise specified (see §2.5.3), the graph is obtained by arbitrarily generating a spanning tree with K nodes and then randomly and independently connecting 2 nodes with

probability 1/2. It is demonstrated for $K = 8$ in Fig. 2.1c.

2.5.2 Demonstration on Synthetic Data

We study the effect of the network topology and the step-size on the convergence rate of CBGA-GMS in §2.5.3 and §2.5.4 respectively. In §2.5.5 we compare the accuracy of a CADMM version of GMS with CBGA-GMS. In §2.5.6 we compare our proposed distributed RSR algorithms. In each experiment 50 random samples are generated according to the model of §2.5.1. The recovery error of the tested algorithm is averaged over the random 50 samples. For Figs. 2.2a-2.2c we further average the recovery error over the K processors to demonstrate the average rate of convergence. We remark that in all experiments, the data is full rank at each processor, so there was no need to initially apply dimension reduction.

2.5.3 The Influence of the Network Topology on Convergence

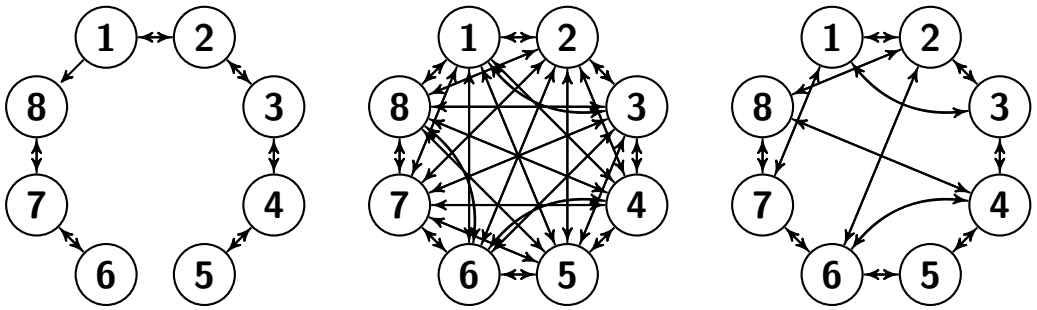


Figure 2.1: Three types of connected networks with 8 nodes. Fig. 2.1a: sparsely connected network; Fig. 2.1b: fully connected network; and Fig. 2.1c: randomly connected network.

To check the effect of the network topology on the convergence rate we use three different networks, whose graphs are shown in Fig. 2.1. The graph in Fig. 2.1a is sparse, the graph in Fig. 2.1b is fully connected and the graph in Fig. 2.1c is generated according to the recipe described in §2.5.1. We generate data according to the model of §2.5.1, where $K = 10$, $N^1 = 200$, $N^0 = 2000$, $D = 50$, $d = 3$, $\sigma = 0.1$ and $\mu = 100$. The average recovery error as a function of the number of iterations for the 3 different networks is

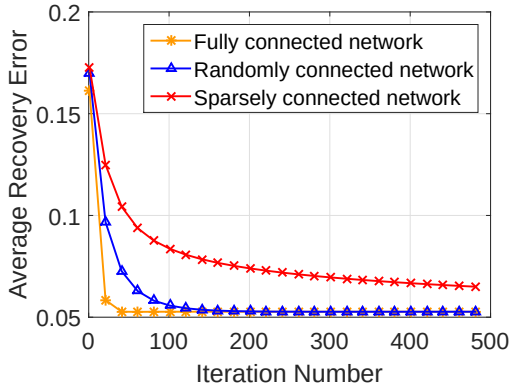
shown in Fig. 2.2a. The fully connected network has the fastest convergence and as the network gets sparser, the convergence rate decreases.

2.5.4 The Influence of the Step-size on the Convergence Rate

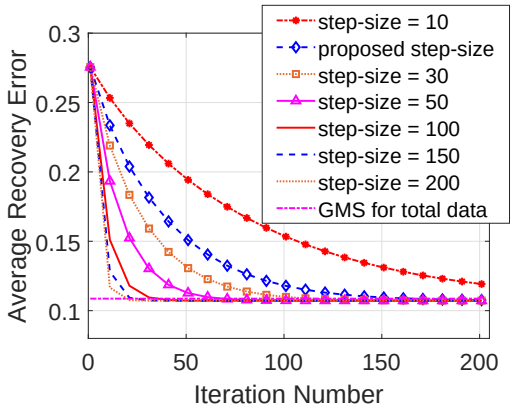
We generate data according to the model of §2.5.1, where $K = 10$, $N^1 = 200$, $N^0 = 2000$, $D = 50$, $d = 3$, and $\sigma = 0.1$. Fig. 2.2b shows the average recovery error for CBGA-GMS as a function of the number of iterations for 7 different step-sizes: 10, 30, 50, 100, 150, 200 and the one proposed in (2.36), whose value here is 22.5. The average error of GMS for the total data is included as a baseline. These results imply that the convergence rate increases with the step-size. However, additional experiments, not reported in here, indicate that for a very large step-size the algorithm does not converge. We also note that for large step-sizes, the increase of the step-size does not significantly change the convergence rate, for example, for step-sizes 150 and 200 we see almost the same result, while the difference between convergence results is obvious for smaller step-sizes.

2.5.5 Comparing CBGA-GMS with CADMM-GMS

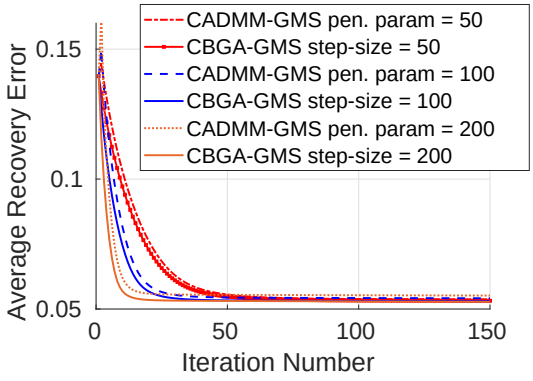
A CADMM scheme for GMS, which directly follows [12], is described in §2.6.3. It is referred to as CADMM-GMS. Both CBGA-GMS and CADMM-GMS are somewhat parallel and it follows from (2.17) and (2.24) that their corresponding parameters μ and ρ play similar roles. We compare them using data generated from the model described in §2.5.1, where $K = 5$, $D = 50$, $d = 3$, $\sigma = 0.05$, $N^0 = 5000$ and $N^1 = 500$. We tested the following same values of ρ and μ: 50, 100 and 200. We remark that the μ proposed in (2.36) obtained the value 51.1. Since both algorithms performed similarly when using this value and 50, we did not report the performance with this value. Fig. 2.2c shows the recovery errors vs. the number of iterations for both algorithms with these step-sizes. We note that both algorithms converge with very similar speed, where CBGA-GMS converges slightly faster. For the smaller values of μ and ρ (100 and 200) the algorithms achieve the same recovery error. However, for the larger value of the parameter (300), the recovery error of CADMM-GMS is slightly higher than the recovery error of CBGA-GMS.



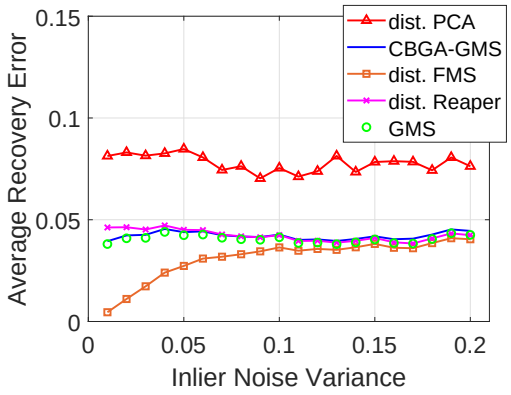
(a) Influence of the network topology on the convergence rate of CBGA-GMS



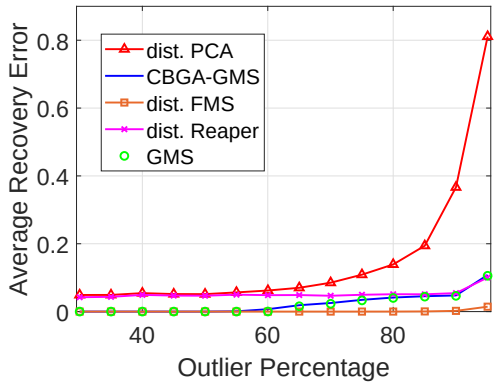
(b) Influence of different step-sizes on the convergence rate of CBGA-GMS



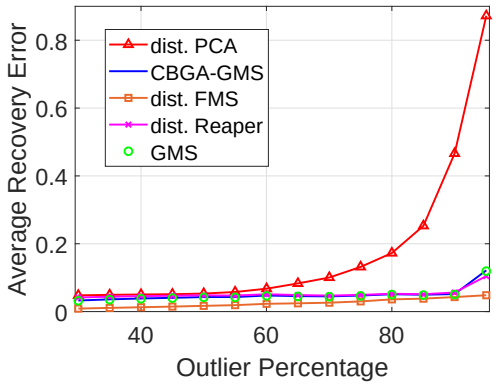
(c) CBGA-GMS vs CADMM-GMS



(d) Influence of inlier noise variance on distributed PCA, Reaper, GMS and FMS



(e) Influence of outlier percentage on distributed PCA, Reaper, GMS, FMS; $\sigma = 0$



(f) Influence of outlier percentage on distributed PCA, Reaper, GMS, FMS; $\sigma = 0.05$

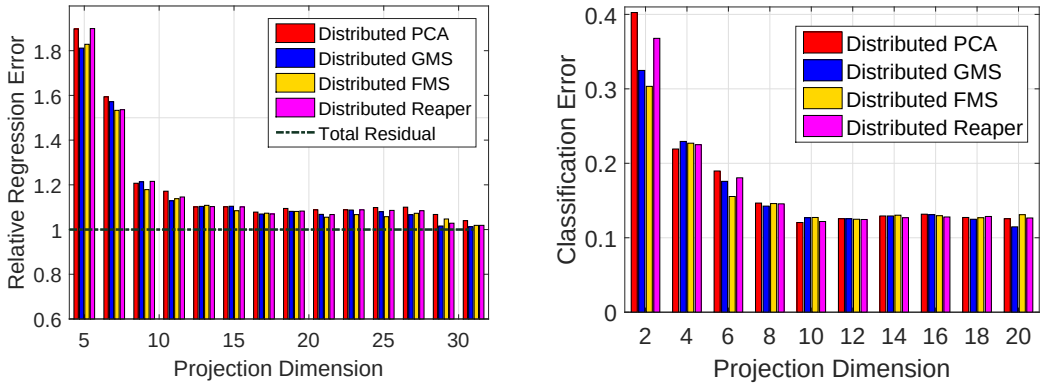
Figure 2.2: Demonstration of properties of the distributed algorithms on synthetic data.

2.5.6 Comparison of the Proposed Algorithms

We compare GMS, the 3 proposed distributed RSR algorithms and distributed PCA in different settings and report the results in Figs. 2.2d-2.2f. Fig. 2.2d demonstrates how the inlier noise variance σ affects the convergence of the four methods. The data for this figure was created according to the model described in §2.5.1, where $D = 50$, $d = 3$, $K = 5$, $N^0 = 3000$, $N^1 = 1000$ and σ varies between 0 and 0.2 with increments of 0.01. In this figure, for all tested values of σ, CBGA-PCA performs the worst and distributed FMS performs the best, where CBGA-GMS and distributed Reaper are somewhat comparable.

Figs. 2.2e and 2.2f demonstrate the influence of the outlier percentage on the average recovery error of the four distributed methods and GMS (for the total data) with and without inlier noise. We generate data according to the model of §2.5.1, where $D = 50$, $d = 3$, $K = 10$, $\sigma = 0$ for Fig. 2.2e, $\sigma = 0.5$ for Fig. 2.2f, $N^0 = 5000$ and N^1 is chosen such that the outlier percentage in the total data varies between 30% to 95% with increments of 5%. For both cases ($\sigma = 0$ and $\sigma = 0.05$) and for all percentages of outliers, the recovery error for distributed FMS is the smallest one and that of CBGA-PCA is the largest one. Figs. 2.2e and 2.2f also demonstrate that when the data is corrupted with outliers (percentage of outliers higher than 65%), the distributed RSR algorithms perform significantly better than distributed PCA. For the case of $\sigma = 0$, distributed FMS and CBGA-GMS succeed with exact recovery up to 90% and 55% of outliers respectively, whereas distributed Reaper could not exactly recover the subspace in the tested range.

In Figs. 2.2d, 2.2e and 2.2f, the recovery errors obtained by CBGA-GMS and GMS are comparable. We remark that the distributed implementations of PCA, Reaper and FMS also obtain similar recovery errors as the non-distributed ones in all of these experiments. However, since these figures are already dense, we do not report the results of the latter non-distributed implementations.



(a) CTslices (b) Human Activity Recognition (HAR)

Figure 2.3: Demonstration of the proposed distributed algorithms on two real datasets: CTslices and HAR.

2.5.7 Real Data Experiments

Distributed RSR algorithms can be used as a preprocessing step for clustering, classification and regression. We apply our proposed distributed algorithms as a preprocessing step for two different tasks: linear regression, where we use the CTslices dataset ($N = 53{,}500, D = 386$) [33], and classification (multiclass SVM), where we use the Human Activity Recognition (HAR) dataset ($N = 10{,}299, D = 561$) [33, 34]. For both datasets we apply initial centering by the geometric median and to ensure full-rank data in all processors we reduce dimension to $D = 150$ by distributed exact PCA (see §2.6.1). We remark that higher values of reduced dimensions D were also possible. We report the results for one of the processors as they are the same for all of them.

For the CTslices data, the algorithms are trained on 50,000 data points and tested on 3500 data points. The training data is divided between 5 processors, each containing 10,000 data points. We apply CBGA-PCA, CBGA-GMS, distributed FMS and distributed Reaper to reduce the dimension of the dataset to lie between 5 and 30. We then apply linear least squares regression in the reduced dimension. Fig. 2.3a reports the relative regression error for the different projected dimensions. The relative regression error is the regression error for the data with the reduced dimension divided by the relative error for the data in 150 dimensions. We notice that for almost all dimensions, the relative errors of distributed FMS and GMS are lower than those of distributed

PCA, and the relative errors of distributed Reaper are either lower or comparable to those of distributed PCA.

For the HAR data, the algorithms are trained on 7352 data points and tested on 2947 data points. The training data is divided between 8 processors, each containing 919 data points. We apply CBGA-PCA, CBGA-GMS, distributed FMS and distributed Reaper to reduce the dimension of the dataset to lie between 2 and 20. We then apply classification in the reduced dimensions. Fig. 2.3b reports classification error for the different projected dimensions. It demonstrates that in dimension 2, the distributed RSR algorithms, in particular, distributed FMS and GMS, have a clear advantage over distributed PCA. In other dimensions, distributed RSR algorithms perform at least as good as distributed PCA.

We comment that for all real datasets, the results of the distributed algorithms are very similar to those of the non-distributed ones. Differences between all distributed and non-distributed implementations may exist when the initial dimension D is large and an initial dimension reduction by OSE is applied (see §2.6.1). An effect of OSE on the performance of PCA in a distributed setting is documented in [19].

2.6 Solutions of Related Problems

We first use the idea of CBGA-GMS to solve two simpler problems: distributed computation of the PCA subspace and distributed computation of the geometric median. We then describe a CADMM solution for distributed GMS.

2.6.1 Distributed PCA for Arbitrarily Distributed Network

Before describing the CBGA procedure for PCA, we remark that if the dimension D is not high, then the following simple procedure can be applied to solve the problem. One may propagate the local covariance matrices among the network and recover the exact covariance matrix at each processor and use it for PCA computation. We refer to it as exact distributed PCA. If the dimension D is high, then it can be reduced by an OSE procedure described below before applying the exact distributed PCA algorithm.

Our proposed CBGA-PCA algorithm is similar to [4, 21, 22], but uses instead the PCA formulation in (2.7). This formulation leads to a direct solution of the local

optimization problem. In order to apply (2.7), one needs to guarantee that for all $1 \leq k \leq K$, $\mathcal{X}_k$ has full rank. If $\mathcal{X}_k$ is rank-deficient, one can reduce its dimension. If the dimension D is high, one can sample an Oblivious Subspace Embedding (OSE) matrix $\boldsymbol{H}$ [35] and instead of $\mathcal{X}_k$ consider $\boldsymbol{H}\mathcal{X}_k$. One common OSE $\boldsymbol{H}$ has only one non-zero entry per row. By taking an appropriate number of rows for $\boldsymbol{H}$, one can assume that the projected data at each node has full rank. If the dimension D is not high, then the exact distributed PCA, or a faster approximate version of it, can be used to reduce the dimension.

Next, we clarify the application of CBGA to (2.7). In view of §2.2, it is sufficient to compute the dual function of (2.7) at each node, that is, compute for each $1 \leq k \leq K$:

$$d_k(\boldsymbol{\Lambda}) = \min_{\boldsymbol{Q} \in \mathbb{H}} \left(\sum_{\boldsymbol{x} \in \mathcal{X}_k} \|\boldsymbol{Q}\boldsymbol{x}\|^2 + \operatorname{tr}(\boldsymbol{A}_k \boldsymbol{Q}) \right), \text{ where } \boldsymbol{A}_k = \sum_{m \in \mathcal{E}_k} c_{mk} \boldsymbol{\Lambda}_m^T. \tag{2.20}$$

Appendix 2.7.1 guarantees the unique minimizer of (2.20) and explains how to find it.

Since the minimized function in (2.7) is strongly convex, it follows from [36] that its dual function $d(\boldsymbol{\Lambda}) = \sum_{k=1}^{K} d_k(\boldsymbol{\Lambda})$, where $d_k(\boldsymbol{\Lambda})$ is defined in (2.20), is Lipschitz smooth. This implies that the CBGA algorithm for PCA converges to the PCA solution for the total data with rate $O(1/t)$. The complexity of CBGA-PCA is $O(T_{CBGA} \times N_{\max} \times D^2)$ (see §2.7.4). This algorithm is not optimal in terms of complexity and communication. Indeed, the distributed exact PCA algorithm described above is simpler and achieves the exact PCA subspace. Nevertheless, we find this CBGA-PCA interesting for two reasons. First of all, it is similar to previous attempts [4, 21, 22] that did not clarify how to solve the local dual problem. Second of all, CBGA-PCA simply demonstrates the main idea of the more complicated CBGA-GMS procedure.

2.6.2 Distributed Geometric Median

The geometric median of a discrete dataset $\mathcal{X} \subset \mathbb{R}^D$ is defined as

$$\arg\min_{\boldsymbol{y} \in \mathbb{R}^D} \sum_{\boldsymbol{x} \in \mathcal{X}} \|\boldsymbol{x} - \boldsymbol{y}\|. \tag{2.21}$$

Weiszfeld's algorithm [37] is a common numerical approach to approximating (2.21) within a sufficiently small error. It applies an iteratively reweighted least squares (IRLS) procedure. However, if in one of the iterations, the estimate coincides with one of the data points, then Weiszfeld's algorithm fails to converge to the geometric median. To avoid this issue, we consider the following regularized version of (2.21):

$$\arg\min_{\boldsymbol{y}\in\mathbb{R}^D} \sum_{\boldsymbol{x}\in\mathcal{X},\|\boldsymbol{x}-\boldsymbol{y}\|\geq\delta} \|\boldsymbol{x}-\boldsymbol{y}\| + \sum_{\boldsymbol{x}\in\mathcal{X},\|\boldsymbol{x}-\boldsymbol{y}\|<\delta} \left(\frac{\|\boldsymbol{x}-\boldsymbol{y}\|^2}{2\delta}+\frac{\delta}{2}\right), \tag{2.22}$$

where $\delta>0$ is a small regularization parameter. We can solve (2.22) by the generalized Weiszfeld's algorithm [38, §4]. This algorithm runs as follows: it starts with an initial guess $\boldsymbol{y}_0\in\mathbb{R}^D$, and at iteration $s\geq 1$ it computes

$$\boldsymbol{y}_s = \sum_{\boldsymbol{x}\in\mathcal{X}} \frac{\boldsymbol{x}}{\max\left(\|\boldsymbol{x}-\boldsymbol{y}_{s-1}\|,\delta\right)} \Big/ \sum_{\boldsymbol{x}\in\mathcal{X}} \frac{1}{\max\left(\|\boldsymbol{x}-\boldsymbol{y}_{s-1}\|,\delta\right)}.$$

The sequence $\{\boldsymbol{y}_s\}_{s\in\mathbb{N}}$ r-linearly converges to the solution of (2.22) (see [38]).

We assume a dataset $\mathcal{X}$ with $\{\mathcal{X}_k\}_{k=1}^K$ distributed at K nodes, and distributedly compute the regularized geometric median of $\mathcal{X}$ by CBGA. In view of §2.2, it is enough to compute the dual function of (2.22) at each node, that is, compute for each $1\leq k\leq K$

$$d_k(\boldsymbol{\lambda}) = \min_{\boldsymbol{y}\in\mathbb{R}^D} \sum_{\substack{\boldsymbol{x}\in\mathcal{X}_k,\\ \|\boldsymbol{x}-\boldsymbol{y}\|\geq\delta}} \|\boldsymbol{x}-\boldsymbol{y}\| + \sum_{\substack{\boldsymbol{x}\in\mathcal{X}_k,\\ \|\boldsymbol{x}-\boldsymbol{y}\|<\delta}} \left(\frac{\|\boldsymbol{x}-\boldsymbol{y}\|^2}{2\delta}+\frac{\delta}{2}\right) + \sum_{m\in\mathcal{E}_k} c_{mk}\boldsymbol{\lambda}_m^T\boldsymbol{y}, \tag{2.23}$$

where $\boldsymbol{\lambda}=[\boldsymbol{\lambda}_1^T,\ldots,\boldsymbol{\lambda}_M^T]^T\in\mathbb{R}^{MD}$. We suggest solving (2.23) by IRLS as follows: start with an initial guess $\boldsymbol{y}_k^0\in\mathbb{R}^D$ and at iteration $s\geq 1$ compute

$$\boldsymbol{y}_k^s = \left(2\sum_{\boldsymbol{x}\in\mathcal{X}_k} \frac{\boldsymbol{x}}{\max\left(\|\boldsymbol{x}-\boldsymbol{y}_k^{s-1}\|,\delta\right)} - \sum_{m\in\mathcal{E}_k} c_{mk}\boldsymbol{\lambda}_m\right) \Big/ \left(2\sum_{\boldsymbol{x}\in\mathcal{X}_k} \frac{1}{\left(\|\boldsymbol{x}-\boldsymbol{y}_k^{s-1}\|,\delta\right)}\right).$$

The convergence of $\{\boldsymbol{y}_k^s\}_{s\in\mathbb{N}}$ follows from that of IRLS (see [38]) and CBGA (see §2.2).

2.6.3 CADMM Solution for the Distributed GMS Problem

We formulate in Algorithm 3 a CADMM solution of the distributed GMS problem by following the CADMM scheme of [12]. The solution of the local problem is discussed in §2.6.4.

Algorithm 3 CADMM implementation for distributed GMS (CADMM-GMS)

Input: Network with K nodes, $\mathcal{X}_1, \ldots, \mathcal{X}_K$: datasets in the K nodes, T_{CADMM}, T_{GMS}: stopping iteration numbers, δ: regularization parameter (default: 10^{-10}) and ρ: penalty parameter for CADMM

Set: For all $1 \leq k \leq K$, $\boldsymbol{Z}_k^0 = \mathbf{0}$ and $\boldsymbol{Q}_k^0$ is the solution of Algorithm 1 with input $\mathcal{X}_k$, $\boldsymbol{A}_k = \mathbf{0}$, T_{GMS} and δ

for $s = 1 : T_{CADMM}$ **do**

- For $1 \leq k \leq K$ update $\boldsymbol{Z}_k^s$ by

$$\boldsymbol{Z}_k^s = \boldsymbol{Z}_k^{s-1} + \rho \sum_{j \in \mathcal{N}_k} \left(\boldsymbol{Q}_k^{s-1} - \boldsymbol{Q}_j^{s-1} \right) \tag{2.24}$$

- For $1 \leq k \leq K$ apply the algorithm described in §2.6.4 to solve

$$\boldsymbol{Q}_k^s = \arg\min_{\boldsymbol{Q}_k \in \mathbb{H}} \tilde{G}_{ADMM}(\boldsymbol{Q}_k), \text{ where} \tag{2.25}$$

$$\tilde{G}_{ADMM}(\boldsymbol{Q}_k) = F_k(\boldsymbol{Q}_k) + \operatorname{tr}\left(\boldsymbol{Q}_k^T \boldsymbol{Z}_k^s\right) + \rho \sum_{j \in \mathcal{N}_k} \left\| \boldsymbol{Q}_k - \frac{\boldsymbol{Q}_k^{(s-1)} + \boldsymbol{Q}_j^{(s-1)}}{2} \right\|_2^2$$

end for

return $L_k :=$ the span of the bottom d eigenvectors of $\boldsymbol{Q}_k^{T_{CADMM}}, 1 \leq k \leq K$

2.6.4 Algorithm for computing the solution of (2.25)

We propose an iterative scheme for solving (2.25), which is almost identical to Algorithm 1, but at each iteration s instead of finding the trace one solution of (2.14), it

finds the trace one solution of the following Lyapunov equation in $\boldsymbol{P}$:

$$\boldsymbol{P}\left(\sum_{\boldsymbol{x}\in\mathcal{X}_k}\frac{\boldsymbol{x}\boldsymbol{x}^T}{2\max(\|\boldsymbol{Q}\boldsymbol{x}\|,\delta)}+\rho|\mathcal{N}_k|\boldsymbol{I}\right)+\left(\sum_{\boldsymbol{x}\in\mathcal{X}_k}\frac{\boldsymbol{x}\boldsymbol{x}^T}{2\max(\|\boldsymbol{Q}\boldsymbol{x}\|,\delta)}+\rho|\mathcal{N}_k|\boldsymbol{I}\right)\boldsymbol{P} + \boldsymbol{Z}_k^s-\rho\sum_{j\in\mathcal{N}_k}\left(\boldsymbol{Q}_k^{s-1}+\boldsymbol{Q}_j^{s-1}\right)=c\boldsymbol{I}. \quad (2.26)$$

Here, c is chosen so that $\mathrm{tr}(\boldsymbol{P})=1$ and its existence is guaranteed by Lemma 3. The convergence theory for this iterative algorithm is the same as the one developed for Algorithm 1.

2.7 Supplementary Details

2.7.1 On the Minimizer of (2.20)

We first state the main result of this section:

Lemma 2. *If $\mathcal{X}_k\subset\mathbb{R}^D$ is full rank and $\boldsymbol{A}_k\in\mathcal{S}^D$, then the minimizer of (2.20) is unique. Furthermore, there exists a unique $c'\in\mathbb{R}$ such that this minimizer is the unique solution of the following equation with $c=c'$*

$$\boldsymbol{Q}\left(\sum_{\boldsymbol{x}\in\mathcal{X}_k}\boldsymbol{x}\boldsymbol{x}^T\right)+\left(\sum_{\boldsymbol{x}\in\mathcal{X}_k}\boldsymbol{x}\boldsymbol{x}^T\right)\boldsymbol{Q}+\boldsymbol{A}_k=c\boldsymbol{I}. \quad (2.27)$$

Section 2.7.2 states and proves a lemma about the solution of the above Lyapunov equation and §2.7.3 then uses this latter lemma to conclude lemma 2. At last, §2.7.4 briefly discusses the computation of the minimizer of (2.20).

2.7.2 Preliminary lemma

We verify the following lemma.

Lemma 3. *If $c\in\mathbb{R}$, $\boldsymbol{X}\in\mathcal{S}_{++}^D$ and $\boldsymbol{A}\in\mathcal{S}^D$, then the following Lyapunov equation*

$$\boldsymbol{Q}\boldsymbol{X}+\boldsymbol{X}\boldsymbol{Q}+\boldsymbol{A}=c\boldsymbol{I} \quad (2.28)$$

has a unique solution in $\boldsymbol{Q} \in \mathcal{S}^D$. *Furthermore,* $\operatorname{tr}(\boldsymbol{Q})$ *is an increasing linear function of* c *with slope* $\operatorname{tr}(\boldsymbol{X}^{-1})/2$.

Proof. The existence and uniqueness of the solution of (2.28) is well-known [39, page 107]. We thus only need to show that $\operatorname{tr}(\boldsymbol{Q})$ is an increasing linear function of c. Assume that $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$ are the solutions of (2.28) corresponding to c_1 and c_2, that is,

$$\boldsymbol{Q}_1\boldsymbol{X} + \boldsymbol{X}\boldsymbol{Q}_1 + \boldsymbol{A} = c_1\boldsymbol{I} \text{ and } \boldsymbol{Q}_2\boldsymbol{X} + \boldsymbol{X}\boldsymbol{Q}_2 + \boldsymbol{A} = c_2\boldsymbol{I}. \tag{2.29}$$

Subtracting the two equations in (2.29), results in

$$(\boldsymbol{Q}_1 - \boldsymbol{Q}_2)\boldsymbol{X} + \boldsymbol{X}(\boldsymbol{Q}_1 - \boldsymbol{Q}_2) = (c_1 - c_2)\boldsymbol{I}, \tag{2.30}$$

whose unique solution is $(\boldsymbol{Q}_1 - \boldsymbol{Q}_2) = (c_1 - c_2)\boldsymbol{X}^{-1}/2$. By taking traces of both sides of the solution, we get that $(\operatorname{tr}(\boldsymbol{Q}_1) - \operatorname{tr}(\boldsymbol{Q}_2))/(c_1 - c_2) = \operatorname{tr}(\boldsymbol{X}^{-1})/2 > 0$. □

2.7.3 Proof of Lemma 2

Since $\mathcal{X}_k$ is full rank, $\sum_{\boldsymbol{x}\in\mathcal{X}_k} \boldsymbol{x}\boldsymbol{x}^T \in \mathcal{S}^D_{++}$. Hence the minimized function in (2.20) is strongly convex and its minimizer is unique.

We note that (2.27) is a Lyapunov equation in $\boldsymbol{Q}$. Lemma 3 implies that there is a unique value c' for which the unique solution of (2.27) has trace 1. We denote this solution by $\boldsymbol{Q}'$. Next, we show that $\boldsymbol{Q}'$ is the minimizer of (2.20). The following two facts: $\sum_{\boldsymbol{x}\in\mathcal{X}_k} \|\boldsymbol{Q}\boldsymbol{x}_k\|^2 + \operatorname{tr}(\boldsymbol{A}_k\boldsymbol{Q}) = \sum_{\boldsymbol{x}\in\mathcal{X}_k} \operatorname{tr}(\boldsymbol{Q}\boldsymbol{x}_k\boldsymbol{x}_k^T\boldsymbol{Q}) + \operatorname{tr}(\boldsymbol{A}_k\boldsymbol{Q})$ for $\boldsymbol{Q} \in \mathbb{H}$ and $\operatorname{tr}(\boldsymbol{Q}) = 1$ for $\boldsymbol{Q} \in \mathbb{H}$, imply the same minimizer for (2.20) and

$$\min_{\boldsymbol{Q}\in\mathbb{H}} l(\boldsymbol{Q}), \text{ where } l(\boldsymbol{Q}) = \sum_{\boldsymbol{x}\in\mathcal{X}_k} \operatorname{tr}(\boldsymbol{Q}\boldsymbol{x}_k\boldsymbol{x}_k^T\boldsymbol{Q}) + \operatorname{tr}(\boldsymbol{A}_k\boldsymbol{Q}) - c' \operatorname{tr}(\boldsymbol{Q}). \tag{2.31}$$

Since $l(\boldsymbol{Q})$ is convex on $\mathbb{H}$, we conclude that $\boldsymbol{Q}'$ minimizes (2.31) by showing that the derivative of $l(\boldsymbol{Q})$ at $\boldsymbol{Q}'$, when restricted to $\mathbb{H}$, is $\boldsymbol{0}$:

$$\frac{d}{d\boldsymbol{Q}} l(\boldsymbol{Q})\Big|_{\boldsymbol{Q}=\boldsymbol{Q}'} = \boldsymbol{Q}'\left(\sum_{\boldsymbol{x}\in\mathcal{X}_k} \boldsymbol{x}\boldsymbol{x}^T\right) + \left(\sum_{\boldsymbol{x}\in\mathcal{X}_k} \boldsymbol{x}\boldsymbol{x}^T\right)\boldsymbol{Q}' + \boldsymbol{A}_k - c'\boldsymbol{I} = \boldsymbol{0}. \qquad \square$$

2.7.4 Computing the Minimizer of (2.20)

In view of Lemma 3 we compute c' and the corresponding solution of (2.29) as follows. We solve (2.27) with $c = 0$ to obtain $\boldsymbol{Q}_* \in \mathcal{S}^D$. We then use $\operatorname{tr}(\boldsymbol{Q}_*)$ and the slope $\operatorname{tr}(\boldsymbol{X}^{-1})/2$, where $\boldsymbol{X} = \sum_{\boldsymbol{x}\in\mathcal{X}_k} \boldsymbol{x}\boldsymbol{x}^T$, to find c'. Therefore, computing this minimizer requires computing $\boldsymbol{X}$, which costs $O(N_{\max} \times D^2)$, and solving two Lyapunov equations, which costs $O(D^3)$ (see [30]).

2.7.5 Proof of lemma 1

Let $\boldsymbol{X} = \sum_{i=1}^N \boldsymbol{x}_i\boldsymbol{x}_i^T / (2\max(\|\boldsymbol{Q}\boldsymbol{x}_i\|, \delta))$ and note that $\boldsymbol{X} \in \mathcal{S}_{++}^D$. This observation and Lemma 3 imply that there is a unique value $c \in \mathbb{R}$ for which (2.14) has a unique solution in $\mathbb{H}$. We will show that $c > \lambda_1(\boldsymbol{A})$, equivalently $\boldsymbol{A} - c\boldsymbol{I} \preceq \boldsymbol{0}$, and thus in view of [39, page 107], this solution is in $\mathcal{S}_{++}^D$.

To get this estimate, we rewrite (2.14) as $\boldsymbol{P} + \boldsymbol{X}\boldsymbol{P}\boldsymbol{X}^{-1} + \boldsymbol{A}\boldsymbol{X}^{-1} = c\boldsymbol{X}^{-1}$. Applying trace to both sides and using the following facts: $\operatorname{tr}(\boldsymbol{P}) = 1$, $\operatorname{tr}(\boldsymbol{X}\boldsymbol{P}\boldsymbol{X}^{-1}) = \operatorname{tr}(\boldsymbol{X}^{-1}\boldsymbol{X}\boldsymbol{P}) = 1$ and $\operatorname{tr}(\boldsymbol{A}\boldsymbol{X}^{-1}) \geq \lambda_D(\boldsymbol{A}) \operatorname{tr}(\boldsymbol{X}^{-1})$ yields $c \geq 2/\operatorname{tr}(\boldsymbol{X}^{-1}) + \lambda_D(\boldsymbol{A})$.

Let $\boldsymbol{X}_* = \sum_{i=1}^N \boldsymbol{x}_i\boldsymbol{x}_i^T / (2\max(\|\boldsymbol{x}_i\|, \delta))$. Since $\boldsymbol{Q} \in \mathcal{S}_+^D \cap \mathbb{H}$ and $\max(\|\boldsymbol{Q}\boldsymbol{x}_i\|, \delta) \leq \max(\|\boldsymbol{x}_i\|, \delta)$ for $1 \leq i \leq N$, $\boldsymbol{X} - \boldsymbol{X}_* \in \mathcal{S}_+^D$, which implies that $\boldsymbol{X}_*^{-1} - \boldsymbol{X}^{-1} \in \mathcal{S}_+^D$. Combining the last result with (2.13), $\|\boldsymbol{A}\|_2 > \lambda_1(-A)$ and the estimate of c we obtain that $c \geq 2/\operatorname{tr}(\boldsymbol{X}_*^{-1}) + \lambda_D(\boldsymbol{A}) \geq 2\|\boldsymbol{A}\|_2 - \lambda_1(-\boldsymbol{A}) \geq \lambda_1(\boldsymbol{A})$.

The last statement of the lemma is a direct application of Lemma 3. □

2.7.6 On the Choice of the Step-Size

In view of lemma 1, we require that condition (2.13) holds at each iteration of Algorithm 2 and each node k. The following lemma shows that a choice of a sufficiently small step-size μ guarantees this requirement. After verifying this lemma, we discuss weaker restrictions on the step-size as well as a weaker practical version of condition (2.13).

Lemma 4. *If $\{\mathcal{X}_k\}_{k=1}^K \subset \mathbb{R}^D$ are datasets distributed at K nodes, $n \in \mathbb{N}$ and*

$$\mu \leq \frac{1}{n \cdot \max\limits_{1\leq k\leq K} |\mathcal{E}_k| \cdot \operatorname{tr}\left(\left(\sum\limits_{x\in\mathcal{X}_k} \frac{\boldsymbol{x}\boldsymbol{x}^T}{\max(\|\boldsymbol{x}\|,\delta)}\right)^{-1}\right)}, \tag{2.32}$$

then at each iteration $s \leq n$ of Algorithm 2 and node k, $\boldsymbol{A}_k^s$ satisfies condition (2.13).

Proof. We estimate the LHS of (2.13) at iteration s as follows:

$$\|\boldsymbol{A}_k^s\|_2 = \left\|\sum_{m\in\mathcal{E}_k} c_{mk}\boldsymbol{\Lambda}_m^s\right\|_2 \leq \sum_{m\in\mathcal{E}_k} \|\boldsymbol{\Lambda}_m^s\|_2. \tag{2.33}$$

In order to evaluate $\|\boldsymbol{\Lambda}_m^s\|_2$ for $1 \leq m \leq M$, we apply (2.16) and basic inequalities:

$$\begin{aligned}\|\boldsymbol{\Lambda}_m^s\|_2 = \|\boldsymbol{\Lambda}_m^{s-1} + \mu(c_{mk}\boldsymbol{Q}_k^s - c_{mk}\boldsymbol{Q}_q^s)\|_2 \leq \|\boldsymbol{\Lambda}_m^{s-1}\|_2 + \mu\|\boldsymbol{Q}_k^s - \boldsymbol{Q}_q^s\|_2 \leq \\ \|\boldsymbol{\Lambda}_m^{s-1}\|_2 + \mu\max\{\|\boldsymbol{Q}_k^s\|_2, \|\boldsymbol{Q}_q^s\|_2\} \leq \|\boldsymbol{\Lambda}_m^{s-1}\|_2 + \mu \leq \cdots \leq s\mu \leq n\mu.\end{aligned} \tag{2.34}$$

Combining (2.32), (2.33) and (2.34) results in

$$\|\boldsymbol{A}_k^s\|_2 \leq \sum_{m\in\mathcal{E}_k} \|\boldsymbol{\Lambda}_m^s\|_2 \leq |\mathcal{E}_k| n\mu \leq 1 \Big/ \operatorname{tr}\left(\left(\sum_{\boldsymbol{x}\in\mathcal{X}} \frac{\boldsymbol{x}\boldsymbol{x}^T}{2\max(\|\boldsymbol{x}\|,\delta)}\right)^{-1}\right). \tag{2.35}$$

□

In practice one may apply several iterations with the same fixed step-size and gradually reduce it until it satisfies the estimate above. Nevertheless, this estimate represents a worse-case scenario and typically we expect an improved one. Indeed, first note that condition (2.13) represents a worse-case scenario. In the proof of lemma 1 we used the worst-case estimate $\|\boldsymbol{Q}\| \leq 1$. However, typically $\|\boldsymbol{Q}\| \sim 1/D$. This will introduce a multiplicative factor D for the RHS of (2.13) and thus of (2.32). Second, in (2.34) we used the estimate $\|\boldsymbol{Q}_k^s - \boldsymbol{Q}_q^s\| \leq \max\{\|\boldsymbol{Q}_k^s\|_2, \|\boldsymbol{Q}_q^s\|_2\} \leq 1$. However, typically for $\boldsymbol{Q}_k^s, \boldsymbol{Q}_q^s \in \mathbb{H} \cap \mathcal{S}_{++}^D$, $\max\{\|\boldsymbol{Q}_k^s\|_2, \|\boldsymbol{Q}_q^s\|_2\} \sim 1/D$. This observation introduces another multiplicative factor D for the RHS of (2.32). These two observations suggest, in

practice, the following choice of a step-size:

$$\mu = \frac{D^2}{n \cdot \max\limits_{1\le k\le K} |\mathcal{E}_k| \cdot \operatorname{tr}\left(\left(\sum\limits_{x\in\mathcal{X}_k} \frac{\boldsymbol{x}\boldsymbol{x}^T}{\max(\|\boldsymbol{x}\|,\delta)}\right)^{-1}\right)}. \tag{2.36}$$

Third of all, we note that for sufficiently small step-sizes the gradient descent gets closer to the solution, that is, $\|\boldsymbol{Q}_k^s - \boldsymbol{Q}_q^s\|_2 \to 0$, for $1 \le k, q \le K$. However, we used $1/D$ as an upper bound for $\|\boldsymbol{Q}_k^s - \boldsymbol{Q}_q^s\|_2$. At last, we comment that while the above analysis aims to guarantee that at each iteration the solution is in $\mathbb{H} \cap \mathcal{S}_{++}^D$ (since (2.13) guarantees this), in practice it is not a main concern for small step-sizes and large number of iterations. Indeed, the solution of (2.4) coincides with the solution of GMS for the total data, which is in $\mathbb{H} \cap \mathcal{S}_{++}^D$. Thus, by choosing the step-size small enough we will always converge to the solution.

2.8 Proof of Theorem 1

We establish an auxiliary lemma in §2.8.1 and conclude Theorem 1 in §2.8.2 by following ideas of [24, 38] and using this lemma.

2.8.1 Preliminary Proposition

We first apply Lemma 3 to define the mapping $T_{\boldsymbol{A}}(\boldsymbol{Q})$ and then establish the continuity of $T_{\boldsymbol{A}}(\boldsymbol{Q})$ in $\mathcal{S}_{++}^D$.

Definition 3 (The mapping $T_{\boldsymbol{A}}(\boldsymbol{Q})$)**.** *If $\{\boldsymbol{x}_i\}_{i=1}^N \subset \mathbb{R}^D$, $\delta > 0$, $\boldsymbol{Q} \in \mathcal{S}_+^D \cap \mathbb{H}$ and $\boldsymbol{A} \in \mathcal{S}^D$ with $\operatorname{tr}(\boldsymbol{A}) = 0$, then $T_{\boldsymbol{A}}(\boldsymbol{Q})$ is the solution of the following equation in $\boldsymbol{P}$*

$$\boldsymbol{P}\left(\sum_{i=1}^N \frac{\boldsymbol{x}_i\boldsymbol{x}_i^T}{\max(\|\boldsymbol{Q}\boldsymbol{x}_i\|,\delta)}\right) + \left(\sum_{i=1}^N \frac{\boldsymbol{x}_i\boldsymbol{x}_i^T}{\max(\|\boldsymbol{Q}\boldsymbol{x}_i\|,\delta)}\right)\boldsymbol{P} + \boldsymbol{A} = c\boldsymbol{I}, \tag{2.37}$$

where $c = c(\boldsymbol{Q}) \in \mathbb{R}$ is uniquely chosen so that the solution has trace 1.

Lemma 5. *Assume a sequence $\{\boldsymbol{Q}^t\}_{t\in\mathbb{N}} \subset \mathcal{S}_{++}^D \cap \mathbb{H}$, $\boldsymbol{A} \in \mathcal{S}^D$ with $\operatorname{tr}(\boldsymbol{A}) = 0$, $\{\boldsymbol{x}_i\}_{i=1}^N \subset \mathbb{R}^D$ and $\delta > 0$. If $\boldsymbol{Q}^t \to \hat{\boldsymbol{Q}}$, then $T_{\boldsymbol{A}}(\boldsymbol{Q}^t) \to T_{\boldsymbol{A}}(\hat{\boldsymbol{Q}})$.*

Proof. For $t \in \mathbb{N}$, let $\boldsymbol{P}^t = T_{\boldsymbol{A}}(\boldsymbol{Q}^t)$ be the trace one solution of (2.37) with $\boldsymbol{Q} = \boldsymbol{Q}^t$ and $c = c^t$. Let $\hat{\boldsymbol{P}} = T_{\boldsymbol{A}}(\hat{\boldsymbol{Q}})$ be the trace one solution of (2.37) with $\boldsymbol{Q} = \hat{\boldsymbol{Q}}$ and $c = \hat{c}$. We need to prove that $\boldsymbol{P}^t \to \hat{\boldsymbol{P}}$ as $t \to \infty$. We write (2.37) with $\boldsymbol{P}^t, \boldsymbol{Q}^t$ and c^t as

$$\boldsymbol{P}^t \left(\sum_{i=1}^{N} \frac{\boldsymbol{x}_i \boldsymbol{x}_i^T}{\max(\|\boldsymbol{Q}^t \boldsymbol{x}_i\|, \delta)} \right) + \left(\sum_{i=1}^{N} \frac{\boldsymbol{x}_i \boldsymbol{x}_i^T}{\max(\|\boldsymbol{Q}^t \boldsymbol{x}_i\|, \delta)} \right) \boldsymbol{P}^t + \boldsymbol{A} = c^t \boldsymbol{I}. \tag{2.38}$$

Note that $\boldsymbol{R}^t := \sum_{i=1}^{N} \boldsymbol{x}_i \boldsymbol{x}_i^T / \max(\|\boldsymbol{Q}^t \boldsymbol{x}_i\|, \delta) \to \hat{\boldsymbol{R}} := \sum_{i=1}^{N} \boldsymbol{x}_i \boldsymbol{x}_i^T / \max(\|\boldsymbol{Q} \boldsymbol{x}_i\|, \delta)$ as $t \to \infty$. Also observe that for $\boldsymbol{Q} = \hat{\boldsymbol{Q}}$, $c = \hat{c}$ and $T_{\boldsymbol{A}}(\boldsymbol{Q}^t) = \boldsymbol{P}^t$, (2.37) has the form

$$\boldsymbol{P}^t \boldsymbol{R}^t + \boldsymbol{R}^t \boldsymbol{P}^t + \boldsymbol{A} = c^t \boldsymbol{I}. \tag{2.39}$$

By subtracting $c^t \boldsymbol{I}$ from both sides of (2.39) and rewriting $c^t \boldsymbol{I} = c^t \boldsymbol{R}^{t^{-1}} \boldsymbol{R}^t/2 + \boldsymbol{R}^t c^t \boldsymbol{R}^{t^{-1}}/2$, (2.39) becomes $(\boldsymbol{P}^t - c^t \boldsymbol{R}^{t^{-1}}/2)\boldsymbol{R}^t + \boldsymbol{R}^t(\boldsymbol{P}^t - c^t \boldsymbol{R}^{t^{-1}}/2) + \boldsymbol{A} = \boldsymbol{0}$. Similarly, $(\hat{\boldsymbol{P}} - \hat{c}\hat{\boldsymbol{R}}^{-1}/2)\hat{\boldsymbol{R}} + \hat{\boldsymbol{R}}(\hat{\boldsymbol{P}} - \hat{c}\hat{\boldsymbol{R}}^{-1}/2) + \boldsymbol{A} = \boldsymbol{0}$. Since $\boldsymbol{A}$ is fixed and $\boldsymbol{R}^t \to \hat{\boldsymbol{R}}$ as $t \to \infty$, it follows from the last two expressions that

$$\boldsymbol{P}^t - c^t \boldsymbol{R}^{t^{-1}}/2 \to \hat{\boldsymbol{P}} - \hat{c}\hat{\boldsymbol{R}}^{-1}/2 \text{ as } t \to \infty. \tag{2.40}$$

By taking the trace of both sides of (2.40) and using the facts that $\mathrm{tr}(\boldsymbol{P}^t) = \mathrm{tr}(\hat{\boldsymbol{P}}) = 1$ and $\hat{\boldsymbol{R}}^t \to \boldsymbol{R}$ as $t \to \infty$, we get that $c^t \to \hat{c}$ and consequently $\boldsymbol{P}^t \to \hat{\boldsymbol{P}}$ as $t \to \infty$. □

2.8.2 Conclusion of Theorem 1

We divide the proof of Theorem 1 into the following steps suggested in [24].

Step 1: **The majorizing function H and its minimizer.** Let H_k^δ denote the following function

$$H_k^\delta(\boldsymbol{Q}, \boldsymbol{Q}^*) = \sum_{\boldsymbol{x} \in \mathcal{X}_k} \left(\frac{\|\boldsymbol{Q}\boldsymbol{x}\|^2}{2\max(\|\boldsymbol{Q}^*\boldsymbol{x}\|, \delta)} + \frac{\max(\|\boldsymbol{Q}^*\boldsymbol{x}\|, \delta)}{2} \right) + \mathrm{tr}(\boldsymbol{Q}\boldsymbol{A}_k). \tag{2.41}$$

We show next that H_k^δ majorizes G_k^δ, that is,

$$H_k^\delta(\boldsymbol{Q}, \boldsymbol{Q}) = G_k^\delta(\boldsymbol{Q}) \text{ and } G_k^\delta(\boldsymbol{Q}) \leq H_k^\delta(\boldsymbol{Q}, \boldsymbol{Q}^*). \tag{2.42}$$

The above equality is immediate. To prove the above inequality we define

$$G_k^\delta(\boldsymbol{x},\boldsymbol{Q}) = \begin{cases} \|\boldsymbol{Q}\boldsymbol{x}\|, & \text{if } \|\boldsymbol{Q}\boldsymbol{x}\| \geq \delta; \\ \frac{\|\boldsymbol{Q}\boldsymbol{x}\|^2}{2\delta} + \frac{\delta}{2}, & \text{if } \|\boldsymbol{Q}\boldsymbol{x}\| < \delta, \end{cases}$$

$$H_k^\delta(\boldsymbol{x},\boldsymbol{Q},\boldsymbol{Q}^*) = \frac{\|\boldsymbol{Q}\boldsymbol{x}\|^2}{2\max(\|\boldsymbol{Q}^*\boldsymbol{x}\|,\delta)} + \frac{\max(\|\boldsymbol{Q}^*\boldsymbol{x}\|,\delta)}{2}.$$

We show that $G_k^\delta(\boldsymbol{x},\boldsymbol{Q}) \leq H_k^\delta(\boldsymbol{x},\boldsymbol{Q},\boldsymbol{Q}^*)$ by considering four complementing cases:

Case 1: $\|\boldsymbol{Q}\boldsymbol{x}\| \geq \delta$ and $\|\boldsymbol{Q}^*\boldsymbol{x}\| \geq \delta$. In this case

$$G_k^\delta(\boldsymbol{x},\boldsymbol{Q}) = \|\boldsymbol{Q}\boldsymbol{x}\| = \frac{\|\boldsymbol{Q}\boldsymbol{x}\|\|\boldsymbol{Q}^*\boldsymbol{x}\|}{\|\boldsymbol{Q}^*\boldsymbol{x}\|} \leq \frac{\|\boldsymbol{Q}\boldsymbol{x}\|^2 + \|\boldsymbol{Q}^*\boldsymbol{x}\|^2}{2\|\boldsymbol{Q}^*\boldsymbol{x}\|} = H_k^\delta(\boldsymbol{x},\boldsymbol{Q},\boldsymbol{Q}^*).$$

Case 2: $\|\boldsymbol{Q}\boldsymbol{x}\| \geq \delta$ and $\|\boldsymbol{Q}^*\boldsymbol{x}\| < \delta$. We conclude the desired property as follows $0 \leq (\|\boldsymbol{Q}\boldsymbol{x}\| - \delta)^2 = \|\boldsymbol{Q}\boldsymbol{x}\|^2 - 2\|\boldsymbol{Q}\boldsymbol{x}\|\delta + \delta^2 = \delta\left(H_k^\delta(x,\boldsymbol{Q},\boldsymbol{Q}^*) - G_k^\delta(\boldsymbol{x},\boldsymbol{Q})\right)$.

Case 3: $\|\boldsymbol{Q}\boldsymbol{x}\| < \delta$ and $\|\boldsymbol{Q}^*\boldsymbol{x}\| \geq \delta$. In this case

$$G_k^\delta(\boldsymbol{x},\boldsymbol{Q}) - H_k^\delta(\boldsymbol{x},\boldsymbol{Q},\boldsymbol{Q}^*) = \frac{1}{2}\left(\frac{\|\boldsymbol{Q}\boldsymbol{x}\|^2}{\delta} + \delta - \frac{\|\boldsymbol{Q}\boldsymbol{x}\|^2}{\|\boldsymbol{Q}^*x\|} - \|\boldsymbol{Q}^*x\|\right) = \frac{\|\boldsymbol{Q}^*\boldsymbol{x}\| - \delta}{2}\left(\frac{\|\boldsymbol{Q}\boldsymbol{x}\|^2}{\delta\|\boldsymbol{Q}^*\boldsymbol{x}\|} - 1\right) \leq 0.$$

Case 4: $\|\boldsymbol{Q}\boldsymbol{x}\| < \delta$ and $\|\boldsymbol{Q}^*\boldsymbol{x}\| < \delta$. Then $G_k^\delta(\boldsymbol{x},\boldsymbol{Q}) = H_k^\delta(\boldsymbol{x},\boldsymbol{Q},\boldsymbol{Q}^*)$.

We thus conclude (2.42) as follows

$$G_k^\delta(\boldsymbol{Q}) = \sum_{\boldsymbol{x}\in\mathcal{X}_k} G_k^\delta(\boldsymbol{x},\boldsymbol{Q}) + \operatorname{tr}(\boldsymbol{Q}\boldsymbol{A}_k) \leq H_k^\delta(\boldsymbol{x},\boldsymbol{Q},\boldsymbol{Q}^*) + \operatorname{tr}(\boldsymbol{Q}\boldsymbol{A}_k) = H_k^\delta(\boldsymbol{Q},\boldsymbol{Q}^*). \quad (2.43)$$

Next, we claim that the minimizer of $H_k^\delta(\boldsymbol{Q},\boldsymbol{Q}_k^t)$ over all $\boldsymbol{Q} \in \mathbb{H}$ is $\boldsymbol{Q}_k^{t+1}$. First we note that since the data satisfies the two-subspaces criterion and since $\operatorname{tr}(\boldsymbol{A}_k\boldsymbol{Q}_k)$ is a linear function, then according to Theorem 2 of [24], $H_k^\delta(\boldsymbol{Q},\boldsymbol{Q}_k^t)$ is strictly convex over $\boldsymbol{Q} \in \mathbb{H}$. We further note that for $\boldsymbol{Q} \in \mathbb{H}$, $H_k^\delta(\boldsymbol{Q},\boldsymbol{Q}^*) = \tilde{H}_k^\delta(\boldsymbol{Q},\boldsymbol{Q}^*)$, where

$$\tilde{H}_k^\delta(\boldsymbol{Q},\boldsymbol{Q}^*) = \sum_{\boldsymbol{x}\in\mathcal{X}_k}\left(\frac{\operatorname{tr}(\boldsymbol{Q}\boldsymbol{x}\boldsymbol{x}^T\boldsymbol{Q})}{2\max(\|\boldsymbol{Q}^*\boldsymbol{x}\|,\delta)} + \frac{\max(\|\boldsymbol{Q}^*\boldsymbol{x}\|,\delta)}{2}\right) + \operatorname{tr}(\boldsymbol{Q}\boldsymbol{A}_k). \quad (2.44)$$

Therefore, the minimizers over $\mathbb{H}$ of $H_k^\delta(\boldsymbol{Q}, \boldsymbol{Q}_k^t)$ and $\tilde{H}_k^\delta(\boldsymbol{Q}, \boldsymbol{Q}_k^t) - c_k \operatorname{tr}(\boldsymbol{Q})$ are the same. We compute the derivative of the latter term w.r.t. $\boldsymbol{Q}$ as follows:

$$\frac{d}{d\boldsymbol{Q}} \left(\tilde{H}_k^\delta(\boldsymbol{Q}, \boldsymbol{Q}_k^t) - c_k \operatorname{tr}(\boldsymbol{Q}) \right) \Big|_{\boldsymbol{Q}=\boldsymbol{Q}_k^{t+1}} =$$

$$\frac{1}{2} \left(\boldsymbol{Q}_k^{t+1} \sum_{\boldsymbol{x} \in \mathcal{X}_k} \frac{\boldsymbol{x}\boldsymbol{x}^T}{\max(\|\boldsymbol{Q}_k^t \boldsymbol{x}\|, \delta)} + \sum_{\boldsymbol{x} \in \mathcal{X}_k} \frac{\boldsymbol{x}\boldsymbol{x}^T}{\max(\|\boldsymbol{Q}_k^t \boldsymbol{x}\|, \delta)} \boldsymbol{Q}_k^{t+1} \right) + \boldsymbol{A}_k - c_k \boldsymbol{I} = \boldsymbol{0}. \quad (2.45)$$

The last equation follows from the definition of $\boldsymbol{Q}_k^{t+1}$ (see (2.12)). Combining this with the fact that $H^\delta(\boldsymbol{Q}, \boldsymbol{Q}_k^t)$ is strictly convex when restricted to $\boldsymbol{Q} \in \mathbb{H}$, we conclude that $\boldsymbol{Q}_k^{t+1}$is the unique minimizer of $H^\delta(\boldsymbol{Q}, \boldsymbol{Q}_k^t)$ for $\boldsymbol{Q} \in \mathbb{H}$.

Step 2: **Convergence of** $\{G_k^\delta(\boldsymbol{Q}_k^t)\}_{t \in \mathbb{N}}$. We first note that G_k^δ is bounded from below on $\mathbb{H}$. Indeed, $G_k^\delta(\boldsymbol{Q}) \geq \operatorname{tr}(\boldsymbol{Q}\boldsymbol{A}_k) \geq \operatorname{tr}(\boldsymbol{Q}) \times \min \operatorname{eig}(\boldsymbol{A}_k) = \min \operatorname{eig}(\boldsymbol{A}_k)$.

Next, we show that $G_k^\delta(\boldsymbol{Q}_k^t)$ decreases with t. By using (2.42) and the fact that $\boldsymbol{Q}_k^{t+1}$ is the minimizer of $H_k^\delta(\boldsymbol{Q}, \boldsymbol{Q}_k^t)$ for $\boldsymbol{Q} \in \mathbb{H}$, we get that

$$G_k^\delta(\boldsymbol{Q}_k^{t+1}) \leq H_k^\delta(\boldsymbol{Q}_k^{t+1}, \boldsymbol{Q}_k^t) \leq H_k^\delta(\boldsymbol{Q}_k^t, \boldsymbol{Q}_k^t) = G_k^\delta(\boldsymbol{Q}_k^t). \quad (2.46)$$

Since $\{G_k^\delta(\boldsymbol{Q}_k^t)\}_{t \in \mathbb{N}}$ is bounded from below and decreases, it converges.

Step 3: $\|\boldsymbol{Q}_k^t - \boldsymbol{Q}_k^{t+1}\| \to 0$ **as** $t \to \infty$. It follows from (2.45) and the fact that $\boldsymbol{Q}_k^t - \boldsymbol{Q}_k^{t+1} \in \mathcal{S}^D$ has trace 0, that

$$\operatorname{tr} \left((\boldsymbol{Q}_k^{t+1} \sum_{\boldsymbol{x} \in \mathcal{X}_k} \frac{\boldsymbol{x}\boldsymbol{x}^T}{\max(\|\boldsymbol{Q}_k^t \boldsymbol{x}\|, \delta)} + \sum_{\boldsymbol{x} \in \mathcal{X}_k} \frac{\boldsymbol{x}\boldsymbol{x}^T}{(\|\boldsymbol{Q}_k^t \boldsymbol{x}\|, \delta)} \boldsymbol{Q}_k^{t+1} + 2\boldsymbol{A}_k)(\boldsymbol{Q}_k^t - \boldsymbol{Q}_k^{t+1}) \right) = 0.$$

Simplifying the above equation, we get that

$$\operatorname{tr} \left(\boldsymbol{A}_k \left(\boldsymbol{Q}_k^t - \boldsymbol{Q}_k^{t+1} \right) \right) = - \operatorname{tr} \left(\boldsymbol{Q}_k^{t+1} \sum_{\boldsymbol{x} \in \mathcal{X}_k} \frac{\boldsymbol{x}\boldsymbol{x}^T}{\max(\|\boldsymbol{Q}_k^t \boldsymbol{x}\|, \delta)} \left(\boldsymbol{Q}_k^t - \boldsymbol{Q}_k^{t+1} \right) \right) =$$

$$\operatorname{tr} \left(\sum_{\boldsymbol{x} \in \mathcal{X}_k} \frac{\boldsymbol{Q}_k^{t+1} \boldsymbol{x}\boldsymbol{x}^T (\boldsymbol{Q}_k^{t+1} - \boldsymbol{Q}_k^t))}{\max(\|\boldsymbol{Q}_k^t \boldsymbol{x}\|, \delta)} \right) = \sum_{\boldsymbol{x} \in \mathcal{X}_k} \frac{\boldsymbol{x}^T \boldsymbol{Q}_k^{t+1} (\boldsymbol{Q}_k^{t+1} - \boldsymbol{Q}_k^t) \boldsymbol{x}}{\max(\|\boldsymbol{Q}_k^t \boldsymbol{x}\|, \delta)}. \quad (2.47)$$

It follows from (2.46) and (2.41) that

$$G_k^\delta(\boldsymbol{Q}_k^t) - G_k^\delta(\boldsymbol{Q}_k^{t+1}) \geq H_k^\delta(\boldsymbol{Q}_k^t, \boldsymbol{Q}_k^t) - H_k^\delta(\boldsymbol{Q}_k^{t+1}, \boldsymbol{Q}_k^t) = \frac{1}{2}\sum_{\boldsymbol{x}\in\mathcal{X}_k}\left(\frac{\|\boldsymbol{Q}_k^t\boldsymbol{x}\|^2 - \|\boldsymbol{Q}_k^{t+1}\boldsymbol{x}\|^2}{\max(\|\boldsymbol{Q}_k^t\boldsymbol{x}\|,\delta)}\right) + \mathrm{tr}((\boldsymbol{Q}_k^t - \boldsymbol{Q}_k^{t+1})\boldsymbol{A}) = \frac{1}{2}\sum_{\boldsymbol{x}\in\mathcal{X}_k}\left(\frac{\boldsymbol{x}^T(\boldsymbol{Q}_k^t)^2\boldsymbol{x} - \boldsymbol{x}^T(\boldsymbol{Q}_k^{t+1})^2\boldsymbol{x}}{\max(\|\boldsymbol{Q}_k^t\boldsymbol{x}\|,\delta)}\right) + \mathrm{tr}((\boldsymbol{Q}_k^t - \boldsymbol{Q}_k^{t+1})\boldsymbol{A}). \quad (2.48)$$

The combination of (2.47) and (2.48) yields

$$G_k^\delta(\boldsymbol{Q}_k^t) - G_k^\delta(\boldsymbol{Q}_k^{t+1}) \geq \frac{1}{2}\sum_{\boldsymbol{x}\in\mathcal{X}_k}\left(\frac{\boldsymbol{x}^T(\boldsymbol{Q}_k^t)^2\boldsymbol{x} - \boldsymbol{x}^T(\boldsymbol{Q}_k^{t+1})^2\boldsymbol{x}}{\max(\|\boldsymbol{Q}_k^t\boldsymbol{x}\|,\delta)}\right) + \sum_{\boldsymbol{x}\in\mathcal{X}_k}\frac{\boldsymbol{x}^T\boldsymbol{Q}_k^{t+1}(\boldsymbol{Q}_k^{t+1} - \boldsymbol{Q}_k^t)\boldsymbol{x})}{\max(\|\boldsymbol{Q}_k^t\boldsymbol{x}\|,\delta)} = \frac{1}{2}\sum_{\boldsymbol{x}\in\mathcal{X}_k}\frac{\|(\boldsymbol{Q}_k^t - \boldsymbol{Q}_k^{t+1})\boldsymbol{x}\|^2}{\max(\|\boldsymbol{Q}^t\boldsymbol{x}\|,\delta)} \geq 0. \quad (2.49)$$

Since $\{G(\boldsymbol{Q}_k^t)\}_{t\in\mathbb{N}}$ converges, (2.49) implies that

$$\sum_{\boldsymbol{x}\in\mathcal{X}_k}\frac{\|(\boldsymbol{Q}_k^t - \boldsymbol{Q}_k^{t+1})\boldsymbol{x}\|^2}{\max(\|\boldsymbol{Q}_k^t\boldsymbol{x}\|,\delta)} \to 0 \text{ as } t\to\infty \quad (2.50)$$

and consequently (using the fact that $\mathrm{Span}\{\boldsymbol{x}\}_{\boldsymbol{x}\in\mathcal{X}_k} = \mathbb{R}^D$):

$$\|\boldsymbol{Q}_k^t - \boldsymbol{Q}_k^{t+1}\| \to 0 \text{ as } t\to\infty. \quad (2.51)$$

Step 4: **Convergence of $\{\boldsymbol{Q}_k^t\}_{t\in\mathbb{N}}$ to the minimizer of $G_k^\delta(\boldsymbol{Q})$.** The sequence $\{\boldsymbol{Q}_k^t\}_{t\in\mathbb{N}}$ lies in the compact set of positive semi-definite matrices with trace 1. By Bolzano-Weierstrass theorem, $\{\boldsymbol{Q}_k^t\}_{t\geq 1}$ has a converging subsequence. Let $\tilde{\boldsymbol{Q}}_k$ denote the limit of the subsequence. We show that

$$\tilde{\boldsymbol{Q}}_k = \arg\min_{\boldsymbol{Q}\in\mathbb{H}} G_k^\delta(\boldsymbol{Q}). \quad (2.52)$$

By lemma 5 and the fact that the limits of $G_k^\delta(\boldsymbol{Q}_k^t)$ and $G_k^\delta(\boldsymbol{Q}_k^{t+1}) \equiv G_k^\delta(T_{\boldsymbol{A}}(\boldsymbol{Q}_k^t))$ are the same, we conclude that $G_k^\delta(\tilde{\boldsymbol{Q}}_k) = G_k^\delta(T_{\boldsymbol{A}}(\tilde{\boldsymbol{Q}}_k))$. Combining this result with

(2.46) we get that $H_k^\delta(T_{\boldsymbol{A}}(\tilde{\boldsymbol{Q}}_k), \tilde{\boldsymbol{Q}}_k) = H_k^\delta(\tilde{\boldsymbol{Q}}_k, \tilde{\boldsymbol{Q}}_k)$. Since $T_{\boldsymbol{A}}(\tilde{\boldsymbol{Q}}_k)$ is the unique minimizer of $H_k^\delta(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}_k)$ among all $\boldsymbol{Q} \in \mathbb{H}$ we get that $T_{\boldsymbol{A}}(\tilde{\boldsymbol{Q}}_k) = \tilde{\boldsymbol{Q}}_k$. That is, $\tilde{\boldsymbol{Q}}_k$ is the unique minimizer of $H_k^\delta(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}_k)$ and $\tilde{H}_k^\delta(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}_k)$ among all $\boldsymbol{Q} \in \mathbb{H}$ and thus the directional derivatives of $\tilde{H}_k^\delta(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}_k)$ with respect to $\boldsymbol{Q}$ restricted to $\mathbb{H}$ are $\boldsymbol{0}$. Hence, $\operatorname{tr}\left(\left(\frac{d}{d\boldsymbol{Q}}\tilde{H}_k^\delta(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}_k)|_{\boldsymbol{Q}=\tilde{\boldsymbol{Q}}_k}\right)\left(\boldsymbol{P} - \tilde{\boldsymbol{Q}}_k\right)^T\right) = 0$ and thus there exists $c \in \mathbb{R}$ such that $\frac{d}{d\boldsymbol{Q}}\tilde{H}_k^\delta(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}_k)|_{\boldsymbol{Q}=\tilde{\boldsymbol{Q}}_k} = c\boldsymbol{I}$. This implies that

$$c\boldsymbol{I} = \frac{1}{2}\tilde{\boldsymbol{Q}}_k \sum_{\boldsymbol{x}\in\mathcal{X}_k} \frac{\boldsymbol{x}\boldsymbol{x}^T}{\max(\|\tilde{\boldsymbol{Q}}_k\boldsymbol{x}\|, \delta)} + \frac{1}{2}\sum_{\boldsymbol{x}\in\mathcal{X}_k} \frac{\boldsymbol{x}\boldsymbol{x}^T}{\max(\|\tilde{\boldsymbol{Q}}_k\boldsymbol{x}\|, \delta)}\tilde{\boldsymbol{Q}}_k + \boldsymbol{A}_k = \frac{d}{d\boldsymbol{Q}}\tilde{G}_k^\delta(\boldsymbol{Q})|_{\boldsymbol{Q}=\tilde{\boldsymbol{Q}}_k}, \quad (2.53)$$

where

$$\tilde{G}_k^\delta(\boldsymbol{Q}) = \sum_{\boldsymbol{x}\in\mathcal{X}_k, \|\boldsymbol{Q}\boldsymbol{x}\|\geq\delta} \sqrt{\operatorname{tr}(\boldsymbol{Q}\boldsymbol{x}\boldsymbol{x}^T\boldsymbol{Q})} + \sum_{\boldsymbol{x}\in\mathcal{X}, \|\boldsymbol{Q}\boldsymbol{x}\|<\delta} \left(\frac{\operatorname{tr}(\boldsymbol{Q}\boldsymbol{x}\boldsymbol{x}^T\boldsymbol{Q})}{2\delta} + \frac{\delta}{2}\right) + \operatorname{tr}(\boldsymbol{Q}\boldsymbol{A}_k).$$

The directional derivatives of $\frac{d}{d\boldsymbol{Q}}\tilde{G}_k^\delta(\boldsymbol{Q})|_{\boldsymbol{Q}=\tilde{\boldsymbol{Q}}_k}$ restricted to $\mathbb{H}$ are

$$\operatorname{tr}\left(\left(\frac{d}{d\boldsymbol{Q}}\tilde{G}_k^\delta(\boldsymbol{Q})|_{\boldsymbol{Q}=\tilde{\boldsymbol{Q}}_k}\right)\left(\boldsymbol{P} - \tilde{\boldsymbol{Q}}_k\right)^T\right) = \operatorname{tr}\left(c\boldsymbol{I}\left(\boldsymbol{P} - \tilde{\boldsymbol{Q}}_k\right)^T\right) = 0, \quad (2.54)$$

where for the first equality we used (2.53) and for the last equality we used that $\boldsymbol{P}, \tilde{\boldsymbol{Q}}_k \in \mathbb{H}$ and thus $\operatorname{tr}(\boldsymbol{P}) = \operatorname{tr}(\tilde{\boldsymbol{Q}}_k) = 1$. Equation (2.54) and the fact that $G_k^\delta(\boldsymbol{Q}) = \tilde{G}_k^\delta(\boldsymbol{Q})$ for $\boldsymbol{Q} \in \mathbb{H}$ imply (2.52). Finally, combining (2.51), (2.52), the definition of $\tilde{\boldsymbol{Q}}_k$ and [40, Theorem 2.1], we conclude that $\boldsymbol{Q}_k^t \to \tilde{\boldsymbol{Q}}_k$ as $t \to \infty$.

Step 5: r**-linear Convergence.** The proof of r-linear convergence of $\boldsymbol{Q}_k^t$ follows from Theorem 6.1 of [38] (similarly to the proof of Theorem 11 of [24]). To show that the conditions of the theorem are satisfied we just need to check that the functions G and H satisfy Hypotheses 4.1 and 4.2 of [38] (see proof of Theorem 6.1 in there and note that G and H of this work are parallel to F and H of [38], respectively). We note that [38] states the result for vector-valued functions, which can be easily generalized for matrix-valued functions. Since $\boldsymbol{Q}_k^t$ converges, it is enough to show that Hypotheses 4.1 and 4.2 hold for some local neighborhood $B(\tilde{\boldsymbol{Q}}_k, \epsilon)$ of $\tilde{\boldsymbol{Q}}_k$, for some $\epsilon > 0$. Conditions

1 and 3 of Hypothesis 4.1 are easy to check, since G is twice differentiable on $B(\tilde{\boldsymbol{Q}}_k, \epsilon)$ and G is bounded from below (as we have already shown). There is no need to check condition 2, since $\boldsymbol{Q}$ is restricted to H. To verify condition 1 of Hypothesis 4.2 we need to show that

$$H_k^\delta(\boldsymbol{Q}_1, \boldsymbol{Q}_2) = G_k^\delta(\boldsymbol{Q}_2) + \operatorname{tr}((\boldsymbol{Q}_1 - \boldsymbol{Q}_2)^T \frac{d}{d\boldsymbol{Q}} G_k^\delta(\boldsymbol{Q})|_{\boldsymbol{Q}=\boldsymbol{Q}_2}) + \frac{1}{2}\operatorname{tr}((\boldsymbol{Q}_1 - \boldsymbol{Q}_2)^T C(\boldsymbol{Q}_2)(\boldsymbol{Q}_1 - \boldsymbol{Q}_2)). \quad (2.55)$$

To prove (2.55), we write its RHS as follows:

$$\sum_{\boldsymbol{x}\in\mathcal{X}_k, \|\boldsymbol{Q}_2\boldsymbol{x}\|\geq\delta} \|\boldsymbol{Q}_2\boldsymbol{x}\| + \sum_{\boldsymbol{x}\in\mathcal{X}_k, \|\boldsymbol{Q}_2\boldsymbol{x}\|<\delta} \left(\frac{\|\boldsymbol{Q}_2\boldsymbol{x}\|^2}{2\delta} + \frac{\delta}{2}\right) + \operatorname{tr}(\boldsymbol{Q}_2\boldsymbol{A}_k) +$$
$$\operatorname{tr}\left((\boldsymbol{Q}_1 - \boldsymbol{Q}_2)^T \frac{1}{2}\left(\boldsymbol{Q}_2 \sum_{\boldsymbol{x}\in\mathcal{X}_k} \frac{\boldsymbol{x}\boldsymbol{x}^T}{\max(\|\boldsymbol{Q}_2\boldsymbol{x}\|, \delta)} + \sum_{\boldsymbol{x}\in\mathcal{X}_k} \frac{\boldsymbol{x}\boldsymbol{x}^T}{\max(\|\boldsymbol{Q}_2\boldsymbol{x}\|, \delta)}\boldsymbol{Q}_2\right) + \boldsymbol{A}_k\right) +$$
$$\operatorname{tr}((\boldsymbol{Q}_1 - \boldsymbol{Q}_2)^T \frac{1}{2} C(\boldsymbol{Q}_2)(\boldsymbol{Q}_1 - \boldsymbol{Q}_2)).$$

By setting $C(\boldsymbol{Q}) = \sum_{x\in X_k} \boldsymbol{x}\boldsymbol{x}^T / \max(\|\boldsymbol{Q}\boldsymbol{x}\|, \delta)$, the above equation becomes

$$\sum_{\boldsymbol{x}\in\mathcal{X}_k, \|\boldsymbol{Q}_2\boldsymbol{x}\|\geq\delta} \|\boldsymbol{Q}_2\boldsymbol{x}\| + \sum_{\boldsymbol{x}\in\mathcal{X}_k, \|\boldsymbol{Q}_2\boldsymbol{x}\|<\delta} \left(\frac{\|\boldsymbol{Q}_2\boldsymbol{x}\|^2}{2\delta} + \frac{\delta}{2}\right) + \operatorname{tr}(\boldsymbol{Q}_2\boldsymbol{A}_k) +$$
$$\operatorname{tr}\left((\boldsymbol{Q}_1 - \boldsymbol{Q}_2)^T \boldsymbol{Q}_2 \sum_{\boldsymbol{x}\in\mathcal{X}_k} \frac{\boldsymbol{x}\boldsymbol{x}^T}{\max(\|\boldsymbol{Q}_2\boldsymbol{x}\|, \delta)} + \boldsymbol{A}_k\right) +$$
$$\operatorname{tr}\left((\boldsymbol{Q}_1 - \boldsymbol{Q}_2)^T \sum_{x\in X_k} \frac{\boldsymbol{x}\boldsymbol{x}^T}{2\max(\|\boldsymbol{Q}_2\boldsymbol{x}\|, \delta)}(\boldsymbol{Q}_1 - \boldsymbol{Q}_2)\right) = \sum_{\boldsymbol{x}\in\mathcal{X}_k, \|\boldsymbol{Q}_2\boldsymbol{x}\|\geq\delta} \|\boldsymbol{Q}_2\boldsymbol{x}\| +$$
$$\sum_{\boldsymbol{x}\in\mathcal{X}_k, \|\boldsymbol{Q}_2\boldsymbol{x}\|<\delta} \left(\frac{\|\boldsymbol{Q}_2\boldsymbol{x}\|^2}{2\delta} + \frac{\delta}{2}\right) + \operatorname{tr}(\boldsymbol{Q}_1\boldsymbol{A}_k) - \sum_{\boldsymbol{x}\in\mathcal{X}_k} \frac{\|\boldsymbol{Q}_2\boldsymbol{x}\|^2}{\max(\|\boldsymbol{Q}_2\boldsymbol{x}\|, \delta)} +$$
$$\sum_{\boldsymbol{x}\in\mathcal{X}_k} \frac{\|\boldsymbol{Q}_2\boldsymbol{x}\|^2}{2\max(\|\boldsymbol{Q}_2\boldsymbol{x}\|, \delta)} + \sum_{\boldsymbol{x}\in\mathcal{X}_k} \frac{\|\boldsymbol{Q}_1\boldsymbol{x}\|^2}{2\max(\|\boldsymbol{Q}_2\boldsymbol{x}\|, \delta)} = \sum_{\boldsymbol{x}\in\mathcal{X}_k} \frac{\|\boldsymbol{Q}_1\boldsymbol{x}\|^2}{2\max(\|\boldsymbol{Q}_2 x\|, \delta)} +$$
$$\sum_{\boldsymbol{x}\in\mathcal{X}_k} \frac{\max(\|\boldsymbol{Q}_2\boldsymbol{x}\|, \delta)}{2} + \operatorname{tr}(\boldsymbol{Q}_1\boldsymbol{A}_k) = H(\boldsymbol{Q}_1, \boldsymbol{Q}_2).$$

That is, condition 1 of Hypothesis 4.2 is verified, conditions 2 and 3 follow directly from

the definition of $C(\boldsymbol{Q})$ and condition 4 follows from (2.43). $\square$

Chapter 3

Part II: Solving Jigsaw Puzzles By The Connection Graph Laplacian

3.1 Introduction

Solving jigsaw puzzles is an entertaining task, which is commonly explored by children and adults. It is also a challenging mathematical and engineering problem that occupies researchers in computer science, mathematics and engineering. The solution of this problem is useful for several industrial applications. One example is reassembling archaeological artifacts [41, 42], where one tries to recover the shape of an archaeological object from damaged pieces. Another example is recovering shredded documents or photographs [43, 44], where one tries to recover a document or a picture from small pieces of it. Additional applications appear in biology [45] and speech descrambling [46].

The automatic solution of puzzles, without having any information on the underlying image, is known to be NP hard [47, 48]. The first algorithm that attempted to automatically solve general puzzles was introduced by Freeman and Garder [49] in 1964. It was designed to solve puzzles with 9 pieces by only considering the geometric shapes of the pieces.

A recent setting of "jigsaw" puzzles assumes an image cut into equal square pieces

and the mathematical problem is to recover the original image from the given pieces, which are possibly rotated and shifted along the puzzle grid. Gallagher [50] categorized these kinds of jigsaw puzzles into three types. In type 1 puzzles the pieces are not rotated, but shifted. In type 3 puzzles the pieces are not shifted, but rotated. In type 2 puzzles, the pieces are both shifted and rotated. We later formulate a more general setting, but our current work address this special setting.

Many proposals for solving the latter jigsaw puzzles are based on greedy methods [50, 51, 52, 53, 54]. However, greedy algorithms can easily get trapped in locally optimal solutions, which are not global. Some proposals also involve non-greedy constructive methods [55, 56, 57], which are often combined with greedy procedures. This work proposes a constructive framework for recovering rotations and for improving the metric between puzzle pieces. However, it also relies on common methods for recovering locations, which are either greedy or partly greedy.

3.1.1 Previous Work

Several algorithms have been recently proposed for automatic solution of the latter jigsaw puzzles [46, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59]. The problem gets more challenging when the number of puzzle pieces increases and the sizes of puzzle pieces decrease. Some of these algorithms only consider type 1 puzzles (see e.g., [46, 51, 52, 55, 56]), since recovering orientations increases the possible comparisons between two pieces by four and may also decrease the accuracy of solving the puzzle. The rest of these algorithms focus on type 2 puzzles, where [50] also separately discusses type 3 puzzles.

Cho et al. [55] proposed a probabilistic, graphical model approach to the jigsaw puzzle problem and discussed different compatibility metrics between puzzle pieces. Pomeranz et al. [51] proposed a greedy method, discussed few compatibility metrics and included some analysis on how to pick the correct compatibility metric for their method. Gallagher [50] proposed a tree-based reassembly algorithm, which greedily merges components while respecting the geometric consistence constraints. It runs in three steps: building a constrained tree, trimming and filling. Andalo et al. [52] proposed quadratic assignment approach, which maximizes a constrained quadratic function via constrained gradient ascent. Sholomon et al. [56, 57] proposed a genetic algorithm.

Son et al. [53] proposed finding small loops (4-cycles) of puzzle pieces, which form

consistent cycles, and then hierarchically combining these small loops into higher order loops in a bottom-up fashion. They argued that loop constraints could effectively eliminate pairwise matching outliers. Son et al. [54] proposed a growing consensus approach that assembles pieces by multiple modest bonds and uses a new objective function that maximizes consensus configurations. Yu et al. [59] proposed a linear programming based formulation, which combines global and greedy approaches. Their proposed solver simultaneously exploits all the pairwise matches, and globally computes the location of each piece/component at each step of the algorithm.

A procedure for solving type 3 puzzles was only considered by Gallagher [50] using a greedy method. We are not aware of any previous constructive and non-greedy procedure for solving type 3 puzzles. More importantly, we are not aware of a previous general method for finding the orientations of puzzle pieces with unknown locations. Such a procedure can enhance the solution of type 2 puzzles.

3.1.2 Our Contribution

In this chapter we propose a novel approach to address type 2 and type 3 jigsaw puzzles. For type 3 puzzles, we suggest fast, robust and constructive solution that uses the connection graph Laplacian (CGL) [60] (discussed in §3.3.1). Since the locations of puzzle pieces are given for type 3 puzzles, there is no need to find the metric between puzzle pieces, but only between neighboring pieces. Therefore the complexity of our proposed algorithm for type 3 puzzles is relatively low.

For type 2 puzzles we propose a novel iterative algorithm, which solves the following two subproblems (SPs) iteratively:

SP1 Finding the orientations of all puzzle pieces.

SP2 Finding the locations of all puzzle pieces.

These two steps are iteratively repeated until the desired result is achieved. We solve SP1 by using the CGL. We solve SP2 by incorporating an improved metric, obtained from the solution of SP1, within any state-of-the-art solution of type 1 puzzles. Some information inferred from the solution of SP2 is used to improve the solution of SP1.

All previous algorithms for solving type 2 puzzles simultaneously addressed both subproblems. On the other hand, this work separately solves the two subproblems

and iteratively updates the solutions. The proposed procedure is also faster than the previous simultaneous procedures as long as the solver for SP1 is relatively accurate. Indeed, SP2 asks to solve type 1 puzzles, which are easier than type 2 puzzles. Moreover, most previous algorithms are greedy, whereas the one proposed here is not.

3.1.3 Structure of This Chapter

This chapter is organized as follows: §3.2 discusses a general mathematical setting for the jigsaw puzzle problem and a special case of it, which is studied in the work; §3.3 presents a solution for SP1, which assumes the existence of a "connection graph"; §3.4 shows how to construct the connection graph for type 2 and type 3 puzzles; §3.5 presents a solution of SP2 that uses the solution of SP1 and also suggests how to update the solution of SP1 based on the solution of SP2; §3.6 concludes with numerical experiments that test the proposed algorithm using digital images; at last, §3.7 concludes with a short discussion that includes possible extensions of this work.

3.2 The Mathematical Setting for Jigsaw Puzzles

Here we mathematically formalize the jigsaw puzzle problem. We first formulate a general abstract setting of this problem in §3.2.1. We then describe a specific special case of interest in §3.2.2, where we also discuss possible generalization and the direct application to the discrete setting of the application area of this work. At last, §3.2.3 discusses the main challenge of addressing the specific setting.

3.2.1 A General Mathematical Formulation

Our general mathematical formulation assumes a d-dimensional compact manifold M embedded in $\mathbb{R}^q$ via the inclusion map ι. For simplicity, we refer to the embedded manifold by M instead of $\iota(M)$. For this embedded M, we further assume a sufficiently smooth function $f : M \to \mathbb{R}^k$, where $k \geq 1$. One may assume different smoothness classes containing f depending on the application domain. For the application we consider, which has a discrete setting with discontinuities of f, the assumption $f \in L^2(M, \mathbb{R}^k)$ seems natural.

We will first discuss the notion of patches partitioning the embedded M as well as image patches. Generally, a patch is a subset of the embedded M. Since our mathematical setting is continuous we assume that patches are open sets. We later explain how this assumption does not matter to the discrete setting of this work. An image patch on the embedded M is a pair of a patch and the restriction of f on it. For simplicity, we denote a patch by P, even though $\iota(P)$ is more precise. Similarly, we denote an image patch by $(P, f|_P)$, even though $(\iota(P), f|_{\iota(P)})$ is more precise.

We partition M into open patches $\{P_i\}_{i=1}^n$ so that $M = \cup_{i=1}^n \bar{P}_i$, where for $1 \leq i \leq n$, $P_i \subset M$ and $\bar{P}_i$ is the closure of P_i, and also for $1 \leq i \neq j \leq n$, $P_i \cap P_j = \emptyset$. When defining the corresponding image patches we allow local rigid transformations, such as rotations and translations. We make the problem formulation even more general by considering local diffeomorphic transformations. For each $1 \leq i \leq n$, consider a transform D_i on $\mathbb{R}^q$, so that $D_i(P_i) \subset \mathbb{R}^q$ is diffeomorphic to P_i. For $\boldsymbol{x} \in \mathbb{R}^q$, define $(D_i \circ f|_{P_i})(\boldsymbol{x}) := f(D_i^{-1}(\boldsymbol{x}))$ when $D_i^{-1}(\boldsymbol{x}) \in P_i$ and $\mathbf{0}$ otherwise.

There are three jigsaw puzzle problems we could formulate:

P0: Given a set of image patches $\mathcal{Q} := \{(D_i(P_i), D_i \circ f|_{P_i})\}_{i=1}^n$ and M, recover f.

P1: Given a set of image patches $\mathcal{Q} := \{(D_i(P_i), D_i \circ f|_{P_i})\}_{i=1}^n$, recover f and M.

P2: Given a set of patches $\mathcal{P} := \{D_i(P_i)\}_{i=1}^n$, recover M.

In general these are ill-defined and challenging problems, since more conditions may be needed. For example, if f is a constant function on a sufficiently large region of M and the shapes of the puzzle patches are not sufficient to uniquely determine neighboring patches, then there is no information available for reconstructing f. Similarly, estimating the unknown local diffeomorphic functions is a challenging problem, and it makes sense to further restrict them. On the other hand, there are simplified, well-defined versions of these problems. In this work we address P0 in the very specific setting of the 2-*dimensional square jigsaw puzzle problem*, which we describe next.

3.2.2 A Special Setting and its Generalization

In the 2-dimensional square jigsaw puzzle problem, M is a rectangle in $\mathbb{R}^2$, $M = [a_1, b_1] \times [a_2, b_2]$ and $\{P_i\}_{i=1}^n$ form a square tiling of M. That is, the open patches partitioning M

are shifted versions of the same square. We assume that $f \in L^2(M, \mathbb{R}^k)$ and $k \geq 1$. We think of the graph $\{(\boldsymbol{x}, f(\boldsymbol{x})) : \boldsymbol{x} \in M \subset \mathbb{R}^2$ as a continuous version of an image. One may use $k = 1$ for gray-scale images, $k = 3$ for color images and higher k for multispectral and hyperspectral images. We further assume that the diffeomorphic transformations D_i, $1 \leq i \leq n$, are proper rigid transformations from one patch to another. That is, they are combinations of rotations and translations, where a rotation is by $0°$, $90°$, $180°$ or $270°$, and all shifts of patches $\{P_i\}_{i=1}^n$ can be described as $\{P_{\sigma(i)}\}_{i=1}^n$, where σ is a permutation of degree n. This assumes that the grid is labeled by numbers and the goal is to find the correct permutation for the indexes of all patches that would place each square in the correct position of the tile. Therefore, we can write the set of image patches as $\mathcal{Q} = \{(R_{\sigma_i}(P_{\sigma_i}), R_{\sigma_i} \circ f|_{P_{\sigma_i}})\}_{i=1}^n$, where σ is a permutation of degree n, R_{σ_i} is an element of the cyclic group $\mathbb{Z}_4$ and the action $\circ$ was defined above. The problem of interest in this setting is P0. Note that its solution requires recovering $\{R_i\}_{i=1}^n$ and σ. We also remark that in this case finding M in P1 is unique up to a proper rigid transformation, however, the extra component of P1 is artificial for this setting. Furthermore, in this setting, P2 is ill-defined as it has many possible solutions. In general, a solution of P2 requires stronger assumptions, for example, on the shape of puzzle patches that can be in the standard shape of jigsaw puzzles or on the manifold that can be asked to be closed.

One can consider an equivalent formulation, where instead of having patches initially on the grid, the patches are arbitrarily shifted and rotated within $\mathbb{R}^2$. In this new formulation, $D_i = T_{\boldsymbol{x}_i} \circ R_i$, where $\boldsymbol{x}_i$ is an arbitrary vector in $\mathbb{R}^2$, $T_{\boldsymbol{x}_i}(\boldsymbol{x}) = \boldsymbol{x}_i + \boldsymbol{x}$ for each $\boldsymbol{x} \in M \subset \mathbb{R}^2$, and R_i is an arbitrary element of SO(2). The equivalence of the two formulations is evident. Indeed, given patches with any choice of centers and rotations, one can arbitrarily assign them to a grid and use the former formulation, and vice versa. Nevertheless, the latter formulation can apply to more general settings. For example, settings with more complicated shapes of patches, such as polygonal shapes, which are common in tangram puzzles, or shapes with curvy edges, which are common in commercial jigsaw puzzles. Mathematical ideas for solving these two kinds of puzzles appear in [61] and [58] respectively. We remark that there are cases of more complicated shapes that are easier to solve. For example, if the shapes of the patches lead to unique determination of the neighboring patches, then exact reconstruction is easier.

We remark that whenever we refer to neighboring patches in the two-dimensional square jigsaw puzzle, we mean the four nearest neighbors of the given patch (left, right, top or bottom). On the other hand, there are clearly very difficult cases of complicated shapes with many possibilities of aligning them together. In general, one may also consider various 3D puzzles or more complicated problems. Note that most of the ideas discussed in this chapter can be well suited for puzzles with non-square patches and higher dimensional non-flat manifold.

The particular instance of the 2-dimensional square jigsaw puzzle problem we discuss in this work is demonstrated in Figure 3.1. In this setting of RGB images $k = 3$. While the digital images and their patches are discrete, they have a corresponding continuous description. That is, f can be viewed as a piecewise constant function with constant values on squares corresponding to image pixels. This is the common way of presenting discrete images, and is evident in the presentation of patches at the top right image of Figure 3.2 where the number of pixels per unit length is low relative to the demonstration on top left image of Figure 3.2. We remark that the last column of this figure, illustrates the image patches $\mathcal{Q} = \{(R_{\sigma_i}(P_{\sigma_i}), R_{\sigma_i} \circ f|_{P_{\sigma_i}})\}_{i=1}^{n}$ discussed above.

Figure 3.1: Examples of puzzles with 12 patches. Left column: the original image; Central column: division of the image into 12 square patches of the same size. Right column: The 12 patches are randomly reordered and rotated.

3.2.3 The Main Challenge of the Special Setting

We recall that the formulation of the 2-dimensional square jigsaw puzzle problem requires finding a permutation σ and rotations $\{R_i\}_{i=1}^n \subset \mathrm{SO}(2)$. Equivalently, one may solve for locations $\{\boldsymbol{x}_i\}_{i=1}^n$ on a uniform grid, representing the centers of the patches, and rotations $\{R_i\}_{i=1}^n$. In order to estimate these from $\mathcal{Q}$ for general functions f, one needs to rely on the similar function values on edges of neighboring patches. In our setting of digital images, we should often expect discontinuities in values of f on neighboring edges. The top right image of Figure 3.2 demonstrates this phenomenon for two patches selected from the puzzles shown in top left image with lower resolution. Such discontinuity can result in loss of information for determining neighbors and may lead to ill-posed problems.

There are also special images for which the puzzle problem is ill-posed. For example, the bottom left image of Figure 3.2 demonstrates a case where several patches look very similar to each other and it is impossible to determine the right permutation. Nevertheless, the output of common algorithms given this particular puzzle is often visually acceptable. On the other hand, the bottom right image of Figure 3.2 demonstrates a case where the image consists of two parts that are disconnected by a uniform background. In this particular instance, the background is the white sky, one part is the main scene of the image and the other part includes two short branches of another tree at the top left corner of the image. In this case it would be impossible to figure out the exact position of the latter part of the image.

The following definition quantifies an ideal type of metric between edges of image patches that if exists, that is, if the problem is well-posed, it can be used to solve the two-dimensional square jigsaw puzzle problem.

Definition 4. *Fix an image I and a set of image patches $\mathcal{Q} := \{P_i, f|_{P_i}\}_{i=1}^n$. A metric defined on $\mathcal{Q}$ is called perfect if there exists $c > 0$ so that two neighboring patches have a distance less than c and two non-neighboring patches have a distance greater than c.*

The main challenge of solving reasonable instances of the 2-dimensional square jigsaw puzzle problem, is to find a nearly perfect metric. Empirically, we have found that the the Mahalanobis Gradient Compatibility (MGC) metric, described in §3.4.1, is often near perfect in well-posed cases.

3.3 Frameworks for Recovering Rotations of Puzzle Pieces

This section applies the framework of [60, 62] for recovering the global orientations of puzzle patches. It also mentions another framework. These frameworks require the construction of a graph whose vertices correspond to the puzzle patches and its edges connect neighboring patches. The rest of the section is organized as follows: §3.3.1 forms the connection graph Laplacian (CGL) and explains how to estimate the rotations of puzzle patches by this graph; §3.3.2 describes an equivalent framework for solving this problem; §3.3.3 theoretically justifies the method described in §3.3.1.

3.3.1 Estimation of Orientations Using the Connection Graph

The general connection graph [60] $G = (V, E, W, R)$ consists of four components: vertices V, edges E, *affinity function* (or weight function) $W : E \to [0, 1]$ and the *connection function* $R : E \to \mathbb{G}$, where $\mathbb{G}$ is a given group. The first three components are determined by the weighted graph and the fourth one depends on the application in which the graph is used. In the case of the two-dimensional square jigsaw puzzles with a perfect metric (recall Definition 4), the ideal connection graph is formed as follows. The vertices represent patches in $\mathcal{Q}$, the edges connect neighboring patches and the weights are 1 for all edges and 0 otherwise. The group $\mathbb{G}$ is the cyclic group $\mathbb{Z}_4$ which we can represent either by the four complex numbers $\{1, i, -1, -i\}$ with complex multiplication or by the following four 2×2 matrices:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

with matrix multiplication. Note that for each edge $\{i, j\}$ with $i < j$, which connects patches P_i and P_j, an element $\boldsymbol{R}(i, j)$ of $\mathbb{G} = \mathbb{Z}_4$ is a rotation whose application to P_j results in a match with P_i after an appropriate plane translation.

For possibly imperfect scenarios of the two-dimensional square jigsaw puzzles, the vertices are formed as above, but one needs to construct meaningful edges, affinity function and connection function (with $\mathbb{G} = \mathbb{Z}_4$). A heuristic construction of these is suggested for type 2 and type 3 puzzles in §3.4.3 and §3.4.2 respectively. Here we propose a general heuristic that uses a given connection graph of two-dimensional square jigsaw

puzzles to estimate the unknown orientations of the patches. This heuristic is later justified in §3.3.3 under special assumptions. The main idea of this heuristic is to use the CGL for inferring global information (in the form of a certain eigendecomposition) from local information (needed to form the CGL).

Next, we review several matrices associated with a general connection graph. Since for two-dimensional square jigsaw puzzles, W and R are defined on the set $\{1,\ldots,n\}\times\{1,\ldots,n\}$, where n is the number of puzzle pieces, we denote them from now on by their corresponding matrices $\boldsymbol{W}\in\mathbb{R}^{n\times n}$ and $\boldsymbol{R}\in\mathbb{R}^{2n\times 2n}$, respectively. Note that $\boldsymbol{R}$ is a block matrix whose 2×2 blocks represent rotations. For $1\le i,j\le n$, we denote by $\boldsymbol{R}(i,j)$ the i,j-th 2×2 block of $\boldsymbol{R}$. The connection graph is thus $G=(V,E,\boldsymbol{W},\boldsymbol{R})$. The *connection adjacency matrix* is an $n\times n$ block matrix $\boldsymbol{S}$ with 2×2 submatrices, where for $1\le i,j\le n$ the (i,j)-th submatrix is

$$\boldsymbol{S}_{ij}=\begin{cases}\boldsymbol{W}(i,j)\boldsymbol{R}(i,j), & \text{when } (i,j)\in E;\\ \boldsymbol{0}, & \text{otherwise.}\end{cases} \tag{3.1}$$

The *degree matrix* is an $n\times n$ block diagonal matrix $\boldsymbol{D}$, where for $1\le i\le n$, its i-th diagonal submatrix is

$$\boldsymbol{D}_{ii}=\sum_{j\neq i}\boldsymbol{W}(i,j)\boldsymbol{I}_2, \tag{3.2}$$

where $\boldsymbol{I}_2$ is the 2×2 identity matrix. We define $\boldsymbol{C}:=\boldsymbol{D}^{-1}\boldsymbol{S}$ and $\widetilde{\boldsymbol{C}}:=\boldsymbol{D}^{-1/2}\boldsymbol{S}\boldsymbol{D}^{-1/2}$, and refer to the matrix $\boldsymbol{I}-\boldsymbol{C}$ as the *CGL matrix* and $\boldsymbol{I}-\widetilde{\boldsymbol{C}}$ as the normalized CGL matrix.

The CGL matrix is associated with a random walk, whose transition probability matrices are $\boldsymbol{W}(i,j)$, $1\le i,j\le n$. This can be seen by its action on a block vector $\boldsymbol{v}\in\mathbb{R}^{2n\times 2}$, whose n submatrices are

$$\boldsymbol{v}[j]=\begin{bmatrix}\boldsymbol{v}_{2j-1,1} & \boldsymbol{v}_{2j-1,2}\\ \boldsymbol{v}_{2j,1} & \boldsymbol{v}_{2j,2}\end{bmatrix}\in\mathbb{R}^2,\ 1\le j\le n$$

in the following way

$$(\boldsymbol{C}\boldsymbol{v})[i] = \frac{\sum_{j:(i,j)\in E} \boldsymbol{W}(i,j)\boldsymbol{R}(i,j)\boldsymbol{v}[j]}{\sum_{k:(i,k)\in E} \boldsymbol{W}(i,k)}.$$

To recover the global orientations of the puzzle patches, we follow the procedures of [60,62]. We form the block vector $\boldsymbol{U} \in \mathbb{R}^{2n\times 2}$ whose columns are the top 2 eigenvectors of $\boldsymbol{C}$. We then project each of the 2×2 blocks of $\boldsymbol{U}$ onto $\mathbb{Z}_4$ and use the resulting blocks as the global orientations. Algorithm 4 summarizes the above straightforward procedure of recovering the unknown orientations of the image patches for a given two-dimensional square jigsaw puzzles.

Algorithm 4 The CGL Algorithm

Input: Connection graph: $G = (V, E, \boldsymbol{W}, \boldsymbol{R})$

- Construct the Connection Adjacency Matrix $\boldsymbol{S}$ by (3.1)
- Construct the degree matrix $\boldsymbol{D}$ by (3.2)
- Let $\boldsymbol{C} = \boldsymbol{D}^{-1}\boldsymbol{S}$
- Form $\boldsymbol{U} \in \mathbb{R}^{2n\times 2}$ whose columns are the 2 top eigenvector of $\boldsymbol{C}$
- For $1 \le i \le n$, let $\boldsymbol{R}_i \in \mathbb{Z}_4$ be the projection of the ith block of $\boldsymbol{U}$ onto $\mathbb{Z}_4$

Return: Global rotation matrices $\boldsymbol{R}_1, \ldots, \boldsymbol{R}_n$

We emphasize that the CGL algorithm for recovering the orientations of patches is non-greedy. Indeed, it directly constructs the orientation of patches using the information in the connection graph. On the other hand, other methods, such as [50,51,53,54], try to greedily match pieces based on their relative orientations. We also mention that the CGL algorithm does not use any knowledge of the size of the puzzle image, or equivalently, of the number of puzzle pieces per length or width of the image.

3.3.2 Another Formulation

The general problem we have addressed in §3.3.1 is referred to as synchronization. That is, one assumes a connection graph $G = (V, E, W, R)$ and needs to estimate for all vertices $i \in V$ a group element $g_i \in \mathbb{G}$ from noisy or wrong measurements of $g_i g_j^T \in \mathbb{G}$.

In the particular case of the two-dimensional square jigsaw puzzle, the graph is $G = (V, E, \boldsymbol{W}, \boldsymbol{R})$, $\mathbb{G} = \mathbb{Z}_4$ and $g_i g_j^T = \boldsymbol{R}(i, j)$, which was defined in §3.3.1. In this case the synchronization problem is referred to as angular.

We describe here a least-squares formulation for this problem, review two common solutions for it and discuss the similarities and differences of one of them with the method above. Using the matrix $\boldsymbol{S}$ defined in (3.1), the least-squares formulation for angular synchronization asks to solve the optimization problem

$$\arg\min_{\boldsymbol{u}\in\mathbb{Z}_4^n} \|\boldsymbol{u}\boldsymbol{u}^T - \boldsymbol{S}\|^2, \tag{3.3}$$

or equivalently,

$$\arg\max_{\boldsymbol{u}\in\mathbb{Z}_4^n} \operatorname{tr}(\boldsymbol{u}\boldsymbol{u}^T\boldsymbol{S}). \tag{3.4}$$

We remark that sometimes the problem above is formulated with $\boldsymbol{R}$ instead of $\boldsymbol{S}$ when the affinities are ones for edges in E and zeros otherwise, so that $\boldsymbol{R} = \boldsymbol{S}$.

This problem is NP-hard [63]. Nevertheless, approximate solutions were proposed, in particular, semidefinite programming and spectral relaxation [64]. The semidefinite programming method suggests to remove the rank 2 constraint on the PSD (positive semi-definite) matrix $\boldsymbol{u}\boldsymbol{u}^T$ in (3.4) and consequently solve

$$\arg\max_{\boldsymbol{H}\succeq\boldsymbol{0},\boldsymbol{H}(i,i)=\boldsymbol{I}_2} \operatorname{tr}(\boldsymbol{H}\boldsymbol{S}). \tag{3.5}$$

The solution $\boldsymbol{v} \in \mathbb{R}^{2n\times 2}$ is recovered by projecting the blocks of top 2 eigenvectors of solution (3.4) into $\mathbb{Z}_4$.

The spectral relaxation method suggests to relax the set $\mathbb{Z}_4^n$ into $\mathbb{R}^{2n\times 2}$ and solve the following eigenvalue/eigenvector problem

$$\arg\max_{\boldsymbol{u}\in\mathbb{R}^{2n\times 2}:\,\|\boldsymbol{u}\|=2n} \operatorname{tr}(\boldsymbol{u}^T\boldsymbol{S}\boldsymbol{u}). \tag{3.6}$$

The block vector $\tilde{\boldsymbol{v}}$, which contains the top 2 eigenvectors of $\boldsymbol{S}$, solves (3.6). To recover the g_is one can project the 2×2 blocks of $\tilde{\boldsymbol{v}}$ into $\mathbb{Z}_4$. The spectral relaxation is faster and better suited for higher-volume data. The SDP relaxation is often more accurate than the spectral relaxation for SO(2), however, for the special case of $\mathbb{Z}_4$ their accuracy should be comparable, since there are only four, well-separated elements of $\mathbb{Z}_4$. We note

that the spectral relaxation method is very similar to the method described in §3.3.1, but directly uses the matrix $\boldsymbol{S}$ instead of $\boldsymbol{C}$. In fact, the method proposed in §3.3.1 is a spectral relaxation of (3.3) when $\boldsymbol{S}$ is replaced by $\boldsymbol{C}$. One of the advantages of using $\boldsymbol{C}$ instead of $\boldsymbol{S}$ is that it gives rise to a natural diffusion distance, which is discussed later in §3.5.1.

3.3.3 Theoretical Justification of the CGL Algorithm

In this section we show that the CGL algorithm for two-dimensional square jigsaw puzzles is robust to noise and incorrect measurements. By incorrect measurements we mean that there are mistakes in estimating the connection graph. The three puzzles in Figure 3.2 exemplify cases where incorrect measurements are expected due to indistinguishability of some patches or low-resolution of patches.

For a given two-dimensional square jigsaw puzzle, let $G_{\text{true}} = (V, E_{\text{true}}, \boldsymbol{W}_{\text{true}}, \boldsymbol{R}_{\text{true}})$ denote the true connection graph. Note that the graph (V, E_{true}) is a grid, the true affinity function $\boldsymbol{W}_{\text{true}}$ is defined by

$$\boldsymbol{W}_{\text{true}}(i,j) = \begin{cases} 1, \text{ if } \{i,j\} \in E_{\text{true}}; \\ 0, \text{ otherwise,} \end{cases} \tag{3.7}$$

and the true connection function $\boldsymbol{R}_{\text{true}}$ is defined by

$$\boldsymbol{R}_{\text{true}}(i,j) = \begin{cases} \boldsymbol{R}_i \boldsymbol{R}_j^T, \text{ if } \{i,j\} \in E_{\text{true}}; \\ \boldsymbol{0}, \text{ otherwise,} \end{cases} \tag{3.8}$$

where $\boldsymbol{R}_1, \ldots, \boldsymbol{R}_n$ are the rotation matrices of the rotations $R_1, \ldots, R_n$ defined in §3.2.2. Let $G_{\text{est}} = (V, E_{\text{est}}, \boldsymbol{W}_{\text{est}}, \boldsymbol{R}_{\text{est}})$ denote the estimated connection graph. At last, denote by $\boldsymbol{C}_{\text{est}}$ and $\boldsymbol{C}_{\text{true}}$ the CGL matrices corresponding to G_{est} and G_{true} respectively.

The following lemma shows that if the estimated connection graph is a good approximation of the true connection graph, then the estimated CGL matrix is a good approximation of the true CGL matrix. It is analogous to Lemmas 2.1 and 2.2 of Karoui and Wu [62], but assumes a different noise model. In fact, its proof is parallel to the proofs of the latter lemmas and is thus omitted here.

Lemma 6. *Suppose that G_{true} and G_{est} are the true and estimated connection graphs respectively. Assume that there exist $\epsilon > 0$ and $f_1, \ldots, f_n > 0$ such that*

$$\sup_{1 \le i,j \le n} \left| \boldsymbol{W}_{\text{true}}(i,j) - \frac{\boldsymbol{W}_{\text{est}}(i,j)}{f_i} \right| < \epsilon \tag{3.9}$$

and there exists $\gamma > \epsilon$ such that $\inf_i \sum_{j \neq i} w_{ij}/n > \gamma$. Assume further that there exists a set $E' \in E_{\text{true}} \cap E_{\text{est}}$ such that (V, E') is a connected graph and such that

$$\boldsymbol{R}_{\text{est}}(i,j) = \begin{cases} \boldsymbol{R}_{\text{true}}(i,j), & \text{if } \{i,j\} \in E' \\ \text{arbitrary element of } \mathbb{Z}_4, & \text{otherwise.} \end{cases}$$

Then

$$\|\boldsymbol{C}_{\text{true}} - \boldsymbol{C}_{\text{est}}\|_2 \le \frac{2\epsilon}{\gamma} + \frac{4\epsilon}{\gamma(\gamma - \epsilon)}. \tag{3.10}$$

Note that if two patches are wrongly connected as neighbors in the estimated graph, then (3.9) enforces their affinity function to be small. Also note that each of f_i cannot be too large, otherwise (3.9) cannot be satisfied when two patches are neighbors.

Recall that according to [60] the top 2 eigenvectors of the true CGL matrix $\boldsymbol{C}_{\text{true}}$ recovers the global orientations of puzzle patches up to global rotation. Thus, if the top 2 eigenvectors of the estimated CGL matrix approximate well the top 2 eigenvectors of the true CGL matrix, they would recover the global orientations of puzzle patches. The Davis-Kahan $\sin\Theta$ Theorem [1, 65] guarantees such a good approximation when (3.10) holds for a small enough $\epsilon > 0$. Indeed, the matrix $\boldsymbol{V}_{\text{true}}$ of top 2 column eigenvectors of $\boldsymbol{C}_{\text{true}}$ and the matrix $\boldsymbol{V}_{\text{est}}$ of top 2 column eigenvectors of $\boldsymbol{C}_{\text{est}}$ satisfy

$$\|\sin\Theta(\boldsymbol{V}_{\text{true}}, \boldsymbol{V}_{\text{est}})\|_2 \le \frac{\|\boldsymbol{C}_{\text{true}} - \boldsymbol{C}_{\text{est}}\|_2}{n - \lambda_3(C_{\text{est}})}, \tag{3.11}$$

where $\lambda_3(\boldsymbol{C}_{\text{est}})$ is the third largest eigenvalue of C_{est}. To bound $\lambda_3(\boldsymbol{C}_{\text{est}})$, we first note that $\lambda_3(\boldsymbol{C}_{\text{true}}) = 0$ since $\boldsymbol{C}_{\text{true}}$ is a positive semidefinite matrix of rank 2. Furthermore, by Weyl's inequality [66],

$$|\lambda_3(\boldsymbol{C}_{\text{est}})| = |\lambda_3(\boldsymbol{C}_{\text{true}}) - \lambda_3(\boldsymbol{C}_{\text{est}})| \le \|\boldsymbol{C}_{\text{true}} - \boldsymbol{C}_{\text{est}}\|_2. \tag{3.12}$$

Combining (3.11) and (3.12) yields

$$\|\sin\Theta(\boldsymbol{V}_{\text{true}}, \boldsymbol{V}_{\text{est}})\|_2 \leq \frac{\|\boldsymbol{C}_{\text{true}} - \boldsymbol{C}_{\text{est}}\|_2}{n - \|\boldsymbol{C}_{\text{true}} - \boldsymbol{C}_{\text{est}}\|_2}, \tag{3.13}$$

We thus conclude by (3.10) and (3.13) that if ϵ is sufficiently small then $\boldsymbol{V}_{\text{est}}$ closely approximates $\boldsymbol{V}_{\text{true}}$. We remark that this analysis generalizes to other puzzles and not just the two-dimensional square jigsaw puzzle.

3.4 Connection Graph Construction For Type 2 and Type 3 puzzles

As we have discussed in §3.3.1, if we are given a perfect metric, we can easily construct the connection graph. However, there is no perfect metric that would work for all images. For example, if part of the image contains a region with a uniform color, such as sky or ocean (see the images on the second row of Figure 3.2), the metric between the edges of the image patches from this region will be close to zero. Thus, all these patches should be wrongly identified by a perfect metric as neighbors. Furthermore, if all edges of patches are similar to each others, then the patches are indistinguishable and the problem is ill-posed. Therefore, the idea of finding a perfect metric and using a threshold to identify neighbors may not lead to a correct affinity graph. Instead, we suggest to iteratively update the graph construction, while identifying possibly incorrect edges and reassigning zero or small affinities to them. The rest of this section is organized as follows: §3.4.1 reviews the Mahalanobis Gradient Compatibility (MGC) metric that is used for the proposed graph construction; §3.4.3 describes a construction of the connection graph for type 2 puzzles; and §3.4.2 proposes a construction of the connection graph for type 3 puzzles.

3.4.1 Approximate a Perfect Metric Between Image Patches

To automatically assemble a jigsaw puzzle, no matter what algorithm is used, one needs to have a measure that can indicate whether two patches are neighbors or not. As we can see in the first row of Figure 3.2, it can be challenging to compare patches. We recall that the discrete values of the digital image at two sides of an edge between two

patches are not the same. The right image of the first row of Figure 3.2 shows two neighboring patches at high resolution, where the difference between the image values at the two sides of the edge (left and right) is noticeable. On the other hand, in the left image of the first row of Figure 3.2, this difference is hard to notice in the printed resolution. Nevertheless, we emphasize here the existing difference of numerical values at two sides of edges, which is challenging for any algorithm that needs to align patches.

In this work, we align patches by using the MGC metric, which was proposed in [50]. It is based on two main ideas. The first idea is that the derivatives of RGB values in the perpendicular direction to the edge are similar in both sides of that edge. The second idea is that these values can be compared by using the covariance between the color channels and the corresponding Mahalanobis distance.

To review Gallagher's precise definition, we assume two neighboring image patches P_i and P_j of size $s \times s$. There are four different relative positions of P_i and P_j, namely, left-right, right-left, top-bottom and bottom-top, and the computation needs to adapt to each case. We assume without loss of generality we the left-right relative position, that is, P_i on the left and P_j on the right, and compute the corresponding MGC, which we denote by $\mathrm{MGC}_{\mathrm{lr}}(P_i, P_j)$, as follows. For each color channel c (red, green and blue), each row r, $1 \leq r \leq s$ of the patch P_i, or equivalently P_j, and the last column s of the left patch P_i we find the derivatives near the right edge of the image patch P_i in the direction left-right as follows:

$$G_{iL}(r, c) = P_i(r, s, c) - P_i(r, s-1, c).$$

The subscript L in the above equation indicates that patch P_i is on the left side of the patch P_j. Note that the matrix G_{iL} is in $\mathbb{R}^{s\times 3}$ and can be singular. Gallagher [50] suggests regularizing it by adding the following 9 additional rows $(0, 0, 0)$, $(1, 1, 1)$, $(-1, -1, -1)$, $(0, 0, 1)$, $(0, 1, 0)$, $(1, 0, 0)$, $(-1, 0, 0)$, $(0, -1, 0)$ and $(0, 0, -1)$. The resulting regularized matrix in $\mathbb{R}^{(s+9)\times 3}$ is denoted by $\tilde{\boldsymbol{G}}_{iL}$.

Next, for each color channel c we define the mean distribution for those derivatives on the right side of the $s \times s$ patch P_i as

$$\boldsymbol{\mu}_{iL}(c) = \frac{1}{s} \sum_{r=1}^{s} \boldsymbol{G}_{iL}(r, c).$$

The regularized covariance matrix $\Sigma_{iL} \in \mathbb{R}^{3\times 3}$ between color channels is

$$\boldsymbol{\Sigma}_{iL} = \frac{1}{s+8}(\tilde{\boldsymbol{G}}_{iL} - \text{mean}(\tilde{\boldsymbol{G}}_{iL}))^T(\tilde{\boldsymbol{G}}_{iL} - \text{mean}(\tilde{\boldsymbol{G}}_{iL})),$$

where

$$\text{mean}(\tilde{\boldsymbol{G}}_{iL}) = \sum_{r=1}^{s+9} \tilde{\boldsymbol{G}}_{iL}(r,c)/(s+9) = \sum_{r=1}^{s} \boldsymbol{G}_{iL}(r,c)/(s+9).$$

We also define $G_{ij\text{LR}}(p)$, the derivative from the left $s \times s$ image patch P_i to the right $s \times s$ image patch P_j at row r and color c, by

$$\boldsymbol{G}_{ij\text{LR}}(r,c) = P_j(r,1,c) - P_i(r,s,c).$$

The left-to-right compatibility measure from P_i to P_j is defined by

$$D_{\text{LR}}(P_i,P_j) = \sum_{r=1}^{s}(\boldsymbol{G}_{ij\text{LR}}(r) - \boldsymbol{\mu}_{iL})\boldsymbol{\Sigma}_{iL}^{-1}(\boldsymbol{G}_{ij\text{LR}}(r) - \boldsymbol{\mu}_{iL})^T.$$

Similarly, one can defined the right-to-left compatibility measure from P_j to P_i in the same left-right setting, where P_i is to the left of P_j. The left-right MGC metric then has the symmetrized form

$$\text{MGC}_{\text{lr}}(P_i,P_j) = D_{\text{LR}}(P_i,P_j) + D_{\text{RL}}(P_j,P_i). \tag{3.14}$$

The right-left, top-bottom and bottom-top MGC's, denoted by $\text{MGC}_{\text{rl}}(P_i,P_j)$, $\text{MGC}_{\text{tb}}(P_i,P_j)$ and $\text{MGC}_{\text{bt}}(P_i,P_j)$ respectively, are similarly computed.

3.4.2 Connection Graph Construction for Type 3 Puzzles

For type 3 puzzles, the locations of patches are given. Furthermore, edges are drawn between neighboring patches. The affinity function is set by $\boldsymbol{W}(i,j) = 1$ for all $\{i,j\} \in E$. One need only find the unknown orientations, that is, the unknown connection matrix $\boldsymbol{R}$.

To construct the connection function we propose to use the MGC metric, described in §3.4.1. For all neighboring patches P_i and P_j, we calculate the possible 16 values of the MGC metric (for all possible 16 relative positions) and select the smallest of these

numbers and its corresponding rotation $\boldsymbol{R}(i,j)$. If there is no unique minimum among these 16 values we suggest assigning $\boldsymbol{W}(i,j) = 1/2$ (or another value smaller than 1) and letting $\boldsymbol{R}_{i,j}$ be the mean of the candidate rotations that obtain the minimal value.

3.4.3 Connection Graph Construction for Type 2 Puzzles

We propose the following step-by-step procedure for constructing the affinity graph, the affinity function and the connection function for type 2 puzzles and then summarize this procedure in Algorithm 5.

Initial Step

We start with an initial construction of the directed graph $G = (V, E_{\text{est}})$. The vertex set V contain the patches in $\mathcal{Q}$. The edge set E_{est} is updated by the following procedure. In order to describe it, we denote by $\boldsymbol{R} \cdot P$ the action of the rotation $\boldsymbol{R} \in \mathbb{Z}_4$ on the patch P. For a patch P_i, we find the patches $P_{i_{\text{t}}}, P_{i_{\text{l}}}, P_{i_{\text{b}}}, P_{i_{\text{r}}}$ and the corresponding rotations $\boldsymbol{R}(i,i_{\text{t}}), \boldsymbol{R}(i,i_{\text{l}}), \boldsymbol{R}(i,i_{\text{r}}), \boldsymbol{R}(i,i_{\text{b}}) \in \mathbb{Z}_4$ such that

$$\begin{aligned}
\{P_{i_{\text{t}}}, \boldsymbol{R}(i,i_t)\} &\in \arg\min_{P\in\mathcal{Q},\boldsymbol{R}\in\mathbb{Z}_4} \text{MGC}_{\text{bt}}(P_i, \boldsymbol{R}\cdot P),\\
\{P_{i_{\text{l}}}, \boldsymbol{R}(i,i_{\text{l}})\} &\in \arg\min_{P\in\mathcal{Q},\boldsymbol{R}\in\mathbb{Z}_4} \text{MGC}_{\text{rl}}(P_i, \boldsymbol{R}\cdot P),\\
\{P_{i_{\text{b}}}, \boldsymbol{R}(i,i_{\text{b}})\} &\in \arg\min_{P\in\mathcal{Q},\boldsymbol{R}\in\mathbb{Z}_4} \text{MGC}_{\text{tb}}(P_i, \boldsymbol{R}\cdot P),\\
\{P_{i_{\text{r}}}, \boldsymbol{R}(i,i_{\text{r}})\} &\in \arg\min_{P\in\mathcal{Q},\boldsymbol{R}\in\mathbb{Z}_4} \text{MGC}_{\text{lr}}(P_i, \boldsymbol{R}\cdot P).
\end{aligned} \tag{3.15}$$

The set E_{est} of directed edges contains the edges that connect each vertex with index i, $1 \le i \le n$, to the vertices with indices i_{t}, i_{l}, i_{b} and i_{r}. Note that these indices solving (3.15) may not be unique and we consider all solutions of (3.15) when forming E_{est}.

We next modify the directed graph $G = (V, E_{\text{est}})$ into an undirected graph. We fix two values of weights: $w_1 = 1$ and $w_2 = 0.01$, and define an initial affinity function $\boldsymbol{W}_{\text{init}}$ by setting $\boldsymbol{W}_{\text{init}}(i,j) = \boldsymbol{W}_{\text{init}}(j,i) = w_1$ if both (i,j) and $(j,i) \in E_{\text{est}}$ and $\boldsymbol{W}_{\text{init}}(i,j) = \boldsymbol{W}_{\text{init}}(j,i) = w_2$ if only one of (i,j) or (j,i) is in E_{est}. Figure 3.3 demonstrates this construction for a fixed patch.

Next, we enforce the constraint that for two-dimensional square jigsaw puzzles each patch can have at most one neighbor for each direction and trim some edges that are likely not neighbors. This is done as follows. Assume without loss of generality that

patch P_i has more than one neighbor in the top direction and denote these neighbors by $P_{i_1}, \ldots, P_{i_k}$ where $k > 1$. Then we solve the minimization problem

$$j \in \arg\min_{1 \le j \le k} \mathrm{MGC}_{\mathrm{bt}}(P_i, P_{i_j}). \tag{3.16}$$

If (3.16) has a unique solution, we keep the edge $\{i, i_j\}$ and remove rest of the edges. Otherwise, we remove all edges $\{i, i_j\}_{j=1}^{k}$ from E. The procedure is analogous if P_i has more than one neighbors from left, bottom or right. If edges were eliminated from E_{est}, then the matrix $\boldsymbol{W}_{\mathrm{init}}$ is updated so it is zero on the corresponding indices. This process results in the following initial connection graph $G = (V, E_{\mathrm{est}}, \boldsymbol{W}_{\mathrm{init}}, \boldsymbol{R})$.

The construction of this graph uses a nearest-neighbor construction. For high-noise regime, Karoui and Wu [62] recommend avoiding a nearest-neighbor construction. However, due to the special lattice structure of the true graph, the nearest-neighbor initial construction is natural for the two-dimensional square jigsaw puzzle problem.

Use of Jaccard Index to refine the graph

Next, we refine the connection graph by trying to assess the validity of the edges and decrease the weights of edges that do not seem valid, that is, they may not appear in the true connection graph. The idea is to check whether after removing an edge, its neighbors are still connected in some weak sense to each others. If so, then the edge seems to be valid and otherwise, it may not be valid. For this purpose, we use the Jaccard index [67].

The description of this index uses the following notation in a graph $G = (V, E)$. Given a vertex i, $1 \le i \le |V|$, let $N^1_{G,i}$ denote the set of vertices in V which are connected to vertex i, that is, $N^1_{G,i} = \{j \in V | \{i, j\} \in E\}$. Using our terminology, $N^1_{G,i}$ contains the neighbors of i. The set $N^2_{G,i}$ contains all vertices, but vertex i, which are at most 2 steps away from i. That is, $N^2_{G,i} = \bigcup_{j \in N^1_{G,i}} N^1_{G,j} \setminus \{i\}$. At last, denote $G^{\setminus(i,j)} = (V, E \setminus (i, j))$. The set $N^1_{G,i}$ and $N^2_{G,i}$ are demonstrated in Figure 3.4.

By using this notation, we define the Jaccard index between vertices i and j as

$$\mu_{\mathrm{Jaccard}}(i, j) = |N^2_{G^{\setminus(i,j)},i} \cap N^2_{G^{\setminus(i,j)},j}|. \tag{3.17}$$

This definition is similar to the one in [67]. The difference is that we consider the

graph $G^{\setminus(i,j)}$ instead of G and we do not divide by $|N^2_{G^{\setminus(i,j)},i} \cup N^2_{G^{\setminus(i,j)},j}|$. The latter division does not matter to us as we only care about the positivity of this index. Figure 3.5 demonstrates calculation of the Jaccard index for a special example. Note that the chance of two vertices i and j to be neighbors in the graph (V, E_{est}) is higher if $\mu_{\text{Jaccard}}(i,j) > 0$ than $\mu_{\text{Jaccard}}(i,j) = 0$. Thus, we propose to use the Jaccard indices to refine the connection graph. We use another weight matrix $\boldsymbol{W}_{\text{Jaccard}} \in \mathbb{R}^{n\times n}$. If $\{i,j\} \in E_{\text{est}}$ and $\mu_{\text{Jaccard}}(i,j) = 0$, then $\boldsymbol{W}_{\text{Jaccard}}(i,j) = \boldsymbol{W}_{\text{Jaccard}}(j,i) = 0$. If on the other hand $\mu_{\text{Jaccard}}(i,j) > 0$, then $\boldsymbol{W}_{\text{Jaccard}}(i,j) = \boldsymbol{W}_{\text{Jaccard}}(j,i) = \boldsymbol{W}_{\text{init}}(i,j)$.

Since this procedure might also remove a lot of correct edges by zeroing out the corresponding values of the affinity function, we propose using an affinity function, which is a linear combination of $\boldsymbol{W}_{\text{init}}$ and $\boldsymbol{W}_{\text{Jaccard}}$ with larger coefficient given to $\boldsymbol{W}_{\text{Jaccard}}$. In our experiments we set $\boldsymbol{W}_{\text{nb}} = 0.2 \times \boldsymbol{W}_{\text{init}} + 0.8 \times \boldsymbol{W}_{\text{Jaccard}}$ and use the following affinity graph $G = (V, E, \boldsymbol{W}_{\text{nb}}, R)$.

Making the Affinity Graph Connected

The procedures described in §3.4.3 and §3.4.3 might result in a disconnected affinity graph G as demonstrated in the left image of Figure 3.6. To complete G so it is connected, we first find all connected components of G. Assume that they are k connected components with corresponding vertices $V_1, \ldots, V_k$ that partition the set of vertices V. Assume further that they are labeled by descending size order, i.e. $|V_1| \geq |V_2| \cdots \geq |V_k|$. Next, we find vertices $i \in V_1$ and $j \in V \setminus V_1$, that minimize the MGC metric between the patches P_i and P_j. Mathematically, we find

$$\{i, j, \boldsymbol{R}\} \in \arg\min_{i\in V\setminus V_1, j\in V_1, \boldsymbol{R}\in\mathbb{Z}_4} \min\{\text{MGC}_{\text{lr}}(P_i, \boldsymbol{R}\cdot P_j)), \text{MGC}_{\text{tb}}(P_i, \boldsymbol{R}\cdot P_j)), \\ \text{MGC}_{\text{rl}}(P_i, \boldsymbol{R}\cdot P_j)), \text{MGC}_{\text{bt}}(P_i, \boldsymbol{R}\cdot P_j))\}. \tag{3.18}$$

If the solution of (3.18) is not unique, we randomly choose one solution. We then add the edge $\{i,j\}$ of the chosen solution to E_{est} and update the weight as follows: $\boldsymbol{W}(i,j) = \boldsymbol{W}(j,i) = w_3$, where $w_3 = w_2/2 = 0.005$ and also set $\boldsymbol{R}(i,j) = \boldsymbol{R}$ and $\boldsymbol{R}(j,i) = \boldsymbol{R}^T$. We iterate the procedure described above until the graph becomes connected. The number of iterations needed are $k-1$ since there are k connected components and at iteration

we connect the largest component with a remaining component.

Taking Advantage of 4-Loops

We refine the constructed connection graph by using the following property of the two-dimensional square jigsaw puzzle: If two patches P_i and P_j are diagonal neighbors, then there exists exactly two other patches P_{n_1} and P_{n_2} and a cycle containing the vertices i, n_1, j and n_2, this idea is demonstrated in Figure 3.7. Such a cycle of 4 vertices is referred to as a 4-loop by [53]. In this latter work, 4-loops were used to solve the puzzle problem. We use them to define a better connection graph. As we have already discussed for the true grid, each patch can have at most 4 direct neighbors (right, top, left or bottom). Furthermore, each patch has at most 4 diagonal neighbors. Exactly four diagonal neighbors are obtained for patches in the interior of the puzzle, a single diagonal neighbor occurs for a corner patch and there are 2 diagonal neighbors when the patch lies on the boundary of the grid but not on a corner.

For patches P_i and P_j we define

$$\delta_{\text{diag}}(i,j) = |N^1_{G,i} \cap N^1_{G,j}|. \tag{3.19}$$

We observe that patches P_i and P_j are diagonal neighbors in the true grid if and only if $\delta_{\text{diag}}(i,j) = 2$. To find the diagonal neighbors for graph $G = (V, E)$ we propose a two step procedure. First, we find the set of all pairs of vertices $\{i,j\} \in V \times V$ for which $\delta_{\text{diag}}(i,j) = 2$. For each such pair $\{i,j\}$ there exists another pair $\{n_1, n_2\}$ such that

$$N^1_{G,i} \cap N^1_{G,j} = \{n_1, n_2\}, \tag{3.20}$$

or equivalently, i, n_1, j and n_2 are contained in a 4-loop. We set

$$\boldsymbol{W}_{\text{diag}}(i,j) = \begin{cases} 1, & \text{when } \delta_{\text{diag}}(i,j) = 2 \text{ and } \boldsymbol{R}(i,n_1)\boldsymbol{R}(n_1,j) = \boldsymbol{R}(i,n_2)\boldsymbol{R}(n_2,j); \\ 0, & \text{otherwise.} \end{cases} \tag{3.21}$$

The condition $\boldsymbol{R}(i,n_1)\boldsymbol{R}(n_1,j) = \boldsymbol{R}(i,n_2)\boldsymbol{R}(n_2,j)$ in (3.21) is explained below after the whole procedure is clarified. We further update blocks of the matrix $\boldsymbol{W}_{\text{nb}}$ as follows,

where we denote by $\boldsymbol{W}_{\mathrm{nb}}([i,j],[n_1,n_2])$ the 2×2 block of $\boldsymbol{W}_{\mathrm{nb}}$ with indices (i,n_1), (i,n_2), (j,n_1) and (j,n_2),

$$\boldsymbol{W}_{\mathrm{nb}}([i,j],[n_1,n_2]) = \begin{cases} \frac{\boldsymbol{W}_{\mathrm{nb}}([i,j],[n_1,n_2])}{3}, & \text{if } \delta_{\mathrm{diag}}(i,j)=2 \text{ and} \\ & \boldsymbol{R}(i,n_1)\boldsymbol{R}(n_1,j) \neq \boldsymbol{R}(i,n_2)\boldsymbol{R}(n_2,j); \\ \mathbf{1}_{2\times 2}, & \text{if } \delta_{\mathrm{diag}}(i,j)=2 \text{ and} \\ & \boldsymbol{R}(i,n_1)\boldsymbol{R}(n_1,j) = \boldsymbol{R}(i,n_2)\boldsymbol{R}(n_2,j); \\ \frac{2\boldsymbol{W}_{\mathrm{nb}}([i,j],[n_1,n_2])}{3}, & \text{otherwise.} \end{cases} \tag{3.22}$$

We note that the support sets of $\boldsymbol{W}_{\mathrm{nb}}$ and $\boldsymbol{W}_{\mathrm{diag}}$ are disjoint. We set $\boldsymbol{W} = \boldsymbol{W}_{\mathrm{nb}} + \boldsymbol{W}_{\mathrm{diag}}$ and this is the final set of constructing the connection graph $G = (V, E_{\mathrm{est}}, \boldsymbol{W}, \boldsymbol{R})$ for two-dimensional square jigsaw puzzles. The full algorithm of this construction is summarized in Algorithm 5.

The condition for rotations in (3.21) and (3.22), that is, $\boldsymbol{R}(i,n_1)\boldsymbol{R}(n_1,j) = \boldsymbol{R}(i,n_2)\boldsymbol{R}(n_2,j)$, is naturally satisfied in a true graph as demonstrated in Figure 3.8. Therefore, when it is satisfied and also $\delta_{\mathrm{diag}}(i,j)=2$, the maximal weight of 1 is assigned to the corresponding diagonal edge. Equation (3.22), on the other hand, considers the case where this condition is violated, but $\delta_{\mathrm{diag}}(i,j)=2$. In this case, there is an evidence for a mismatch between puzzle pieces and therefore the weight is reduced by a factor of 2 so that the diagonal edge is less valid.

3.5 Estimation of Locations for Type 2 puzzles and Update of the Connection Graph

This section completes the solution of type 2 jigsaw puzzles by estimation of the correct locations of the patches. This is done after forming the connection graph according to Algorithm 5 and estimating the correct orientations by Algorithm 4. The section is organized as follows. In §3.5.1 we review the vector diffusion map and distance and explain how to use them for updating the MGC metric. The updated metric is later

Algorithm 5 Connection Graph Construction for Type 2 puzzles

Input: Puzzle Patches: $\{P_i\}_{i=1}^{n} \subset \mathbb{R}^{p\times p\times 3}$

- For all $1 \leq i < j \leq n$ calculate the 16 MGC metric values between patches P_i and P_j as explained in §3.4.1
- Construct $G = (V, E_{\text{est}}, \boldsymbol{W}_{\text{init}}, \boldsymbol{R})$ according to the procedure described in §3.4.3 with the following three stages: nearest-neighbors construction based on (3.15), symmetrization of $\boldsymbol{W}_{\text{init}}$ and pruning extra neighbors with the use of (3.16)
- For all $\{i, j\} \in E_{\text{est}}$, calculate $\mu_{\text{Jaccard}}(i, j)$ according to (3.17)
- For all $\{i, j\} \in E_{\text{est}}$, if $\mu_{\text{Jaccard}}(i, j) = 0$, set $\boldsymbol{W}_{\text{Jaccard}}(i, j) = 0$; otherwise, $\boldsymbol{W}_{\text{Jaccard}}(i, j) = 1$
- Set $\boldsymbol{W}_{\text{nb}} = 0.8 \times \boldsymbol{W}_{\text{Jaccard}} + 0.2 \times \boldsymbol{W}_{\text{init}}$
- If the graph G is disconnected, iteratively connect the largest connected component to smaller connected components as explained in §3.4.3
- For all $i,\ j \in V$, calculate $\delta_{\text{diag}}(i, j)$ according to (3.19) and if $\delta_{\text{diag}}(i, j) = 2$, calculate n_1 and n_2 according to (3.20)
- Form $\boldsymbol{W}_{\text{diag}}$ according to (3.21) and update $\boldsymbol{W}_{\text{nb}}$ according to (3.22)
- Set $\boldsymbol{W} = \boldsymbol{W}_{\text{nb}} + \boldsymbol{W}_{\text{diag}}$

Return: $G = (V, E_{\text{est}}, \boldsymbol{W}, \boldsymbol{R})$, MGC values for all pairs of patches

used for better estimation of locations. In §3.5.2 we discuss the problem of recovering the location of patches. We propose a mathematical idea for solving the problem by applying a quadratic assignment formulation with respect to the affinity function $\boldsymbol{W}$ defined in §3.4.3. However, the solver of the combinatorial optimization problem is not sufficiently fast and accurate. We thus recommend applying some other existing algorithms for finding locations of patches. In §3.5.2 we propose an idea for recovering the location of patches by applying a quadratic assignment problem with respect to the affinity function $\boldsymbol{W}$ defined in §3.4.3. In §3.5.3 we propose a procedure for updating the affinity function and the connection function based on the estimated rotations and locations. At last, §3.5.4 summarizes the complete algorithm to solve type 2 puzzles.

We remark that the main contribution of the work to type 2 puzzles is in the application of Algorithm 5 and Algorithm 4, which were already described in earlier sections. As explained below, the practical code does not use the new ideas in §3.5.1 and §3.5.2, but follows a previous procedure for estimating locations, which is mentioned in §3.5.2. Nevertheless, our numerical experiments indicate that the procedure in §3.5.3 improves the performance of the whole algorithm.

3.5.1 Updating the Metric between Puzzle Pieces by Vector Diffusion Distances

The MGC metric defined in §3.4.1 is not a perfect metric, but it provides some information whether two patches are neighbors or not. However, if two patches are not neighbors, the MGC metric between them does not provide any information about their distance in the image. Such information can be helpful since the estimated information on neighboring patches can be wrong. For this purpose, we suggest updating the MGC metric by considering the diffusion process associated with the random walk determined by $\boldsymbol{C}$. The diffusion vector framework for doing this was suggested in [60]. This part is performed after the rotations of the patches were estimated according to Algorithm 5.

For $t > 0$, the *vector diffusion map* (VDM) [60] in our setting is a function $V_{t,n} : \mathcal{Q} \to \mathbb{R}^{2n \times 2n}$ defined by

$$V_{t,n} : P_i \mapsto \left((\mu_{\boldsymbol{C},l}\mu_{\boldsymbol{C},r})^t \langle v_{\boldsymbol{C},l}[i], v_{\boldsymbol{C},r}[i] \rangle \right)_{l,r=1}^{2n},$$

where $\mu_{\boldsymbol{C},l}$ and $v_{\boldsymbol{C},l}$ are the l-th eigenvalue and eigenvector, respectively, of $\boldsymbol{C}$, and $v_{\boldsymbol{C},l}[i]$ is a 2-dim vector containing the $(2(i-1)+1)$-th and $(2i)$-th entries of $v_{\boldsymbol{C},l}$. The *vector diffusion distance* (VDD) [60] between two patches indexed by i and j is

$$d_{\boldsymbol{C},t,n}(i,j) := \|V_{t,n}(P_i) - V_{t,n}(P_j)\|^2. \tag{3.23}$$

This distance converts the local information into a global information and provides an estimate of the distance between patches in the original grid. Based on this distance one can infer whether two patches are close to each other in the original image or far away. As demonstrated in Figure 3.9, this distance is not sufficiently accurate to infer nearness when patches have comparable distances. In particular, it cannot be used to infer whether two patches are neighbors or diagonal neighbors.

Nevertheless, it is still possible to use the VDD for improving the MGC metric in the following way. We first compute the VDD between all patches. For each image patch $P_i \in \mathcal{Q}$, we sort the VDD distances of all other patches to P_i and record the patches in the 0.1-quantize of largest distances. The MGC metric between these patches and P_i is then increased by the factor $\alpha = 2$. This ensures that patches that are not likely neighbors of P_i are penalized by a larger distance and thus have a smaller chance of becoming neighbors in the final solution.

Our numerical tests did not indicate any significant improvement when using this procedure. In order to reduce the computational time of the algorithm, we do not apply it in practice and mention it as an optional step for the whole algorithm.

3.5.2 Possible Estimation of Locations by Quadratic Assignment Problem

As we have already mentioned, the main contribution of this work is to introduce a new approach for the recovery of the unknown orientations of patches in type 2 puzzles by using the CGL. After one recovers the unknown orientations, the problem reduces to solving a type 1 puzzle. According to our numerical tests the following algorithms for type 1 puzzles are highly competitive: Gallagher [50] and Yu et al. [59]. We have often noticed a slight advantage of the latter algorithm, which applies a linear programming procedure. Therefore, in the experiments reported in this chapter we use the algorithm

of Yu et al. [59]. However, one could use instead any algorithm that solves type 1 puzzles.

Nevertheless, one can try to take advantage of the affinity function $\boldsymbol{W}_{\text{est}}$, whose construction is described in §3.4.3, and the fact that for two-dimensional square jigsaw puzzles the true affinity function $\boldsymbol{W}_{\text{true}}$ is known. Therefore, one may try to match $\boldsymbol{W}_{\text{est}}$ with $\boldsymbol{W}_{\text{true}}$. This gives rise to the problem of finding a permutation matrix $\boldsymbol{P}$, which corresponds to the permutation σ described in §3.4.3, such that $\boldsymbol{W}_{\text{est}}$ and $\boldsymbol{P}^T\boldsymbol{W}_{\text{true}}\boldsymbol{P}$ match. The desired permutation can be expressed as the solution of the following optimization problem:

$$\arg\min_{\boldsymbol{P}\in\text{Perm}(n)} \|\boldsymbol{W}_{\text{est}} - \boldsymbol{P}^T\boldsymbol{W}_{\text{true}}\boldsymbol{P}\|_2^2. \tag{3.24}$$

Note that (3.24) is the Quadratic Assignment Problem (QAP) for the matrices $\boldsymbol{W}_{\text{true}}$ and $\boldsymbol{W}_{\text{est}}$. However, existing solvers are slow as the number of patches increase and thus we are not sure how to make this procedure practical for large puzzles. A similar idea has been proposed by Andalo et al. [52] for solving type 1 puzzles. They suggest solving a QAP with different weight matrices by using constrained gradient descent. It is unclear to us if their procedure is applicable to the QAP problem in (3.24).

3.5.3 Updating the Affinity Function and the Connection Function

After estimating the orientations and locations of the puzzle pieces, we recommend updating the values of the affinity and connection functions and estimate again the orientations and locations. This procedure can be iterated several times. The idea is that given the estimated puzzle solution, one can infer possible mismatches. These identified mismatches could be used to reassign values for the affinity and connection functions that may lead to a more accurate solution.

First, we figure out which patches are wrongly placed in the assembled puzzle and remove them from the grid. For this purpose we use the following kinds of metrics, which we refer to as NAM (Neighbor-Averaged Metric). For each patch i, $1 \leq i \leq n$, with neighbors i_{t}, i_{l}, i_{b} and i_{r}, from top, left, bottom and right respectively, we define

$$\text{NAM}_{\text{all}}(i) = (\text{MGC}_{\text{bt}}(i, i_{\text{t}}) + \text{MGC}_{\text{rl}}(i, i_{\text{l}}) + \text{MGC}_{\text{tb}}(i, i_{\text{b}}) + \text{MGC}_{\text{lr}}(i, i_{\text{r}}))/4. \tag{3.25}$$

If a patch i is at the edge or corner of the puzzle grid, then it has 3 or 2 neighbors respectively. In this case, we only sum up the respective MGC values and divide the sum by the number of neighbors. Similarly we define the following four metrics:

$$\begin{aligned}
\mathrm{NAM}_{\mathrm{ltr}}(i) &= (\mathrm{MGC}_{\mathrm{rl}}(i, i_{\mathrm{l}}) + \mathrm{MGC}_{\mathrm{bt}}(i, i_{\mathrm{t}}) + \mathrm{MGC}_{\mathrm{lr}}(i, i_{\mathrm{r}}))/3, \\
\mathrm{NAM}_{\mathrm{trb}}(i) &= (\mathrm{MGC}_{\mathrm{bt}}(i, i_{\mathrm{t}}) + \mathrm{MGC}_{\mathrm{lr}}(i, i_{\mathrm{r}}) + \mathrm{MGC}_{\mathrm{tb}}(i, i_{\mathrm{b}}))/3, \\
\mathrm{NAM}_{\mathrm{blt}}(i) &= (\mathrm{MGC}_{\mathrm{tb}}(i, i_{\mathrm{b}}) + \mathrm{MGC}_{\mathrm{rl}}(i, i_{\mathrm{l}}) + \mathrm{MGC}_{\mathrm{bt}}(i, i_{\mathrm{t}}))/3, \\
\mathrm{NAM}_{\mathrm{lbr}}(i) &= (\mathrm{MGC}_{\mathrm{rl}}(i, i_{\mathrm{l}}) + \mathrm{MGC}_{\mathrm{tb}}(i, i_{\mathrm{b}}) + \mathrm{MGC}_{\mathrm{lr}}(i, i_{\mathrm{r}}))/3.
\end{aligned} \tag{3.26}$$

Again, if the patch is at the edge or corner of the puzzle grid, then we sum the appropriate MGC values and divide by the corresponding number of neighbors.

If the puzzle is correctly assembled, the NAM values of all patches are relatively small as demonstrated for $\mathrm{NAM}_{\mathrm{all}}$ values in last figure of the first row of Figure 3.10. Otherwise, if there are some wrongly placed patches, their corresponding NAM values should be relatively higher. This is demonstrated in first and second figures of the first row of Figure 3.10, where the spikes of $\mathrm{NAM}_{\mathrm{all}}$ values correspond to wrongly orientated or placed patches. Based on this observation, we suggest to find all patches, for which the corresponding $\mathrm{NAM}_{\mathrm{all}}$ value and at least one of the $\mathrm{NAM}_{\mathrm{ltr}}$, $\mathrm{NAM}_{\mathrm{trb}}$, $\mathrm{NAM}_{\mathrm{blt}}$ and $\mathrm{NAM}_{\mathrm{lbr}}$ values exceed 1.5 times the median of all corresponding NAM values. We remove the corresponding edges from the grid. For example, for $\mathrm{NAM}_{\mathrm{ltr}}$ we remove the edges connecting vertex i with its left, top and right neighbors. We refer to a location as empty if all edges connecting the patch in this location to its top, bottom, left and right neighbors were removed. Patches at empty locations at each iteration of this procedure are demonstrated in the second row of Figure 3.10. Their removal, which literally creates empty locations, is demonstrated in the last row of this figure.

Next, we identify all the empty locations for which at least 2 of 4 neighboring locations are not empty. We consider the neighboring locations in the puzzle grid, regardless of edges that were removed in the current process. For each fixed empty location, denote the set of neighboring patches (according to locations) by S_{nb}. Note that S_{nb} contains either 2, 3 or 4 indices of patches. We identify the empty location with the set S_{nb}. For the same empty location, find an oriented patch that minimizes the

averaged MGC metric with respect to the vertices in S_{nb} among all patches with non-empty locations and with their all four orientations. The corresponding minimal value of the averaged MGC metric for a specified empty location with neighboring patches S_{nb} is denoted by $\text{NAM}_{S_{\text{nb}}}$. Assuming the index of this latter patch is i, we denote its pairwise orientation with respect to patch $j \in S_{\text{nb}}$ by $\boldsymbol{R}_{i,j}$. Let $\text{med}(\text{NAM}_{\text{all}})$ denote the median value of all NAM_{all} values. We update the affinity and connection functions for i and $j \in S_{\text{nb}}$ as above by

$$\boldsymbol{W}_{\text{est}}(i,j) = \boldsymbol{W}_{\text{est}}(j,i) = \begin{cases} 0.6, & \text{if } \text{NAM}_{S_{\text{nb}}} < \text{med}(\text{NAM}_{\text{all}}); \\ 0.3, & \text{if } \text{med}(\text{NAM}_{\text{all}}) < \text{NAM}_{S_{\text{nb}}} < 2\text{med}(\text{NAM}_{\text{all}}) \end{cases} \tag{3.27}$$

and

$$\boldsymbol{R}_{\text{est}}(i,j) = \boldsymbol{R}_{i,j} \text{ and } \boldsymbol{R}_{\text{est}}(i,j) = \boldsymbol{R}_{i,j}^T \text{ if } \text{med}(\text{NAM}_{\text{all}}) < 2\text{med}(\text{NAM}_{\text{all}}). \tag{3.28}$$

Algorithm 6 summarizes this update procedure.

Most state-of-the-art methods use a greedy step to make final corrections to the solved puzzle. On the other hand, the step discussed here only updates the connection graph and is thus non-greedy. It is possible to incorporate greedy procedures that may improve the performance of our algorithm, however, we would like to show that a more principled method can be competitive.

3.5.4 Improvement and Final Solution for Type 2 Puzzles

As we have discussed in §3.3.3, if the constructed connection graph is good enough, the top 2 eigenvectors of the CGL matrix can recover the orientations of puzzle patches. However, when it is impossible to construct an accurate affinity graph (see, for example, Figure 3.2), one might consider top few eigenvectors, as they might also contain some useful information about the orientations of patches. For some puzzles and poorly-estimated connection graphs, the orientations recovered by the top 3-rd and 4-th eigenvectors are more accurate than the ones recovered by the top 2 eigenvectors. We thus suggest that in the initial step of solving the puzzle (before applying the updates

Algorithm 6 Updating the affinity function and the connection function at a given iteration

Input: MGC metric values between all patches, current solution to the puzzle problem

- Calculate the NAM values for all patches according to (3.25) and (3.26)
- Remove all edges from the solution grid for which the corresponding $\mathrm{NAM}_{\mathrm{all}}$ value and at least one of the $\mathrm{NAM}_{\mathrm{ltr}}$, $\mathrm{NAM}_{\mathrm{trb}}$, $\mathrm{NAM}_{\mathrm{blt}}$ and $\mathrm{NAM}_{\mathrm{lbr}}$ values exceed 1.5 times the median of all corresponding NAM values
- For any empty location (that is, for any location whose all edges were removed) which has at least two non-empty neighboring locations (according to the natural grid of locations), find the patch with the correct rotation which best fits in that position and update the affinity function and the connection function according to (3.27) and (3.28).

Return: $G = \{V, E_{\mathrm{est}}, \boldsymbol{W}_{\mathrm{est}}, \boldsymbol{R}_{\mathrm{est}}\}$.

described in §3.5.3), one should compare the accuracy of the solved puzzles according to top 2 eigenvectors and the top 3-rd and 4-th eigenvectors. For this purpose, we recommend using the following metric

$$\sum_{i=1}^{n} \left(\mathrm{MGC}_{\mathrm{lr}}(P_i, P_{i_r}) + \mathrm{MGC}_{\mathrm{tb}}(P_i, P_{i_b}) + \mathrm{MGC}_{\mathrm{rl}}(P_i, P_{i_l}) + \mathrm{MGC}_{\mathrm{bt}}(P_i, P_{i_b})\right), \quad (3.29)$$

where i_t, i_l, i_b and i_r are the corresponding neighbors of patch i from top, left, bottom and right; if patch i is at the edge or corner of the puzzle grid, we only sum the respective MGC values. We remark that this procedure is not needed at the later updates of §3.5.3 since the connection graphs are nicely approximated then. Our experiments indicate that even for the initial stage, the use of this procedure is beneficial for only few images. We thus leave this step as optional.

Finally, our proposed algorithm for reassembling two-dimensional square jigsaw puzzles is summarized in Algorithm 7.

Algorithm 7 Solution of type 2 puzzles

Input: Puzzle Patches: $\{P_i\}_{i=1}^{N} \subset \mathbb{R}^{p\times p\times 3}$

- Apply Algorithm 5 with $\{P_i\}_{i=1}^{n}$ to construct the Affinity Graph $G = (V, E, \boldsymbol{W}, \boldsymbol{R})$ and obtain MGC values between all patches
- Run Algorithm 4 with $G = (V, E, \boldsymbol{W}, \boldsymbol{R})$ to find the orientations $\{R_i\}_{i=1}^{N}$
- **Optional**: Apply Algorithm 4 with $G = (V, E, \boldsymbol{W}, \boldsymbol{R})$, but use top 3-4 eigenvectors to find the orientations $\{R_{i,2}\}_{i=1}^{N}$, and check by (3.29) whether this solution is better than the one from the previous step. If it is, set $R_i = R_{i,2}$, $1 \le i \le N$.
- **Optional:** Update the MGC metric according to the procedure explained in §3.5.1
- Apply the type 1 jigsaw puzzle solver of [59] to solve the type 1 puzzle with patches $\{R_i \cdot P_i\}_{i=1}^{n}$ and obtain their estimated permutation vector σ.
- **for** iterations 1:5 **do**
  Apply Algorithm 6 with σ, $\{R_i\}_{i=1}^{N}$ and the MGC values to obtain the updated connection graph $G = (V, E, \boldsymbol{W}, \boldsymbol{R})$
  Apply Algorithm 4 with $G = (V, E, \boldsymbol{W}, \boldsymbol{R})$ to recover the orientations $\{R_i\}_{i=1}^{N}$

  Optional: Update the MGC metric according to the procedure explained in §3.5.1
  Apply the type 1 jigsaw puzzle solver of [59] to solve the type 1 puzzle with patches $\{R_i \cdot P_i\}_{i=1}^{n}$ and obtain their estimated permutation vector σ.
- **end for**

Return: $\{R_i\}_{i=1}^{n}$ and σ.

3.5.5 Time Complexity

The most time consuming step is to find the MGC metric between all puzzle pieces. The order of operations for this is step is $O(n^2 d)$, where n is the number of image patches and d is the size of the square image patches. However, one can parallelize this procedure and achieve faster computation. We would like to mention that this step is vital for all jigsaw puzzle solvers.

After finding the MGC metric between all puzzle pieces, our proposed algorithm finds the orientations of all patches and converts a type 2 puzzle into a type 1 puzzle. To find the orientations of puzzle patches we need only construct the connection graph, which requires nearest neighbors computation for each patch. The worst case complexity for this is $O(n^2)$ and the average complexity is $O(n \log(n))$. Then, it finds the top eigenvector of a sparse symmetric matrix with 4 nonzero elements in each column and row, which would take $O(n)$ time. For the type 1 puzzle, the complexity depends on the state-of-the-algorithm being used. It is faster then using the latter algorithm for directly solving the type 2 puzzle.

One may suggest using subsampling to speed up the computation. That is, instead of calculating the MGC metric between a given patch and all other patches, one may only consider a fraction p of the other patches. However, this procedure would only speed up the computation by a constant factor, so the order of time complexity will still remain the same. Also, one needs to be cautious when applying this idea since for a 2-dimensional square jigsaw puzzle each patch has at most 4 neighbors, and if for each patch one subsamples 50% of patches, there are on average only 2 neighbors for a central patch. This may yield a disconnected graph and may also result in sensitivity to individual mistakes.

3.6 Numerical Experiments

We apply our proposed algorithm to solve two-dimensional square jigsaw puzzles of the following standard image datasets: The MIT dataset from Cho et al. [55], which contains 20 images, each with 432 patches, and three datasets from Pomeranz et al. [51], where the first two, which are referred to as McGill and Pomeranz, include 20 images with 540 and 805 patches, respectively, and the third one has 3 images with 3300

patches, which is also referred to as Pomeranz or large Pomeranz. For all datasets, the patches are of size 28×28. Figure 3.11 demonstrates the application of our proposed algorithm to four images that represent the four datasets. To test the accuracy of our proposed algorithm we use the following four metrics, defined in Gallagher [50] and Cho et al. [55]: The direct comparison, the neighbors comparison, the largest component and the perfect reconstruction. The direct comparison measures the percentage of image patches whose location and orientation are correct. The neighbors comparison calculates the percentage of pairs of image patches that are matched correctly. The largest component calculates the percentage of patches in the largest correctly assembled component of the solved puzzle. Finally, the perfect reconstruction of a puzzle is 1 if it is solved correctly and 0 otherwise.

We compared our algorithm with the algorithms of Gallagher [50] and Yu et al. [59] since these were the only algorithms with available codes (we have requested codes from all authors of published algorithms). While our algorithm is deterministic, we noticed some randomness in the results of these two algorithms and we thus report their averaged results over 20 different instances. Nevertheless, this randomness is not significant. For example, the standard deviations for these 20 instances, averaged over the 20 images of the MIT dataset, are 0.055 and 0.015 for Gallagher [50] and Yu et al. [59], respectively.

Table 3.1 compares the four metrics of our proposed algorithm with some state-of-the-art algorithms. For the first three metrics, which obtain percentages, we report the means and standard deviations among each of the four datasets. We clarify that here the means and standard deviations are with respect to the results of the various images in the datasets, whereas for Gallagher [50] and Yu et al. [59] they are averaged over the 20 instances mentioned above. On the other hand, the standard deviations mentioned above are with respect to these 20 instances, while we averaged them over the images in the MIT dataset. For the fourth metric of perfect reconstruction, we report its sum, that is the numbers of perfectly solved images in each dataset. Figure 3.12 presents histograms of the metrics of accuracy of the algorithm for the first three datasets with 20 images. The fourth dataset is excluded from this figure since it only has three images.

As we can see, our results are comparable with those of state-of-the-art methods. The mean error of the first three metrics are slightly better for [59], but with relatively

Table 3.1: Comparison of results for type 2 puzzles for the four datasets. For the first three metrics, we report the mean values and standard deviations over all the images in a dataset. For the fourth metric we report the sum over all images in a dataset. Due to randomness, the results of the algorithms of Gallagher [50] and Yu et al. [59] are averaged over 20 instances of solving a given puzzle.

Dateset	Method	Direct		Neighbor		Largest		Perfect
		mean	std	mean	std	mean	std	
MIT dataset,	Gallagher [50]	84.2	19.7	89.1	12.4	87.2	14.3	9
20 images,	Yu et al. [59]	95.5	13.0	95.4	8.7	95.4	13.2	13
432 patches (28×28)	Our method	95.3	9.5	95.0	9.7	95.2	9.6	13
McGill dataset,	Gallagher [50]	77.2	35.3	85.8	19.8	84.6	21.3	7
20 images,	Yu et al. [59]	92.9	24.6	93.5	14.8	93.1	15.4	13
540 patches (28×28)	Our method	86.9	30.4	91.6	16.4	90.8	18.7	13
Pomeranz dataset,	Gallagher [50]	77.5	27.8	85.3	15.5	79.3	22.6	5
20 images	Yu et al. [59]	91.8	14.2	92.7	13.0	91.7	14.2	9
805 patches (28×28)	Our method	86.4	24.6	90.0	14.7	89.3	15.7	8
Pomeranz dataset,	Gallagher [50]	82.9	15.6	84.2	14.2	82.8	15.7	1
3 images	Yu et al. [59]	89.7	12.3	90.2	11.0	89.7	12.3	1
3300 patches (28×28)	Our method	86.4	14.0	88.1	11.7	86.4	14.0	1

large standard deviations. The histograms in Figure 3.12 indicate that are results are comparable to those of state-of-the-art methods. We remark that images with low percentages of recovered puzzle pieces have large portions of patches with the same uniform color. For example, in the MIT dataset, the puzzle that our proposed algorithm assembled with lowest percentage of 65% contains a lot of patches that are uniformly white and are identical. This puzzle, with the solution of our proposed algorithm, is presented in the first row of Figure 3.11. In this scenario, there is no way to find the exact original positions of all patches. However, the solution obtained by our proposed algorithm is visually identical with the original one. On the other hand, in puzzles where most of the patches have non-zero gradients around their boundaries, we get perfect recovery.

At last, we would like to mention that our proposed algorithm for recovering the unknown orientations of the puzzle patches has no assumption on the shape and size of the puzzle, unlike [50,59]. Furthermore, it is non-greedy. On the other hand, most state-of-the-art algorithms, in particular [50,53,59], use a greedy step to make final corrections.

We believe that by using that final step of corrections we could further improve our results, however, we would like to avoid any greedy or semi-greedy procedure.

3.7 Discussion and Conclusion

This chapter introduced a novel, non-greedy mathematical approach for solving two-dimensional square jigsaw puzzles. More specifically, its main contribution is a theoretically-guaranteed strategy for recovering the unknown orientations of type 2 puzzle patches. Furthermore, it also suggests a non-greedy step for updating the full puzzle solution based on the latter strategy for solving orientations. Some components of the proposed algorithm, in particular, the strategy for recovery of orientations are relatively fast, though the main bottleneck in the computational complexity is shared by all existing algorithms. Numerical experiments on datasets of two-dimensional square jigsaw puzzles indicate competitive results, that is, results, which are at least comparable to state-of-the-art methods.

We expect some possible extensions of the proposed algorithm. First of all, we believe that the ideas pursued in this work could be extended to puzzles that come from more complicated manifolds, such as the two-dimensional sphere or a three-dimensional cube jigsaw puzzle, or to puzzles with more complicated shapes of patches, such as tangrams. The CGL algorithm should be the same, however, instead of considering the group $\mathbb{Z}_4$, one needs to consider the corresponding rotation group. One of the main challenges would be to define a metric between puzzle pieces and construct the connection graph. By doing this, one will extend the applicability of this work to various real-world applications, such as three-dimensional image reconstruction from two-dimensional images.

In terms of theory, it is interesting to analyze our proposed CGL algorithm with more complicated noise models. It is also interesting to see if one can utilize the information from the VDD distances in a more effective way. Another question is whether one may modify the VDD distances and resolve the problem with the current definition discussed in §3.5.1.

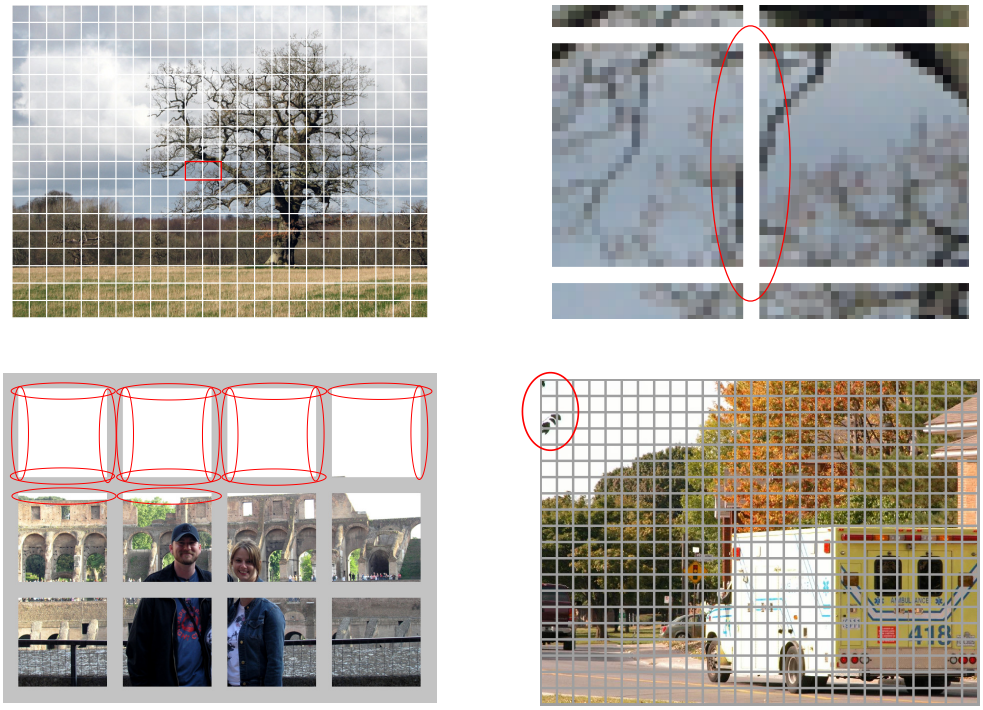


Figure 3.2: Examples of two-dimensional square jigsaw puzzles, where the comparison of two neighboring patches is challenging or impossible. The top left image shows a puzzle with 432 pieces, each of size 28×28. The top right image demonstrates an example of 2 neighboring patches in the latter puzzle that have different pixel values around the boundaries due to the discrete nature of a digital image. These patches are circled with red in the original puzzle (top left image) and their nearby edges are circled with red in the top right image. The bottom two images demonstrate examples of puzzles that have patches with uniformly white edges (circled with red in the bottom left image) and also have some uniformly white patches. Natural solutions of the bottom left puzzle seem to yield visually correct images that may not coincide with the original assignment. However, there are natural solutions of the bottom right puzzle that result in different images than the original one. Indeed, the small component of the image circled with red can be placed in different area within the skies.

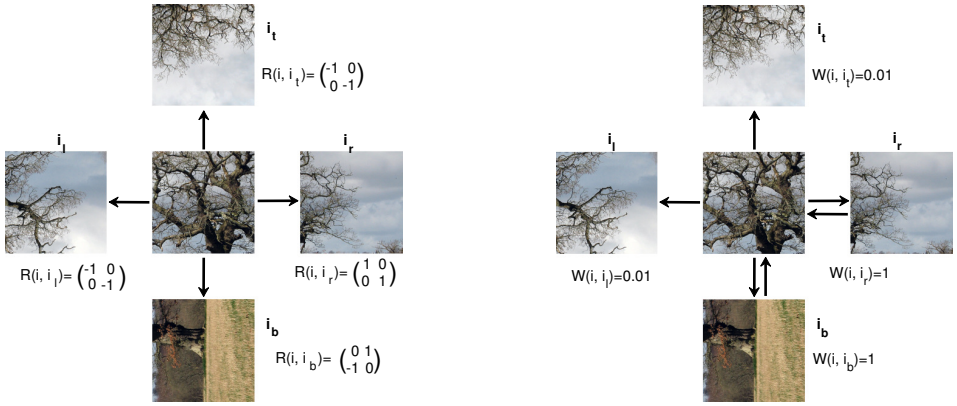


Figure 3.3: Demonstration of the initial step for the construction of the connection graph. The left figure demonstrates the best matches for a given patch from the four directions: top, left, bottom and right. For each matching patch it records the rotation needed to apply to it. The right figure shows the application of these rotations to the matching patches and demonstrates how to assign the weights to undirected graph. In this example, the matching patches from top and left were originally connected by a single direction, their weights in the undirected graph are thus 0.01. On the other hand, the patches from right and bottom are connected in both directions and thus their weights in the undirected graph are 1.

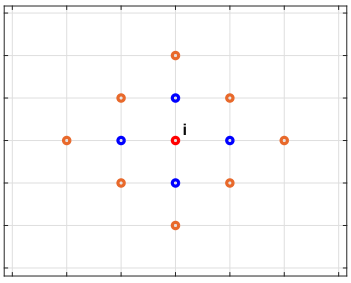


Figure 3.4: Demonstration of the sets $N^1_{G,i}$ and $N^2_{G,i}$. A given vertex i is colored in red, the elements of the set $N^1_{G,i}$ are colored in blue, and the elements of the set $N^2_{G,i}$ are colored in blue and orange.

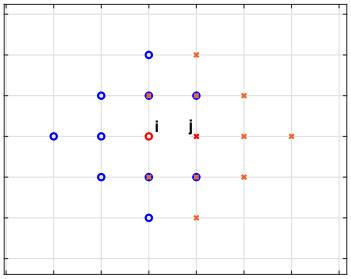


Figure 3.5: Demonstration of Jaccard index. Vertex i is denoted by a red circle and vertex j is denoted by a red cross. The edge between these vertices was removed from the grid. The elements of $N^2_{G^{\setminus(i,j)},i}$ are denoted by blue circles and the elements of $N^2_{G^{\setminus(i,j)},j}$ by orange crosses. The Jaccard index is four since there are four elements in $N^2_{G^{\setminus(i,j)},i} \cap N^2_{G^{\setminus(i,j)},j}$ (denoted by blue circles filled with orange crosses).

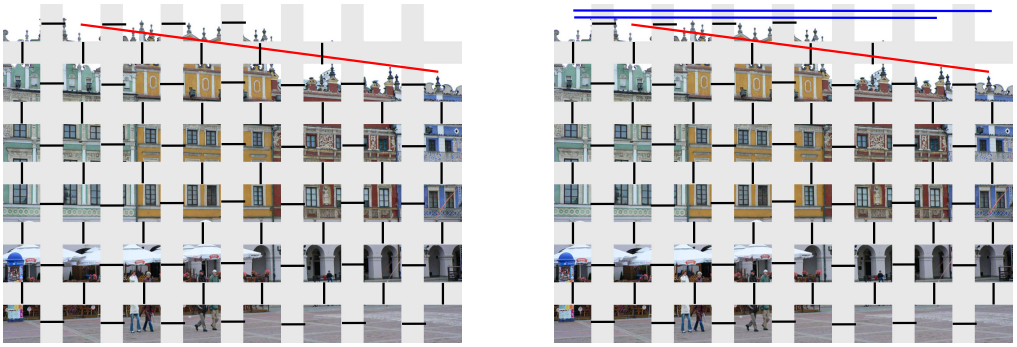


Figure 3.6: Demonstration of a disconnected affinity graph and the way it got connected. The left figure shows an example where the affinity graph resulted by our method is disconnected. Indeed, the two top right patches are not connected to any of the other patches. The black edges connect between true neighbors and the only red edge is a wrongly determined edge. The right figure demonstrates the result of the simple procedure described in §3.4.3. The connected graph has two new blue edges. While these blue edges connect between non-neighboring patches, the originally disconnected patches are uniformly white and thus their rotations do not matter for the reconstruction of the image.

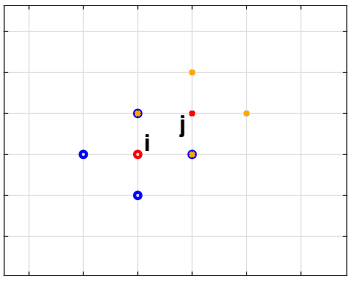


Figure 3.7: Demonstration of finding diagonally neighboring vertices in the grid. Two vertices i and j are denoted by a red circle and a red cross respectively. The elements of the sets $N^1_{G,i}$ and $N^1_{G,j}$ are colored by blue and orange respectively. The intersection of these sets yield the two diagonally neighboring vertices to i and j.

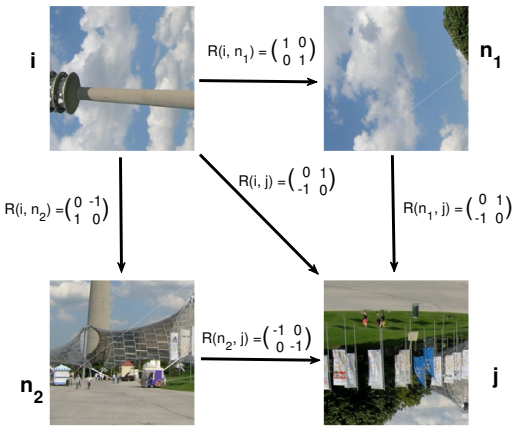


Figure 3.8: Intuition for the condition in (3.21) and (3.22). The two vertices i and j are diagonal neighbors and the vertices n_1 and n_2 satisfy (3.20). Thus, i, j, n_1 and n_2 form a cycle of size 4, that is, a 4-loop. The relative rotations between vertices are indicated on the corresponding edges. We note that both $\boldsymbol{R}(i,n_1)\boldsymbol{R}(n_1,j)$ and $\boldsymbol{R}(i,n_2)\boldsymbol{R}(n_2,j)$ are equal to the relative rotation $\boldsymbol{R}(i,j)$ shown on edge (i,j). In particular, $\boldsymbol{R}(i,n_1)\boldsymbol{R}(n_1,j) = \boldsymbol{R}(i,n_2)\boldsymbol{R}(n_2,j)$. The assigned weights thus try to encourage this constraint on rotations and penalize cases where it is not satisfied.

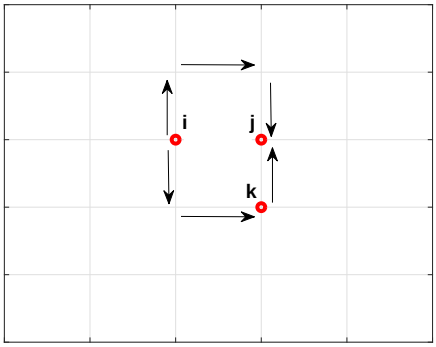


Figure 3.9: An example where VDD fails to reflect the distance between nearby patches. The graph is a grid with one missing edge between vertices i and j (all other neighboring edges are connected by an edge). Due to the structure of the grid, the shortest path between vertices i and j is of length 3, whereas the shortest path between vertices i and k is 2. Thus the use of VDD leads to the wrong conclusion that vertex i is closer to vertex k than to vertex j.

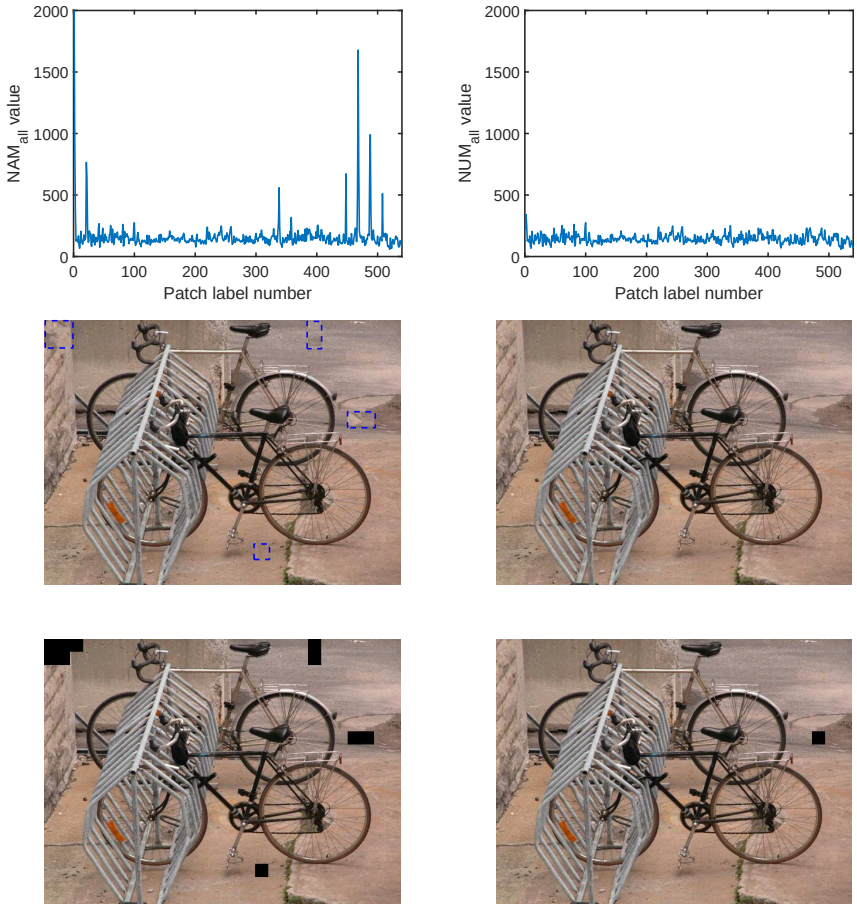


Figure 3.10: Demonstration of the update step for 2 iterations, described in §3.5.3, of a type 2 puzzle with 540 pieces each with sizes of 28×28. The first row shows the histograms of the NAM_{all} metric values for all patches, defined in (3.25). The second row shows the solution of the puzzle after each iteration of assembling the puzzle and the third row shows the remaining patches of an assembled puzzle after removing the patches that are wrongly placed or oriented.

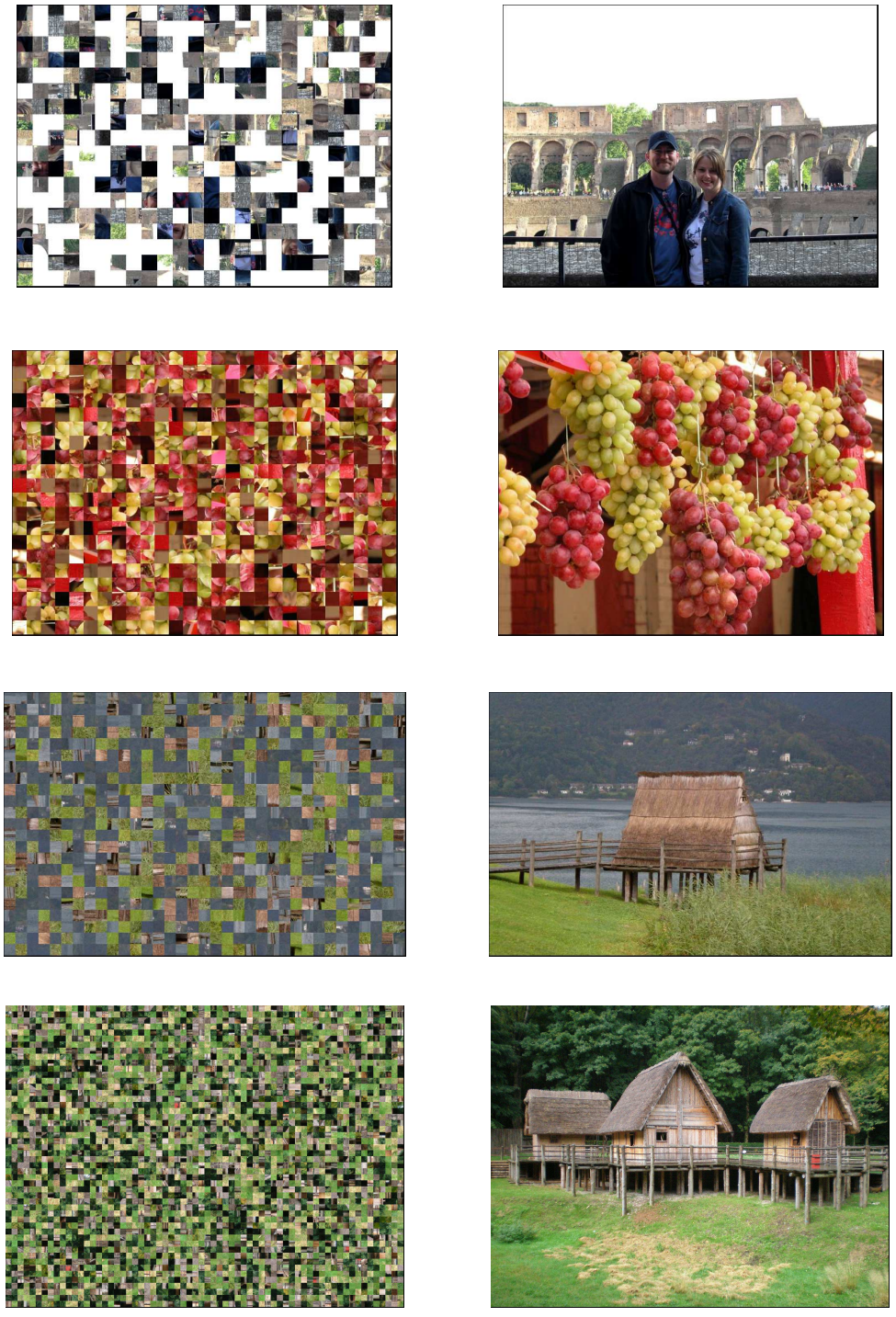


Figure 3.11: Reconstruction results of our algorithm for type 2 puzzles representing the four datasets. The images in the left column are the inputs for the algorithm and the ones in the right column are the outputs generated by our proposed algorithm. All the patches are of size 28×28. The puzzle in first row is from the MIT dataset with 432 patches, the puzzle in the second row is from the McGill dataset with 540 patches, the puzzle in the third row is from the Pomeranz dataset with 805 patches and the puzzle in the fourth row is from the Pomeranz dataset with 3300 patches.

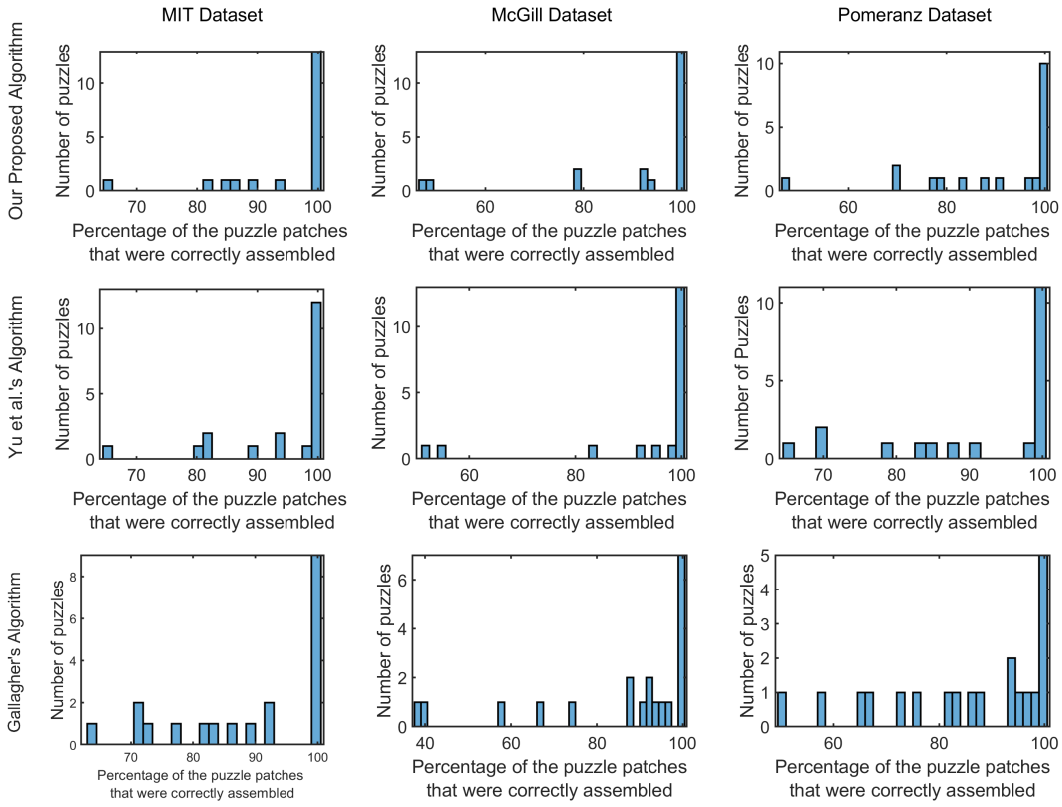


Figure 3.12: Histograms of the percentages of the recovered patches for type 2 jigsaw puzzle. The three rows correspond to results by our proposed algorithm, the algorithm of [59] and the algorithm of [50], respectively. The three columns correspond to the MIT, McGill and (small) Pomeranz datasets, respectively.

Chapter 4

Part III: Non-convex Analysis of Multi-Graph Matching

4.1 Introduction

Multi-Graph Matching (MGM) is a well-known problem that arises naturally in many computer vision applications. Its setting assumes n graphs each with m vertices. Furthermore, it assumes that for each pair of graphs there exists a one-to-one correspondence map between their vertices. The MGM problem asks to find the complete set of correspondence maps between all graphs given only noisy measurements of the mutual correspondences. The solution of this setting is useful for several applications, including 3D shape matching [68], 3D reconstruction/Stereo Matching [69], Structure from Motion (SfM) [70] etc.

The following computer vision application helps to motivate the problem: Assume we take pictures of the same scene from different angles and that this scene has m points of interests that we can identify among all images. Next, for each image we construct a graph that connects between these points of interests and define a vector of characteristics (one possible option is SIFT features) for each point. Assume we are given an algorithm that matches between each of these two graphs (corresponding to two images) and finds the correspondence map, possibly with some mistakes in it. The goal is to correctly register the same points of interests across all images.

If the measurements are corrupted, naive methods, such as trying to start with one

graph and chase the points by mutual correspondences often fail. Indeed, one mistake would lead to many wrong assignments. Thus, more robust algorithms are needed. Some algorithms have already addressed the MGM problem [2, 71, 72, 73]: Pachauri et al. [72] propose a harmonic analysis approach; Pachauri et al. [2] propose the eigenvector synchronization approach; Shi et al. [73] propose tensor power iteration method; Chen et al. [71] propose semidefinite relaxation method; Yan et al. [74] propose graduated consistency-regularized boosting method.

In this work we propose to apply the Projected Power Method (PPM) to MGM problem. The PPM is an iterative algorithm that starts at an initial guess and improves the guess at each iteration. This framework has been applied and analyzed for synchronization problem for various groups: Chen and Candes [75] apply PPM to joint alignment problem (the group is $\mathbb{Z}/n\mathbb{Z}$), Boumal [76] applies PPM to phase synchronization problem (the group is $SO(2)$).

We show that under some conditions on the initial guess and for a particular data model the PPM algorithm for MGM linearly converges to the unique solution of the problem and show how to find such an initial guess. Furthermore, we present numerical experiments for both synthetic and real world datasets and compare the results with a state-of-the-art algorithm.

4.1.1 Structure of This Chapter

The rest of this chapter is organized as follows: §4.2 contains a short description of the mathematical framework of MGM; §4.3 reviews the projected power method for MGM; §4.4 theoretically analyses the PPM algorithm for MGM; and §4.5 concludes with numerical experiments that test the accuracy of the proposed algorithm.

4.2 The Mathematical Setting for MGM

In this section we summarize the mathematical formulation of the MGM problem. Consider a set of permutation matrices $\boldsymbol{P}_1, \dots \boldsymbol{P}_n \in \mathrm{Perm}(m)$, where $\mathrm{Perm}(m)$ denotes the set of all $m \times m$ permutation matrices. Next, assume we are given noisy measurements of the relative permutations, that is, for all $1 \leq i \neq j \leq n$ we are given a noisy

measurement of $\boldsymbol{P}_i\boldsymbol{P}_j^{-1}$, which is the same as $\boldsymbol{P}_i\boldsymbol{P}_j^T$ (indeed, for a permutation matrix $\boldsymbol{P} \in \mathrm{Perm}(m)$, $\boldsymbol{P}^{-1} = \boldsymbol{P}^T$). We register these measurements in a block matrix $\boldsymbol{L} \in \mathbb{R}^{mn\times mn}$ (with $m \times m$ blocks) as its ij-th block.

The noise model that we consider in this work is the random corruption model. It assumes $0 \leq \pi_0 \leq 1$ and for each $1 \leq i \neq j \leq n$, with probability π_0 the measured relative permutation is the correct one, otherwise it is a uniformly random permutation matrix. That is

$$\boldsymbol{L}[i,j] = \begin{cases} \boldsymbol{P}_i\boldsymbol{P}_j^T & \text{with probability } \pi_0, \\ \mathrm{Unif}(m) & \text{otherwise,} \end{cases} \tag{4.1}$$

where $\mathrm{Unif}(m)$ denotes the random variable corresponding to the uniform distribution on $m \times m$ permutation matrices. The parameter π_0, introduced in (4.1) is called the non-corruption rate and $1 - \pi_0$ is called the corruption rate.

Note, that if $\pi_0 = 1$, that is, we did not make any mistakes in measurements, then $\boldsymbol{L} = \bar{\boldsymbol{P}}\bar{\boldsymbol{P}}^T$, where

$$\bar{\boldsymbol{P}} = \begin{bmatrix} \boldsymbol{P}_1 \\ \vdots \\ \boldsymbol{P}_n \end{bmatrix}. \tag{4.2}$$

The MGM problem asks to determine the permutation matrices $\boldsymbol{P}_1, \ldots, \boldsymbol{P}_n \in \mathrm{Perm}(m)$, up to a global permutation, from $\boldsymbol{L}$. To tackle this problem we consider the following optimization problem:

$$\arg\min_{\bar{\boldsymbol{Q}}\in \mathrm{Perm}^n(m)} \|\boldsymbol{L} - \bar{\boldsymbol{Q}}\bar{\boldsymbol{Q}}^T\|_F^2, \tag{4.3}$$

where $\mathrm{Perm}^n(m)$ represents the set of matrices in $\mathbb{R}^{mn\times n}$ which has $m \times m$ permutation matrices as its blocks,

$$\bar{\boldsymbol{Q}} = \begin{bmatrix} \boldsymbol{Q}_1 \\ \vdots \\ \boldsymbol{Q}_n \end{bmatrix}, \tag{4.4}$$

and $\boldsymbol{Q}_i \in \mathrm{Perm}(m)$ for all $1 \leq i \leq n$. Since the set of all permutation matrices of size m is discrete and n is finite, the set $\mathrm{Perm}^n(m)$ is discrete, thus, non-convex. Note, that

(4.3) is equivalent to

$$\arg \max_{\bar{\boldsymbol{Q}} \in \mathbb{R}^{nm \times m}} \operatorname{tr}(\bar{\boldsymbol{Q}}^T \boldsymbol{L} \bar{\boldsymbol{Q}}). \tag{4.5}$$

If we relax the constraint $\bar{\boldsymbol{Q}} \in \mathbb{R}^{nm \times m}$ in (4.3) and (4.5) they would become an eigenvalue-eigenvector problem and the matrix $\boldsymbol{\lambda}$ that has its columns as the top m eigenvectors of $\boldsymbol{L}$ would solve them. Pachauri et al. [2] propose an algorithm, which relaxes the condition $\bar{\boldsymbol{Q}} \in \mathbb{R}^{nm \times m}$, solves the eigenvector problem to obtain $\boldsymbol{\lambda}$ and projects the blocks of $\boldsymbol{\lambda}$ into the set of permutation matrices.

We propose to apply the Projected Power Method (PPM) to solve the MGM problem. Note, that in Pachauri et al [2], to find the top m eigenvectors of matrix $\boldsymbol{L}$ they use the Power Method (PM) algorithm. The PPM is a similar iterative procedure as the PM, but instead, at each iteration it projects the result into the given set (in our case it would be $\text{Perm}^n(m)$). Thus, it preserves the non-convexity of the problem.

4.3 Projected Power Method for Multi-Graph Matching

In this section we review the PPM for MGM algorithm. Assume we are given the matrix $\boldsymbol{L}$, defined in (4.1), and a starting point (initial guess)

$$\bar{\boldsymbol{Q}}^0 = \begin{bmatrix} \boldsymbol{Q}_1^0 \\ \vdots \\ \boldsymbol{Q}_n^0 \end{bmatrix}, \tag{4.6}$$

where $\boldsymbol{Q}_1^0, \ldots, \boldsymbol{Q}_n^0 \in \text{Perm}(m)$ are permutation matrices. The PPM is an iterative algorithm, which has two main steps in each of its iteration: First, for iteration $1 \leq t$ calculate the following matrix $\tilde{\boldsymbol{Q}}_i^t = \boldsymbol{L} \bar{\boldsymbol{Q}}_i^{t-1}$. Second, project the $m \times m$ blocks of $\tilde{\boldsymbol{Q}}_i^t$ into the set of $m \times m$ permutation matrices $\text{Perm}(m)$.

Before we summarize the algorithm, there are two things that we need to justify. First, at each iteration of the algorithm we need to project the blocks of $\tilde{\boldsymbol{Q}}$ into the set $\text{Perm}(m)$. We define the projection of a square matrix $\boldsymbol{A} \in \mathbb{R}^{m \times m}$ into the set of permutation matrices $\boldsymbol{P}$ as a permutation matrix $\boldsymbol{P}$ which solves the following problem:

$$\mathcal{P}_\Delta(\boldsymbol{A}) = \arg\max_{\boldsymbol{Q} \in \text{Perm}(m)} \operatorname{tr}(\boldsymbol{Q}^T \boldsymbol{A}). \tag{4.7}$$

This problem is known as linear assignment problem, with an assignment matrix $\boldsymbol{A}$. There are some well-known algorithms for solving (4.7), the one that we use in this work is the well-known Hungarian algorithm, which is also known as Kuhn-Munkres algorithm or Munkres assignment algorithm [77].

Second, since the problem is non-convex, in order to guarantee convergence, we need to find a proper starting point for the algorithm. We propose to use the spectral initialization. That is, let the matrix $\boldsymbol{\lambda} \in \mathbb{R}^{nm \times m}$ has its columns as the top m eigenvectors of $\boldsymbol{L}$. Furthermore, set the $m \times m$ blocks of $\bar{\boldsymbol{Q}}^0$ as $\mathcal{P}_\Delta(\boldsymbol{\lambda}_1^{-1}\boldsymbol{\lambda}_i)$, where $\boldsymbol{\lambda}_i$ is the i-th $m \times m$ block of $\boldsymbol{\lambda}$. In §4.4 we show that such initialization guarantees the convergence of the algorithm. However, in our numerical experiments we see that under some conditions on m, n and π_0 random initialization would just be as successful as the spectral initialization.

The PPM for MGM algorithm is summarized in Algorithm 8 and the theoretical analysis of the algorithm is presented in §4.4.

Algorithm 8 Projected Power Method for Multi-Graph Matching

Input: The block matrix $\boldsymbol{L} \in \mathbb{R}^{mn \times mn}$, where for all $1 \leq i, j \leq n$ $\boldsymbol{L}[i,j] \in \text{Perm}(m)$, T_{ppm}: stopping parameter.
Find $\boldsymbol{\lambda} \in \mathbb{R}^{mn \times m}$ which has top m eigenvectors of $\boldsymbol{L}$ as its columns
Set: $\boldsymbol{Q}_i^0 = \mathcal{P}_\Delta(\boldsymbol{\lambda}_1^{-1}\boldsymbol{\lambda}_i)$ for $1 \leq i \leq n$ and combine $\boldsymbol{Q}_1^0, \ldots, \boldsymbol{Q}_n^0$ into $\bar{\boldsymbol{Q}}^0$ as in (4.6).
for $1 \leq t \leq T_{\text{ppm}}$ **do**
  Set: $\boldsymbol{Q}_i^t = \mathcal{P}_\Delta((\boldsymbol{L}\bar{\boldsymbol{Q}}^{t-1})_i)$ for all $1 \leq i \leq n$
  Combine: $\boldsymbol{Q}_1^t, \ldots, \boldsymbol{Q}_n^t$ into

$$\bar{\boldsymbol{Q}}^t = \begin{bmatrix} \boldsymbol{Q}_1^t \\ \vdots \\ \boldsymbol{Q}_n^t \end{bmatrix},$$

end for
Output: $\bar{\boldsymbol{Q}}^T$

4.4 Theoretical analysis

This section provides theoretical analysis for the PPM for MGM algorithm. In §4.4.1 we state and prove a theorem for the noise free case, and in §4.4.2 we provide with

theoretical analysis for the random corruption model.

4.4.1 A Theorem with its Proof for the Noise Free Case

As a quick warm up before we start the analysis of the random corruption model, we analyze the algorithm for noise free case. The following theorem states that if the data is noise free (that is $\pi_0 = 1$) no matter which initial point we start with the proposed algorithm will converge to the true solution.

Theorem 2 (Projected power method for noise free data)**.** *Assume we are given* $\boldsymbol{P}_1$, $\ldots$, $\boldsymbol{P}_n \in Perm(m)$, $m \times m$ *permutation matrices and that the matrix* $\boldsymbol{L} \in \mathbb{R}^{mn \times mn}$ *for all* $1 \leq i, j \leq n$ *has its* ij*-th* $m \times m$ *block as* $\boldsymbol{P}_i \boldsymbol{P}_j^T$*. For any* $\bar{\boldsymbol{Q}}^0 \in \boldsymbol{R}^{mn \times m}$ *with* $\boldsymbol{Q}_1$, $\ldots$, $\boldsymbol{Q}_n \in Perm(m)$ *the iterates produced by Algorithm 8 will converge to* $\bar{\boldsymbol{P}}$*.*

Proof. We start the proof by showing that at each iteration of Algorithm 8 the value of the following function, $f(\bar{\boldsymbol{Q}}) = \operatorname{tr}(\bar{\boldsymbol{Q}}^T \boldsymbol{L} \bar{\boldsymbol{Q}})$ either increases or stays the same. That is $f(\bar{\boldsymbol{Q}}^{t+1}) \geq f(\bar{\boldsymbol{Q}}^t)$ for all $t \geq 1$.

To prove this we use the fact that f is convex. Note, that since $\pi_0 = 1$, $\boldsymbol{L} = \bar{\boldsymbol{Q}} \bar{\boldsymbol{Q}}^T$, thus $\boldsymbol{L}$ is positive semi-definite matrix on the convex hull of permutation matrices (which is the set of doubly stochastic matrices). Which implies, that f is convex on the set of doubly stochastic matrices. Thus, the following inequality holds

$$f(\bar{\boldsymbol{Q}}^{t+1}) - f(\bar{\boldsymbol{Q}}^t) \geq 2 \operatorname{tr}((\bar{\boldsymbol{Q}}^{t+1} - \bar{\boldsymbol{Q}}^t) \boldsymbol{L} \bar{\boldsymbol{Q}}^t) \geq 0. \tag{4.8}$$

For the second inequality we used the fact that each block of $\boldsymbol{Q}^{t+1}$ is the projection of a corresponding block of $\boldsymbol{L}\boldsymbol{Q}^t$. Thus, at each iteration of Algorithm 8 the value of f either increases or stays the same. We also note that since $\operatorname{Perm}^n(m)$ is a discrete set, $f|_{\operatorname{Perm}^n(m)}$ is bounded above.

Next, we show that if for two consecutive iterations, the value of f is the same, that is for some $t \geq 1$, $f(\bar{\boldsymbol{Q}}^t) = f(\bar{\boldsymbol{Q}}^{t+1})$ then the algorithm has reached to a fixed point, that is $\bar{\boldsymbol{Q}}^t = \bar{\boldsymbol{Q}}^{t+1}$. To prove this claim we write

$$\operatorname{tr}(\bar{\boldsymbol{Q}}^{t^T} \boldsymbol{L} \bar{\boldsymbol{Q}}^t) = \operatorname{tr}(\bar{\boldsymbol{Q}}^{t+1^T} \boldsymbol{L} \bar{\boldsymbol{Q}}^{t+1}). \tag{4.9}$$

Furthermore, using the fact that $\boldsymbol{Q}^{t+1}$ is the projection of $\boldsymbol{L}\boldsymbol{Q}^t$ and using the definition

of the projection operator we obtain

$$\mathrm{tr}(\boldsymbol{Q}^{t^T}\boldsymbol{L}\boldsymbol{Q}^t) \leq \mathrm{tr}(\boldsymbol{Q}^{t+1^T}\boldsymbol{L}\boldsymbol{Q}^t). \tag{4.10}$$

Next, we combine (4.9) and (4.10) to conclude

$$\mathrm{tr}\left((\boldsymbol{Q}^{t+1}-\boldsymbol{Q}^t)^T\boldsymbol{L}(\boldsymbol{Q}^{t+1}-\boldsymbol{Q}^t)\right) = \mathrm{tr}(\boldsymbol{Q}^{t+1^T}\boldsymbol{L}\boldsymbol{Q}^{t+1}) + \\ \mathrm{tr}(\boldsymbol{Q}^{t^T}\boldsymbol{L}\boldsymbol{Q}^t) - 2\,\mathrm{tr}(\boldsymbol{Q}^{t+1^T}\boldsymbol{L}\boldsymbol{Q}^t) \leq 0. \tag{4.11}$$

Since $\boldsymbol{L}$ is a positive semi-definite matrix, it implies that $\boldsymbol{Q}^{t+1}-\boldsymbol{Q}^t \in \ker(\boldsymbol{L})$ and since the blocks of $\boldsymbol{Q}^{t+1}$ and $\boldsymbol{Q}^t$ are permutation matrices we get that $\boldsymbol{Q}^{t+1}=\boldsymbol{Q}^t$. Thus, the algorithm converges to a fixed point.

Next, we analyze the fixed points of the projection operator and prove that if $\boldsymbol{L} = \bar{\boldsymbol{P}}\bar{\boldsymbol{P}}^T$, there is only one fixed point which is $\bar{\boldsymbol{P}}$ (up to a global permutation). Assume $\bar{\boldsymbol{Q}}$ is a fixed point, note that the ij-th block of $\boldsymbol{L}$, $\boldsymbol{L}[i,j] = \boldsymbol{P}_i\boldsymbol{P}_j^T$. Which means that if we denote $\boldsymbol{X} := \boldsymbol{L}\boldsymbol{Q}$ and write the projection for the i-th block of $\boldsymbol{X}$ we will obtain

$$\mathcal{P}_\Delta(\boldsymbol{X}_i) = \mathcal{P}_\Delta(\sum_{k=1}^n \boldsymbol{L}_{ik}\boldsymbol{Q}_i) = \mathcal{P}_\Delta(\sum_{k=1}^n \boldsymbol{P}_i\boldsymbol{P}_k^T\boldsymbol{Q}_k) = \\ \mathcal{P}_\Delta(\boldsymbol{P}_i\sum_{k=1}^n \boldsymbol{P}_k^T\boldsymbol{Q}_k) = \boldsymbol{P}_i \times \mathcal{P}_\Delta(\sum_{k=1}^n \boldsymbol{P}_k^T\boldsymbol{Q}_k) = \boldsymbol{Q}_i. \tag{4.12}$$

For the last equation we used the fact that $\bar{\boldsymbol{Q}}$ is a fixed point. We conclude from (4.12) that for all $1 \leq i \leq n$, $P_i \times \mathcal{P}_\Delta(\sum_{k=1}^n P_k^T Q_k) = \boldsymbol{Q}_i$, which means that for all $1 \leq i \leq n$

$$\mathcal{P}_\Delta(\sum_{k=1}^n \boldsymbol{P}_k^T\boldsymbol{Q}_k) = \boldsymbol{P}_i^T\boldsymbol{Q}_i. \tag{4.13}$$

Note, that the LHS of (4.13) does not depend on i, which implies that for all $1 \leq i \leq n$, $\boldsymbol{P}_i^T\boldsymbol{Q}_i = \text{const}$. This concludes the proof. □

4.4.2 Theoretical Analysis of Algorithm 8 for Random Corruption Model

In this section we state and prove a theorem which claims that at each iteration of Algorithm 8, it improves the current guess. Then, we discuss how to apply this theorem and why the spectral initialization works.

To state the main theorem, we need the following definition:

Definition 5. *For* $\bar{\boldsymbol{P}}, \bar{\boldsymbol{Q}}, \in \mathbb{R}^{mn\times m}$ *with* $\boldsymbol{P}_1, \ldots, \boldsymbol{P}_n \in Perm(m)$ *and* $\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_n \in Perm(m)$ *by* $\delta(\bar{\boldsymbol{P}}, \bar{\boldsymbol{Q}})$ *we denote*

$$\delta(\bar{\boldsymbol{P}}, \bar{\boldsymbol{Q}}) = \frac{\|\bar{\boldsymbol{P}} - \bar{\boldsymbol{Q}}\|_{0,m}}{n}, \tag{4.14}$$

where $\|.\|_{0,m}$ *is the* 0*-norm over the* $m \times m$ *blocks of a block vector.*

Theorem 3. *Assume we are given*

$$\bar{\boldsymbol{P}} = \begin{bmatrix} \boldsymbol{P}_1 \\ \vdots \\ \boldsymbol{P}_n \end{bmatrix}, \qquad \bar{\boldsymbol{Q}} = \begin{bmatrix} \boldsymbol{Q}_1 \\ \vdots \\ \boldsymbol{Q}_n \end{bmatrix}, \tag{4.15}$$

with $\boldsymbol{P}_1, \ldots, \boldsymbol{P}_n \in Perm(m)$ *and* $\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_n \in Perm(m)$*. Furthermore assume that* $\boldsymbol{L}$ *is constructed according to the random corruption model* (4.1) *for some* $\pi_0 > 0$ *and for all* $1 \le i, j \le n$*, if* $\boldsymbol{L}[i,j] \neq \boldsymbol{P}_i \boldsymbol{P}_j^t$ *then* $\boldsymbol{Q}_j$ *is independent of* $\boldsymbol{L}[i,j]$*. If* $\alpha = 1 - \delta(\bar{\boldsymbol{P}}, \bar{\boldsymbol{Q}}) > 0.5$*, then for all* $1 \le i \le n$ *and sufficiently large* n

$$Prob\left(\mathcal{P}_\Delta\left((\boldsymbol{L}\bar{\boldsymbol{Q}})_i\right) = \boldsymbol{P}_i\right) > 1 - \frac{1}{2} e^{-\frac{(2\alpha-1)^2 n \pi_0}{2(1-\pi_0)}}.$$

Proof. Assume that the matrix (block vector) $\bar{\boldsymbol{R}}$ is obtained by projecting the blocks $\boldsymbol{L}\bar{\boldsymbol{Q}}$ into the set Perm(m). That is for each $1 \le i \le n$, $\boldsymbol{R}_i = \mathcal{P}_\Delta((\boldsymbol{L}\bar{\boldsymbol{Q}})_i)$. We aim to analyze $\delta(\bar{\boldsymbol{P}}, \bar{\boldsymbol{R}})$ and show that with high probability for large n if $\delta(\bar{\boldsymbol{P}}, \bar{\boldsymbol{Q}}) < 0.5$, $\delta(\bar{\boldsymbol{P}}, \bar{\boldsymbol{R}}) < \rho\delta(\bar{\boldsymbol{P}}, \bar{\boldsymbol{Q}})$.

We start with the computation of the i-th block of $\boldsymbol{L}\bar{\boldsymbol{Q}}$.

$$(\boldsymbol{L}\bar{\boldsymbol{Q}})_i = \sum_{j=1}^{n} \boldsymbol{L}[i,j]\boldsymbol{Q}_j. \tag{4.16}$$

Now, if for all $1 \leq i \leq n$ by $J_i \subset \{1,\ldots,n\}$ we denote the set of indices j for which $\boldsymbol{L}[i,j] = \boldsymbol{P}_i\boldsymbol{P}_j^T$, that is the set of indices for which the measurement is correct. We can break the sum in (4.16) into the following:

$$(\boldsymbol{L}\bar{\boldsymbol{Q}})_i = \sum_{j\in J_i}^{n} \boldsymbol{P}_i\boldsymbol{P}_j^T\boldsymbol{Q}_j + \sum_{j\notin J_i}^{n} \boldsymbol{L}[i,j]\boldsymbol{Q}_j = \boldsymbol{P}_i \sum_{j\in J_i}^{n} \boldsymbol{P}_j^T\boldsymbol{Q}_j + \sum_{j\notin J_i}^{n} \boldsymbol{L}[i,j]\boldsymbol{Q}_j. \tag{4.17}$$

Furthermore, by $S \subset \{1,\ldots,n\}$ let us denote the set of indices for which $\boldsymbol{Q}_i = \boldsymbol{P}_i$. We proceed with (4.17) by

$$(\boldsymbol{L}\bar{\boldsymbol{Q}})_i = |J_i \cap S|\boldsymbol{P}_i + \boldsymbol{P}_i \sum_{j\in J_i\setminus S}^{n} \boldsymbol{P}_j^T\boldsymbol{Q}_j + \sum_{j\notin J_i}^{n} \boldsymbol{L}[i,j]\boldsymbol{Q}_j. \tag{4.18}$$

Now, since for $j \notin J_i$, Ł$[i,j]$ is a uniformly distributed permutation independent of $\boldsymbol{Q}_j$, we get that $\boldsymbol{L}[i,j]\boldsymbol{Q}_j$ is also uniformly distributed. Furthermore, since for all $j_1, j_2 \notin J_i$, Ł$[i,j_1]$ and Ł$[i,j_2]$ are independent and $\boldsymbol{Q}_j$ is fixed for all $j \in J_i$ we can claim that

$$\sum_{k\notin J_i} \boldsymbol{L}[i,j]\boldsymbol{Q}_j \approx \frac{|J_i|}{m}\mathbf{1}^T \times \mathbf{1}. \tag{4.19}$$

To make (4.19) rigorous we use the matrix Bernstein inequality [78], which concludes that

$$\text{Prob}\left(\left\|\sum_{j\in J_i}\left(\boldsymbol{L}[i,j] - \frac{1}{m}\mathbf{1}\mathbf{1}^T\right)\right\| \geq t\right) \leq 2me^{-\frac{t^2/2}{v(Z)+Lt/3}} \tag{4.20}$$

Thus, if n is large enough, for $1 \leq i \leq n$, with high probability we can claim that

$$\mathcal{P}_\Delta\left((\boldsymbol{L}\bar{\boldsymbol{Q}})_i\right) = \mathcal{P}_\Delta\left(|J_i \cap S|\boldsymbol{P}_i + \boldsymbol{P}_i \sum_{j\in J_i\setminus S}^{n} \boldsymbol{P}_j^T\boldsymbol{Q}_j + \sum_{j\notin J_i}^{n} \boldsymbol{L}[i,j]\boldsymbol{Q}_j\right) =$$

$$\mathcal{P}_\Delta\left(|J_i \cap S|\boldsymbol{P}_i + \boldsymbol{P}_i \sum_{j\in J_i\setminus S}^{n} \boldsymbol{P}_j^T\boldsymbol{Q}_j\right) =$$

$$\boldsymbol{P}_i\mathcal{P}_\Delta\left(|J_i \cap S|\boldsymbol{I}_m + \sum_{j\in J_i\setminus S}^{n} \boldsymbol{P}_j^T\boldsymbol{Q}_j\right). \quad (4.21)$$

Note, that if $|J_i \cap S| > |J_i \setminus S|$ then $\mathcal{P}_\Delta\left(|J_i \cap S|\boldsymbol{I}_m + \sum_{j\in J_i\setminus S}^{n} \boldsymbol{P}_j^T\boldsymbol{Q}_j\right) = \boldsymbol{I}_m$ and thus $\mathcal{P}_\Delta\left((\boldsymbol{L}\bar{\boldsymbol{Q}})_i\right) = \boldsymbol{P}_i$. This implies that

$$\text{Prob}\left(\mathcal{P}_\Delta\left((\boldsymbol{L}\bar{\boldsymbol{Q}})_i\right) = \boldsymbol{P}_i\right) \geq \text{Prob}\left(|J_i \cap S| > |J_i \setminus S|\right)$$

Note, that $|J_i \cap S| \sim \text{Binom}(\pi_0, n\alpha)$ and $|J_i \setminus S| \sim \text{Binom}(\pi_0, n(1-\alpha))$. Thus,

$$\text{Prob}\left(\mathcal{P}_\Delta\left((\boldsymbol{L}\bar{\boldsymbol{Q}})_i\right) = \boldsymbol{P}_i\right) \geq \text{Prob}\left(\text{Binom}(\pi_0, n\alpha) - \text{Binom}(\pi_0, n(1-\alpha) > 0\right).$$

Next, we use the normal approximation for binomials to conclude that if n is large enough

$$\text{Prob}\left(\text{Binom}(\pi_0, n\alpha) - \text{Binom}(\pi_0, n(1-\alpha) > 0\right) \approx \text{Prob}(\text{N}(n\alpha\pi_0, n\alpha\pi_0(1-\pi_0)) -$$
$$\text{N}(n(1-\alpha)\pi_0, n(1-\alpha)\pi_0(1-\pi_0) > 0) = \text{Prob}\left(\text{N}((2\alpha-1)n\pi_0, n\pi_0(1-\pi_0))\right).$$

Combining all these results yields

$$\text{Prob}\left(\mathcal{P}_\Delta\left((\boldsymbol{L}\bar{\boldsymbol{Q}})_i\right) = \boldsymbol{P}_i\right) > \text{Prob}\left(\text{N}((2\alpha-1)n\pi_0, n\pi_0(1-\pi_0))\right), \quad (4.22)$$

for large enough n. We conclude by noting that if $\alpha > 1/2$, that is $2\alpha - 1 > 0$ we get that

$$\text{Prob}\left(\text{N}((2\alpha-1)n\pi_0, n\pi_0(1-\pi_0))\right) \geq 1 - \frac{1}{2}e^{-\frac{(2\alpha-1)^2 n\pi_0}{2(1-\pi_0)}}. \quad (4.23)$$

Combining (4.22) and (4.23) we conclude the proof. □

4.4.3 Discussion on Spectral Initialization and Independence

Next, we would like to mention that Chen and Candes [75][Theorem 9] prove a general result for spectral initialization, which is applicable for our problem. It guarantees that the initial guess that we obtain from the top m eigenvectors of $\boldsymbol{L}$ is in the range where Theorem 3 is applicable.

The last thing to justify is that in Algorithm 8, during the iterations for all $1 \leq i, j \leq n$ if $\boldsymbol{L}[i,j] \neq \boldsymbol{P}_i \boldsymbol{P}_j^T$ then $\boldsymbol{L}[i,j]$ and $\boldsymbol{Q}_j^{t+1}$ stay independent, given that $\delta(\bar{\boldsymbol{P}}, \bar{\boldsymbol{Q}}^{t+1}) < 0.5$. We observe that similar to (4.17) we can write

$$(\boldsymbol{L}\bar{\boldsymbol{Q}})_i = \sum_{j=1}^{n} \boldsymbol{L}[i,j]\boldsymbol{Q}_j \tag{4.24}$$

and thus, if we follow the analysis in the proof of Theorem 3, removing 1 summand, with high probability for sufficiently large n will not change $\mathcal{P}_\Delta((\boldsymbol{L}\bar{\boldsymbol{Q}})_i)$.

4.5 Numerical Experiments

In this section we test the PPM for MGM algorithm, proposed in this work, by using both synthetic and real datasets. This section is organized as follows: §4.5.1 contains experiments on datasets generated from random corruption model (4.1) and §4.5.2 contains experiments on real data sets.

4.5.1 Synthetic Data Experiments

In Figure 4.1 we compare the PPM for MGM algorithm with the algorithm for permutation synchronization proposed by Pachauri et al. [2] (the top eigenvectors method). We create the datasets according to the random corruption data model, defined in Equation (4.1), with fixed permutation size $m = 5$ and for each subfigure we vary the number of permutations $n = 80$, 100, 150 and 200. As we can see in all four subfigures of Figure 4.1 PPM for MGM performes better than the method proposed by Pachauri et al. [2]. Note that for larger number of permutation matrices the algorithm can handle higher corruption rate.

In Figure 4.2 we show that the PPM for MGM algorithm converges even if instead

of spectral initialization we use random initialization. We initialize the algorithm with a uniformly random $\bar{\boldsymbol{Q}}$, with $\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_n \in \text{Perm}(m)$ and run it for 100 uniformly random initial points for 3 different values of the corruption rate. We report the average results in Figure 4.2. For all experiments we fix the number of permutations to be $n = 500$ and the size of permutations to be $m = 5$. For the left figure the corruption rate is 0.89 for the middle figure the corruption rate is 0.90 and for the right figure the corruption rate is 0.91. As we can see the smaller the corruption rate is the faster algorithm converges. However, for larger corruption rates we see a slow convergence region for initial iterations. We describe this by the fact that if the initial point is not in the region of fast convergence, it needs to slowly improve the guess and eventually end up in the fast convergence region. Note, that for all 3 subfigures of Figure 4.2 there is a point after which the algorithm converges very fast, for smaller corruption rate values it happens faster than for larger corruption rate values.

4.5.2 Real Data Experiments

To test the accuracy of PPM for MGM we use the WILLOW-Object Class dataset released in [79]. There are 5 groups of images in this dataset, the first one has 109 images of faces, the second one has 50 images of ducks, the third one has 66 images of Wine Bottles, the fourth one contains 40 images of motorbikes and the fifth one contains 40 images of cars. For each image 10 feature points are manually labeled. The goal is to register the points among images that correspond to the same area of the image.

For each of the 5 datasets we start by preprocessing the images and feature points as follows: We first creat the graphs between the feature points and match pairwisely between them according to the procedure described in Cho et al. [80], which uses re-weighted random walks to match between graphs. Then, after obtaining the mutual correspondences we use the PPM for MGM algorithm. The results are presented in Table 4.1.

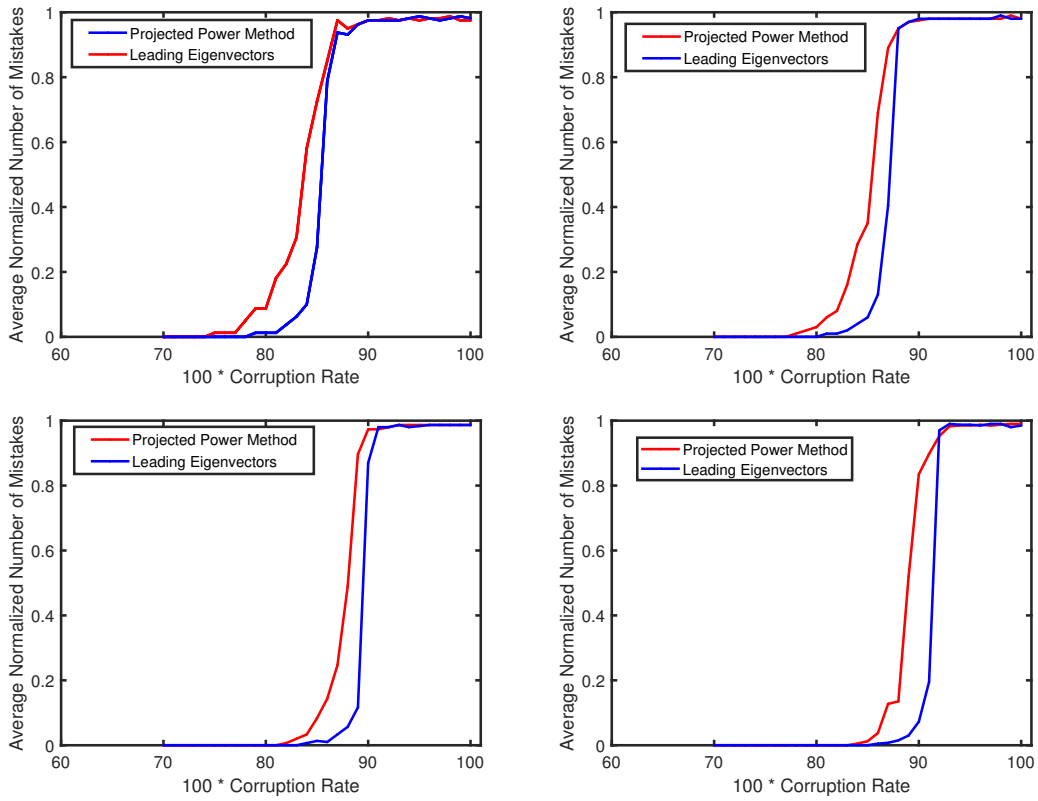


Figure 4.1: This figure demonstrates a comparison between PPM for MGM and [2], the number of permutation size is fixed to 5 and the number of permutation varies as follows: for the left figure of the first row it is 80, for the right figure of the first row it is 100, for the left figure of the second row it is 150 and for the right figure of the second row it is 200.

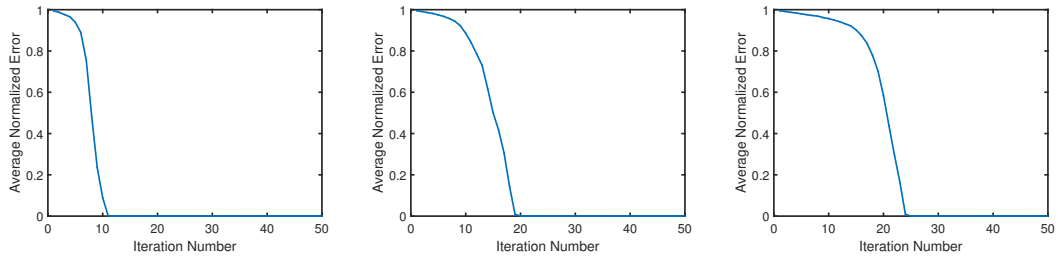


Figure 4.2: This figure demonstrates the convergence speed of the PPM for MGM with random initialization for different corruption rates. For the first figure the corruption rate is 0.89, for the second figure the corruption rate is 0.9 and for the third figure the corruption rate is 0.91.

Table 4.1: Results of PPM for MGM algorithm on the real image datasets.

Dataset	# images	# feat. points	mistake % in est.	recovery %
Car	40	10	58.1	92.5
Duck	50	10	64.2	84.0
Wine Bottle	66	10	45.5	94.0
Motorbike	40	10	45.3	97.5
Face	109	10	20.1	100.0

References

[1] Y. Yu, T. Wang, and R. Samworth. A useful variant of the Davis–Kahan theorem for statisticians. *Biometrika*, 102(2):315–323, 2014.

[2] D Pachauri, R. Kondor, and V. Singh. Solving the multi-way matching problem by permutation synchronization. In *27th NIPS: Lake Tahoe, Nevada, USA*, pages 1860–1868, 2013.

[3] J. Chen and A. H. Sayed. Diffusion adaptation strategies for distributed optimization and learning over networks. *IEEE Transactions on Signal Processing*, 60(8):4289–4305, 2012.

[4] A. Bertrand and M. Moonen. Consensus-based distributed total least squares estimation in ad hoc wireless sensor networks. *IEEE Trans. Signal Processing*, 59(5):2320–2330, 2011.

[5] B. Johansson, C.M. Carretti, and M. Johansson. On distributed optimization using peer-to-peer communications in wireless sensor networks. In *Sensor, Mesh and Ad Hoc Communications and Networks*, pages 497–505, June 2008.

[6] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.*, 3(1):1–122, January 2011.

[7] M. G. Rabbat, R. D. Nowak, and J. A. Bucklew. Generalized consensus computation in networked systems with erasure links. In *IEEE 6th Workshop on Signal Processing Advances in Wireless Communications, 2005.*, pages 1088–1092, June 2005.

[8] A. Nedić and A. E. Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Trans. Automat. Contr.*, 54(1):48–61, 2009.

[9] A. Nedić and A. Ozdaglar. Cooperative distributed multi-agent optimization. In *Convex Optimization in Signal Processing and Communications*. Cambridge University Press, 2010.

[10] W. Shi, Q. Ling, K. Yuan, G. Wu, and W. Yin. On the linear convergence of the ADMM in decentralized consensus optimization. *IEEE Trans. Signal Processing*, 62(7):1750–1761, 2014.

[11] T.H. Chang, M. Hong, and X. Wang. Multi-agent distributed optimization via inexact consensus ADMM. *IEEE Trans. Signal Processing*, 63(2):482–497, 2015.

[12] G. Mateos, J. Bazerque, and G. Giannakis. Distributed sparse linear regression. *IEEE Transactions on Signal Processing*, 10(58):5262–5276, 2010.

[13] I. Schizas, A. Ribeiro, and G. Giannakis. Consensus in ad hoc WSNs with noisy links - Part I: Distributed estimation of deterministic signals. *Signal Processing, IEEE Transactions on*, 56(1):350–364, 2008.

[14] H. Zhu, A. Cano, and G. Giannakis. Distributed consensus-based demodulation: algorithms and error analysis. *IEEE Transactions on Wireless Communications*, 9(6):2044–2054, 2010.

[15] P. Forero, A. Cano, and G. Giannakis. Consensus-based distributed support vector machines. *J. Mach. Learn. Res.*, 11:1663–1707, 2010.

[16] Y. Qu, G. Ostrouchov, N. Samatova, and A. Geist. Principal component analysis for dimension reduction in massive distributed data sets. In *SIAM International Conference on Data Mining*, 2002.

[17] H. Qi, T. Wang, and D. Birdwell. *Global Principal Component Analysis for Dimensionality Reduction in Distributed Data Mining*, chapter 19, pages 327–342. CRC Press, 2004.

[18] Z. Bai, H. C. Raymond, and T. L. Franklin. Principal component analysis for distributed data sets with updating. In *Proceedings of International workshop on Advanced Parallel Processing Technologies (APPT)*, 2005.

[19] Y. Liang, M. Balcan, V. Kanchanapally, and D. Woodruff. Improved distributed principal component analysis. In *Advances in Neural Information Processing Systems*, pages 3113–3121, 2014.

[20] Z. Meng, A. Wiesel, and A. Hero III. Distributed principal component analysis on networks via directed graphical models. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 2877–2880. IEEE, 2012.

[21] M. Valcarcel, P. Belanovic, and S. Zazo. Consensus-based distributed principal component analysis in wireless sensor networks. In *11th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 1–5. IEEE, 2010.

[22] A. Aduroja, I. D. Schizas, and V. Maroulas. Distributed principal components analysis in sensor networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 5850–5854, May 2013.

[23] G. Lerman and T. Maunu. An Overview of Robust Subspace Recovery. *ArXiv e-prints*, 2018, 1803.01013.

[24] T. Zhang and G. Lerman. A novel M-estimator for robust pca. *J. Mach. Learn. Res.*, 15(1):749–808, January 2014.

[25] G. Lerman, M. McCoy, J. Tropp, and T. Zhang. Robust computation of linear models by convex relaxation. *Foundations of Computational Mathematics*, 15(2):363–410, 2015.

[26] G. Lerman and T. Maunu. Fast, robust and non-convex subspace recovery. *Information and Inference: A Journal of the IMA*, pages 1–60, 2017.

[27] D. P. Bertsekas. *Convex analysis and optimization.* Athena Scientific, Belmont, MA, 2003. With Angelia Nedić and Asuman E. Ozdaglar.

[28] R. T. Rockafellar. *Convex analysis*. Princeton Mathematical Series, No. 28. Princeton University Press, Princeton, N.J., 1970.

[29] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, Cambridge, 2004.

[30] R. H. Bartels and G. W. Stewart. Solution of the matrix equation AX+XB=C [F4] (algorithm 432). *Commun. ACM*, 15(9):820–826, 1972.

[31] K. L. Clarkson and D. P. Woodruff. Input sparsity and hardness for robust subspace approximation. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 310–329. IEEE, 2015.

[32] N. Halko, P. Martinsson, and J. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[33] M. Lichman. UCI Machine Learning Repository, 2013.

[34] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In *21st European Symposium on Artificial Neural Networks ESANN, Bruges, Belgium*, 2013.

[35] T. Sarlos. Improved approximation algorithms for large matrices via random projections. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 143–152, Oct 2006.

[36] S. M. Kakade, S. Shalev-Shwartz, and A. Tewari. Regularization techniques for learning with matrices. *J. Mach. Learn. Res.*, 13:1865–1890, 2012.

[37] E. Weiszfeld. Sur le point pour lequel la somme des distances de n points donnes est minimum. *Tohoku Mathematical Journal*, 43:355 – 386, 1937.

[38] T. Chan and P. Mulet. On the convergence of the lagged diffusivity fixed point method in total variation image restoration. *SIAM Journal on Numerical Analysis*, 36(2):354–367, 1999, http://dx.doi.org/10.1137/S0036142997327075.

[39] R. Bhatia and L. Elsner. Positive linear maps and the Lyapunov equation. In I. Gohberg and H. Langer, editors, *Linear Operators and Matrices*, volume 130 of *Operator Theory: Advances and Applications*, pages 107–120. Birkhuser, Basel, 2002.

[40] M. Ostrowski. *Solution of equations and systems of equations.* Pure and applied mathematics. Academic Press, 1966.

[41] B. J Brown, C. Toler-Franklin, D. Nehab, M. Burns, D. P. Dobkin, A. Vlachopoulos, C. Doumas, S. Rusinkiewicz, and T. Weyrich. A system for high-volume acquisition and matching of fresco fragments: reassembling Theran wall paintings. *ACM Trans. Graph.*, 27(3):84:1–84:9, 2008.

[42] D. Koller and M. Levoy. Computer-aided reconstruction and new matches in the forma urbis romae. *Bullettino Della Commissione Archeologica Comunale di Roma*, pages 103–125, 2006.

[43] H. Liu, S. Cao, and S. Yan. Automated assembly of shredded pieces from multiple photos. *IEEE Transactions on Multimedia*, 13(5):1154–1162, 2011.

[44] A. Deever and A. Gallagher. Semi-automatic assembly of real cross-cut shredded documents. In *Image Processing (ICIP), 2012 19th IEEE International Conference on*, pages 233–236. IEEE, 2012.

[45] W. Marande and G. Burger. Mitochondrial dna as a genomic jigsaw puzzle. *Science*, 318(5849):415–415, 2007.

[46] Z. Yu-Xiang, S. Mu-Chun, C. Zhong-Lie, and L. Jonathan. A puzzle solver and its application in speech descrambling. In *Proceedings of the 2007 WSEAS International Conference on Computer Engineering and Applications, Gold Coast, Australia, January 17-19, 2007*, pages 171–176, 2007.

[47] T. Altman. Solving the JIGSAW puzzle problem in linear time. *Applied Artificial Intelligence*, 3(4):453–462, 1989.

[48] E. D. Demaine and M. L. Demaine. Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity. *Graphs and Combinatorics*, 23(Supplement-1):195–208, 2007.

[49] H. Freeman and L. Garder. Apictorial jigsaw puzzles: The computer solution of a problem in pattern recognition. *IEEE Transactions on Electronic Computers*, 13(2):118–127, 1964.

[50] A. C. Gallagher. Jigsaw puzzles with pieces of unknown orientation. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pages 382–389, 2012.

[51] D. Pomeranz, M. Shemesh, and O. Ben-Shahar. A fully automated greedy square jigsaw puzzle solver. In *The 24th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2011, Colorado Springs, CO, USA, 20-25 June 2011*, pages 9–16, 2011.

[52] F. A Andaló, G. Taubin, and S. Goldenstein. Solving image puzzles with a simple quadratic programming formulation. In *Graphics, Patterns and Images (SIBGRAPI), 2012 25th SIBGRAPI Conference on*, pages 63–70. IEEE, 2012.

[53] K. Son, J. Hays, and D. B Cooper. Solving square jigsaw puzzles with loop constraints. In *European Conference on Computer Vision*, pages 32–46. Springer, 2014.

[54] K. Son, D. Moreno, J. Hays, and D. B. Cooper. Solving small-piece jigsaw puzzles by growing consensus. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1193–1201, 2016.

[55] T. S. Cho, S. Avidan, and W. T Freeman. A probabilistic image jigsaw puzzle solver. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 183–190. IEEE, 2010.

[56] D. Sholomon, O. E. David, and N. S. Netanyahu. A generalized genetic algorithm-based solver for very large jigsaw puzzles of complex types. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 2839–2845, 2014.

[57] D. Sholomon, O. E. David, and N. S. Netanyahu. An automatic solver for very large jigsaw puzzles using genetic algorithms. *Genetic Programming and Evolvable Machines*, 17(3):291–313, 2016.

[58] D. J. Hoff and P. J. Olver. Automatic solution of jigsaw puzzles. *Journal of Mathematical Imaging and Vision*, 49(1):234–250, 2014.

[59] R. Yu, C. Russell, and L. Agapito. Solving jigsaw puzzles with linear programming. In *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*, 2016.

[60] A. Singer and H-T Wu. Vector diffusion maps and the connection laplacian. *Communications on pure and applied mathematics*, 65(8):1067–1144, 2012.

[61] S. Z. Kovalsky, D. Glasner, and R. Basri. A global approach for solving edge-matching puzzles. *SIAM J. Imaging Sciences*, 8(2):916–938, 2015.

[62] N. El Karoui, H-T Wu, et al. Graph connection laplacian methods can be made robust to noise. *The Annals of Statistics*, 44(1):346–372, 2016.

[63] S. Zhang and Y. Huang. Complex quadratic optimization and semidefinite programming. *SIAM Journal on Optimization*, 16(3):871–890, 2006.

[64] A. Singer. Angular synchronization by eigenvectors and semidefinite programming. *Applied and computational harmonic analysis*, 30(1):20–36, 2011.

[65] C. Davis and W. M. Kahan. The rotation of eigenvectors by a perturbation. iii. *SIAM Journal on Numerical Analysis*, 7(1):1–46, 1970.

[66] R. Bhatia. *Matrix Analysis*. Graduate Texts in Mathematics. Springer New York, 1996.

[67] P. Jaccard. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bulletin del la Société Vaudoise des Sciences Naturelles*, 37:547–579, 1901.

[68] Q. Huang and Guibas L. J. Consistent shape maps via semidefinite programming. *Comput. Graph. Forum*, 32(5):177–186, 2013.

[69] M. Goesele, N. Snavely, Curless B., H. Hoppe, and S. M. Seitz. Multi-view stereo for community photo collections. In *IEEE 11th International Conference on Computer Vision, ICCV 2007, Rio de Janeiro, Brazil, October 14-20, 2007*, pages 1–8, 2007.

[70] S. Agarwal, Y. Furukawa, N. Snavely, I. Simon, B. Curless, S. M. Seitz, and R. Szeliski. Building rome in a day. *Commun. ACM*, 54(10):105–112, 2011.

[71] Y. Chen, L. J. Guibas, and Q. Huang. Near-optimal joint object matching via convex relaxation. In *Proceedings of the 31th ICML, Beijing, China*, pages 100–108, 2014.

[72] D. Pachauri, M. D. Collins, and V. Singh. Incorporating domain knowledge in matching problems via harmonic analysis. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*, 2012.

[73] X. Shi, H. Ling, W. Hu, J. Xing, and Y. Zhang. Tensor power iteration for multi-graph matching. In *2016 IEEE Conference on CVPR, Las Vegas, NV, USA*, pages 5062–5070, 2016.

[74] J. Yan, M. Cho, H. Zha, X. Yang, and S. M. Chu. A general multi-graph matching approach via graduated consistency-regularized boosting. *CoRR*, abs/1502.05840, 2015, 1502.05840.

[75] Y. Chen and E. Candes. The projected power method: An efficient algorithm for joint alignment from pairwise differences. *arXiv preprint arXiv:1609.05820*, 2016.

[76] N. Boumal. Nonconvex phase synchronization. *SIAM Journal on Optimization*, 26(4):2355–2377, 2016.

[77] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[78] J. A. Tropp. An introduction to matrix concentration inequalities. *Foundations and Trends in Machine Learning*, 8(1-2):1–230, 2015.

[79] M. Cho, Alahari K., and Ponce J. Learning graphs to match. In *IEEE International Conference on Computer Vision, ICCV 2013, Sydney, Australia, December 1-8, 2013*, pages 25–32, 2013.

[80] M. Cho, J. Lee, and K. M. Lee. Reweighted random walks for graph matching. In *Computer Vision - ECCV 2010 - 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part V*, pages 492–505, 2010.