POLITECNICO DI MILANO
DIPARTIMENTO DI ELETTRONICA INFORMAZIONE E BIOINGEGNERIE
MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

SYNCHRONIZATION ON GROUP-LABELLED MULTIGRAPHS

M.Sc. Thesis by:
Andrea Porfiri Dal Cin, 927948

Supervisor:
Prof. Luca Magri

Co-supervisor:
Prof. Giacomo Boracchi

Academic Year 2020-2021

Abstract

GROUP synchronization refers to the problem of inferring the unknown values attached to vertices of a graph where edges are labelled with the ratio of the incident vertices and labels belong to a group. This work addresses for the first time the group synchronization problem on multigraphs, that is, graphs with more than one edge connecting the same pair of vertices. The problem arises naturally when multiple measures are available to model the relationship between two vertices. This happens when different sensors measure the same quantity or when the original graph is partitioned into subgraphs that are solved independently. In this case the relationships among subgraphs give rise to multi-edges and the problem can be traced back to a multigraph synchronization problem. The solutions appeared so far reduce multigraphs to simple ones by averaging their multi-edges, however this approach falls short because: i) has been studied only for some groups and ii) the resulting estimator is less statistically efficient than our solution, as we prove empirically. Specifically, we present a solution based on a principled constrained eigenvalue optimization that copes with general groups and can be profitably used both on synthetic and real multigraph synchronization problems.

Sommario

La sincronizzazione su gruppi è il problema di riconstruire i valori sconosciuti corrisponenti ai vertici di un grafo etichettato, i cui archi sono anch'essi etichettati dal rapporto tra gli elementi associati ai due vertici incidenti. In questo elaborato si considera, per la prima volta, il caso in cui la sincronizzazione è applicata a multigrafi, ovvero grafi che ammettono più di un singolo arco tra una qualsiasi coppia di vertici. Questo problema si presenta naturalmente quando sensori diversi misurano la stessa quantità o quando il grafo originale è partizionato in sotto-grafi indipendentemente sincronizzati. In questo caso, le relazioni tra sotto-grafi danno origine a multi-archi e il problema può essere ricondotto a un problema di sincronizzazione su multigrafo. Le soluzioni attuali riducono il multigrafo ad un grafo semplice facendo la media degli elementi associati ai multi-archi. Questo approccio presenta serie limitazioni in quanto: i) è stato finora considerato solo per alcuni gruppi and ii) lo stimatore che si ottiene è meno efficiente a livello statistico della nostra soluzione, come dimostrato empiricamente. Nello specifico, in questo elaborato presentiamo una soluzione basata su un problema di ottimizzazione di autovalori vincolati che non è limitata a gruppi specifici e che garantisce ottimi risultati su grafi sia sintetici sia reali.

Acknowledgements

I cannot express enough thanks to the incredible team who helped me throughout this journey. To Luca, Giacomo, Federica and Andrea, thank you very much for your incredible support and the opportunity you have given me.

To my family and friends who supported me for the last 5 years, thank you from the bottom of my heart.

Contents

Notation

Multidimensional objects

- a: is a generic element, be it in scalar, nominal or vectorial form
- $\mathbf{a}$: is a vector
- $\mathbf{A}$: is a matrix
- $\mathbf{I}_d$: is an identity $d \times d$ matrix
- $\mathbf{1}_d$: is a $1 \times d$ vector filled with ones

Sets

- $\mathcal{S}$: is a finite set. The cardinality of $\mathcal{S}$ is $|\mathcal{S}|$. In addition, $\mathcal{S}$ can be an empty set $\emptyset$ or $\{\}$
- $\mathbb{N}, \mathbb{R}, \mathbb{Z}$: are the infinite numerical sets for natural, real and integer respectively

Operations

- $\mathbf{A}^\intercal$: is the transposition for a matrix $\mathbf{A}$
- $\text{tr}(\mathbf{A})$: is the linear trace operator of a square matrix $\mathbf{A}$
- $||\mathbf{A}||_F$: is the Frobenius norm of $\mathbf{A}$
- $\mathbf{A} \otimes \mathbf{B}$: is the Kronecker product of $\mathbf{A}$ and $\mathbf{B}$
- $\mathbf{A} \circ \mathbf{B}$: is the Hadamard product (entry-wise product) of $\mathbf{A}$ and $\mathbf{B}$

- $\mathcal{S} \times \mathcal{R}$: is the Cartesian product between sets $\mathcal{S}$ and $\mathcal{R}$

Matrix groups and semigroups

- $GL(d) = \{\mathbf{M} \in \mathbb{R}^{d \times d} \text{ s.t. } det(\mathbf{M}) \neq 0\}$: the General Linear Group is the set of invertible matrices
- $SL(d) = \{\mathbf{M} \in \mathbb{R}^{d \times d} \text{ s.t. } det(\mathbf{M}) = 1\}$: the Special Linear Group is the set of invertible matrices with unit determinant
- $O(d) = \{\mathbf{M} \in \mathbb{R}^{d \times d} \text{ s.t. } \mathbf{M}^\top \mathbf{M} = \mathbf{M}\mathbf{M}^\top = \mathbf{I}_d\}$: the Orthogonal Group is the set of rotations and reflections
- $SO(d) = \{\mathbf{M} \in O(d) \text{ s.t. } det(\mathbf{M}) = 1\}$: the Special Orthogonal Group is the set of rotations
- $SE(d)$: the Special Euclidean Group is the set of direct isometries
- $GA(d)$: the General Affine Group is the set of affine maps
- $Sym(d) = \{\mathbf{M} \in \{0,1\}^{d \times d} \text{ s.t. } \mathbf{M}\mathbf{1}_d = \mathbf{1}_d, \mathbf{1}_d\mathbf{M} = \mathbf{1}_d\}$: the Symmetric Group is the set of total permutations
- $\mathbb{R}^*$: the set of projectively extended real numbers

Any exception of the notation as above is locally explained in this thesis to avoid misunderstanding.

CHAPTER 1

Introduction

Many tasks in Computer Vision can be formulated as the *synchronization* [34] of group-labelled graphs: given a network of nodes labeled with unknown elements of a group Σ, the goal is to estimate them from a set of noisy relative measurements expressed as ratios (or differences) attached to edges. Depending on the group Σ, the synchronization formulation can be exploited to model several relevant problems in Computer Vision, including structure from motion, simultaneous localization and mapping (SLAM), multi-view matching and image mosaicking. A prominent example is when the vertices of the graph are sensors and the goal is to estimate the unknown attitude and location of each sensor in a common reference frame [20]. In this case, labels are in the Special Euclidean Group $\Sigma = SE(3)$, and the pairwise measurements are relative orientations between sensors.

1.1 Scenario and Problem statement

Traditionally, the synchronization problem is defined on *simple* graphs, that is graphs in which any pair of vertices is connected by at most one edge. In this work, we explore the possibility of dealing with the case in which *multiple* measurements are available for the same pair of vertices, meaning that

multiple edges (multi-edges) can exist in the graph. Thus, synchronization is performed on a *multigraph* rather than on a simple graph.

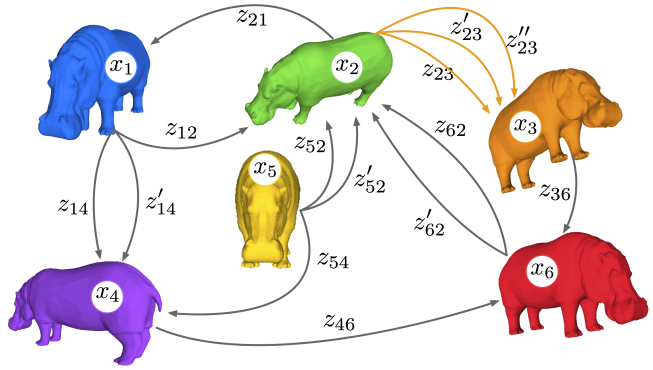

Figure 1.1: *A multigraph is a graph that admits multiple edges between its vertices. In this figure, vertices correspond to unknown group elements $x_i \in SO(3)$ and edges correspond to known relative measures. A multi-edge with cardinality 3, resulting for instance from different estimates of relative transformations, is depicted in orange.*

This novel and general multigraph synchronization framework allows us to account for multiple measurements between the same pair of vertices, which often happens in many applications. For instance, in SLAM multiple sensors (cameras, IMU, GPS, ...) can estimate the 6 d.o.f motion of a vehicle. Another scenario where multigraphs naturally arise is in large-scale problems, where the original measurement graph is partitioned into smaller subgraphs that are solved independently to speed up performance. In this context, the vertices shared across sub-problems or the cut-edges connecting vertices from different sub-graphs generate multi-edges and yield a multigraph synchronization problem, as will be clarified in Chapter 6. This approach not only reduces memory and processing time, but also enables multi-threading and parallelism to a greater extent.

The naive solution to synchronization on multigraphs – henceforth named *edge averaging* – consists in turning all multi-edges in the graph into simple ones by averaging their measurements. This approach suffers from several shortcomings. First, averaging is well defined only for some groups: while it is possible to average rotations [22], there is not a principled solution for homographies. Secondly, even when it is possible to average multi-edges, the resulting estimator has sub-optimal statistical properties. As an example, consider the problem of estimating a scale factor s that links two matrices with noisy entries: $\mathbf{A} = s\mathbf{B}$. The optimal estimate is the least squares solution $s = \text{tr}(\mathbf{B}^\intercal\mathbf{A})/\text{tr}(\mathbf{B}^\intercal\mathbf{B})$ which is different from, *e.g.*, tak-

ing the average of the entry-wise division $\mathbf{A}./\mathbf{B}$. Our experiments confirm that this intuition holds also for synchronization.

1.2 Contributions

Our solution approaches synchronization of group-labeled multigraphs from a new perspective: rather than averaging measures to collapse a multi-edge to a simple one, it expands the multigraph by replicating vertices with incoming or outgoing multi-edges and enforces identity constraints between the replicas of the same vertices. This leads to a constrained optimization problem for which we derive a general closed-form spectral solution that can be applied to graphs labeled with any linear group. Our experiments, performed on both synthetic and real data sets, demonstrate that our solution outperforms edge averaging in terms of accuracy and precision. In the context of partitioned problems, our solution strikes a good balance between accuracy and complexity, as opposed to performing synchronization on the whole graph. To summarize, our contribution is three-fold:

- we present, for the first time, a formal definition of the synchronization of group-labeled multigraphs, which is a significant extension of the synchronization of simple graphs;
- we derive a practical algorithm for solving a synchronization problem on a multigraph, which is based on an expansion algorithm coupled with a constrained spectral solution to deal with replicated vertices;
- we demonstrate how the multigraph framework can be conveniently used to partition classical synchronization tasks, achieving a good trade-off between accuracy and complexity.

1.3 Structure of Thesis

The work is organized as follows. Chapter (2) reviews previous works related to group synchronization. Chapter (3) formally introduces the group synchronization problem and closed-form solutions for it. Chapter (4) provides the theoretical footing and introduces an algorithm for turning any measurement multigraph into a simple measurement graph. Chapter (5) presents our solution to the synchronization problem on multigraphs and reports the results of experiments performed on synthetic graphs. Chapter (6) describes partitioned synchronization as a possible application of multigraph synchronization and reports the results obtained from experiments on real-world datasets. Conclusions are drawn in Chapter (7).

CHAPTER 2

Related work

The synchronization problem has a rich literature thanks to its many applications in the field of Computer Vision. The name *synchronization* is derived from clock synchronization, that is, the problem of finding the time offset with respect to a global reference for multiple connected clocks. Depending on the group chosen for synchronization, we obtain specific instances of the problem, which relate to a specific application.

In this section, we review the existing applications of synchronization in Computer Vision and the approaches for solving the problem on a *simple graph*. The case of a multigraph is not considered in previous works as it is introduced in this work for the first time.

If we consider the Special Orthogonal Group $SO(3)$, solving the problem results in rotation synchronization, which is also known as multiple rotation averaging or rotation optimization. Although only closed-form solutions will be covered in this work, rotation synchronization can be solved using a vast array of techniques, including: spectral decomposition [34], the Weiszfeld algorithm [21], Lie-group optimization [12], semidefinite programming [41], distributed optimization [39], low-rank decomposition [2], Riemannian optimization [9], deep learning [28] and message passing [32].

If the Special Euclidean Group (i.e., $\Sigma = SE(3)$) is considered, it results

in *rigid-motion synchronization*, which is also called *motion averaging* or *pose-graph optimization*. Existing techniques include spectral decomposition [1], Lie-group optimization [19], diffusion over dual quaternions [37], Riemannian optimization [38], semidefinite programming [29], distributed optimization [36], Bayesian optimization [11] and deep learning [24]. Both rotation and rigid-motion synchronizations can be applied to structure from motion, registration of 3D point clouds and simultaneous localization and mapping.

If we consider the Symmetric Group (i.e., $\Sigma = Sym(d)$), then we get *permutation synchronization*, which finds application in multi-view matching. Available approaches include spectral decomposition [6], Gauss-Seidel relaxation [43], distributed optimization [26], and Riemannian optimization [10].

Other instances of synchronization concern the Special Linear Group (i.e., $\Sigma = SL(3)$), which is used to represent homographies in image mosaicking [5], and the General Affine Group (i.e., $\Sigma = GA(3)$), which has been used to solve for global color matching in image mosaicking [30].

CHAPTER 3

Group synchronization

3.1 Introduction

In this chapter, we provide an overview of the group synchronization problem, how it can be solved and its applications in the field of Computer Vision. Most of the concepts introduced in this section can be found in the literature and can be skipped if the reader is already familiar with group synchronization and its closed-form solutions. In the following chapters, we will refer to the notation and properties introduced in the following sections.

Let us begin by providing an intuition that should help the reader familiarize with the basic idea behind the group synchronization problem. Consider a situation in which there are multiple clocks placed at different locations. The goal is to make sure that all clocks display the same time, an idea that is akin to the Poincaré-Einstein synchronization for clocks. This problem can be formulated as a network in which each node represents a clock and each link identifies the ability of two clocks to communicate and thus measure clock differences. Without any loss of generality, we can assume that the time for any clock A is defined as an offset o_A from a global reference. We can estimate the propagation delay between any two clocks

A and B, that is the time that it takes for a message sent from A to be received by B and vice versa, as follows:

$$o_{AB} = \frac{t_{B,1} - t_{A,1} + t_{B,2} - t_{A,2}}{2} \tag{3.1}$$

Solving the synchronization problem means finding the offset o_A of each clock or node from the relative propagation delays o_{AB} that are available.

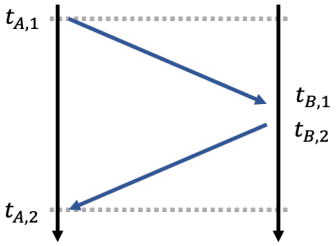

Figure 3.1: *A graph showing how two clocks A and B can exchange messages and measure the propagation delay between them.*

Many tasks in Computer Vision can be formulated as group synchronization problems on graphs. A prominent example is registration, in which nodes represent sensors and edges represent relative measurements, that is pairwise rotations, between nodes. The goal is to recover the unknown attitude of each sensor to register them in a common reference frame. Depending on what the nodes in the graph represent, synchronization can be used to model several other well-known Computer Vision problems, including multi-view matching, structure from motion, image mosaicking and color correction.

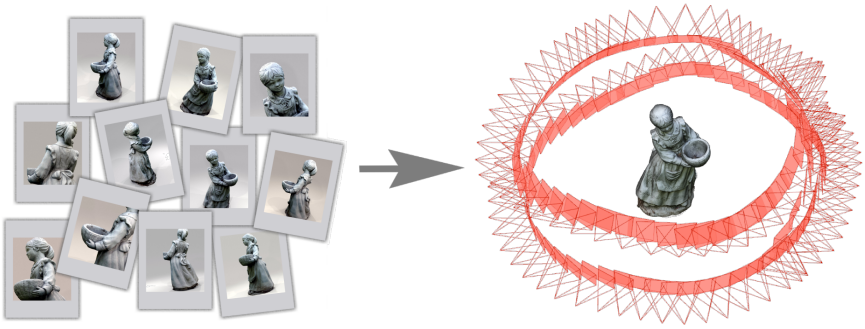

Figure 3.2: *Structure from Motion (SfM), one of the most prominent applications of group synchronization. Image from [8]*

In this chapter we provide the theoretical framework for group synchro-

nization on measurement graphs and introduce two well-known closed-form solutions to such problem.

3.2 Theoretical framework

In this section, we provide a brief theoretical background that will cover the main concepts and definitions concerning the group synchronization problem. The notation and structure used throughout this section is inspired by the work in [3], in which the reader can find more detailed descriptions and formal proofs of the following theoretical results.

Let us describe in an informal way what the problem of *group synchronization* is. Consider a network of nodes, each characterized by an unknown state, and assume that the only information available is a set of relative measurements between such nodes. The goal of the synchronization problem is to provide an estimate for the states of the nodes from the pairwise measurements available in the network.

Specifically, the group synchronization problem is defined on a measurement graph in which states are represented by elements of a group Σ. The goal of such problem is to recover the absolute or global states from a set of pairwise measurements, which take the form of edges connecting any two vertices in the graph. Thus, a measurement graph is not simply a graph defined by a vertex set $\mathcal{V}$ and an edge set $\mathcal{E}$ with $(i,j) \in \mathcal{E}$ and $i,j \in \mathcal{V}$. Instead, it can be seen as a group-labeled graph in which vertices and edges are assigned labels or elements of a group. Let us provide a formal definition:

Definition 3.2.1. ***(Σ-labeled graph)***
Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with $\mathcal{V}$ being its vertex set and $\mathcal{E}$ being its edge set, and a group Σ with unit element 1_Σ. A Σ-labeled graph is described as a tuple $\Gamma = (\mathcal{V}, \mathcal{E}, z)$ where:

$$z : \mathcal{E} \to \Sigma \tag{3.2}$$

In addition, if $(i,j) \in \mathcal{E}$, then $(j,i) \in \mathcal{E}$ and z satisfies the following property:

$$z(j,i) = z(i,j)^{-1} \tag{3.3}$$

Therefore, since all edges have a counterpart that connects the same vertices but with opposite direction, a group-labeled graph can be seen as an undirected graph.

Let us define what a null cycle is in the context of group-labeled graphs and group synchronization:

Definition 3.2.2. ***(Null cycle)***
Consider a Σ-labeled graph $\Gamma = (\mathcal{V}, \mathcal{E}, z)$, a circuit $\{(i_1, i_2), (i_2, i_3), \ldots, (i_l, i_1)\}$ with $i_k \in \mathcal{V}$ is a null cycle if and only if the composition of the edge labels for such path is equal to the identity for group Σ.

$$z(i_1, i_2) * z(i_2, i_3) * \ldots * z(i_l, i_1) = \mathbf{1}_\Sigma \tag{3.4}$$

Intuitively, the definition of null cycle resembles that of the Kirchoff's voltage law. The concept of null cycle is very important in the context of group synchronization: if we can find a null cycle in a measurement graph, we can recover the exact vertex labeling for the vertices along that path. It is possible to extend this idea to the whole graph by assuming that, if a measurement graph does not contain non-null cycles, it is possible to recover a vertex labeling that induces an edge labeling such that all cycles are still null.

Definition 3.2.3. ***(Consistent labeling)***
Consider a Σ-labeled graph $\Gamma = (\mathcal{V}, \mathcal{E}, z)$, a consistent labeling for Γ is a function $x : \mathcal{V} \to \Sigma$ such that:

$$z(e) = x(i) * x(j)^{-1} \quad \forall e = (i, j) \in \mathcal{E} \tag{3.5}$$

Equation (3.5) is referred to as the *consistency constraint* and states that a vertex labeling is consistent for a group-labeled graph if every induced edge label is equal to the ratio of the labels of the vertices that are connected by such edge.

Corollary 1. *A group-labeled graph admits a consistent labeling if and only if it does not contain a non-null cycle.*

In other words, only measurement graphs that contain non-null cycles, meaning that they do not include any noise in their measurements, admit a consistent labeling. Solving the synchronization problem on a group-labeled graph is equivalent to finding a consistent vertex labeling for such graph. Of course, we wish to solve the synchronization problem even on noisy measurement graphs, for which it is not possible to recover a perfectly consistent vertex labeling. Thus, we can introduce the concept of *consistency error* to evaluate how much a vertex labeling and its induced edge labeling are compatible with the labels assigned to the edges of a measurement graph.

Definition 3.2.4. ***(Consistency error)***
Let Γ be a Σ-labeled graph and $\tilde{x} : \mathcal{V} \to \Sigma$ be a vertex labeling. The

consistency error of $\tilde{x}$ is defined as such:

$$\epsilon(\tilde{x}) = \sum_{(i,j)\in\mathcal{E}} \rho(\delta(\tilde{z}(i,j), z(i,j))) \tag{3.6}$$

where ρ is a loss function $\rho : \mathbb{R}^+ \to \mathbb{R}^+$ with the properties described in [3] and δ is a metric function $\delta : \Sigma \times \Sigma \to \mathbb{R}^+$.

If a vertex labeling is consistent we can see that the edge labeling $\tilde{z}$ induced by $\tilde{x}$ coincides with the measured edge labeling z. Thus, if a vertex labeling is consistent it introduces a null consistency error. In the general case, if a measurement graph is noisy, the consistency error ϵ for such graph is not null.

3.3 Synchronization over GL(d)

In this section, we introduce two well-known closed-form solutions to the synchronization problem on $GL(d)$-labeled measurement graphs, namely the *spectral solution* and the *null-space solution*. $GL(d)$ is a very general group that includes all real $d \times d$ invertible matrices:

$$GL(d) = \{\mathbf{M} \in \mathbb{R}^{d\times d} \mid \det(\mathbf{M}) \neq 0\} \tag{3.7}$$

The goal is to provide an overview of the main synchronization techniques that are going to be employed for the rest of this work. For further details and formal proofs of the following concepts, we invite the reader to refer to [3].

3.3.1 Definitions

Let us consider a $GL(d)$-labeled graph Γ. If Γ is complete, it is possible to define matrices $\mathbf{X} \in \mathbb{R}^{dn\times d}$ and $\mathbf{Z} \in \mathbb{R}^{dn\times dn}$ that collect the vertex labels and the edge labels for Γ respectively:

$$\mathbf{X} = \begin{pmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \\ \dots \\ \mathbf{X}_n \end{pmatrix} \tag{3.8}$$

$$\mathbf{Z} = \begin{pmatrix} \mathbf{I}_d & \mathbf{Z}_{12} & \dots & \mathbf{Z}_{1n} \\ \mathbf{Z}_{21} & \mathbf{I}_d & \dots & \mathbf{Z}_{2n} \\ \dots & \dots & \dots & \dots \\ \mathbf{Z}_{n1} & \mathbf{Z}_{n2} & \dots & \mathbf{I}_d \end{pmatrix} \tag{3.9}$$

Let us rewrite the consistency constraint in matrix form as follows:

$$\mathbf{Z} = \begin{pmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \\ \dots \\ \mathbf{X}_n \end{pmatrix} \begin{pmatrix} \mathbf{X}_1^{-1} & \mathbf{X}_2^{-1} & \dots & \mathbf{X}_n^{-1} \end{pmatrix} = \mathbf{X}\mathbf{X}^{-} \tag{3.10}$$

where $\mathbf{X}^{-}$ is not the inverse of $\mathbf{X}$, but the matrix obtained by inverting the sub-matrices $\mathbf{X}_i$ in $\mathbf{X}$.

We can prove that Equation (3.10) holds by considering the consistency constraint defined in Equation (3.5) and its matrix form $\mathbf{Z}_{ij} = \mathbf{X}_i\mathbf{X}_j^{-1}$:

$$\begin{aligned} \mathbf{Z} &= \begin{pmatrix} \mathbf{X}_1\mathbf{X}_1^{-1} & \mathbf{X}_1\mathbf{X}_2^{-1} & \dots & \mathbf{X}_1\mathbf{X}_n^{-1} \\ \mathbf{X}_2\mathbf{X}_1^{-1} & \mathbf{X}_2\mathbf{X}_2^{-1} & \dots & \mathbf{X}_2\mathbf{X}_n^{-1} \\ \dots & \dots & \dots & \dots \\ \mathbf{X}_n\mathbf{X}_1^{-1} & \mathbf{X}_n\mathbf{X}_2^{-1} & \dots & \mathbf{X}_n\mathbf{X}_n^{-1} \end{pmatrix} = \\ &= \begin{pmatrix} \mathbf{I}_d & \mathbf{Z}_{12} & \dots & \mathbf{Z}_{1n} \\ \mathbf{Z}_{21} & \mathbf{I}_d & \dots & \mathbf{Z}_{2n} \\ \dots & \dots & \dots & \dots \\ \mathbf{Z}_{n1} & \mathbf{Z}_{n2} & \dots & \mathbf{I}_d \end{pmatrix} \end{aligned} \tag{3.11}$$

If Γ is not complete, the matrix of edge labels $\mathbf{Z}$ should not contain labels $\mathbf{Z}_{ij}$ and $\mathbf{Z}_{ji}$ if $(i, j) \notin \mathcal{E}$. Therefore, we wish to mask or zero the corresponding elements from matrix $\mathbf{Z}$. In order to achieve this result, we define a new matrix $\mathbf{Z}_{\mathbf{A}}$ for incomplete graphs:

$$\mathbf{Z}_{\mathbf{A}} = \mathbf{Z} \circ (\mathbf{A} \otimes \mathbf{1}_{d\times d}) \tag{3.12}$$

where $\circ$ stands for the Hadamard product and $\otimes$ for the Kronecker product. $\mathbf{A}$ is the adjacency matrix for the graph and is used to determine whether to zero one element of $\mathbf{Z}$ based on the existence or absence of an edge in the graph.

Let us define the consistency error for the group synchronization problem on $GL(d)$:

$$\epsilon(\mathbf{X}) = ||\mathbf{Z}_{\mathbf{A}} - (\mathbf{X}\mathbf{X}^{-}) \circ (\mathbf{A} \otimes \mathbf{1}_{d\times d})||_F^2 \tag{3.13}$$

where $||\cdot||_F$ indicates the Frobenius norm. The goal of group synchronization is to find a vertex labeling $\mathbf{X}$ that minimizes $\epsilon(\mathbf{X})$. Two closed-form solutions exist to solve this problem, the *spectral* solution and the *null-space* solution.

3.3.2 Spectral solution

The spectral solution is the best known closed-form solution for group synchronization on $GL(d)$. If we assume that the graph is complete and that there is no noise in the measurements ($\epsilon = 0$), from the definitions of $\mathbf{Z}$ and $\mathbf{X}$ we obtain:

$$\mathbf{Z}\mathbf{X} = n\mathbf{X} \tag{3.14}$$

Therefore, the problem of recovering $\mathbf{X}$ from the pairwise measurements $\mathbf{Z}$ becomes a matter of finding the d independent eigenvectors of $\mathbf{Z}$ that correspond to the eigenvalue n. In Equation (3.10), we see that $\mathbf{Z}$ is of rank d by construction, thus n is the largest eigenvalue of $\mathbf{Z}$ and has multiplicity d.

For the incomplete graph case, Equation (3.14) generalizes to:

$$\begin{aligned} \mathbf{Z}_\mathbf{A}\mathbf{X} &= (\mathbf{D} \otimes \mathbf{I}_d)\mathbf{X} \\ (\mathbf{D} \otimes \mathbf{I}_d)^{-1}\mathbf{Z}_\mathbf{A}\mathbf{X} &= \mathbf{X} \end{aligned} \tag{3.15}$$

where $\mathbf{D}$ is the degree matrix for the measurement graph computed from the adjacency matrix as follows:

$$\mathbf{D} = \text{diag}(\mathbf{A}\mathbf{1}_{n\times 1}) \tag{3.16}$$

given that:

$$\sum_{j|(i,j)\in\mathcal{E}} \mathbf{Z}_{ij}\mathbf{X}_j = [\mathbf{D}]_{ii}\mathbf{X}_i \tag{3.17}$$

It can be proven that matrix $(\mathbf{D} \otimes \mathbf{I}_d)^{-1}\mathbf{Z}_\mathbf{A}$ has real eigenvalues and that the largest eigenvalue is 1 with multiplicity d. Thus, the problem of recovering $\mathbf{X}$ for an incomplete graph in the case of $\epsilon = 0$ can be solved by finding the d eigenvectors that correspond to the leading eigenvalue of $(\mathbf{D} \otimes \mathbf{I}_d)^{-1}\mathbf{Z}_\mathbf{A}$, i.e. 1.

In the noisy case, when $\epsilon \neq 0$, the vertex labels are estimated by computing the eigenvectors corresponding to the d largest eigenvalues of $(\mathbf{D} \otimes \mathbf{I}_d)^{-1}$. The reason is that, in the general case, the leading eigenvalue of $(\mathbf{D} \otimes \mathbf{I}_d)^{-1}\mathbf{Z}_\mathbf{A}$ will not be equal to 1 and it will not necessarily have multiplicity equal to d. The eigenvectors computed from $(\mathbf{D} \otimes \mathbf{I}_d)^{-1}\mathbf{Z}_\mathbf{A}$ may be complex, therefore, in order to obtain real vertex labels, the imaginary part is zeroed.

3.3.3 Null-space solution

The null-space solution is an alternative closed-form solution to the synchronization problem over $GL(d)$. In order to derive its formulation, let us

express the synchronization problem as a null-space problem by rewriting Equation (3.14) as follows:

$$(n\mathbf{I}_{dn} - \mathbf{Z})\mathbf{X} = 0 \tag{3.18}$$

so that $\mathbf{X}$ represents the d-dimensional null-space of $n\mathbf{I}_{dn} - \mathbf{Z}$.

Let us apply the same reasoning to the case of the incomplete graph:

$$(\mathbf{D} \otimes \mathbf{I}_d - \mathbf{Z}_\mathbf{A})\mathbf{X} = 0 \tag{3.19}$$

so that $\mathbf{X}$ belongs to the null-space of $\mathbf{D} \otimes \mathbf{I}_d - \mathbf{Z}_\mathbf{A}$. In presence of noise ($\epsilon \neq 0$), Equation (3.19) is usually solved in a least square sense:

$$\min_{\mathbf{X}^\top \mathbf{X} = \mathbf{I}_d} \|\mathbf{M}\mathbf{X}\|_F^2 \tag{3.20}$$

where $\mathbf{M} = \mathbf{Z}_\mathbf{A} - (\mathbf{D} \otimes \mathbf{I}_d)$ is defined from the matrix of incomplete relative measurements. Hence, a solution can be obtained in closed-form as the null-space of $\mathbf{M}$, which in turn can be derived from the least eigenvectors of $\mathbf{M}^\top\mathbf{M}$. In the presence of outliers, robustness can be easily gained via Iteratively Reweighted Least Squares (IRLS) [23].

3.3.4 Ambiguity

As mentioned in section (3.2), the solution to the synchronization problem is defined up to a global (right) product with any group element. Let us provide a formal explanation on why there are an infinite number of solutions to the same group synchronization problem.

Without any loss of generality, let us consider the formulation of the spectral solution when $\epsilon = 0$ and the measurement graph is not complete. An estimate for the vertex labeling is obtained from the eigenvectors of matrix $(\mathbf{D} \otimes \mathbf{I}_d)^{-1}\mathbf{Z}_\mathbf{A}$ associated to the eigenvalue 1 with multiplicity d. Thus, the eigenvectors corresponding to eigenvalue 1 represent a basis for a d-dimensional linear subspace. Any basis for the eigenspace is a solution to the synchronization problem. A (right) multiplication of the solution by an invertible $d \times d$ matrix is equivalent to a change of basis, meaning that the solution to synchronization is defined up to an element of $GL(d)$. The result is that the number of equivalent solutions is infinite, given that the number of basis for the same eigenspace is also infinite.

The same idea applies to the null-space formulation, for which a solution corresponds to any basis for the null-space of matrix $n\mathbf{I}_{dn} - \mathbf{Z}$ (complete graph) or $\mathbf{D} \otimes \mathbf{I}_d - \mathbf{Z}_\mathbf{A}$ (incomplete graph).

3.4 Synchronization over subgroups of GL(d)

In this section we discuss on how the results obtained in sections (3.3.2) and (3.3.3) can be used to solve the synchronization problem over the subgroups of $GL(d)$. Many tasks in Computer Vision can be modeled as synchronization problems over subgroups of $GL(d)$. Let us provide a short list of the most prominent groups we are interested in:

- $\Sigma = SO(d)$
  It corresponds to *rotation synchronization*, also known as rotation averaging.
- $\Sigma = SE(d)$
  It corresponds to *rigid-motion synchronization*, also known as motion averaging. Common applications include structure from motion, registration of 3D point clouds and simultaneous localization and mapping.
- $\Sigma = SL(d)$
  It corresponds to *homography synchronization* which can be applied to image stitching or mosaicking.

Other examples include $\Sigma = \mathcal{S}_d$ and $\Sigma = \mathcal{I}_d$, which correspond to *permutation synchronization* and *partial permutation synchronization* respectively, but they will not be covered in detail in this work.

Subgroups of $GL(d)$ can be embedded in $\mathbb{R}^{d\times d}$, hence we can treat them as if they were elements of $GL(d)$ itself. This means that the spectral and the null-space solutions work as expected.

When synchronizing over subgroups of $GL(d)$, the main difference to consider is that the estimated labels are only guaranteed to belong to $GL(d)$ and not to one of its subgroups. In other words, if we are synchronizing rotations, we are not guaranteed to obtain estimates that are still rotations. Therefore, after solving the synchronization problem we need to project the solution into the subgroup on which we are performing synchronization. Of course, every subgroup of $GL(d)$ requires its own projection that entails some sort of approximation.

CHAPTER 4

Multigraph formulation and expansion

4.1 Introduction

The problem of group synchronization is defined on measurement graphs in which vertices represent unknown states and edges are labeled with the pairwise measurements between states. So far, we analyzed the case in which only a single relative measurement between any pair of states is available, thus we only considered simple graphs.

In graph theory, a graph which is permitted to have multiple edges between any pair of vertices is referred to as a *multigraph*. The closed-form synchronization techniques introduced in sections (3.3.2) and (3.3.3) are designed for simple graphs and do not cope with synchronization problems on multigraphs. As a matter of fact, the matrix $\mathbf{Z}$ of relative measurements defined in Equation (3.9), which is used to estimate the vertex labeling for the graph, can hold one measurement between any two given states at most.

In this chapter we formally define the group synchronization problem in the presence of multiple relative measurements between unknown global states. In order to achieve this, we adopt a theoretical framework that extends the mathematical background introduced in section (3.2) to support group-labeled multigraphs.

Then, in section (4.5), we introduce an algorithm that transforms a labeled-multigraph into a simple labeled-graph that retains all pairwise information as well as the underlying structure of the original graph. This procedure will be referred to as *multigraph expansion* and will prove to be a fundamental component in the solution of the synchronization problem on group-labeled multigraphs.

4.2 Theoretical framework

In this section we introduce the theoretical framework for group-labeled multigraphs that is referenced throughout this work.

In graph theory, a multigraph is a graph that is allowed to have multiple edges (or parallel edges). We refer to the set of edges between a pair of vertices (i, j) as the *multi-edge* involved with i and j. In order words, a pair of vertices in a multigraph is connected by a multi-edge if there is more than one edge connecting such vertices.

For the purposes of this work, the notion of multigraph will not include *pseudographs*, that is, a multigraph is not allowed to have loops and, therefore, edges cannot connect a vertex to itself. This is consistent with the fact that edges in a group-labeled graph encode pairwise information and loops are not taken into account.

Let us extend the definition of Σ-labelled graph introduced in section (3.2) to multigraphs:

Definition 4.2.1. ***(Multigraph)***
A multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, s, t)$ is a directed graph where $\mathcal{V}$ is the set of vertices, $\mathcal{E}$ is the set of edges, s is a function mapping an edge to its source vertex:

$$s \colon \mathcal{E} \to \mathcal{V} \tag{4.1}$$

and t is a function mapping an edge to its target vertex:

$$t \colon \mathcal{E} \to \mathcal{V} \tag{4.2}$$

satisfying $\forall u, v$:

$$|s^{-1}(v) \cap t^{-1}(u)| = |s^{-1}(u) \cap t^{-1}(v)| \tag{4.3}$$

thus, $\mathcal{G}$ can be seen as an undirected graph. A multigraph is not allowed to have loops, therefore edges cannot connect a vertex to itself.

Definition 4.2.2. ***(Multi-edge)***
Given a multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, s, t)$, the multi-edge $E(i, j)$ is the set:

$$E(i, j) = \{e \in \mathcal{E} \colon s(e) = i \wedge t(e) = j\}. \tag{4.4}$$

Definition 4.2.3. ***(Multiplicity of a multi-edge)***
Given a multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, s, t)$, the multiplicity m of a multi-edge is a function:

$$m : \mathcal{V} \times \mathcal{V} \to \mathbb{N} \tag{4.5}$$

and is defined as:

$$m(i, j) = |E(i, j)| \tag{4.6}$$

In other words the multiplicity of the multi-edge connecting vertices (i, j) is equal to the number of edges $e \in \mathcal{E}$ that connect such vertices.

Definition 4.2.4. ***(Multiplicity of a multigraph)***
Given a multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, s, t)$, the multiplicity $m_{\mathcal{G}}$ of $\mathcal{G}$ is equal to the maximum number of edges that connect any pair of vertices (i, j), that is:

$$m_{\mathcal{G}} = max(\{e \in \mathcal{E} : m(s(e), t(e))\}) \tag{4.7}$$

Strictly speaking, every simple graph is also a multigraph, but hereinafter we consider them as different objects: a simple graph refers to a graph having multiplicity $m_{\mathcal{G}} \leq 1$, whereas multigraphs must have at least one pair of vertices (i, j) with $m_{ij} > 1$.

The elements of a multiplicative group $(\Sigma, *)$ can be used to label the vertices and edges of a multigraph yielding a group-labeled multigraph:

Definition 4.2.5. ***(Group-labeled multigraph)***
Let $(\Sigma, *)$ be a group. A Σ-labeled multigraph is a tuple $\Gamma = (\mathcal{V}, \mathcal{E}, s, t, z)$ where $\mathcal{G} = (\mathcal{V}, \mathcal{E}, s, t)$ is a multigraph and z is a labeling function:

$$z : \mathcal{E} \to \Sigma \tag{4.8}$$

The edge set $\mathcal{E}$ satisfies the following property: if $e \in \mathcal{E}$ with $s(e) = i \wedge t(e) = j$ then $e' \in \mathcal{E}$ with $s(e') = j \wedge t(e') = i$, and the labelling function z satisfies:

$$z(e) = z(e')^{-1}. \tag{4.9}$$

Equation (4.9) means that each edge connecting a pair of vertices (i, j) has a corresponding edge connecting (j, i), which is labelled with the inverse transformation, analogously to the case of the simple graph. This property allows us to see the graph as an undirected multigraph.

Most of the definitions related to simple graphs are still valid for multigraphs, albeit with small differences to account for the possible presence of multiple edges between any pair of vertices.

Definition 4.2.6. ***(Null cycle)***
Let $\Gamma = (\mathcal{V}, \mathcal{E}, s, t, z)$ be a Σ-labeled multigraph. A circuit of edges connecting pairs of vertices such as $(i_1, i_2), (i_2, i_3), \ldots, (i_l, i_1)$ is a null cycle if and only if the composition of the edge labels along the circuit returns the identity for group Σ for every possible choice of edge between any pair of vertices in the circuit.

$$z(e_{12}) * z(e_{23}) * ... * z(e_{l1}) = 1_\Sigma \tag{4.10}$$

with $e_{ij} \in \{e \in \mathcal{E} \mid (s(e), t(e)) = (i, j)\}$.

In practice, to check whether a cycle is null, the edge labels associated to the edges in the cycles are chained and checked against the identity for group Σ. In the case of multigraphs, multiple cycles passing through the same subset of $\mathcal{V}$ can exist. Thus, the fact that a cycle through vertices $\mathcal{V}' \subseteq \mathcal{V}$ is null does not guarantee that all cycles through $\mathcal{V}'$ are null.

As seen for simple graphs, null cycles are employed in order to define what a consistent labeling for a multigraph is.

Definition 4.2.7. ***(Consistent labeling)***
Let $\Gamma = (\mathcal{V}, \mathcal{E}, s, t, z)$ be a Σ-labelled multigraph and let $x : \mathcal{V} \to \Sigma$ be a vertex labeling. x is a consistent labeling if and only if

$$z(e) = x(i) * x(j)^{-1} \tag{4.11}$$

$\forall e \in \mathcal{E}$ such that $((s(e), t(e)) = (i, j)$.

We refer to a Σ-labeled multigraph which admits a consistent labeling as a *balanced* multigraph. Equation (4.11) means that for any pair of vertices $(i, j) \in \mathcal{V}$, with $\mathcal{V}$ being the vertex set of the multigraph, the labels of the edges connecting i and j must be equal to the ratio of the vertex labels $x(i)$ and $x(j)$. Thus, equation (4.11) is also referred to as the *consistency constraint*.

From definition (4.2.7), it is understood that a multigraph can have a consistent labeling only if all the edges between a pair of vertices share the same edge labeling. In order to satisfy the consistency constraint, an edge label must be equal to the ratio of the corresponding vertex labels. Therefore, in a balanced multigraph, the only label assignment for an edge $e \in \mathcal{E}$ involved with i and j is $z(e) = x(i) * x(j)^{-1}$.

Definition 4.2.8. ***(Redundant edge)***
An edge $e \in \mathcal{E}$ is said to be redundant if there exists another edge $e' \in \mathcal{E}$ s.t. $z(e) = z(e') \wedge s(e) = s(e') \wedge t(e) = t(e')$.

In other words, an edge is said to be redundant if there is another edge connecting the same two vertices i and j with the same edge label.

Remark 1. Let $\Gamma = (\mathcal{V}, \mathcal{E}, s, t, z)$ be a Σ-labelled multigraph. Γ admits a consistent labeling if and only if it can be simplified into a simple graph by removing redundant edges between any pair of vertices $(i, j) \in \mathcal{V} \times \mathcal{V}$ and the resulting graph admits a consistent labeling.

The statements above may lead us to believe that the notion of multigraph is redundant. It is understood that only multigraphs which can be mapped to simple graphs by removing redundant edges are the ones that admit a consistent labeling. In practice, though, having a graph, even one without multi-edges, that admits an exactly consistent labeling is unlikely. This is due to the fact that, in real world scenarios, the pairwise measurements encoded in a measurement graph are noisy. Thus, a labeling that is consistent across the entire graph does not exist. As discussed in section (3.3.2), we wish to obtain a labeling that minimizes the consistency error across the graph. If the graph admits a consistent labeling, then it means that we are in the noiseless scenario and that the consistency error will have a minimum in zero.

4.2.1 Group synchronization

Let us define a metric function $\delta : \Sigma \times \Sigma \to \mathbb{R}^+$ and a non-negative monotonically increasing loss function $\rho : \mathbb{R}^+ \to \mathbb{R}^+$ with a single minimum at 0 and $p(0) = 0$.

Definition 4.2.9. ***(Consistency error for multigraphs)*** *Let $\Gamma = (\mathcal{V}, \mathcal{E}, s, t, z)$ be a Σ-labeled multigraph and $\tilde{x}$ be a vertex labeling for Γ. The consistency error for $\tilde{x}$ is defined as follows:*

$$\epsilon(\tilde{x}) = \sum_{e \in \mathcal{E}} \rho\Big(\delta(\tilde{z}(e), z(e))\Big) \tag{4.12}$$

where $\tilde{z}$ is the edge labeling induced by $\tilde{x}$. The metric function δ defines a distance between each pair of elements in Σ. In the context of group synchronization, the distance between two elements is usually defined as the Frobenius norm of the difference between such elements. The loss function ρ can be a quadratic function or a robust function from M-estimators [23].

As mentioned before, finding a consistent vertex labeling in either a multigraph or a simple graph means that such labeling induces a consistency error equal to zero. Given that this rarely happens in practice due to the noise in the measurements, the goal is to find a vertex labeling that

minimizes the consistency error for such graph. Again, this legitimates the notion of multigraph applied in the context of group synchronization: since pairwise measurements are likely to be noisy, then multiple measurements between unknown states are allowed and encouraged in order to provide a more robust and reliable set of information.

As in synchronization on simple graphs, we wish to estimate the unknown vertex labeling starting from a set of noisy pairwise measurements. This is achieved by applying the consistency constraint between pairs of vertices knowing that cycles in the multigraph must be null. A clear advantage of multigraphs compared to simple graphs is that, given a number of vertices, the number of cycles in the graph can increase substantially depending on the number and the multiplicity of multi-edges. The presence of a greater number of cycles should allow for more effective error compensation and, therefore, provide more accurate and robust results when the multigraph is synchronized.

4.3 Challenges related to multigraphs

From the remarks in section (4.2.1), it is clear that the solutions for the synchronization problem from section (3.3) cannot be applied to a labeled multigraph. As a matter of fact, the presence of multi-edges makes it impossible to collect the edge labels into a single consistency constraint matrix $\mathbf{Z}$: the techniques introduced so far allow for a maximum of one consistency constraint (edge) between each pair of vertices, therefore it is not straightforward to solve the synchronization problem on multigraphs.

Remark 2. A naive strategy to handle the multiple measurements in multigraphs is *edge averaging*. This technique consists in collapsing any multi-edge in one simple edge by averaging its labels. While this approach is certainly effective, it does lead to sub-optimal results. In addition, the average is well defined only in certain groups, e.g., average in $SO(3)$ is well studied by [22], whereas similar results are less studied for general groups such as $SL(3)$.

Another factor is the inherent problem related to information loss that emerges when collapsing the multi-edges. While this loss may be small enough to produce acceptable results for certain groups, this approach can introduce significant approximations.

For these reasons, the solution proposed in later sections consists in turning a multigraph into a simple graph without any loss of information so that synchronization techniques designed for simple graphs can be applied to it. This approach will allow us to solve the synchronization problem for a

multigraph without the need to collapse multi-edges into simple edges and lose relative (pairwise) measurements. As we shall discuss later in more details, the benefits provided by this approach are not limited to preserving measurements. As a matter of fact, the ability to turn any multigraph into a simple graph solves the problem of having to define an aggregation function for certain groups, such as $SL(3)$ for homographies.

4.4 Multigraph expansion

Let us consider the definition of group-labeled multigraph introduced in section (4.2). In this section we show that given any group-labeled multigraph it is possible to find a simple labeled graph that retains all the pairwise and global information present in the original multigraph.

The process of transforming a multigraph Γ_{multi} into a simple graph Γ_{simple} is called *multigraph expansion* and Γ_{simple} is referred to as the *expanded graph* of Γ_{multi}. More specifically, multigraph expansion is defined as the process of *expanding* a multigraph into a simple graph by replicating vertices and by introducing additional constraints in order to preserve both the relative (pairwise) information and the underlying structure of the original multigraph. In addition, we present an iterative greedy algorithm that is capable of expanding any group-labeled multigraph while replicating the minimum number of vertices. As we shall explain, minimizing the number of vertices in the expanded graph provides several benefits, including improved computational and memory efficiency for synchronization.

4.4.1 Intuition

Let us begin this introduction to multigraph expansion by analyzing a simple case first. Let us take into account the minimal multigraph in Figure (4.1), that is, a graph made by two vertices x_1 and x_2 connected by a multi-edge of multiplicity 2 with edge labels $z_{12,1}$ and $z_{12,2}$.

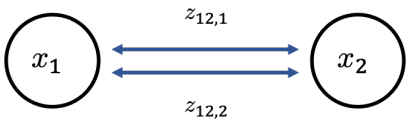

Figure 4.1: *The minimal multigraph considered in this section.*

If the goal is to expand the multigraph, then we need to find a way to remove the multi-edge while preserving the pairwise information in the edge labels. In order to achieve this result, vertex x_1 is replaced by two

separate vertices $x_{1,1}$ and $x_{1,2}$, each connected to x_2. The edge between $x_{1,1}$ and x_2 has label $z_{12,1}$, while the one between $x_{1,2}$ and x_2 has label $z_{12,2}$, as shown in Figure (4.2). The multigraph is now a graph with simple edges and it retains both labels that were previously assigned to the multi-edge.

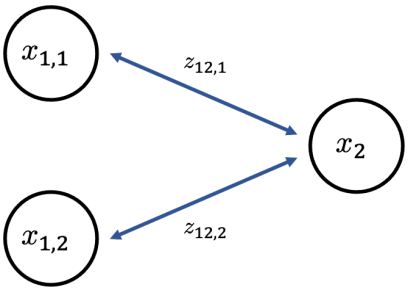

Figure 4.2: *The simple graph that is obtained from the original multigraph by replacing x_1 with its vertex replicas $x_{1,1}$ and $x_{1,2}$.*

An important point to consider is that, because $x_{1,1}$ and $x_{1,2}$ are spawned from the same vertex x_1, the labels assigned to them should be equal to each other. In order to migrate this constraint to the expanded graph, it is sufficient to add a new edge connecting $x_{1,1}$ and $x_{1,2}$ with a label equal to the unit element 1_Σ for group Σ, as shown in Figure (4.3). The vertices $x_{1,1}$ and $x_{1,2}$ are called *replicas* of vertex x_1, given the fact that, although $x_{1,1}$ and $x_{1,2}$ are separate vertices in the expanded graph, they represent a single vertex in the underlying structure of the multigraph. Thus, replicas of a vertex should have the same vertex label to ensure the consistency of labels between the two graphs.

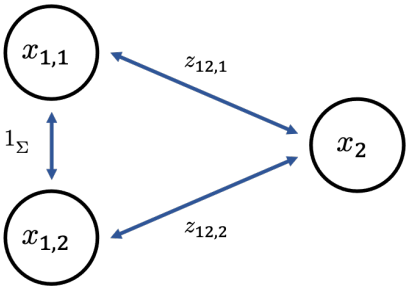

Figure 4.3: *The expanded graph of the multigraph in Figure (4.1) obtained by replacing x_1 with its two vertex replicas and by adding an identity constraint between them.*

After expanding the multigraph, it is possible to collect vertex labels in matrix $\mathbf{X} \in \mathbb{R}^{dn \times d}$ and edge labels in matrix $\mathbf{Z} \in \mathbb{R}^{dn \times dn}$ as seen in the

case of a simple graph with no multi-edges:

$$\mathbf{X} = \begin{pmatrix} \mathbf{X}_{1,1} \\ \mathbf{X}_{1,2} \\ \mathbf{X}_{2} \end{pmatrix} \tag{4.13}$$

$$\mathbf{Z} = \begin{pmatrix} 1_{\Sigma} & 1_{\Sigma} & z_{12,1} \\ 1_{\Sigma} & 1_{\Sigma} & z_{12,2} \\ z_{21,1} & z_{21,2} & 1_{\Sigma} \end{pmatrix} \tag{4.14}$$

Another point to consider is that this method can also be applied when the multi-edge connecting x_1 and x_2 is of an arbitrary multiplicity m_{12} that is greater than 2 (see Figure (4.4)). In order to obtain an expanded graph from a multigraph with two vertices and a multi-edge of an arbitrary multiplicity m_{12}, it is sufficient to replace vertex x_1 with m_{12} vertex replicas $x_{1,1}, x_{1,2}, ..., x_{1,m_{12}}$. Each replica $x_{1,k}$ of x_1 is connected to x_2 by an edge with label $z_{12,k}$, with $k \in \{1, 2, \ldots, m_{12}\}$.The next step is to encode the constraint that the vertex labels of all replicas of x_1 must share the same value in the expanded graph. Thus, identity constraints in the form of edges with label 1_{Σ} are introduced for every pair of vertex replicas. The final expanded graph is shown in Figure (4.5).

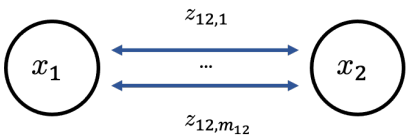

Figure 4.4: *A multigraph that contains multi-edges of an arbitrary multiplicity.*

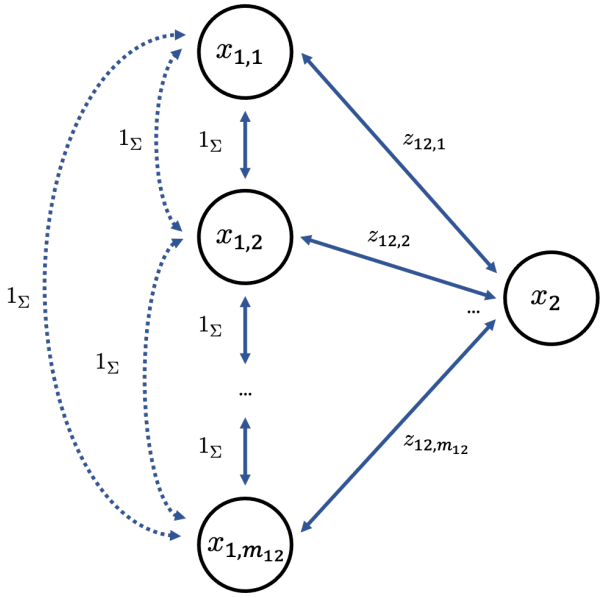

Figure 4.5: *The expanded graph of the multigraph in Figure (4.4).*

As a final note, it is possible to choose x_2 instead of x_1 as the vertex chosen to be replaced by replicas. Of course, the choice does not affect the result, since pairwise information is preserved in either case.

Taking these considerations into account, we can safely assume that every multigraph can be expanded into a simple graph if, for every pair of vertices connected by a multi-edge of multiplicity m_{ij}, we replace at least one vertex with a set of m_{ij} vertex replicas and add the necessary constraints to preserve the information encoded in the original multigraph.

4.4.2 Replicating vertices in a multigraph

So far, we provided an intuition to the idea of multigraph expansion and considered only the case of the minimal multigraph in Figure (4.1). In general, this is not enough given that it is very likely that vertices have more than a single incoming (or outgoing) multi-edge. In this section, we address the general case of expanding a multigraph with an arbitrary number of vertices and multiple incoming (or outgoing) multi-edges.

Let us begin from the intuition provided in section (4.4.1) and generalize from it. Consider the case in Figure (4.6) of a single multi-edge of multiplicity m_{ij} connecting two vertices x_i and x_j. Any of the two vertices is a good candidate for being replicated. Let us select x_i for expansion. x_i is replaced with m_{ij} replicas named $x_{i,1}, x_{i,2}, \ldots, x_{i,m_{ij}}$. The next step is to

add constraints between every pair of replicas to indicate that their labels must be the same, given that they are spawned from the same vertex x_i. Thus, we introduce edges with labels 1_Σ connecting every pair of replicas, for a total of $\frac{m_{ij} \times (m_{ij}-1)}{2}$ constraints. Finally, we introduce constraints of type $z_{ij,k}$ between x_j, which was not expanded, and $x_{i,k}$ for a total of m_{ij} edges as in Figure (4.7).

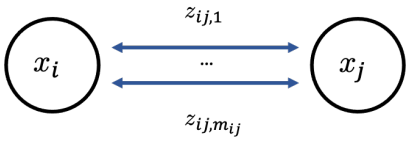

Figure 4.6: *A multigraph with two vertices connected by a multi-edge of an arbitrary multiplicity m_{ij}.*

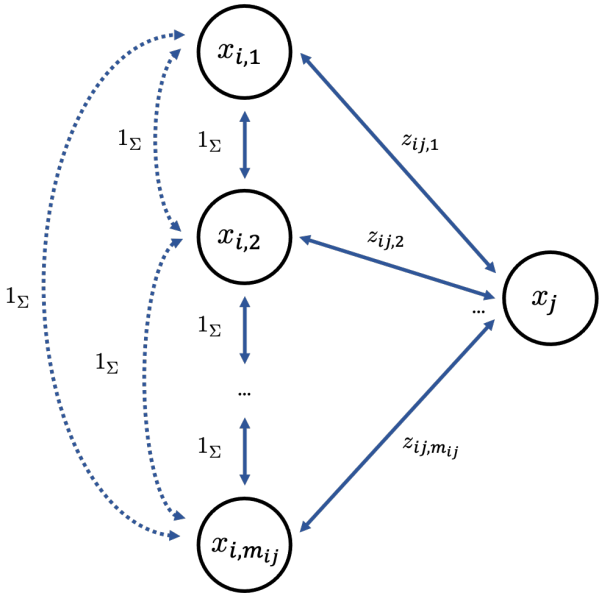

Figure 4.7: *The expanded graph of the multigraph in Figure (4.6) after replicating x_i. Dashed lines represent the identity constraints between vertex replicas.*

Consider the general case in which multi-edges connect x_i to multiple vertices. Let us assume that each multi-edge has an arbitrary multiplicity. The number of replicas that needs to be introduced in order to replicate vertex x_i is equal to the largest multiplicity m_{max} of the incoming multi-edges for x_i. After x_i is replaced by m_{max} replicas, for every vertex previously connected to x_i, we need to introduce a set of constraints between such ver-

tex and the newly added replicas. If the number of replicas m_{max} is larger than the multiplicity m_{ij} of the multi-edge previously connecting x_i to a generic vertex x_j, then the number of constraints added between x_j and the m_{max} replicas of x_i is going to be limited to m_{ij}. Random sampling without replacement can be performed to select a subset with cardinality m_{ij} of the m_{max} replicas of x_i.

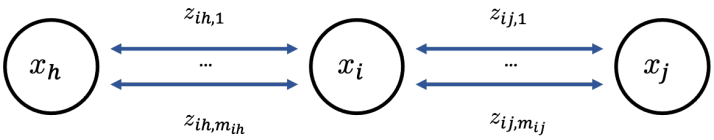

Figure 4.8: *A multigraph with* 2 *multi-edges of arbitrary multiplicity connecting* 3 *vertices. If we assume that* $m_{ij} > m_{ih}$*, then* $m_{max} = m_{ij}$*.*

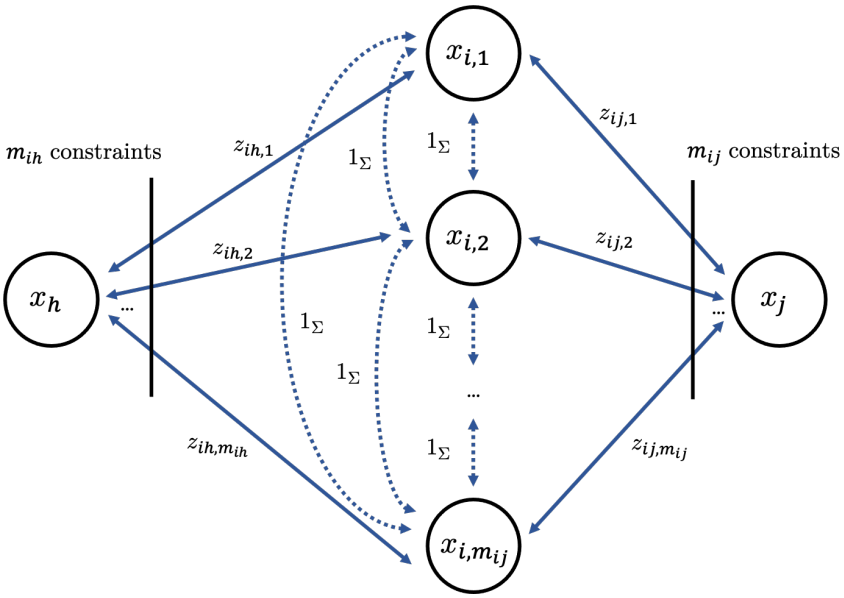

Figure 4.9: *The expanded graph of the multigraph in Figure (4.8).*

In the case in which a replicated vertex is connected to other vertices with a simple edge, then the edge is replicated m_{max} times along with its label, where m_{max} is the number of replicas introduced in place of x_i. Each of these replicated edges will connect the vertex that was not expanded to a different replica of x_i as illustrated in Figure (4.11).

Another approach to the problem of replicating vertices with incoming (outgoing) simple edges is to connect just one of the m_{max} replicas

to the non-expanded vertex, instead of replicating the edge m_{max} times. In practice, the underlying structure of the multigraph is preserved, hence synchronization performance should not be impacted. In addition, this approach provides benefits in terms of time and memory requirements since the matrix of constraints $\mathbf{Z_A}$ will be more sparse. We refer to this approach as *optimized single link vertex replication*.

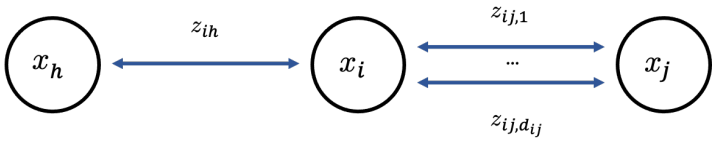

Figure 4.10: *A multigraph containing both a simple edge and a multi-edge.*

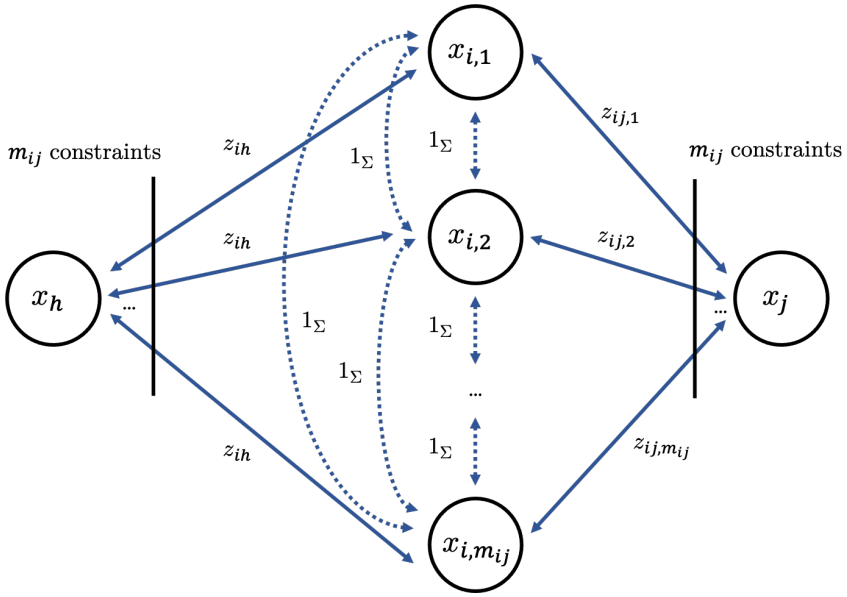

Figure 4.11: *The expanded graph of the multigraph in Figure (4.10) without optimized single link vertex replication.*

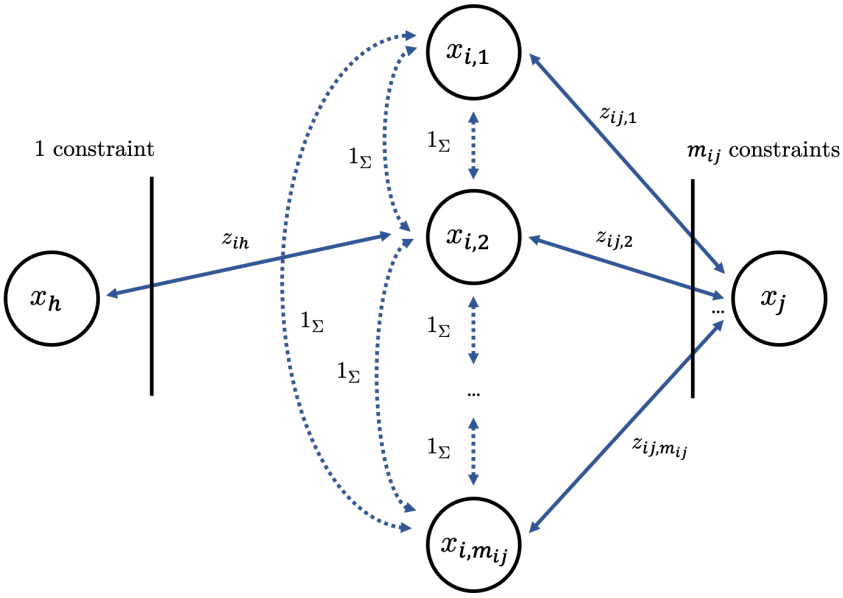

Figure 4.12: *The expanded graph of the multigraph in Figure (4.10) with optimized single link vertex replication.*

4.4.3 Optimized graph expansion

In this section we introduce an efficient iterative greedy algorithm that can be applied in order to find the solution to the problem of replicating the minimum number of vertices to efficiently expand a multigraph.

Let us begin by making an observation regarding vertex replication. After a vertex is replicated in a multigraph, the result is that all vertices that were previously connected to the replicated vertex will have one less incoming (outgoing) multi-edge. Thus, the idea behind optimized multigraph expansion can be seen as the problem of finding those vertices that cause the largest number of multi-edges to be removed from the graph once they are replicated.

In section (4.4.1), we considered a minimal multigraph whose vertex set included only two vertices x_1 and x_2 connected by a single multi-edge with $m_{12} = 2$. In order to expand the multigraph into a simple graph, we demonstrated that it was sufficient to replicate only one of the two vertices and to turn the multi-edge into 2 simple edges connecting the replicas to the non-replicated vertex.

Let us combine the observations made so far in order to devise an optimal strategy for optimized multigraph expansion. The strategy we propose is to give precedence to those vertices whose replication causes two or more

multi-edges to be transformed into simple edges. In practice, the algorithm iterates through all the elements of the vertex set of the input multigraph and proceeds to replicate a vertex if it satisfies the condition that it has at least two incoming (outgoing) multi-edges. Once all vertices are scanned, then only vertices having at most a single incoming (outgoing) multi-edge remain. After that, we arbitrarily choose which vertex to replicate in pairs of adjacent vertices connected by a single multi-edge and obtain the final expanded graph.

Algorithm (1) provides a brief outline of the main steps for optimized graph expansion. Further details will be provided in section (4.5).

Algorithm 1 Outline of optimized graph expansion

Input: A group-labeled multigraph
Output: The optimized expanded graph of the input multigraph

```
S = ∅
for every node i in the multigraph do
    if there is one outgoing multi-edge for i then
        S = S ∪ {i}                          ▷ save node i to expand it later
    else if there are two or more outgoing multi-edges for i then
        replicate node i and apply changes to the graph
    end if
end for
for every node i in S do
    if there is an outgoing multi-edge for i then
        replicate node i and apply changes to the graph
    end if
end for
```

Let us provide a practical example in which the *optimized graph expansion* algorithm can reduce the number of replicated vertices for an expanded multigraph. Consider the case of a multigraph made of a chain of n vertices connected by $n-1$ multi-edges of any multiplicity. For every pair of adjacent vertices in the chain, at least one of the two must be replicated in order to expand all multi-edges. Following the greedy strategy, we iterate through the vertices in the graph, starting from x_1, and expand only the vertices with multiple incoming (outgoing) multi-edges.

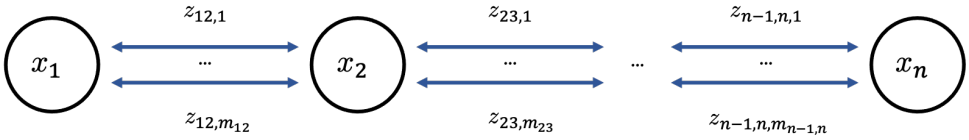

Figure 4.13: *A depiction of the multigraph referenced in this section.*

x_1 is at the edge of the chain, thus it has only one incoming (outgoing) multi-edge and is not chosen for replication. Next is x_2, which satisfies the condition and is replicated. As a result, the incoming (outgoing) multi-edges in x_1 and x_3 are transformed into simple edges. x_3 is skipped for the same reason as x_1 and x_4 is selected for replication. This continues until the end of the chain. The result is that only half of the vertices are selected for replication. More specifically, if n is even, then $\frac{n}{2}$ vertices are expanded, while if n is odd, then $\frac{n-1}{2}$ vertices are expanded. The choice to replicate vertices $x_2, x_4, \ldots$ satisfies the previously identified constraint that, for every pair of adjacent vertices in the original multigraph, at least one vertex should be replicated in order to convert the multigraph into its expanded simple graph.

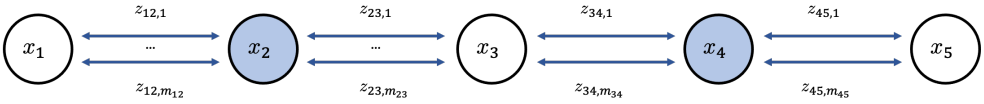

Figure 4.14: *An example of a chain of* 5 *vertices connected by multi-edges. Expanding the* 2 *vertices highlighted in blue is sufficient in order to turn the multigraph into an equivalent simple graph.*

Of course, replicating a vertex at the edge of the chain still produces a valid expanded graph, but it does not guarantee that the set of vertices that are replicated at the end of the expansion procedure contains the minimum number of elements.

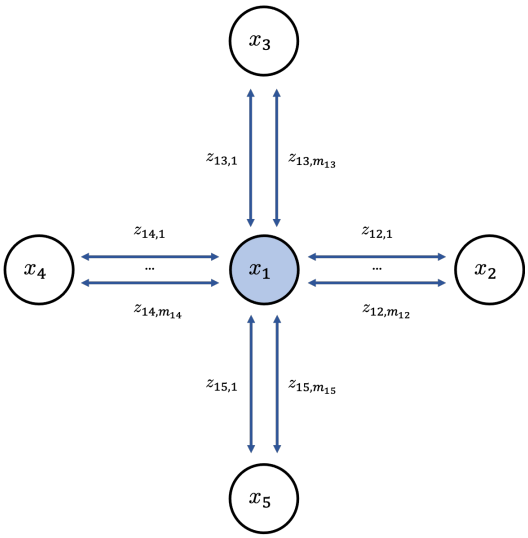

Figure 4.15: *A depiction of a multigraph with a central vertex configuration. Notice how expanding the vertex with more than a single incoming (outgoing) multi-edge allows us to expand the minimum number of vertices.*

4.4.4 Considerations on multigraph expansion

The reader may be questioning why going through the trouble of devising a strategy to minimize the number of additional vertices in the form of replicas introduced by the multigraph expansion algorithm. As established so far, the choice of vertices to replicate does not affect the end result, meaning that multigraph expansion allows us to turn any group-labeled multigraph into a simple graph without any loss of information. The choice of vertices to replicate does affect the computational complexity both in terms of time and memory though, since the more vertex replicas are added to the graph, the larger the matrices encoding the graph are going to be, resulting in both a larger memory footprint and longer processing.

Consider the synchronization problem over a subgroup of the General Linear Group $GL(d)$. As noted in (3.3.2), the problem can be solved by applying a closed-form solution, namely the spectral solution, which consists in finding the eigenvectors associated to the d leading eigenvalues of matrix $(\mathbf{D} \otimes \mathbf{I}_d)^{-1}\mathbf{Z}_\mathbf{A}$. $\mathbf{D}$ is a $n \times n$ matrix and $\mathbf{Z}_\mathbf{A}$ is a $dn \times dn$ matrix, where n is the number of vertices in the graph. This means that, as the number of vertices in the graph grows, the time required to find the d leading eigenvalues of such matrix can increase by a significant amount, even when exploiting optimized sparse solvers such as *eigs* or *svds*. This poses a significant problem, particularly for large graphs with lots of multi-edges,

given that the large number of vertices to replicate may render closed-form solutions impractical. Therefore, we expect that the optimization applied to multigraph expansion will provide significant benefits in the computational complexity of both the expansion of the multigraph itself and subsequent group synchronization techniques applied to the respective expanded graph.

4.5 Algorithm for multigraph expansion

In this section, we expand on the results from section (4.4) in order to define a general algorithm for the expansion of any Σ-labeled multigraph. In addition, we analyze the complexity of such algorithm and present the MATLAB implementation used to conduct experiments in this work.

The multigraph expansion algorithm, henceforth named MULTIGRAPH-EXPAND, is an iterative greedy procedure that turns a Σ-labeled multigraph Γ_{in} given in input into a simple graph $\Gamma_{\text{out}} = (\mathcal{V}_{\text{out}}, \mathcal{E}_{\text{out}}, z)$, as shown in Figure (4.16). Algorithm 2 describes the main steps.

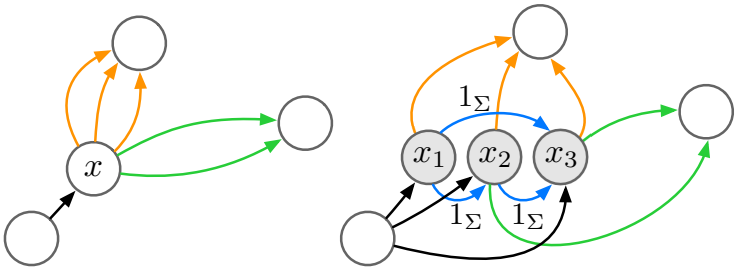

Figure 4.16: *Multigraph expansion: the process of expanding a multigraph into a simple graph without multi-edges by replicating specific vertices (shaded nodes) and by introducing additional constraints to preserve both absolute (global) and relative (pairwise) information between the nodes in the graph.*

At first, in lines 1-3, we initialize the vertex set and the edge set of the output graph to the values of the input multi-graph. At high level, the algorithm cycles through all the vertices of the input graph and populates $\mathcal{V}_{\text{out}}$ and $\mathcal{E}_{\text{out}}$ adding the vertex and its corresponding edges, but when a multi-edge is encountered this is first expanded to simple-edges between replicated vertices, via the EXPANDVERTEX routine, and then it is added to $\mathcal{E}_{\text{out}}$ together with the replicas which are in turn added to $\mathcal{V}_{\text{out}}$. For reasons of efficiency, the procedure starts by expanding the vertices with more than one multi-edge, (lines 4-17) while the expansion of vertices with a single multi-edge is deferred until the end of the procedure (line 19 – 24). This deferred expansion mechanism is implemented by means of a queue $\mathcal{S}$ that

Algorithm 2 MULTIGRAPHEXPAND

Input: A Σ labeled multi-graph $\Gamma_{\text{in}} = (\mathcal{V}_{\text{in}}, \mathcal{E}_{\text{in}}, s, t, z)$
Output: The expanded graph $Go = (\mathcal{V}_{\text{out}}, \mathcal{E}_{\text{out}}, z)$

1: $\mathcal{V}_{\text{out}} = \mathcal{V}_{\text{in}}$
2: $\mathcal{E}_{\text{out}} = \mathcal{E}_{\text{in}}$
3: $\mathcal{S} = \emptyset$ ▷ queue
4: **for** $v \in \mathcal{V}_{\text{in}}$ **do**
5: $\quad \mathcal{E}_v = \text{GETMULTIEDGES}(v, \Gamma_{\text{out}})$
6: $\quad$ **if** $|\mathcal{E}_v| > 1$ **then** ▷ there are many multi-edges
7: $\quad\quad$ ▷ Remove vertices and edges
8: $\quad\quad \mathcal{V}_{\text{out}} = \mathcal{V}_{\text{out}} \setminus \{v\}$
9: $\quad\quad \mathcal{E}_{\text{out}} = \mathcal{E}_{\text{out}} \setminus \{e \in \mathcal{E}_{\text{out}} \colon s(e) = v \text{ or } t(e) = v\}$
10: $\quad\quad$ ▷ Add replicated vertices and edges
11: $\quad\quad (\mathcal{V}_{\text{rep}}, \mathcal{E}_{\text{rep}}) = \text{EXPANDVERTEX}(v, \Gamma_{\text{out}})$
12: $\quad\quad \mathcal{V}_{\text{out}} = \mathcal{V}_{\text{out}} \cup \mathcal{V}_{\text{rep}}$
13: $\quad\quad \mathcal{E}_{\text{out}} = \mathcal{E}_{\text{out}} \cup \mathcal{E}_{\text{rep}}$
14: $\quad$ **else if** $|\mathcal{E}_v| = 1$ **then**
15: $\quad\quad \mathcal{S} = \mathcal{S} \cup \{v\}$ ▷ add to queue
16: $\quad$ **end if**
17: **end for**
18: ▷ Process vertices with a single multi-edge
19: **for** $v \in \mathcal{S}$ **do**
20: $\quad \mathcal{E}_v = \text{GETMULTIEDGES}(v, \Gamma_{\text{out}})$
21: $\quad$ **if** $|\mathcal{E}_v| > 0$ **then**
22: $\quad\quad$ repeat lines 8-13
23: $\quad$ **end if**
24: **end for**
25: $\Gamma_{\text{out}} = (\mathcal{V}_{\text{out}}, \mathcal{E}_{\text{out}}, z)$

is initialized as empty (line 3).

The first loop (lines 4-17) consists in iterating through the vertices in the input multigraph and replicating those with more than one multi-edge updating $\mathcal{V}_{\text{out}}$ and $\mathcal{E}_{\text{out}}$ accordingly for the output graph. Specifically, in line 5 we gather the multi-edges for a vertex v and, if their count is greater than 1, we proceed to replicate v. In lines 8-9, we remove v from $\mathcal{V}_{\text{out}}$ and all its edges from $\mathcal{E}_{\text{out}}$. In line 11, we invoke the EXPANDVERTEX procedure, which returns $\mathcal{V}_{\text{rep}}$ and $\mathcal{E}_{\text{rep}}$. $\mathcal{V}_{\text{rep}}$ contains the replicas of v, while $\mathcal{E}_{\text{rep}}$ contains the previous edges of v distributed for $\mathcal{V}_{\text{rep}}$, including the identity constraints between replicas (more details in Alg.3).

The elements of $\mathcal{V}_{\text{rep}}$ and $\mathcal{E}_{\text{rep}}$ are merged with those of $\mathcal{V}_{\text{out}}$ (line 12) and $\mathcal{E}_{\text{out}}$ (line 13) respectively, updating the output graph. If v has only a single incoming or outgoing multi-edge, then v is added to the queue $\mathcal{S}$ (line 15) and its replication is deferred after the other vertices with a greater number of multi-edges have been expanded.

The second loop (lines 19-24) consists in iterating through the vertices v in the queue $\mathcal{S}$. The procedure is similar to the first loop: v is replicated when it is still involved with a multi-edge. This check (line 21) is necessary because, during the first pass, the multi-edge involved with v may have been expanded by the vertices adjacent to v.

The algorithm outputs the expanded graph Γ_{out}, which is defined by the updated vertex set $\mathcal{V}_{\text{out}}$ and edge set $\mathcal{E}_{\text{out}}$.

Expansion of vertex and multi-edge conversion The procedure EXPANDVERTEX, detailed in Algorithm 3, aims at replicating a vertex i originally involved in a multi-edge with a set of returned replicas $\mathcal{V}_{\text{rep}}$. In addition, EXPANDVERTEX converts the original multi-edges involving i in simple edges connecting the replicas. Specifically, incoming edges, outgoing edges and identity constraints between replicas are instantiated and added to the returned set of simple edges $\mathcal{E}_{\text{rep}}$.

First, we compute the maximum number m between the cardinality of incoming and the cardinality of outgoing multi-edges involved with i (lines 2 - 7). This value is used to initialize an ordered sequence $\mathcal{V}_{\text{rep}} = \{v_1, v_2, \ldots, v_m\}$ with m replicas of i (line 8). The r-th element of $\mathcal{V}_{\text{rep}}$ is referenced by using the array-like notation $\mathcal{V}_{\text{rep}}[r]$. The set of edges for the expanded vertex is initially empty (line 9).

The next step is to distribute the original constraints among replicas, starting from the incoming edges (lines 11-19). For every incoming multi-edge E, we take every simple edge $e \in E$ and add a new edge f to $\mathcal{E}_{\text{rep}}$ with the same label and source node as e, but with a different target vertex in $\mathcal{V}_{\text{rep}}$

Algorithm 3 Expand vertex and convert multi-edges

function EXPANDVERTEX(i, $\mathcal{V}$, $\mathcal{E}$)
  ▷ Get max cardinality of multi-edges
  $\mathcal{E}_{\to i} = \text{GETINCOMINGMULTIEDGES}(i, \mathcal{V}, \mathcal{E})$
  $m_{\to i} = \max_{e \in \mathcal{E}_{\to i}} |e|$
  $\mathcal{E}_{i \to} = \text{GETOUTGOINGMULTIEDGES}(i, \mathcal{V}, \mathcal{E})$
  $m_{i \to} = \max_{e \in \mathcal{E}_{i \to}} |e|$
  $m = \max(1, m_{\to i}, m_{i \to})$ ▷ Number of replicas
  $\mathcal{V}_{\text{rep}} = \{v_1, \ldots, v_m\}$ ▷ Ordered set of replicas
  $\mathcal{E}_{\text{rep}} = \emptyset$
  ▷ Distribute incoming edges among replicas
  for $E \in \mathcal{E}_{\to i}$ **do** ▷ E is a multi-edge
    $r = 1$ ▷ replicas' counter
    for $e \in E$ **do** ▷ e is a simple edge in E
      instantiate f s.t. $s(f) = t(e), t(f) = \mathcal{V}_{\text{rep}}[r]$
      $\mathcal{E}_{\text{rep}} = \mathcal{E}_{\text{rep}} \cup \{f\}$
      $z(f) = z(e)$
      $r = r + 1$
    end for
  end for
  ▷ Distribute outgoing edges among replicas
  for $E \in \mathcal{E}_{i \to}$ **do**
    $r = 1$
    for $e \in E$ **do**
      instantiate f s.t. $s(f) = \mathcal{V}_{\text{rep}}[r], t(f) = t(e)$
      $\mathcal{E}_{\text{rep}} = \mathcal{E}_{\text{rep}} \cup \{f\}$
      $z(f) = z(e)$
      $r = r + 1$
    end for
  end for
  ▷ Add identity constraints between replicas
  for $(v_1, v_2) \in \mathcal{P}_2(\mathcal{V}_{\text{rep}})$ **do**
    instantiate f s.t. $s(f) = v_1, t(f) = v_2$
    $\mathcal{E}_{\text{rep}} = \mathcal{E}_{\text{rep}} \cup \{f\}$
    $z(f) = 1_\Sigma$
  end for
  return $(\mathcal{V}_{\text{rep}}, \mathcal{E}_{\text{rep}})$
end function

(lines 11- 19). In other words, we are rerouting every edge that makes up the multi-edge so that each of them has a different replica $\mathcal{V}_{\text{rep}}[r]$ of i, as target vertex. Notice that simple edges can be seen as multi-edges with multiplicity equal to 1, thus they are distributed as well.

We apply the same steps to distribute the constraints among replicas for outgoing multi-edges. The main difference is that the re-routed edges f have the same label and target vertex as their counterparts, but have a replica $\mathcal{V}_{\text{rep}}[r]$ of i as a source vertex (lines 20-35) (see Figure 4.16).

Finally, we add the identity constraints between replicas (lines 30 - 35). To this end, we iterate through all possible pairs of replicas (v_1, v_2), indicated as $\mathcal{P}_2(\mathcal{V}_{\text{rep}})$ (line 31), and a new edge f is added to $\mathcal{V}_{\text{rep}}$ with source vertex v_1, target vertexv_2 with the identity 1_Σ as label.

Auxiliary function for multi-edges Algorithm 4 collects auxiliary functions that are used to get the incoming, the outgoing and all the multi-edges respectively for a vertex i in the graph Γ. We recall that the notation $E(i, j)$ denotes the multi-edges connecting vertex i and j.

Algorithm 4 Auxiliary function for multi-edges

function GETINCOMINGMULTIEDGES(i, $\mathcal{V}$, $\mathcal{E}$)
Input: A node i and a multi-graph $\Gamma = (\mathcal{V}, \mathcal{E}, s, t)$.
Output: The set of incoming multi-edges for i.
    $\mathcal{T} = \{e \in \mathcal{E} \mid t(e) = i\}$
    $\mathcal{U} = \{v \in \mathcal{V} \mid \exists e \in \mathcal{T} \text{ s.t. } s(e) = v\}$
    return $\mathcal{E}_{\to i} = \{E(v, i) : v \in \mathcal{U}\}$
end function

function GETOUTGOINGMULTIEDGES(i, $\mathcal{V}$, $\mathcal{E}$)
Input: A node i and a multi-graph $\Gamma = (\mathcal{V}, \mathcal{E}, s, t)$.
Output: The set of outgoing multi-edges for i.
    $\mathcal{S} = \{e \in \mathcal{E} \mid s(e) = i\}$
    $\mathcal{U} = \{v \in \mathcal{V} \mid \exists e \in \mathcal{S} \text{ s.t. } t(e) = v\}$
    return $\mathcal{E}_{i \to} = \{E(i, v) : v \in \mathcal{U}\}$
end function

function GETMULTIEDGES(i, $\mathcal{V}$, $\mathcal{E}$)
Input: A node i and a multi-graph $\Gamma = (\mathcal{V}, \mathcal{E}, s, t)$.
Output: The set of multi-edges for i.
    $\mathcal{E}_{\to i}$ = GETINGOINGMULTIEDGES(i, Γ).
    $\mathcal{E}_{i \to}$ = GETOUTGOINGMULTIEDGES(i, Γ).
    return $\mathcal{E}_{\to i} \cup \mathcal{E}_{i \to}$.
end function

4.5.1 Computational complexity

In this section, we analyze the computational complexity of the algorithm.

Let us begin by analyzing the complexity of the EXPANDVERTEX procedure. EXPANDVERTEX includes three for-loops. The first and second loops iterate through the incoming and outgoing multi-edges of vertex i respectively. The number of outgoing and incoming edges for i depends on its degree d_i:

$$d_i = \sum_{j}^{n} a_{ij} \tag{4.15}$$

where the adjacency matrix $\mathbf{A} = [a_{ij}]$. The upper bound for the degree of a vertex is the number of vertices n in the graph. Each iteration contains another loop that, for every edge e in E, adds a new edge to $\mathcal{E}_{rep}$. Thus, the average number of iterations for this loop is equal to the average multiplicity of the multi-edges in the multigraph. Let us assume that the cost function c represents the unit time taken to run an operation. From these considerations, we estimate the run time t_1 and t_2 for the first two loops as follows:

$$t_1 = t_2 = nm\mathsf{c} \tag{4.16}$$

where m is the average multiplicity of the multi-edges and n is the number of vertices in the graph. The third loop iterates through all the unordered pairs in $\mathcal{V}_{rep}$, thus the run time t_3 is estimated as:

$$t_3 = \frac{m \times (m-1)}{2}\mathsf{c} \tag{4.17}$$

and the overall execution time t_{exp} for EXPANDVERTEX is estimated as:

$$t_{exp} = nm\mathsf{c} + \frac{m^2}{2}\mathsf{c} - \frac{m}{2}\mathsf{c} \tag{4.18}$$

and the procedure is of time complexity $\mathcal{O}(nm + m^2)$

The MULTIGRAPHEXPAND algorithm is made of two for-loops. The first loop iterates over n elements, where n is the number of vertices in the input multigraph. The second loop iterates over a subset of the vertex set of the input multigraph, hence it iterates over a maximum of n elements as well. Both loops perform a similar set of operations. For every iteration, the multi-edges of the current vertex v are queried. We assume that this operation is supported by a data structure that can access the edges of a vertex directly, thus getting the multi-edges of v in constant time. The

EXPANDVERTEX procedure is invoked once per iteration and the vertex and edge sets of the graph are updated accordingly.

The overall run time estimate for MULTIGRAPHEXPAND is therefore:

$$t = 2n(\mathsf{c} + t_{exp}) = 2n\mathsf{c} + 2n^2m\mathsf{c} + 2n\frac{m^2}{2}\mathsf{c} - 2n\frac{m}{2}\mathsf{c} \tag{4.19}$$

and the algorithm if of time complexity $\mathcal{O}(n^2m + nm^2)$.

In general, the number of vertices n in the input multigraph is much larger than the average multiplicity of the multi-edges in the same graph ($n \gg m$), therefore:

$$n^2m + nm^2 = nm(n + m) \approx n^2m \tag{4.20}$$

and MULTIGRAPHEXPAND is of time complexity $\mathcal{O}(n^2m)$ in general, as we shall demonstrate empirically in section (4.6).

4.5.2 Implementation

In this section we present a MATLAB implementation of the MULTIGRAPH-EXPAND and EXPANDVERTEX algorithms. This implementation is going to be employed for our experimental validation in later chapters.

Listing 4.1: MULTIGRAPHEXPAND *algorithm - MATLAB implementation*

```
function [E, corr] = multigraph_expand(EdgeTable)
% Perform the Multigraph Expansion algorithm.
%
% [E, corr] = multigraph_expand(EdgeTable) returns an edge table that
% encodes the simple graph that results from performing the Multigraph
% Expansion algorithm on the multigraph passed as input. EdgeTable is
% a table encoding the multigraph we wish to expand. E is the edge
% table that encodes the expanded simple graph and corr is an array or
% size 1xn, with n equal to the number of nodes in the original
% multigraph, storing the correspondences between the indices of the
% original vertices and the indices of their replicas: e.g. if vertex
% at index 4 has replicas with indices 8,9,10, then
% corr{4} = [8, 9, 10].

E = sort_edges(EdgeTable);
n = max([EdgeTable.i; EdgeTable.j]);

% Store correspondences between nodes and replicas
corr = cell(1,n);


% Expand all nodes that have 1+ incoming/outgoing multi-edges
node_cache = [];

for i = 1:n-1
    me_count = count_multiedges(E,i);
    if me_count > 1
        % Replicate node
        [S, c] = replicate_node(E,i);

        % Delete starting node before adding in order not to overwrite
        E(E.i == i,:) = [];

        % Add new edges to the graph and save indexes of node replicas
        E = [E; S];
        corr{i} = c;
    elseif me_count > 0
        % Save node id for later
        node_cache = [node_cache; i];
    end
end


% Expand the remaining nodes that an incoming/outgoing multi-edge
for i = 1:length(node_cache)
    nid = node_cache(i);
    if count_multiedges(E,nid) > 0
        % Replicate node
```

```
49          [S, c] = replicate_node(E,nid);
50
51          % Delete starting node before adding in order not to overwrite
52          E(E.i == nid,:) = [];
53
54          % Add new edges to the graph and save indexes of node replicas
55          E = [E; S];
56          corr{nid} = c;
57      end
58  end
59
60
61  % Add identity constraints between replicas of the same node
62  for i = 1:n
63      if ¬isempty(corr{i})
64          pairs = nchoosek(corr{i}, 2);
65          for j = 1:size(pairs,1)
66              r = edgetable_row(pairs(j,1),pairs(j,2),...
67                  mat2cell(eye(3),3,3),mat2cell(eye(3),3,3));
68              E = [E; r];
69          end
70      end
71  end
72
73  E = sort_edges(E);
74  end
```

Listing 4.2: EXPANDVERTEX *algorithm - MATLAB implementation*

```
1   function [S, corr] = replicate_node(E,i)
2   % Perform the optimized node replication algorithm.
3   %
4   % [S, corr] = replicate_node(E,i) returns the edges that result
5   % from the expansion of node with index i in edge table E.
6   % Correspondences between original nodes and its replicas are
7   % stored in corr. E is an edge table representing the original
8   % multigraph, while i is a int32 scalar representing the index
9   % of the node to replicate
10
11  S = empty_edgetable();
12  corr = [];
13
14  % Get edges starting from node i
15  E_i = E(E.i == i,:);
16  last_idx = max([E.i; E.j]);
17
18  % Loop through all nodes with index higher than i to convert ...
        multi-edges
19  % into simple edges
20
21  for j = i+1:max(E_i.j)
22      % Get edges between nodes i and j
23      E_ij = E_i(E_i.j == j,:);
24
25      for idx = 1:size(E_ij,1)
26          if idx == 1
27              % Recycle the original vertex index not to leave blank ...
```

```
                    indices
28              k = i;
29          else
30              k = last_idx + idx - 1;
31          end
32
33          Z_ij = E_ij(idx,:).Z_ij;
34          Z_ji = E_ij(idx,:).Z_ji;
35
36          % Register a new edge
37          if k > j
38              r = edgetable_row(j,k,Z_ji,Z_ij);
39              S = [S; r];
40          else
41              r = edgetable_row(k,j,Z_ij,Z_ji);
42              S = [S; r];
43          end
44
45          % Register a correspondence between the original vertex and its
46          % replica
47          corr = unique([corr, k]);
48      end
49  end
50  end
```

Data structure

Let us describe the data structure used to represent the input group-labeled multigraph for the purposes of this implementation.

Since a labeled multigraph admits multiple edges between any pair of vertices, it is not possible to write a matrix $\mathbf{Z}$, as defined in section (3.3), to encode all of the consistency constraints represented by the multi-edges. Therefore, we encode labeled multigraphs in a table-like data structure in which each row represents an edge in the graph. This allows for an unlimited number of edges between any pair of vertices in the graph. We call this data structure *edge table*.

In an edge table a row represents an edge and it contains the following columns:

- i: the source vertex of the edge
- j: the target vertex of the edge
- $\mathbf{Z}_{ij}$: the label of the edge
- $\mathbf{Z}_{ji}$: the inverse label of the edge

Let us give a brief example of how an edge table can be used to encode a multigraph and its expanded representation. Consider the minimal multigraph in Figure (4.1), it can be encoded in an edge table as follows:

Table 4.1: *An edge table encoding the minimal multigraph*

i	j	$\mathbf{Z}_{ij}$	$\mathbf{Z}_{ji}$
x_1	x_2	$z_{12,1}$	$z_{12,1}^{-1}$
x_1	x_2	$z_{12,2}$	$z_{12,2}^{-1}$

Notice how the edge table can encode multi-edges by adding multiple rows with the same *source* and *target* vertices, each with their own edge label. If we were to expand the graph in Figure (4.1), the resulting edge table would be as follows:

Table 4.2: *An edge table encoding the expanded graph of the multigraph in Table (4.1)*

i	j	$\mathbf{Z}_{ij}$	$\mathbf{Z}_{ji}$
x_{11}	x_2	$z_{12,1}$	$z_{12,1}^{-1}$
x_{12}	x_2	$z_{12,2}$	$z_{12,2}^{-1}$
x_{11}	x_{12}	1_Σ	1_Σ

Given that this edge table does not contain multiple rows with the same *source* and *target* vertices, it can be converted into a matrix of consistency constraints $\mathbf{Z}$ without any loss of information.

4.6 Experimental validation

In this section, we validate the theoretical results regarding the complexity of the MULTIGRAPHEXPAND algorithm and provide a comparison between *optimized graph expansion* and a naive expansion approach that does not follow the greedy strategy presented in section (4.4.3).

4.6.1 Computational complexity

We ran experiments in order to estimate the computational complexity of our implementation of the MULTIGRAPHEXPAND algorithm and whether the theoretical results in section (4.5.1) check out. In order to achieve this, we ran the algorithm on a number of synthetic multigraphs with a variable number of vertices n and a variable average multiplicity m_{avg} of the multi-edges. Each experiment consisted of 50 runs of the algorithm on randomly generated multigraphs.

The plot in Figure (4.17) report the average execution time of the algorithm as a function of the number of vertices n and the average multiplicity m_{avg} of the multi-edges. As expected, the execution time shows a quadratic growth as n increases. Higher values of m_{avg} have an effect on the slope of the curve, making it grow faster, while still being quadratic.

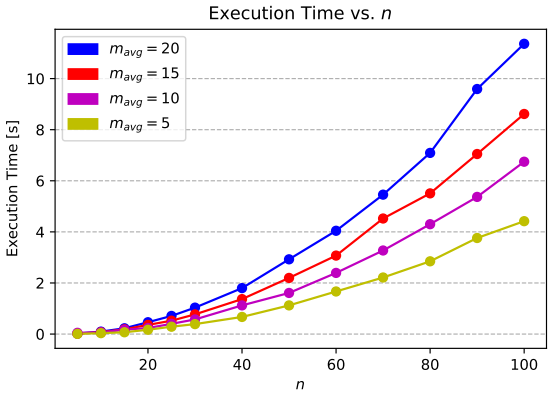

Figure 4.17: *The plot shows the execution time of the* MULTIGRAPHEXPAND *algorithm as a function of the number of vertices and the average multiplicity of the multi-edges.*

The plot in Figure (4.18) shows how the execution time of the algorithm responds to a change in the average multiplicity m_{avg} of the multi-edges for a multigraph with $n = 15$ vertices. As long as $n \gg m_{avg}$, processing time shows a growth that is linear in the multiplicity of the multi-edge since com-

plexity is $\mathcal{O}(n^2 m_{avg})$. When the value of m_{avg} approaches or exceeds n, then the execution time starts to show a quadratic growth, since the approximation that $n + m_{avg} \approx n$ in Eq. (4.20) is no longer valid. Asymptotically, the algorithm is of time complexity $\mathcal{O}(nm_{avg}^2)$.

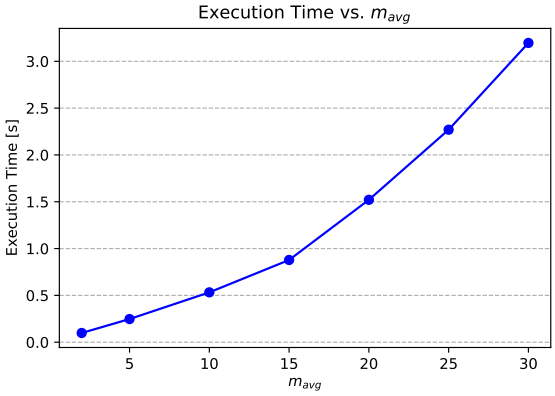

Figure 4.18: *The plot shows the execution time for the* MULTIGRAPHEXPAND *algorithm on a multigraph with* $n = 15$ *as a function of* m_{avg}.

4.6.2 Optimized vs. naive graph expansion

We ran experiments to show how the *optimized graph expansion* strategy introduced in section (4.4.3) and the MULTIGRAPHEXPAND algorithm generate expanded simple graphs that contain fewer vertices compared to those obtained from naive expansion. We refer to naive expansion as the non-greedy method that expands all vertices in order, without considering the number of incoming (outgoing) multi-edges for a vertex.

Table (4.3) shows the number of output vertices n_{out} for both the optimized and naive graph expansion strategies. These results are obtained from synthetic multigraphs with a varying number of vertices n and average multiplicity of the multi-edges m_{avg}. As expected, results show a reduction in the size of the expanded graph. Overall, since the optimized approach never performs worse compared to the naive solution and it does not affect computational complexity significantly, we suggest to use the optimized approach in all circumstances.

Table 4.3: *Optimized vs. naive graph expansion in terms of vertices in the output expanded graph. n is the number of vertices in the original multigraph and m_{avg} is the average multiplicity of the multi-edges. n_{out} stands for the number of vertices in the output graph.*

Optimized vs. naive graph expansion			
n	m_{avg}	n_{out} optimized	n_{out} naive
10	2	17	18
10	5	37	40
10	10	74	78
20	5	87	93
30	5	134	145

CHAPTER 5

Synchronization on multigraphs

In this chapter, we expand on the theoretical background introduced in chapters (3) and (4) to devise a principled and efficient solution to the problem of synchronization on group-labeled multigraphs.

The current state of the art for synchronizing measurement multigraphs is to reduce them by combining the multi-edges and their labels into simple edges. Once simplified, closed-form synchronization techniques designed for simple graphs are applied in order to estimate the vertex labeling for the original multigraph. For the purposes of this work, we refer to this technique as *edge averaging*. However, this approach falls short because: i) averaging is defined only for some groups (*e.g.* rotations in SO(3)) and ii) the solution is sub-optimal, leading to an estimator that is less precise and accurate. For instance, consider the problem of estimating a scale factor s that links two matrices with noisy entries: $\mathbf{A} = s\mathbf{B}$. The optimal estimate is the least squares solution $s = \text{tr}(\mathbf{B}^\intercal\mathbf{A})/\text{tr}(\mathbf{B}^\intercal\mathbf{B})$ which is different from, *e.g.*, taking the average of the entry-wise division $\mathbf{A}./\mathbf{B}$.

The MULTIGRAPHEXPAND algorithm in chapter (4) allows us to turn any labeled multigraph into a simple graph, while, at the same time, preserving the pairwise measurements and the underlying structure of the original multigraph. As we shall discuss, applying closed-form solutions to an

expanded graph is not straightforward, given that, in the general case, the estimated labeling for the replicas of the same vertex will not assign them the same group element. As a matter of fact, the noise in the relative measurements causes the integrity of the consistency constraints, including the identity ones, to break when the graph is synchronized.

The main focus of this chapter is the introduction of the first synchronization algorithm for multigraphs that is based on a principled constrained eigenvalue optimization problem. Our method is a general solution that can cope with any linear group, thus solving the intrinsic issues of edge averaging.

Finally, in order to validate our theoretical results, we report the results of synthetic experiments aimed at comparing the precision and accuracy of our method with respect to edge averaging.

5.1 Challenges of multigraph synchronization

In this section, we give an overview of the possible approaches to synchronization on measurement multigraphs, discussing the challenges that arise when synchronizing graphs expanded with the MULTIGRAPHEXPAND algorithm and exploring the current state of the art techniques such as *edge averaging*. Our analysis will focus on the limitations of these approaches and their inherent issues, while introducing the reader to the reasons behind our method for synchronization on multigraphs.

For the purposes of this analysis, we are going to consider multigraphs whose measurements are noisy, given that the noiseless case is not relevant in real-world scenarios.

5.1.1 Synchronization on expanded graphs

We already established that any measurement multigraph can be expanded into a simple graph by applying the MULTIGRAPHEXPAND algorithm. In order to obtain an estimate for the labeling of the original multigraph, we can apply the spectral or null-space solutions to the expanded graph.

In the general case, the labeling estimated using this approach does not assign the same label to the replicas of a vertex in the original multigraph. This is because, if $\epsilon \neq 0$, a vertex labeling $\tilde{x}$ that satisfies all the consistency constraints at once does not exist. Therefore, the vertex labeling $\tilde{x}$ obtained by applying a closed-form solution will minimize ϵ at the cost of breaking pairwise constraints. The result is that the identity constraints 1_Σ introduced by the MULTIGRAPHEXPAND algorithm between replicas are also broken,

meaning that there is no guarantee that the same label will be assigned to those replicas.

Let us provide a practical example to show how these issues impact on the applicability of existing synchronization techniques in the context of measurement multigraphs. Consider a simple $\mathbb{R}^*$-labeled multigraph with two global states and two noisy pairwise measurements $z_{12,1} = 2.1$ and $z_{12,2} = 1.8$ and its expanded simple graph. It is clear that no vertex label assignment exists such that $\epsilon = 0$, i.e. there is no label assignment for which all consistency constraints can be satisfied.

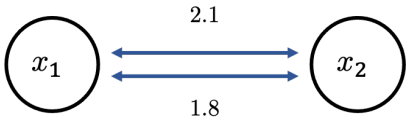

Figure 5.1: *A simple measurement multigraph with noisy measurements. Notice how the labels assigned to the edges that connect the same two vertices x_1 and x_2 are different.*

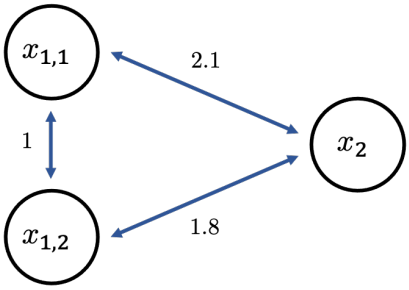

Figure 5.2: *The measurement graph obtained by applying the* MULTIGRAPHEXPAND *algorithm to the measurement multigraph in Figure (5.1).*

If we apply a closed-form solution on the expanded graph, in this case the spectral solution, we obtain the following vertex labeling $\tilde{x}$:

$\tilde{x}_{1,1}$	0.6741
$\tilde{x}_{1,2}$	0.6509
$\tilde{x}_2$	0.3491

Let us compute the edge labeling induced by $\tilde{x}$ as follows:

$\tilde{x}_{1,1} * \tilde{x}_2^{-1}$	1,9309
$\tilde{x}_{1,2} * \tilde{x}_2^{-1}$	1.8645
$\tilde{x}_{1,1} * \tilde{x}_{1,2}^{-1}$	1.0356

Intuitively, these results show that the spectral solution is trying to find a compromise between the pairwise measurements in order to minimize the overall consistency error in the graph, which can be computed as follows:

$$\epsilon = (2.1 - 1,9309)^2 + (1.8 - 1.8645)^2 + (1 - 1.0356)^2$$

This has the inevitable side-effect that the vertex labels assigned to replicas of x_1, that is $x_{1,1}$ and $x_{1,2}$, are not equal, given that the constraint that links them is broken to minimize ϵ.

The problem of aggregating or choosing the best label among the ones assigned to replicas is not straightforward and can lead to sub-optimal results, especially for those groups for which averaging is not well defined.

For these reasons, we would like to have a guarantee that the algorithm will not break the identity constraints between replicas. In other words, we wish to find a compromise aimed at minimizing ϵ only on the edges that do not connect replicas. The goal is to find a vertex labeling that assigns the same exact labels to vertex replicas.

5.1.2 Sub-optimal solutions

In order to better understand the reasoning behind our contribution to the problem of multigraph synchronization, let us introduce the current approaches to this problem and explain why their solutions are sub-optimal.

Vertex averaging

An approach to the problem of finding a common labeling for replicas is to combine their labels by averaging them. This procedure is sub-optimal for a number of reasons, including the fact that it requires a method to combine an arbitrary number of elements of a group Σ. This is not always straightforward for groups for which averaging is not well defined. In addition, this approach is all but general, since it requires modifications for each group Σ on which synchronization is performed.

Edge averaging

Edge averaging consists in reducing the multi-edges in a multigraph to simple ones by averaging their measurements. This suffers from several shortcomings. First of all, averaging is well defined only for some groups. Secondly, even when it is possible to average the multi-edges, the resulting estimator has sub-optimal statistical properties.

Weighted synchronization

So far, we assumed that all pairwise measurements in a group-labeled graph are equally reliable. In some cases, measurements can be associated to a weight w_{ij}, that indicates the importance or reliability of such measurement. This results in a weighted graph $\mathcal{G}$, which can be defined by an adjacency matrix $\mathbf{A} = [w_{ij}]$, meaning that $\mathbf{A}$ will contain the weight of an edge instead of just binary values to indicate the presence of an edge between vertices. In order to solve the problem of group-synchronization for a weighted graph, we employ the same techniques seen for simple graphs, with the only difference that the adjacency matrix is not binary anymore, but instead contains the weights associated to each edge. The result is a degree matrix $\mathbf{D}$ of the graph which differs slightly depending on the weight assigned to each edge.

An interesting possibility is to increase the weight of the identity constraints 1_Σ, introduced between the replicas, with respect to other pairwise measurements. Theoretically, as the weight of the edges between replicas increases, the vertex labeling for such vertices should converge to the same value: breaking the identity constraints will negatively add to the consistency error to a larger extent compared to other constraints, meaning that they should become harder to break.

The downside of this approach is the need to set a dissimilarity threshold between the labels of replicas under which we are satisfied with the results. For instance, we may expect the average squared Frobenius norm of the difference between the vertex labels of replicas to be lower than a certain threshold μ:

$$\frac{2}{n \times (n-1)} \sum_{i=1}^{n-1} \left(\sum_{j=i+1}^{n} \|x_{k,i} - x_{k,j}\|_F \right) < \mu \tag{5.1}$$

where $x_{k,i}$ and $x_{k,j}$ are the i-th and j-th replicas of vertex k respectively and n is the number of replicas of k. Finding a set of weights w_{ij} that ensures that this condition is satisfied is not straightforward and requires manual tuning of the weights. Finally, it is important to stress that no matter the weights, the labels obtained with this method will never converge to the same exact values. Therefore, this approach suffers from the same problems introduced for the previous approaches, albeit to a limited extent.

5.2 Constrained synchronization

In this section we introduce a novel method for multigraph synchronization – named *constrained synchronization* – which approaches group-labeled multigraphs from a new perspective: rather than averaging measures to collapse a multi-edge to a simple one, it works on expanded multigraphs and enforces identity constraints between the replicated vertices. This leads to a constrained optimization problem for which we derive a general closed-form solution that can be applied to graphs labeled with any linear group.

In section (5.2.1), we introduce a constrained eigenvalue optimization problem and its solution which serves as the theoretical foundation of constrained synchronization. In section (5.2.2) we provide an intuition of how constrained synchronization works and what it accomplishes. In section (5.2.3) we describe how to determine the set of linear constraints that the estimated vertex labeling must satisfy. Finally, in section (5.2.4) we formally define and solve the constrained synchronization problem.

5.2.1 Constrained eigenvalue optimization problem

In this section, we introduce a variant of the following eigenvalue optimization problem in [17]:

$$\max_{\mathbf{x}} \quad \mathbf{x}^\intercal \mathbf{A} \mathbf{x} \tag{5.2}$$

with the main difference being that $\mathbf{x}$ is subject to a set of linear constraints. This problem is introduced in [15] and admits a closed-form solution.

Consider a symmetric matrix $\mathbf{A} \in \mathbb{R}^d$ and a vector $\mathbf{c}$ such that $\mathbf{c}^\intercal \mathbf{c} = 1$. Let us define the following optimization problem:

$$\max_{\mathbf{x}} \quad \mathbf{x}^\intercal \mathbf{A} \mathbf{x} \tag{5.3}$$

subject to the constraints:

$$\mathbf{x}^\intercal \mathbf{x} = 1 \tag{5.4}$$

$$\mathbf{c}^\intercal \mathbf{x} = 0 \tag{5.5}$$

Let:

$$\mathcal{L} = \mathbf{x}^\intercal \mathbf{A} \mathbf{x} - \lambda(\mathbf{x}^\intercal \mathbf{x} - 1) + 2\mu \mathbf{x}^\intercal \mathbf{c} \tag{5.6}$$

where λ and μ are the Lagrange multipliers. Consider the derivative of (5.6) and set it to zero:

$$\mathbf{A}\mathbf{x}^\intercal - \lambda \mathbf{x} + \mu \mathbf{c} = 0; \tag{5.7}$$

Let us multiply equation (5.7) on the left by $\mathbf{c}^\intercal$. Referencing equation (5.14) and using the fact that $\mathbf{c}^\intercal\mathbf{c} = 1$, we obtain:

$$\mu = -\mathbf{c}^\intercal\mathbf{A}\mathbf{x} \tag{5.8}$$

From (5.7) and (5.8), we obtain:

$$\mathbf{P}\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \tag{5.9}$$

where $\mathbf{P} = (\mathbf{I} - \mathbf{c}^\intercal\mathbf{c})$. It is clear that Equation (5.9) is an eigenvalue problem with $\mathbf{P}$ being a projection matrix since $\mathbf{P}^2 = \mathbf{P}$.

The authors of [15] discuss the case in which $\mathbf{x}$ is subject to a set of linear constraints $\mathbf{C}$, instead of a single constraint $\mathbf{c}$, a case that is of particular interest for the purposes of constrained synchronization. Let us replace the constraint (5.14) with the set of constraints:

$$\mathbf{C}^\intercal\mathbf{x} = \mathbf{0} \tag{5.10}$$

where $\mathbf{C} \in \mathbb{R}^{d\times p}$ with rank r. Then, the projection matrix $\mathbf{P}$ can be computed as follows:

$$\mathbf{P} = (\mathbf{I} - \mathbf{C}\mathbf{C}^-) \tag{5.11}$$

where $\mathbf{C}^-$ is the generalized inverse of matrix $\mathbf{C}$. Therefore, the solution $\hat{\mathbf{x}}$ to the maximization problem (5.3) subject to the set of linear constraints encoded in $\mathbf{C}$ is equal to the eigenvector associated to the leading eigenvalue of $\mathbf{PA}$.

The same reasoning can be applied for the following minimization problem as well:

$$\min_{\mathbf{x}} \quad \mathbf{x}^\intercal\mathbf{A}\mathbf{x} \tag{5.12}$$

subject to the constraints:

$$\mathbf{x}^\intercal\mathbf{x} = 1 \tag{5.13}$$

$$\mathbf{C}^\intercal\mathbf{x} = \mathbf{0} \tag{5.14}$$

In this case, the solution to the problem is given by the eigenvector associated to the smallest eigenvalue of $\mathbf{PA}$, bearing in mind that, according to [15], $\mathbf{PA}$ has at least $r = \text{rank}(\mathbf{C})$ zero eigenvalues that should not be considered.

5.2.2 Intuition for constrained synchronization

In this section we introduce the idea behind constrained synchronization by analyzing a simple case on a minimal multigraph. The goal of this section

is to give the reader an insight into how constrained synchronization works before introducing it formally in the next sections.

Let us consider a synchronization problem defined on the minimal $GL(2)$-labeled multigraph, containing vertices x_1 and x_2 connected by a single multi-edge with multiplicity 2. We can expand it into a simple graph by replicating x_1 with two replicas, as shown in Figure (5.4).

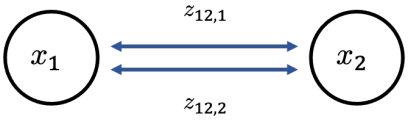

Figure 5.3: *The multigraph taken as an example in this section*

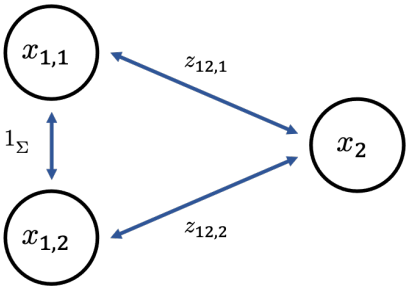

Figure 5.4: *The simple graph that is obtained by expanding the multigraph in Figure (5.3)*

We write the matrix $\mathbf{X}$ of vertex labels and the matrix $\mathbf{Z}$ of edge labels for the expanded graph in Figure (5.4):

$$\mathbf{X} = \begin{pmatrix} \mathbf{X}_{1,1} \\ \mathbf{X}_{1,2} \\ \mathbf{X}_2 \end{pmatrix}, \quad \mathbf{Z} = \begin{pmatrix} \mathbf{I}_2 & \mathbf{I}_2 & \mathbf{Z}_{12,1} \\ \mathbf{I}_2 & \mathbf{I}_2 & \mathbf{Z}_{12,2} \\ \mathbf{Z}_{21,1} & \mathbf{Z}_{21,2} & \mathbf{I}_2 \end{pmatrix} \tag{5.15}$$

$\mathbf{X}_{1,1}$, $\mathbf{X}_{1,2}$ and $\mathbf{X}_2$ are elements of $GL(2)$, so $\mathbf{X}$ can be rewritten as follows:

$$\mathbf{X} = \begin{pmatrix} x_{(1,1),1} & x_{(1,1),2} \\ x_{(1,1),3} & x_{(1,1),4} \\ x_{(1,2),1} & x_{(1,2),2} \\ x_{(1,2),3} & x_{(1,2),4} \\ x_{2,1} & x_{2,2} \\ x_{2,3} & x_{2,4} \end{pmatrix} \tag{5.16}$$

where $x_{(i,k),j}$ is the j-th element of the vertex label of the k-th replica of vertex i. k is omitted if a vertex is not a replica of another vertex.

Consider the null-space solution for solving the synchronization problem on $GL(d)$ introduced in section (3.3.3). In general, *i.e.* $\mathbf{Z}_{12,1} \neq \mathbf{Z}_{12,2}$, the labels $\mathbf{X}_{1,1}$ and $\mathbf{X}_{1,2}$ obtained by computing the eigenvectors corresponding to the $d = 2$ smallest eigenvalues of $\mathbf{D} \otimes \mathbf{I}_d - \mathbf{Z}_\mathbf{A}$ are not equal.

If we consider each column of $\mathbf{X}$ as an independent eigenvector, it is possible to write a set of linear constraints to enforce the identity constraint between $\mathbf{X}_{1,1}$ and $\mathbf{X}_{1,2}$. Let us write the constraints for the first eigenvector:

$$\begin{cases} x_{(1,1),1} = x_{(1,2),1} \\ x_{(1,1),3} = x_{(1,2),3} \end{cases} \tag{5.17}$$

and for the second eigenvector:

$$\begin{cases} x_{(1,1),2} = x_{(1,2),2} \\ x_{(1,1),4} = x_{(1,2),4} \end{cases} \tag{5.18}$$

If we consider the eigenvectors separately, it becomes clear that both systems of equations (5.17) and (5.18) enforce the same constraints on the elements of their respective vectors. Given their equivalence, we consider only the constraints referring to the first eigenvector in $\mathbf{X}$.

Let us rewrite the constraints in (5.17) as follows:

$$\begin{cases} x_{(1,1),1} - x_{(1,2),1} = 0 \\ x_{(1,1),3} - x_{(1,2),3} = 0 \end{cases} \tag{5.19}$$

Referring to the definitions introduced in section (5.2.1), we introduce a matrix $\mathbf{C}$ of linear constraints:

$$\mathbf{C} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \tag{5.20}$$

so that the system of equations in (5.19) can be rewritten as $\mathbf{C}^\intercal \mathbf{x} = \mathbf{0}$, where $\mathbf{x} = \left(x_{(1,1),1}, x_{(1,1),3}, x_{(1,2),1}, x_{(1,2),3}, x_{2,1}, x_{2,3}\right)^\intercal$.

After properly defining $\mathbf{C}$, it is sufficient to solve a constrained version of the original problem by finding the eigenvectors $\mathbf{x}$ corresponding to the

$d = 2$ smallest eigenvalues of $\mathbf{D} \otimes \mathbf{I}_d - \mathbf{Z}_\mathbf{A}$ subject to $\mathbf{C}^\intercal \mathbf{x}$. In section (5.2.4) we formally introduce a closed-form solution to this problem.

Having discussed a simple example on a minimal multigraph, in the next section we show how to build the matrix of linear constraints $\mathbf{C}$ in the general case of a $GL(d)$-labeled multigraph.

5.2.3 Constraints between vertex replicas

In this section, we discuss the general case of defining equality constraints between vertex replicas when synchronization is performed on $GL(d)$ and an arbitrary number of vertices in the multigraph is replaced by an also arbitrary number of replicas.

Let us consider two matrices $\mathbf{A}$, $\mathbf{B}$ both belonging to group $GL(d)$, i.e. $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{d \times d}$ and $det(\mathbf{A}) \neq 0$ and $det(\mathbf{B}) \neq 0$:

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & \dots & a_{1,d} \\ \dots & \ddots & \dots \\ a_{d,1} & \dots & a_{d,d} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} b_{1,1} & \dots & b_{1,d} \\ \dots & \ddots & \dots \\ b_{d,1} & \dots & b_{d,d} \end{pmatrix} \tag{5.21}$$

In order for the first column of $\mathbf{A}$ to be equal to the first column of $\mathbf{B}$ we need to define a set of d linear constraints as follows:

$$\begin{cases} a_{1,1} - b_{1,1} = 0 \\ a_{2,1} - b_{2,1} = 0 \\ \dots \\ a_{d,1} - b_{d,1} = 0 \end{cases} \tag{5.22}$$

The same constraints apply to the other $d-1$ columns of $\mathbf{A}$ and $\mathbf{B}$, bearing in mind a change in the subscript of $a_{i,j}$ and $b_{h,k}$.

Let us consider the case in which we need to set an equality constraint between the elements of the first column of k different matrices instead of just two. Consider k invertible matrices $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_k \in \mathbb{R}^{d \times d}$. The elements in matrices $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_k$ must satisfy the following conditions:

$$\begin{cases} x_{1,1} = x_{1,2} = \dots = x_{1,k} \\ \dots \\ x_{d,1} = x_{d,2} = \dots = x_{d,k} \end{cases} \tag{5.23}$$

where $x_{i,j}$ is the i-th element of the first column of matrix $\mathbf{X}_j$. Let us rewrite

the system of equations as follows:

$$\begin{cases} x_{1,1} - x_{1,2} = 0 \\ x_{1,2} - x_{1,3} = 0 \\ \dots \\ x_{1,k-1} - x_{1,k} = 0 \end{cases}, \begin{cases} x_{2,1} - x_{2,2} = 0 \\ x_{2,2} - x_{2,3} = 0 \\ \dots \\ x_{2,k-1} - x_{2,k} = 0 \end{cases}, \dots \begin{cases} x_{d,1} - x_{d,2} = 0 \\ x_{d,2} - x_{d,3} = 0 \\ \dots \\ x_{d,k-1} - x_{d,k} = 0 \end{cases} \tag{5.24}$$

where $x_{i,j}$ is the same as above. Please consider that, in the equation above, the system of equations is split just for clarity and should not be interpreted as d separate systems.

From (5.24), we know that the number of linear constraints that need to be introduced in order to ensure that the columns of matrices $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_k$ are the same is equal to $d \times (k-1)$, where d is the number of rows (and columns) of matrices in $GL(d)$ and k is the number of matrices to constrain.

Let us extend these results to the problem of synchronization on expanded multigraphs. In section (3.3), we discussed how vertex labels can be collected in a matrix $\mathbf{X} \in \mathbb{R}^{dn \times d}$ by stacking the labels of the n vertices in the graph. Consider the case in which a vertex x_j is replicated k times, so that, in $\mathbf{X}$, label $\mathbf{X}_j$ is replaced by k labels $\mathbf{X}_{j,1}, \dots, \mathbf{X}_{j,k}$. We can assume without any loss of generality that all labels of replicas are placed adjacently in $\mathbf{X}$ as follows:

$$\mathbf{X} = \begin{pmatrix} \vdots \\ \mathbf{X}_{j,1} \\ \mathbf{X}_{j,2} \\ \vdots \\ \mathbf{X}_{j,k} \\ \vdots \end{pmatrix} \tag{5.25}$$

Referencing the system of equations in (5.24), we can write a set of $d(k-1)$ linear constraints. Each constraint $\mathbf{c}_{i,j}$ is a $\mathbb{R}^{dn \times 1}$ vector defined as follows:

$$\mathbf{c}_{i,j} = [\ 0, \dots, 1, \dots, -1, \dots, 0\ ]^\intercal \tag{5.26}$$

with it having only two elements different from 0, corresponding to the elements of the labels appearing in the constraint that $\mathbf{c}$ encodes. Vectors $\mathbf{c}_{i,j}$ are stacked horizontally in a matrix $\mathbf{C}_j$ of size $dn \times d(k-1)$ encoding the set of constraints that enforce the equality between replicas of $\mathbf{X}_j$:

$$\mathbf{C}_j = [\ \mathbf{c}_{1,j}, \mathbf{c}_{2,j}, \dots, \mathbf{c}_{k-1,j}\ ] \tag{5.27}$$

If more than one vertex in a multigraph is expanded, then we obtain m matrices $\mathbf{C}_j$, where m is the number of replicated vertices. The number of columns of each matrix $\mathbf{C}_j$ varies depending on the number of replicas the vertex x_j is replaced with.

Let $\mathbf{C}$ be the matrix containing the final set of linear constraints. $\mathbf{C}$ is obtained by stacking the m matrices $\mathbf{C}_j$ horizontally as follows:

$$\mathbf{C} = [\ \mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_m\ ] \tag{5.28}$$

Let us define an algorithm that, given a mapping M between vertices in a multigraph and their replicas in the expanded graph, returns the matrix of linear constraints $\mathbf{C}$ that enforces the equality between labels of replicas for the same vertex. In order to access the replicas of a vertex i in M, we use the array-like notation $\mathsf{M}[i]$: an index i corresponds to a replicated vertex and $\mathsf{M}[i]$ returns the vector of replicas for i.

In Algorithm 5 we iterate through each replicated vertex v in M (lines 2 - 14) and get the vector of replicas $\mathbf{m}_v$ for v (line 4). In lines 6 - 13 we encode the constraints for the replicas of v, as specified in Eq. (5.24). In line 8 we initialize $\mathbf{c}$ as a $dn \times 1$ vector of zeros and in lines 9 - 10 we update $\mathbf{c}$ in order to encode one of the equality constraints between the labels of the replicas $\mathbf{m}_v$ for v. The linear constraint in vertex form $\mathbf{c}$ is appended to $\mathbf{C}$ (line 11).

Algorithm 5 Building the matrix of linear constraints

Input: A mapping M between vertices in the underlying multigraph and their replicas in the expanded graph

Output: A matrix $\mathbf{C}$ containing the set of linear constraints that enforce the equality between the labels of replicas of the same vertex in the underlying multigraph

1: $d =$ number of rows (columns) of matrices in $GL(d)$
2: **for** every replicated vertex $v \in \mathsf{M}$ **do**
3: ▷ Get the vector of vertices that are replicas of v in the expanded graph
4: $\mathbf{m}_v = \mathsf{M}[v]$
5: ▷ Add linear constraints between the elements of replicas
6: **for** j from 1 to d **do**
7: **for** k from 1 to length$(\mathbf{m}) - 1$ **do**
8: $\mathbf{c} = \mathbf{0}_{dn}$ ▷ $\mathbf{c}$ is a $dn \times 1$ vector filled with zeros
9: $\mathbf{c}(d \times \mathbf{m}_v[k] + j - d) = 1$
10: $\mathbf{c}(d \times \mathbf{m}_v[k+1] + j - d) = -1$
11: $\mathbf{C} = [\mathbf{C},\ \mathbf{c}]$
12: **end for**
13: **end for**
14: **end for**

5.2.4 Solving the constrained synchronization problem

In this section, we define the algorithm for constrained synchronization applied to any Σ-labeled measurement multigraph by combining the results introduced in sections (5.2.1) and (5.2.3).

Consider a Σ-labeled multigraph Γ. Let us apply the MULTIGRAPH-EXPAND algorithm from section (4.5) and obtain the expanded graph of Γ. This graph is described by $\mathbf{Z_A}$, the matrix containing the consistency constraints or edge labels, and $\mathbf{A}$, its adjacency matrix.

Let us consider the definition of null-space solution from section (3.3.3).

Proposition 1 ([1]). A consistent vertex labelling $\mathbf{X}$ satisfies the following equation:

$$\mathbf{Z_A X} - (\mathbf{D} \otimes \mathbf{I}_d)\mathbf{X} = 0 \tag{5.29}$$

where $\mathbf{D}$ denotes the degree matrix of the graph.

This proposition is at the basis of the null-space solution for synchronization on a simple graph, which, due to noise, solves Equation (5.29) by least squares:

$$\min_{\mathbf{X}^\top \mathbf{X} = \mathbf{I}_d} \|\mathbf{M X}\|_F^2 \tag{5.30}$$

where $\mathbf{M} = \mathbf{Z_A} - (\mathbf{D} \otimes \mathbf{I}_d)$ is defined from the matrix of incomplete relative measurements. It can be proved that synchronization admits a closed-form solution as the null-space of $\mathbf{M}$ [1], which in turn can be derived from the least eigenvectors of $\mathbf{M}^\top \mathbf{M}$.

Synchronizing the *expanded* graph by solving Equation (5.30) for the unknown labels $\mathbf{X}$ falls short, as it does not guarantee that replicated vertices have been assigned the same label. Indeed, the constraints given by edges labeled with the identity are treated as "soft" ones, like all the others.

In order to solve this problem, let us apply the algorithm introduced in section (5.2.3) and obtain the matrix $\mathbf{C}$ containing the set of linear constraints that enforce the identity constraint between the labels of replicas. We are therefore led to a constrained version of Equation (5.30):

$$\min_{\mathbf{X}} \|\mathbf{M X}\|_F^2 \quad \text{subject to } \mathbf{X}^\top \mathbf{X} = \mathbf{I}_d, \quad \mathbf{C}^\top \mathbf{X} = 0, \tag{5.31}$$

where $\mathbf{C}^\top \mathbf{X} = 0$ enforces the equality between replicated vertices. The constrained problem (5.31) admits a closed-form solution as demonstrated in the following theorem.

Theorem 1. *The solution to problem* (5.31) *is given by the eigenvectors corresponding to the* d *smallest non-zero eigenvalues of* $\mathbf{P}\mathbf{M}^\top\mathbf{M}$*, where* $\mathbf{P} = \mathbf{I} - \mathbf{C}\mathbf{C}^-$ *and* $\mathbf{C}^- = (\mathbf{C}^-\mathbf{C})^{-1}\mathbf{C}^-$ *is the pseudo-inverse of* $\mathbf{C}$*.*

Proof. Problem (5.31) is equivalent to

$$\min_{\mathbf{X}} \operatorname{tr}\left(\mathbf{X}^\top(\mathbf{M}^\top\mathbf{M})\mathbf{X}\right) \text{ s. t. } \mathbf{X}^\top\mathbf{X} = \mathbf{I}_d,\ \mathbf{C}^\top\mathbf{X} = 0. \tag{5.32}$$

The Lagrangian of the cost function to this problem is

$$\begin{aligned}\mathcal{L} = \operatorname{tr}\left(\mathbf{X}^\top(\mathbf{M}^\top\mathbf{M})\mathbf{X}\right) + \operatorname{tr}\left(\Delta(\mathbf{X}^\top\mathbf{X} - \mathbf{I}_d)\right) + \\ \operatorname{tr}\left(\Gamma\mathbf{C}^\top\mathbf{X}\right),\end{aligned} \tag{5.33}$$

where Δ and Γ are matrices of unknown Lagrange multipliers. Setting to zero the derivatives with respect to $\mathbf{X}$ we have

$$\frac{\partial\mathcal{L}}{\partial\mathbf{X}} = 2\mathbf{M}^\top\mathbf{M}\mathbf{X} + 2\mathbf{X}\Delta + \mathbf{C}\Gamma^\top = 0. \tag{5.34}$$

Revisiting the approach described in [15] and section (5.2.1), we left-multiply by $\mathbf{C}^\top$, then using $\mathbf{C}^\top\mathbf{X} = 0$, we obtain

$$2\mathbf{C}^\top\mathbf{M}^\top\mathbf{M}\mathbf{X} + \mathbf{C}^\top\mathbf{C}\Gamma^\top = 0, \tag{5.35}$$

from which we get

$$\Gamma^\top = -2\mathbf{C}^{-}\mathbf{M}^\top\mathbf{M}\mathbf{X}. \tag{5.36}$$

Plugging (5.36) into (5.34) yields

$$\left(\mathbf{I} - \mathbf{C}\mathbf{C}^\dagger\right)\mathbf{M}^\top\mathbf{M}\mathbf{X} = -\mathbf{X}\Delta. \tag{5.37}$$

Let $\mathbf{P} = \mathbf{I} - \mathbf{C}\mathbf{C}^{-}$ and $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_d]$, the last equation implies that the eigenvectors of $\mathbf{P}\mathbf{M}^\top\mathbf{M}$ are stationary points for (5.33), and the minimum is obtained when $\mathbf{x}_i$ are the orthogonal eigenvectors corresponding to the d smallest non-zero eigenvalues of $\mathbf{P}\mathbf{M}^\top\mathbf{M}$. From [15], at least $r = \operatorname{rank}(\mathbf{C})$ eigenvalues of $\mathbf{P}\mathbf{M}^\top\mathbf{M}$ are equal to 0 and their corresponding eigenvectors are the columns of $\mathbf{C}$. □

Even if $\mathbf{P}$ is symmetric, $\mathbf{P}\mathbf{M}^\top\mathbf{M}$ may not be necessarily symmetric, nonetheless:

Lemma 1. *The matrix* $\mathbf{P}\mathbf{M}^\top\mathbf{M}$ *has real eigenvalues.*

Proof. Since $\mathbf{P}$ is a projection matrix ($\mathbf{P}^2 = \mathbf{P}$) we get:

$$\operatorname{eig}(\mathbf{P}\mathbf{M}^\top\mathbf{M}) = \operatorname{eig}(\mathbf{P}^2\mathbf{M}^\top\mathbf{M}) = \operatorname{eig}(\mathbf{P}\mathbf{M}^\top\mathbf{M}\mathbf{P}). \tag{5.38}$$

Thus the eigenvalues of $\mathbf{P}\mathbf{M}^\top\mathbf{M}$ are the same of those of $\mathbf{P}\mathbf{M}^\top\mathbf{M}\mathbf{P}$ which is symmetric (and hence has real eigenvalues). □

Let us construct $\mathbf{X}$ as a $dn \times d$ matrix by juxtaposing the least d eigenvectors of $\mathbf{P}\mathbf{M}^\top\mathbf{M}$, which are real thanks to the above lemma. According to Theorem 1, such a matrix solves Problem (5.31): this is a relaxed problem, as the feasible set is given by $\mathbf{X} \in \mathbb{R}^{dn\times dn}$ with $\mathbf{X}^\top\mathbf{X} = \mathbf{I}_d$, instead of $\mathbf{X} \in \Sigma^n$. In order to recover the block-wise structure of $\mathbf{X}$ in Equation (3.8), it is hence necessary to project each $d \times d$ block of $\mathbf{X}$ onto the group Σ. When $\Sigma = SO(3)$, for instance, the final projection can be done via Singular Value Decomposition. This produces our closed-form solution to multigraph synchronization. In the presence of outliers, robustness can be easily gained via Iteratively Reweighted Least Squares (IRLS), as in [1].

5.2.5 Summary

To sum up, in order to apply *constrained synchronization*, we first expand a Σ-labeled measurement multigraph with the MULTIGRAPHEXPAND algorithm of section (4.5) to dilate it to a simple graph with replicated vertices. The hard constraints between replicas are then integrated with the constrained Problem (5.31), for which a closed-form solution is derived using Theorem 1. Finally, each block of the solution is projected on the group Σ. This approach is therefore general and can be applied to any linear group, such as $SO(3)$, which is relevant for a variety of vision applications.

5.2.6 Computational complexity

Constrained synchronization is generally more computationally expensive with respect to the spectral and null-space methods for simple measurement graphs.

From a memory complexity point of view, constrained synchronization requires to store an additional matrix $\mathbf{C}$ which, asymptotically, contains a maximum of dn^2 linear constraints, hence $\mathbf{C} \in \mathbb{R}^{dn\times dn^2}$, although $\mathbf{C}$ is usually very sparse. In addition, constrained synchronization is applied to expanded graphs obtained from labeled multigraphs, which can be significantly larger than simplified measurements graphs obtained by combining multi-edges.

Time complexity is also affected by the computation of the generalized inverse $\mathbf{C}^-$ of $\mathbf{C}$ needed to obtain the projection matrix $\mathbf{P}$. This inversion can result in a potentially significant performance penalty if $\mathbf{C}$ is a very sizable matrix.

5.2.7 Implementation

The following MATLAB code snippet show how *constrained synchronization* can be adapted to solve the synchronization problem on $SO(3)$. In the code snippet below, M represents the mapping between the replicas in the expanded simple graph and its underlying multigraph.

Listing 5.1: *Constrained synchronization - SO(3)*

```
function R = constrained_synch(Z,A,C,M)
n = size(A,1);
D = kron(diag(sum(A,2)), eye(3));
B = Z - D;
P = eye(size(B,1)) - C * pinv(C);
[Q,O] = eigs(P*(B'*B),rank(C)+3,'smallestabs');

% Filter the smallest rank(C) eigenvalues according to Golub
[¬,ord] = sort(diag(O));
Q = Q(:,ord);
Q = Q(:,rank(C)+1:end);

% Remove ambiguity
i = M{1}(:,1);
Q = Q / (Q(i*3-2:i*3,:));

% Projection onto SO(3)
synch = cell(1,n);
for i = 1:n
    [U,¬,V] = svd(Q(3*i-2:3*i,:));
    synch{i} = U * diag([1,1,det(U*V')]) * V';
end

% Merge labels from replicas and correct for index differences
R = cell(1,size(M,2));
k = 1;
for i = 1:size(M,2)
    if size(M{i},1) > 0
        R{i} = R{M{i}(:,1)};
    else
        R{i} = R{k};
        k = k + 1;
    end
end
end
```

5.3 Experimental validation

We evaluate the performance of *constrained synchronization* and compare it to existing techniques for handling measurement multigraphs. The current state of the art for group synchronization on measurement multigraphs is represented by *edge averaging*. It consists in turning a measurement multigraph into a simple graph by simplifying its multi-edges by averaging their respective edge labels in order to apply spectral formulation introduced in section (3.3.2).

Constrained synchronization is a general technique that is compatible with $GL(d)$ and its subgroups, therefore, for the purposes of this section, the tests will be performed on group SO(3), given its many applications in the field of Computer Vision, including rotation synchronization, structure from motion, 3D point cloud registration and SLAM.

We compare the estimated and ground-truth orientations by considering the optimal isometry that aligns them as the error metric. As a matter of fact, we use the Φ_6 distance function $\Phi_6 : \text{SO(3)} \times \text{SO(3)} \to \mathbb{R}^+$ defined in [13] as follows:

$$\epsilon_i = ||\log(x(i)\tilde{x}(i)^\intercal)||_F. \tag{5.39}$$

Edge averaging in SO(3) is achieved by averaging the labels of the multi-edges using the (non-robust) chordal ℓ_2-mean [22].

All the methods were implemented in MATLAB and were tested on an Intel Core i9 9900k at stock speeds coupled with 16 GB of 3200 MHz RAM.

We ran synthetic experiments on SO(3)-labeled multigraphs. For each run, we built a measurement multigraph $\Gamma = (\mathcal{V}, \mathcal{E}, s, t, z)$ with parameters (n, p, m) where n indicates the number of vertices in $\mathcal{V}$, p indicates the probability that any two vertices are connected by an edge and m indicates the average multiplicity of the multi-edges in Γ. The ground truth rotation matrices $x(i)$ were instantiated by uniformly sampling Euler angles, hence relative rotations were generated as:

$$z_{ij} = x(i)x(j)^\intercal e_{ij} \tag{5.40}$$

with $e_{ij} \in$ SO(3) being a small multiplicative noise. The Euler angles for the perturbation e_{ij} are sampled from a Gaussian distribution with zero mean and a standard deviation σ_R chosen based on the experiment. Each experiment consisted of 100 random runs and the results were averaged to get more accurate results.

We performed the first batch of experiments on SO(3)-labeled multigraphs with $n = 10$ vertices, $p = 0.75$ and a varying average multiplicity

for the multi-edges m. For each run, a multigraph is built by generating n random ground truth rotation matrices and by corrupting relative orientations with a small multiplicative noise. The perturbation is generated by sampling Euler angles from a Gaussian distribution with zero mean and $\sigma_R = \pi/8$. Results of these tests are reported in Figure (5.5) and show that constrained synchronization and edge averaging are tied in terms of accuracy for low multi-edge multiplicity values m, while constrained synchronization delivers consistently better results for larger values of m. If we look at the precision, constrained synchronization delivers significantly better results for all values of m, which gives credit to our hypothesis that constrained synchronization is intrinsically more robust to noise and outliers compared to edge averaging.

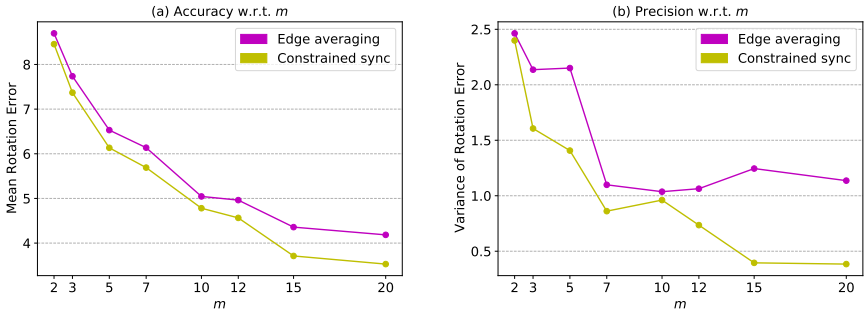

Figure 5.5: *The plots shows the accuracy and the precision of constrained synchronization with respect to the average multiplicity m of the multigraph*

For the second batch of tests, we ran experiments on 12 levels of Gaussian noise σ and measured how the accuracy and precision scale for both techniques. The experiments were run on a SO(3)-labeled multigraph with parameters $n = 10$, $p = 0.75$ and $m = 5$.

In Figure (5.6), results show that both the accuracy and the variance of the rotation error tend to increase less rapidly for constrained synchronization compared to edge averaging with respect to σ. Again, these results give credit to our hypothesis that constrained synchronization is, in general, more robust and returns results that are more reliable compared to those from edge averaging.

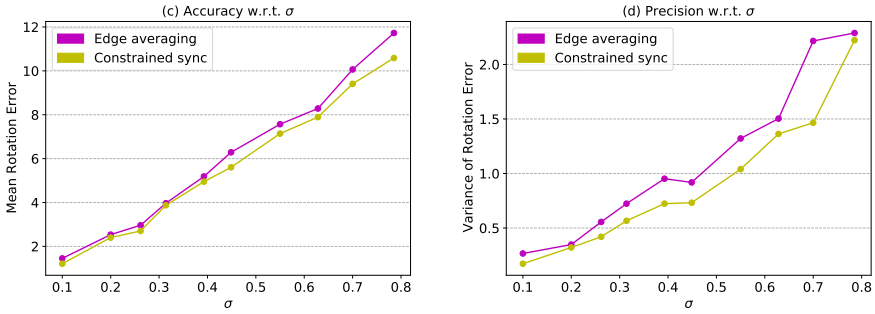

Figure 5.6: *The plots shows the accuracy and the precision of constrained synchronization with respect to the standard deviation σ of the additive Gaussian noise*

CHAPTER 6

Application: Partitioned synchronization

6.1 Introduction

The goal of this chapter is to introduce a novel and efficient approach to group synchronization on large-scale data sets by solving the synchronization problem on subgraphs of the original measurement graph. Partitioning a synchronization problem into multiple sub-problems and combining their solutions is not trivial, as the labeling estimated for each problem is defined up to a global (right) product with any group element. The multigraph formulation introduced in the previous chapters allows us to remove the local ambiguities and lift the multiple labelings for the subgraphs to a single consistent labeling on the original graph. To this end, we build and synchronize a multigraph, from now on called *patch multigraph*, in which a vertex represents the transformation that must be applied to a subgraph in order to fix its group ambiguity.

This novel approach to group synchronization allows us to overcome the most prominent limitations related to existing closed-form solutions when dealing with large-scale data sets. As a matter of fact, traditional synchronization techniques tend to be quite slow, especially robust ones, when measurement graphs are large and dense. Partitioning the problem allows us to

better exploit parallelism and multi-threading in modern computer architectures, achieving a good trade-off between accuracy and execution time. In [3], the authors discuss how, in general, the computational complexity of solving the synchronization problem grows quadratically with respect to the number of vertices in the input graph. Therefore, an added benefit of partitioned synchronization is that it allows us to divide the large data set in chunks and store them on separate processing nodes.

Furthermore, in this chapter we introduce an efficient and effective approach based on spectral clustering that addresses the issue of finding a partition of a measurement graph so that the induced subgraphs are as densely connected as possible to exploit the error compensation properties of synchronization techniques. The main idea is that if the local solutions to the sub-problems are of better quality, then it will be easier to extract more accurate results from the combined solution.

Finally, we include experiments and results that show how our approach performs on real large-scale data sets and how it compares to partitioned synchronization performed by employing current state of the art and competing techniques such as *edge averaging*.

6.2 Reasons for partitioned synchronization

The idea of partitioning a synchronization problem into smaller sub-problems (which are easier to solve) is present in a number of works in the context of structure from motion [7, 14], simultaneous localization and mapping [27] and motion segmentation [4]. Such techniques, however, do not exploit the multigraph formulation – that is introduced in this work for the first time in the literature – but they (either implicitly or explicitly) turn the multigraph into a simple one by averaging multi-edges.

In this section, we summarize the most prominent reasons for partitioned synchronization and why it can be particularly helpful for handling large-scale data sets.

Efficiency

Partitioned synchronization brings several benefits from an efficiency point of view, in terms of both time and memory required to solve the synchronization problem, especially on large data sets. In general, the execution time of synchronization techniques, especially robust ones, grows quadratically with respect to the number of vertices in the graph. Even the spectral and null-space solutions, which are not robust and use efficient sparse

solvers such as *eigs* or *svds*, can see their execution time grow quadratically in the number of vertices if the measurement graph is densely connected.

From a memory complexity point of view, it is worth remembering that, depending on the size of the dataset, the spectral and null-space solutions may require to store two potentially massive matrices, that is the vertex labeling $\mathbf{X}$, which is a $dn \times d$ matrix, and the edge labeling $\mathbf{Z}$, which is a $dn \times dn$ matrix. Having the ability to partition both $\mathbf{X}$ and $\mathbf{Z}$ in multiple sub-matrices and process them independently on separate processing nodes can result in dramatic savings in terms of memory and execution time.

Exploiting multithreading and parallelism

As mentioned earlier, if the synchronization problem can be divided into smaller, independent sub-problems, then it is possible to synchronize the disjoint sub-problems in parallel, fully exploiting the multithreading capabilities of modern computer architectures.

Data safety and privacy

Partitioned synchronization allows for data sets to be segregated so that each processing node can see only a portion of the whole measurement graph (see *e.g.*, [16]) The ability not to expose the whole dataset to all the processing nodes can be important when the safety of data and privacy are of significant importance.

6.3 Outline

In this section we provide a brief overview of the algorithm for partitioned synchronization applied to a Σ-labeled measurement graph $\Gamma = (\mathcal{V}, \mathcal{E}, z)$. The idea behind partitioned synchronization is to decompose the problem of group synchronization on the whole measurement graph Γ into an arbitrary number of smaller synchronization problems on subgraphs of Γ. The solutions to the sub-problems are then combined in order to obtain a vertex labeling for the original graph Γ. Thus, we can say that partitioned synchronization is loosely inspired by the well known divide-and-conquer paradigm for algorithm design.

In this work, we introduce two variants of partitioned synchronization: i) with subgraph augmentation and ii) without subgraph augmentation. Both variants tackle the problem of removing local ambiguities and lifting the labelings obtained for the synchronization sub-problems to a single consistent labeling on the original graph. Each variant has its own strengths and

weaknesses and will be presented as a sequence of steps or a *pipeline* that takes as input a measurement graph, partitions it and estimates its vertex labeling by combining the solutions to the independent sub-problems.

6.3.1 Partitioned synchronization with subgraph augmentation

The first variant we introduce is partitioned synchronization *with subgraph augmentation*. Given a partition of the input measurement graph, subgraph augmentation consists in turning the partition into overlapping subsets that, in turn, induce overlapping subgraphs.

Starting from an input measurement graph Γ, the following steps are performed sequentially in order to estimate the vertex labeling for the input graph:

1. **Graph partitioning**
   The input measurement graph is partitioned into an arbitrary number of disjoint subgraphs.

2. **Subgraph augmentation**
   Each subgraph is augmented in order for it to share vertices with other subgraphs.

3. **Synchronization on subgraphs**
   The synchronization problem is solved on the independent sub-problems.

4. **Synchronization on patch multigraph**
   The vertices shared between subgraphs yield a labeled multigraph that, once synchronized, provides estimates for the ambiguity factors of each sub-problem.

5. **Merge solutions to sub-problems**
   The solutions from the independent sub-problems are combined into a single vertex labeling for the input measurement graph.

In section (6.4), we introduce the problem of graph partitioning applied to a generic measurement graph Γ, while highlighting the problem of handling the ambiguity for each of the resulting sub-problems. In section (6.5), we introduce a novel approach to solve the issues related to the ambiguity by allowing subgraphs to share vertices (subgraph augmentation) that yield a multigraph synchronization problem. Finally, in section (6.6) we combine these results to provide a formal description of the partitioned synchronization with subgraph augmentation algorithm.

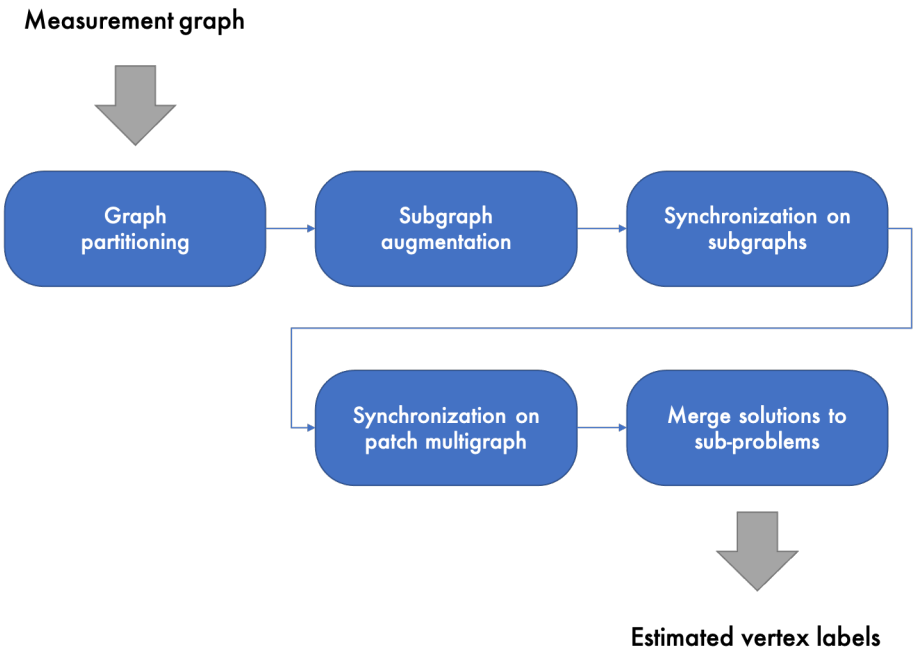

Figure 6.1: *The algorithm for partitioned synchronization with subgraph augmentation illustrated as a pipeline with a measurement graph as the input and the estimated vertex labels as the output.*

6.3.2 Partitioned synchronization w/o subgraph augmentation

The second variant we introduce is partitioned synchronization *without subgraph augmentation*. As opposed to the technique introduced in section (6.3.1), this approach does not require disjoint subgraphs to share vertices to fix the ambiguity in the sub-problems. Instead, the ambiguity problem is solved by synchronizing a multigraph built by considering the cut-edges of the input measurement graph.

Starting from an input measurement graph Γ, the following steps are performed sequentially in order to estimate the vertex labeling for the input graph:

1. **Graph partitioning**
   The input measurement graph is partitioned into an arbitrary number of disjoint subgraphs.

2. **Synchronization on subgraphs**
   The synchronization problem is solved on the independent sub-problems.

3. **Synchronization on patch multigraph**
   The edges cut after partitioning the input graph into disjoint subgraphs yield a labeled multigraph that, once synchronized, provides estimates for the ambiguity factors of each sub-problem.

4. **Merge solutions to sub-problems**
   The solutions from the independent sub-problems are combined into a single vertex labeling for the input measurement graph.

In section (6.7), we provide a formal description of the partitioned synchronization without subgraph augmentation algorithm.

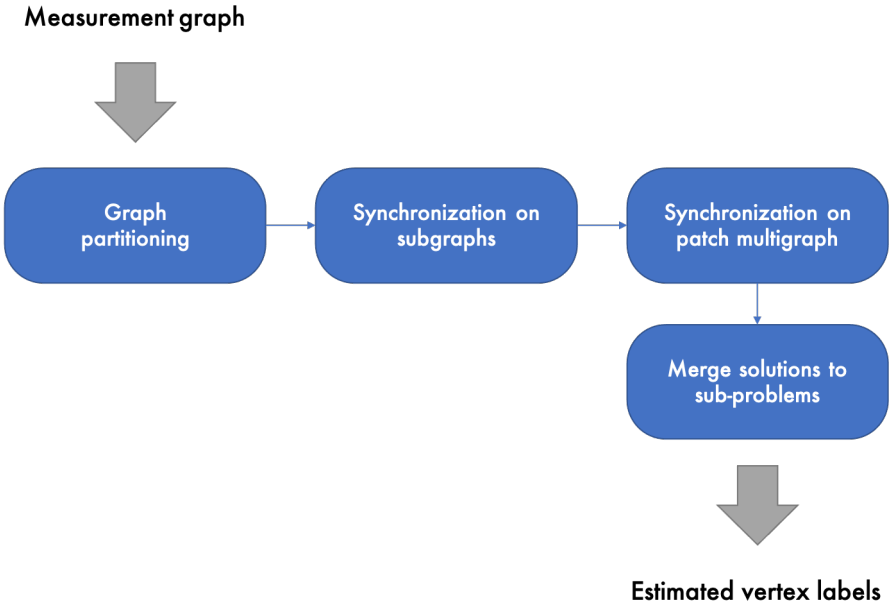

Figure 6.2: *The algorithm for partitioned synchronization without subgraph augmentation illustrated as a pipeline with a measurement graph as the input and the estimated vertex labels as the output.*

6.4 Measurement graph partitioning

In this section we lay the groundwork to partitioned synchronization by presenting our approach to graph partitioning. In particular, we introduce the problem of finding a partition of a measurement graph that induces subgraphs that are as densely connected as possible. The reason behind this requirement is that closed-form synchronization techniques work best when applied to graphs that contain many edges: in section (3.2), we discussed how cycles in graphs can be employed in order to compensate for errors in the pairwise measurements. Therefore, it is reasonable to assume that subgraphs of densely connected vertices should provide better performance when applying synchronization on them.

In order to achieve this goal, we employ the *spectral clustering* algorithm, which provides an efficient solution to the problem of identifying densely connected clusters in a graph.

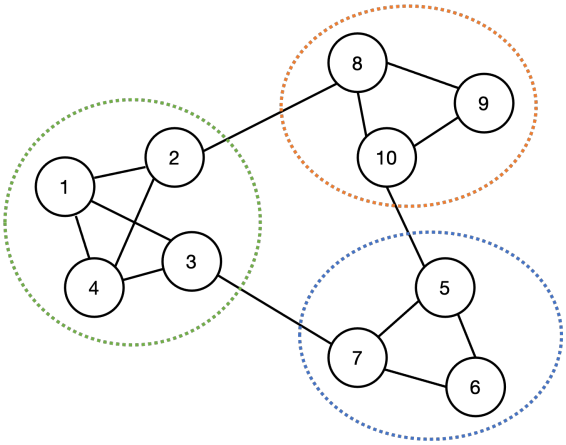

Figure 6.3: *A graph partitioned into 3 densely connected subgraphs.*

6.4.1 Spectral clustering for graph partitioning

The problem of finding densely connected subgraphs has a rich literature in the field of graph theory and can be solved by applying a variety of different techniques [25], [18]. In this work, we chose the spectral clustering algorithm for finding the densest subgraphs of a measurement graph. More specifically, we chose the normalization of [31], also referred to as *normalized spectral clustering according to Shi and Malik*, because it tends to balance the number of edges among clusters, which is the relevant parameter for the computational complexity of the sparse methods we are using

for synchronization.

In [40], the author provides an introduction to spectral clustering with a particular focus on the criteria behind how the algorithm groups elements together. Starting from the definition of similarity s_{ij} between two elements i and j, consider a graph of elements such that edges are assigned weights that are equal to the similarity of the elements they connect. Spectral clustering finds a partition of elements so that the edges that link elements belonging to separate clusters have very low weights and the edges within the same group have high weights. Therefore, if we can successfully define a similarity metric for vertices in a measurement graph, then spectral clustering will output a partition able to satisfy the requirements established so far.

Let us consider a Σ-labeled graph $\Gamma = (\mathcal{V}, \mathcal{E}, z)$. In order to apply the spectral clustering algorithm, the first step is to define how the similarity between vertices in $\mathcal{V}$ is computed. In the context of measurement graphs, the similarity between two vertices i and j depends exclusively on whether there is a labeled edge connecting such vertices, that is if $(i, j) \in \mathcal{E}$. Therefore, s_{ij} can assume only two values: $s_{ij} = 0$ when vertices i and j are not connected by an edge and $s_{ij} = 1$ otherwise. In other words, two vertices share any similarity if and only if there is an edge connecting them in $\mathcal{E}$. From the definition of similarity between vertices s_{ij}, let us define the similarity matrix $\mathbf{S}_\Gamma$ for graph Γ:

$$\mathbf{S}_\Gamma = [s_{ij}] = \begin{cases} 1 & \text{if } (i, j) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

Having defined a metric to establish the similarity between vertices in a Σ-labeled graph, the problem is to partition Γ so that edges connecting vertices in different clusters have small similarity weights, while edges connecting vertices in the same cluster have high similarity values. This problem is equivalent to the one solved by the spectral clustering algorithm.

Let us consider the similarity matrix $\mathbf{S}_\Gamma$. It is straightforward to see that such matrix is equivalent to the adjacency matrix $\mathbf{A}$ defined for the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ obtained from Γ. Let us compute the degree matrix $\mathbf{D}$ from the adjacency matrix $\mathbf{A}$ or, equivalently, $\mathbf{S}_\Gamma$:

$$\mathbf{D} = \text{diag}(\mathbf{A}\mathbf{1}_{n\times 1}) \tag{6.2}$$

where $\mathbf{1}_{n\times 1}$ denotes a $n \times 1$ vector filled with ones. From the degree matrix $\mathbf{D}$, it is possible to compute the unnormalized graph Laplacian as follows:

$$\mathbf{L} = \mathbf{D} - \mathbf{A} \tag{6.3}$$

From the normalized graph Laplacian, we can compute the first k generalized eigenvectors $u_1, ..., u_k$ of the generalized eigenproblem $\mathbf{L}u = \lambda \mathbf{D}u$, where k is the number of clusters we wish to find in the graph. Let $\mathbf{U} \in \mathbb{R}^{n \times k}$ be the matrix containing $u_1, \ldots, u_k$ as columns. We apply the k-means algorithm by considering each row of $\mathbf{U}$ as an independent data point. The result is a set of k clusters that forms a partition of the original pairwise measurement graph so that each cluster induces a densely connected subgraph.

6.4.2 Implementation

The following MATLAB code snippet implements the *Normalized Spectral Clustering according to Shi and Malik* [31] for a labeled graph of pairwise measurements. The algorithm requires two inputs: the adjacency matrix for the graph and the desired number of clusters k.

Listing 6.1: *MATLAB implementation of Normalized spectral clustering according to Ng, Jordan, and Weiss*

```
function groups = spectral_shi_malik(A,k)

% Maximum number of iterations for k-means
maxiter = 1000;

% Normalized spectral clustering according to Shi and Malik (2000)
% using Unnormalized Laplacian L = D - W
D = diag(1 ./ sqrt(sum(A) + eps));
L = D - A;
[U,¬] = eigs(L,D,k,'smallestabs');

groups = kmeans(U,k,'maxiter',maxiter,'Start',U(1:k,:), ...
                'EmptyAction','singleton','Distance','cityblock');

end
```

6.4.3 Ambiguity from graph partitioning

The main challenge that we need to overcome when considering graph partitioning is related to how the solutions of the sub-problems are combined in order to estimate a consistent vertex labeling for the original graph. In this section, we provide a detailed explanation of these inherent problems and discuss a possible solution to overcome these limitations.

So far, we applied spectral clustering in order to partition the vertex set $\mathcal{V}$ of a measurement graph into separate subsets or clusters so that every vertex $v \in \mathcal{V}$ belongs to a single cluster. As mentioned in section (3.2), a consistent labeling in a Σ-labeled graph is defined up to a global (right)

product with any element of group Σ. This means that, for instance, the solution to the synchronization problem in the general case of $GL(d)$ is defined up to a right multiplication by an invertible $d \times d$ matrix.

Let us provide an explanation for the ambiguity in the solution of group synchronization by considering the closed-form solutions introduced in section (3.2) and, in particular, the spectral solution applied to $GL(d)$. The vertex labels for a $GL(d)$-labeled graph are recovered by computing the eigenvectors that correspond to the d leading eigenvalues of matrix $(\mathbf{D} \otimes \mathbf{I}_d)^{-1}\mathbf{Z}_\mathbf{A}$. The d eigenvectors span a linear subspace, therefore, any basis for such space is a solution. A change of the basis in the eigenspace can be seen as a right multiplication by an element in $GL(d)$ that is also invertible. A consequence of such property is that, if the synchronization problem is partitioned into multiple sub-problems, then each solution for a sub-problem is defined up to a $d \times d$ invertible matrix that is unique for such sub-problem. In other words, if we were to solve the synchronization problem on k independent partitions of the measurement graph, then there would be k different elements upon which the solutions would be defined.

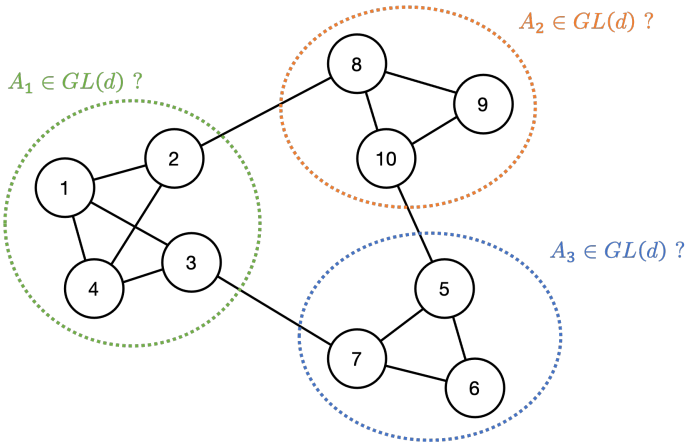

Figure 6.4: *The figure shows how this simple measurement graph is partitioned into 3 subgraphs and how the vertex labels recovered for each sub-problem are defined up to 3 different elements A_1, A_2, A_3 of $GL(d)$.*

Observation 1. The presence of multiple factors on which the individual solutions depend upon makes it impossible to merge the multiple label assignments and to recover a common, consistent vertex labeling for all the vertices in the complete measurement graph.

Proof. Let us consider consider a noiseless Σ-labeled graph $\Gamma = (\mathcal{V}, \mathcal{E}, z)$ with $i, j \in \mathcal{V}$ and $(i, j) \in \mathcal{E}$. Consider the vertex labels estimated for

vertices i and j by solving the synchronization problem on the entire graph Γ, respectively $\mathbf{X}_i$ and $\mathbf{X}_j$. It is possible to write the consistency constraint between i and j as follows:

$$\mathbf{Z}_{ij} = \mathbf{X}_i \mathbf{X}_j^{-1} \tag{6.4}$$

Let us partition Γ into two sub-graphs $\Gamma_1 = (\mathcal{V}_1, \mathcal{E}_1, z_1)$ and $\Gamma_2 = (\mathcal{V}_2, \mathcal{E}_2, z_2)$ so that $\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$, $i \in \mathcal{V}_1$ and $j \in \mathcal{V}_2$. Let us solve the synchronization problem independently for the two sub-graphs and recover labels $\tilde{\mathbf{X}}_i$ and $\tilde{\mathbf{X}}_j$ for i and j respectively. In general, $\mathbf{X}_i \neq \tilde{\mathbf{X}}_i$ and $\mathbf{X}_j \neq \tilde{\mathbf{X}}_j$, given that the solutions found for different graphs (or even sub-graphs of the same graph) are defined up to a global (right) product with an element in Σ.

Let us compute the ratio between the vertex labels recovered from the original measurement graph and the partitioned graphs:

$$\begin{aligned} \mathbf{M}_1 &= \mathbf{X}_i^{-1} \tilde{\mathbf{X}}_i \\ \mathbf{M}_2 &= \mathbf{X}_j^{-1} \tilde{\mathbf{X}}_j \end{aligned} \tag{6.5}$$

Let us write the consistency constraint $\mathbf{Z}_{ij}$ involving the labels of vertices i and j recovered after partitioning the graph and plug in the results of Equation (6.5):

$$\begin{aligned} \tilde{\mathbf{X}}_i &= \mathbf{Z}_{ij} \tilde{\mathbf{X}}_j \\ (\mathbf{X}_i \mathbf{M}_1) &= \mathbf{Z}_{ij} (\mathbf{X}_j \mathbf{M}_2) \\ (\mathbf{X}_i \mathbf{M}_1) &= \mathbf{X}_i \mathbf{X}_j^{-1} (\mathbf{X}_j \mathbf{M}_2) \\ \mathbf{X}_i \mathbf{M}_1 &= \mathbf{X}_i \mathbf{M}_2 \end{aligned} \tag{6.6}$$

Therefore, the vertex labeling for graph Γ obtained by merging the solutions found for the two sub-graphs Γ_1 and Γ_2 satisfies the consistency constraint, i.e. it is consistent, if and only if:

$$\mathbf{M}_1 = \mathbf{M}_2 \tag{6.7}$$

□

Let us make an observation to illustrate how the ambiguity problem can be solved if $\mathbf{M}_1$ and $\mathbf{M}_2$ are known in the previous example. Without any loss of generality, assume that the solution to the synchronization problem for the original measurement graph is defined up to the unit element $\mathbf{I}_d$. If we apply the transformations $\mathbf{M}_1^{-1}$ and $\mathbf{M}_2^{-1}$ to the solutions for Γ_1 and Γ_2 respectively, the result is a change of basis for the eigenspaces of both

solutions. The labels will be defined up to the same element $\mathbf{I}_d$, given that:

$$\begin{aligned} \mathbf{X}_i\mathbf{M}_1\mathbf{M}_1^{-1} &= \mathbf{X}_i\mathbf{I}_d \\ \mathbf{X}_j\mathbf{M}_2\mathbf{M}_2^{-1} &= \mathbf{X}_j\mathbf{I}_d \end{aligned} \tag{6.8}$$

Therefore, the solutions to the sub-problems will be defined up to a (right) product multiplication by $\mathbf{I}_d$, which is also true for the solution to the original measurement graph. In other words, after this change of basis, the solutions for Γ, Γ_1 and Γ_2 will be defined up to the same element of $GL(d)$. Thus, the ambiguity problem is resolved and the solutions can be merged.

Of course, this procedure for merging the labeling for vertices belonging to different sub-graphs requires a solution for the entire measurement graph and knowledge of elements $\mathbf{M}_i$, thus defying the whole purpose of partitioned synchronization. In the next sections we will discuss how the ambiguity problem can be solved either with or without sharing vertices between disjoint subgraphs.

6.5 Subgraph augmentation by vertex sharing

In this section we introduce a novel approach to the solution of the ambiguity problem introduced in section (6.4.3) by formulating it as a group-synchronization problem on a labeled multigraph. More specifically, the issue of ambiguity is solved by allowing subgraphs to have vertices in common. We call this process *subgraph augmentation*.

As described later, shared vertices between subgraphs yield a multigraph synchronization problem that, when solved, allows us to reconstruct the elements of group Σ upon which the solutions to the sub-problems are defined.

6.5.1 Single vertex sharing

Let us introduce subgraph augmentation by considering a simple Σ-labeled graph $\Gamma = (\mathcal{V}, \mathcal{E}, z)$ and a partition $\mathbf{P}$ dividing the graph into two subgraphs or clusters $\Gamma_1 = (\mathcal{V}_1, \mathcal{E}_1, z_1)$ and $\Gamma_2 = (\mathcal{V}_2, \mathcal{E}_2, z_2)$ with $\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$.

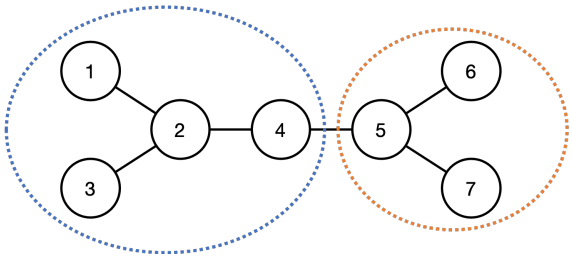

Figure 6.5: *The simple example graph we will refer to in this section.*

As discussed in section (6.4.3), if the synchronization problem is solved independently on the two sub-graphs Γ_1 and Γ_2, then the solution for Γ_1 will be defined up to an element $\mathbf{M}_1 \in \Sigma$, while the solution for Γ_2 will be defined up to $\mathbf{M}_2 \in \Sigma$. In general, $\mathbf{M}_1 \neq \mathbf{M}_2$ and, therefore, in order to merge the solutions, the next step is to apply a transformation to the vertex labels recovered from either of the two clusters in order to make sure that the ambiguity elements $\mathbf{M}_1$ and $\mathbf{M}_2$ for Γ_1 and Γ_2 respectively will converge to the same value. Intuitively, if $\mathbf{M}_1 = \mathbf{M}_2$, then merging the solutions for the two independent sub-problems is straightforward because it is as if the labels estimated from the two separate sub-problems were the result of a single, whole synchronization problem.

Let us consider any vertex $v \in \mathcal{V}$ that is connected to two vertices v_1 and v_2 such that $v \neq v_1 \neq v_2$ and $v_1 \in \mathcal{V}_1$ and $v_2 \in \mathcal{V}_2$.

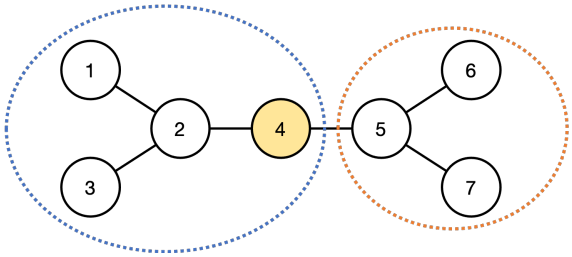

Figure 6.6: *A simple graph partitioned into two subgraphs (blue and orange). In this case, we select $v = 4$, given that it is connected to both $v_1 = 2$ and $v_2 = 5$ which belong to the blue and orange clusters respectively.*

In order to solve the ambiguity problem for the partitioned graph, let us take vertex v and assign it to both clusters before extracting the two sub-graphs Γ_1 and Γ_2.

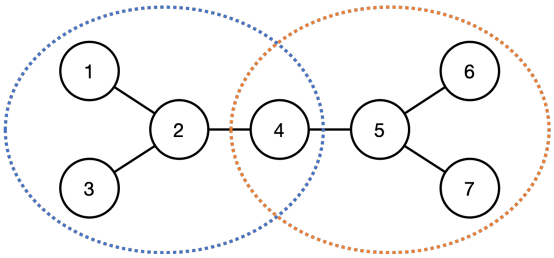

Figure 6.7: *The updated clusters and sub-graphs after assigning $v = 4$ to both the blue and orange clusters.*

Consider the two labels assigned to vertex v from the two synchronization problems, namely $\mathbf{X}_{v,1}$ and $\mathbf{X}_{v,2}$. The transformation that changes the basis for the eigenspace of $\mathbf{X}_{v,1}$ into that of $\mathbf{X}_{v,2}$ is defined as the ratio between the two labels:

$$\mathbf{M}_{12} = (\mathbf{X}_{v,1})^{-1}\mathbf{X}_{v,2} \tag{6.9}$$

given that $\mathbf{X}_{v,1}\mathbf{M}_{12} = \mathbf{X}_{v,1}\mathbf{X}_{v,1}^{-1}\mathbf{X}_{v,2} = \mathbf{X}_{v,2}$.

If the transformation $\mathbf{M}_{12}$ is applied to all labels in the solution for the synchronization problem on Γ_1, then it will result in a change of basis for the eigenspace such that the solutions for both Γ_1 and Γ_2 are defined up to a single element in Σ, thus merging the two solutions found for the independent sub-graphs.

For noiseless measurement graphs, the result is that both vertex labels found for vertex v will converge to the same exact value. In the case of noisy measurement graphs, though, the estimate of the transformation between the basis of the eigenspaces for Γ_1 and Γ_2 will not be perfectly accurate. This estimate is provided by the labels for v recovered from the two noisy measurement sub-graphs. The presence of noise introduces errors in the reconstruction of both the labels and the transformation between the basis for the two eigenspaces. Therefore, merging the solutions for the two sub-problems can introduce errors determined by two factors:

- Errors due to noise in the independent sub-graphs for the recovered vertex labels
- Errors in the reconstruction of the factors upon which the solutions are defined due to the noisy vertex labels on which it is based on

As we shall discuss in later sections, a solution to this problem is to gather more samples in terms of vertex labels to estimate the factors upon which the solutions for the individual sub-graphs are defined.

6.5.2 Handling multiple subgraphs

So far, we considered the case in which the measurement graph is divided into two subgraphs. In the general case of multiple clusters, every pair of subgraphs Γ_i and Γ_j that share a vertex allows us to compute a transformation of type $\mathbf{M}_{ij}$, as described in section (6.5.1). If we consider every ambiguity factor $\mathbf{M}_i$ in the solution for a sub-problem as an unknwon state (see section (6.4.3)) and the transformations of type $\mathbf{M}_{ij}$ as pairwise measurements between such states, it is possible to build a new measurement graph we call *patch graph*.

Definition 6.5.1. ***(Patch graph – for subgraph augmentation)*** *Let us consider a Σ-labeled measurement graph $\Gamma = (\mathcal{V}, \mathcal{E}, z)$ and a set $\mathcal{S} = \{\Gamma_1, \ldots, \Gamma_k\}$ for which every element Γ_i is a sub-graph of Γ.*
The patch graph for Γ and its sub-graph set $\mathcal{S}$ is defined as a Σ-labeled graph $\Phi = (\mathcal{V}_\Phi, \mathcal{E}_\Phi, z_\Phi)$, where $\mathcal{V}_\Phi$ contains a vertex for each $\Gamma_i \in \mathcal{S}$ and $(i, j) \in \mathcal{E}_\Phi$ if and only if $\exists v \in \mathcal{V}$ such that $\exists \Gamma_i \in \mathcal{S} \wedge \exists \Gamma_j \in \mathcal{S}$ such that $v \in \mathcal{V}_i \wedge v \in \mathcal{V}_j$, with $\mathcal{V}_i$ and $\mathcal{V}_j$ the vertex sets for Γ_i and Γ_j respectively. In addition,

$$z : \mathcal{E} \rightarrow \Sigma \tag{6.10}$$

and is defined as follows:

$$z(i, j) = \mathbf{X}_{v,i}^{-1} \mathbf{X}_{v,j} \tag{6.11}$$

where v is the vertex shared by Γ_i and Γ_j, $\mathbf{X}_{v,i}$ and $\mathbf{X}_{v,j}$ are the vertex labels for v estimated for sub-graphs Γ_i and Γ_j respectively.

In other words, a patch graph for a measurement graph Γ contains a number of vertices that is equal to the number of subgraphs identified in it, while every edge (i, j) corresponds to the transformation that needs to be applied to the solution for subgraph Γ_i for the solutions to Γ_i and Γ_j to be defined up to the same element of group Σ.

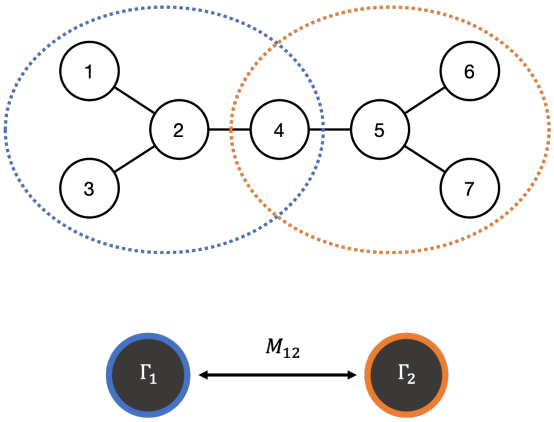

Figure 6.8: *An example of how a patch graph is built starting from a measurement graph and two sub-graphs having a vertex in common.*

The introduction of the patch graph for partitioned synchronization allows us to exploit the properties of measurement graphs and group synchronization introduced in section (3.2) to get more accurate and robust results when merging the solutions for the sub-problems. For starters, it is possible to solve the synchronization problem for the patch graph and reconstruct its global states. The global states for the patch graph represent the elements upon which the solutions for the corresponding sub-graphs are defined. In other words, if we are able to solve the synchronization problem for a patch graph,we can recover the various ambiguity factors and merge the vertex labels from different sub-problem in one sweep. In addition, formulating the problem of merging solutions as a synchronization problem on a measurement graph, allows us to exploit all of the advantages related to graphs, including cycle consistency and the ability to consider multiple pairwise information sources at once. The graph formulation implies that the propagation of the error is mitigated, hence providing better results compared to a more standard spanning tree or cascaded approach to label merging.

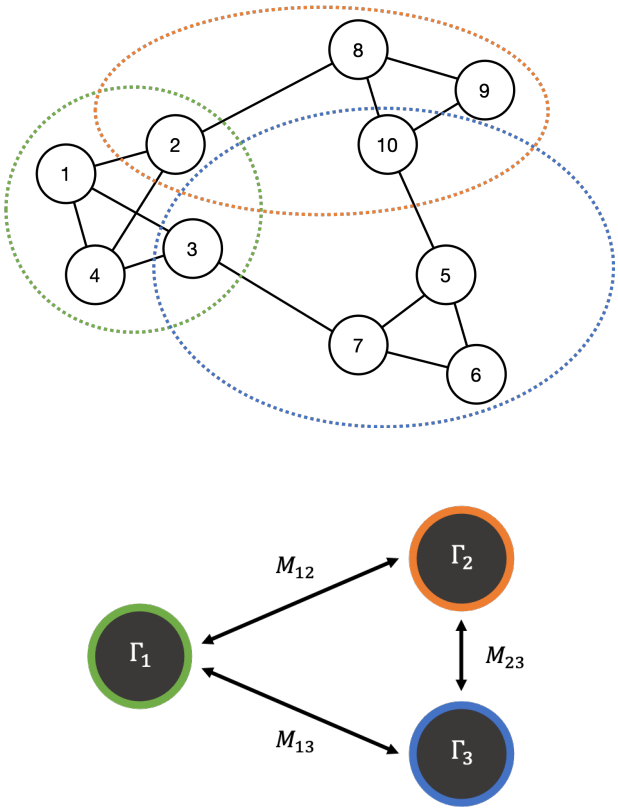

Figure 6.9: *The figure above shows a measurement graph partitioned into* 3 *subgraphs. Each pair of subgraphs shares a vertex. The figure below shows the patch graph resulting from the subgraphs highlighted in the figure above. Notice how the patch graph is complete because every pair of subgraphs has a vertex in common.*

6.5.3 Multiple vertex sharing

In this section, we introduce a novel approach to the problem of merging the solutions of multiple synchronization problems on subgraphs of a single measurement graph by solving a synchronization on a properly defined *patch multigraph*.

In section (6.5.1), we discussed how the quality of the solution for the synchronization problem on the original measurement graph depends not only on the quality of the solutions to the sub-problems, but also on the merging phase of the algorithm. The techniques introduced in sections (6.5.1) and (6.5.2) can be augmented by allowing pairs of subgraphs to share multiple vertices instead of a single one. The main advantage of multiple vertex sharing between subgraphs is the ability to sample more trans-

formations of type $\mathbf{M}_{ij}$ between any pair of subgraphs Γ_i and Γ_j, thus obtaining more pairwise measurements that can be added to the patch graph.

Let us discuss the consequences of multiple vertex sharing on the definition of the patch graph in (6.5.1) by considering a simple Σ-labeled graph $\Gamma = (\mathcal{V}, \mathcal{E}, z)$ and a partition of $\mathcal{V}$ dividing the graph into two subgraphs (or clusters) $\Gamma_1 = (\mathcal{V}_1, \mathcal{E}_1, z_1)$ and $\Gamma_2 = (\mathcal{V}_2, \mathcal{E}_2, z_2)$ with $\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$. Let us consider a set of vertices $\mathcal{S} = \{v_1, v_2, \dots, v_k\}$ with $\mathcal{S} \subseteq \mathcal{V}$ such that $\forall v \in \mathcal{S}$, v is connected to at least two vertices v_a and v_b such that $v_a \in \mathcal{V}_1$ and $v_b \in \mathcal{V}_2$.

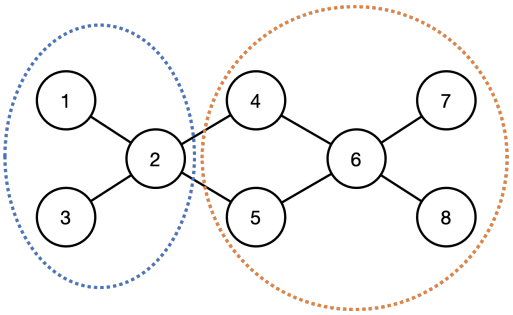

Figure 6.10: *The simple example graph we will refer to in this section. Notice how vertices* 4 *and* 5 *are connected to vertices both in the blue and orange clusters.*

If the synchronization problem is solved independently on the two subgraphs, then the solutions to the sub-problems will be defined up to a (right) product by two different ambiguity elements in Σ, that is $\mathbf{M}_1$ and $\mathbf{M}_2$ for Γ_1 and Γ_2 respectively.

Let us follow the approach introduced in section (6.5.1) and assign some or all of the vertices in $\mathcal{S}$ to both clusters before extracting the two subgraphs Γ_1 and Γ_2. After solving the synchronization problem on Γ_1 and Γ_2 independently, consider the vertex labels recovered for the elements in $\mathcal{S}$. For every $v_i \in \mathcal{S}$, the synchronization sub-problems return two vertex labels of type $\mathbf{X}_{v_i,1}$ and $\mathbf{X}_{v_i,2}$. This means that, for every $v_i \in \mathcal{S}$, we are able to write a transformation of type $\mathbf{M}_{12} = \mathbf{X}_{v_i,1}^{-1}\mathbf{X}_{v_i,2}$ that can be applied to change the basis for the eigenspace associated to the solution for Γ_1, thus merging the two solutions for the independent subgraphs.

A clear implication of this approach is that if we were to build the patch graph for the measurement graph Γ and its subgraphs, the result would not be a simple Σ-labeled graph, but instead a Σ-labeled multigraph. As a matter of fact, the number of transformations available between any two subgraphs (or global states) is no longer limited to 1. Instead, we can estimate up to k different pairwise measurements, where k is the number of

vertices in $\mathcal{S}$ shared between the two subgraphs.

Definition 6.5.2. ***(Patch multigraph – for subgraph augmentation)*** *Let us consider a Σ-labeled measurement graph $\Gamma = (\mathcal{V}, \mathcal{E}, z)$ and a set $\mathcal{S} = \{\Gamma_1, \dots, \Gamma_k\}$ for which every element Γ_i is a sub-graph of Γ.*
The patch multigraph for Γ and its subgraph set $\mathcal{S}$ is defined as a Σ-labeled multigraph $\Phi = (\mathcal{V}_\Phi, \mathcal{E}_\Phi, s_\Phi, t_\Phi, z_\Phi)$, where $\mathcal{V}_\Phi$ contains a vertex for each $\Gamma_i \in \mathcal{S}$ and $\mathcal{E}_\Phi$ contains, for every pair of subgraphs Γ_i and Γ_j, a multi-edge from i to j of multiplicity $m_{ij} = |\mathcal{V}_i \cap \mathcal{V}_j|$, with $\mathcal{V}_i$ and $\mathcal{V}_j$ the vertex sets for Γ_i and Γ_j respectively. In addition,

$$z : \mathcal{E} \to \Sigma \tag{6.12}$$

and is defined as follows:

$$z(e) = \mathbf{X}_{v,i}^{-1} \mathbf{X}_{v,j} \tag{6.13}$$

where $v \in \mathcal{V}_i \cap \mathcal{V}_j$, $s_\Phi(e) = i$, $t_\Phi(e) = j$ and $\mathbf{X}_{v,i}$ and $\mathbf{X}_{v,j}$ are the vertex labels for v estimated for subgraphs Γ_i and Γ_j respectively. s_Φ, t_Φ are the source and target function respectively, as in the definition of the generic group-labeled multigraph.

The fact that, in the general case, the patch graph needs to be upgraded to a multigraph to allow for multiple vertex sharing does not pose a significant problem. As a matter of fact, the *constrained synchronization* technique introduced in section (5.2) allows to efficiently solve the synchronization problem for the patch multigraph independently from the group Σ on which synchronization is performed.

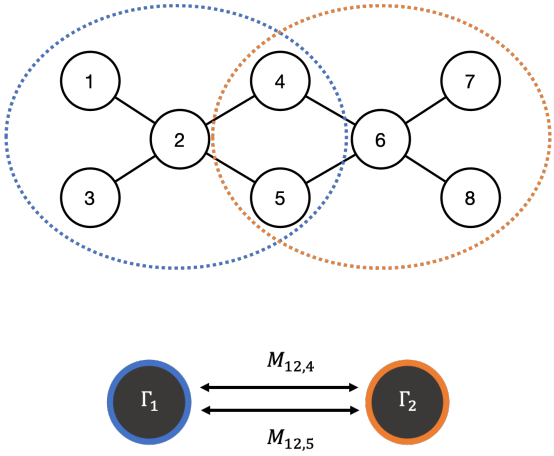

Figure 6.11: *An example of how a patch multigraph is built starting from a measurement graph and two subgraphs having multiple vertices in common.*

Advantages

Multiple vertex sharing between subgraphs brings several advantages in the merging phase of the partitioned synchronization algorithm. The multigraph formulation is more robust to noise and outliers, given that multiedges can encode more information. In addition, the results obtained for the constrained synchronization technique reported in section (5.3) translate to this case, providing concrete proof of the increased robustness of our solution merging technique.

An important point to consider is that the improvements to accuracy related to multiple vertex sharing do not come just from the fact that the patch graph is actually a multigraph, but also from the fact that the subgraphs obtained from the partition of the measurement graph need to be augmented or enlarged for vertices to be shared across clusters. The increased size of the subgraphs means that more pairwise measurements are available for each sub-problem, producing more accurate sub-solutions.

Drawbacks

Although multiple vertex sharing tends to provide more robust results compared to single vertex sharing, it does come at a potentially significant computational cost. Depending on the number of vertices shared between clusters, the size of the subgraphs may become significant enough to offset the increased efficiency provided by partitioned synchronization.

6.6 Partitioned synchronization with subgraph augmentation

In sections (6.4) and (6.5) we discussed about the inherent ambiguity in the solutions to the synchronization sub-problems introduced after partitioning the graph and provided a method for reconstructing those ambiguity factors by solving the synchronization on a patch multigraph. In this section, we piece together the results obtained so far to provide a formal description of the various steps that make up the pipeline for the partitioned synchronization algorithm with subgraph augmentation.

Let us consider a measurement graph $\Gamma = (\mathcal{V}, \mathcal{E}, z)$. In order to solve the partitioned synchronization problem on Γ, let us apply the following steps sequentially.

6.6.1 Graph partitioning

Find a partition of k disjoint subsets of the vertex set $\mathcal{V}$ for Γ by applying the spectral clustering technique for measurement graphs introduced in section (6.4). The result is a set $\mathcal{W}$ containing subsets of $\mathcal{V}$, where $\mathcal{W} = \{\mathcal{V}_1, \dots, \mathcal{V}_k\}$ satisfies the condition that $\forall \mathcal{V}_i \in \mathcal{W} : \mathcal{V}_i \subseteq \mathcal{V}$.

6.6.2 Subgraph augmentation

For every pair of subsets $\mathcal{V}_i, \mathcal{V}_j \in \mathcal{W}$, the set of vertices that can be shared between the induced subgraphs $\Gamma_i = (\mathcal{V}_i, \mathcal{E}_i, z_i)$ and $\Gamma_j = (\mathcal{V}_j, \mathcal{E}_j, z_j)$ is defined as follows:

$$\mathcal{S}_{ij} = \{v \in \mathcal{V}_i \cup \mathcal{V}_j \mid v \in \mathcal{V}_i \wedge v \in \mathcal{V}_j\} \tag{6.14}$$

For every subset $\mathcal{V}_i \in \mathcal{W}$, let us initialize $\mathcal{V}'_i = \mathcal{V}_i$ and add it to the set $\mathcal{W}'$. For every pair of elements $\mathcal{V}_i, \mathcal{V}_j \in \mathcal{W}$, let us choose a subset $\mathcal{V}_{ij} \subseteq \mathcal{S}_{ij}$ and update the elements $\mathcal{V}'_i, \mathcal{V}'_j \in \mathcal{W}'$ as follows:

$$\begin{aligned} \mathcal{V}'_i &= \mathcal{V}'_i \cup \mathcal{V}_{ij} \\ \mathcal{V}'_j &= \mathcal{V}'_j \cup \mathcal{V}_{ij} \end{aligned} \tag{6.15}$$

thus adding shared vertices to both clusters. The result is that the sets $\mathcal{V}'_u \in \mathcal{W}'$ are not disjoint anymore.

In practice, the number of vertices shared between subgraphs is an hyperparameter that can be tuned based on the accuracy required from the reconstruction. In general, a larger number of vertices shared between clusters produces better results at the cost of increased execution time.

6.6.3 Synchronization on subgraphs

Let us build the measurement subgraphs and solve the independent synchronization sub-problems. From the elements of $\mathcal{W}'$, which are still subsets of $\mathcal{V}$ for Γ, let us build k measurement sub-graphs $\Gamma_1, \ldots, \Gamma_k$. Every sub-graph Γ_i is defined as $\Gamma_i = (\mathcal{V}_i', \mathcal{E}_i, z_i)$, with $\mathcal{V}_i' \in \mathcal{W}'$ and $\mathcal{E}_i$ defined as follows:

$$\mathcal{E}_i = \{(u, v) \in \mathcal{E} \mid u \in \mathcal{V}_i' \wedge v \in \mathcal{V}_i'\} \tag{6.16}$$

Let us solve the synchronization problem for every measurement subgraph Γ_i and obtain k matrices of type $\mathbf{X}_i$, each containing the vertex labels estimated for the set of vertices $\mathcal{V}_i'$:

$$\mathbf{X}_i = \begin{pmatrix} \mathbf{X}_{i,1} \\ \vdots \\ \mathbf{X}_{i,|\mathcal{V}_i'|} \end{pmatrix} \tag{6.17}$$

As discussed in section (6.4.3), each solution will be defined up to an element $\mathbf{M}_i$ that is unique for each subgraph Γ_i.

6.6.4 Synchronization on patch multigraph

Let us build the patch multigraph Φ for Γ and its subgraphs $\Gamma_1, \ldots, \Gamma_k$. The patch multigraph is a tuple $\Phi = (\mathcal{V}_\Phi, \mathcal{E}_\Phi, s_\Phi, t_\Phi, z_\Phi)$, in which $\mathcal{V}_\Phi$ contains k elements (or global states), $\mathcal{E}_\Phi$ is a set such that $e \in \mathcal{E}_\Phi$ and $(s_\Phi(e), t_\Phi(e)) = (i, j)$ if and only if $\exists v \in \mathcal{V}$ such that $v \in \mathcal{V}_i' \wedge v \in \mathcal{V}_j'$. The multiplicity m_{ij} of the multi-edge connecting vertices (i, j) is equal to the number of vertices in common between subgraphs Γ_i and Γ_j, that is:

$$m_{ij} = |\mathcal{V}_i' \cap \mathcal{V}_j'| \tag{6.18}$$

z_Φ is a function of type:

$$z_\Phi : \mathcal{E}_\Phi \to \Sigma \tag{6.19}$$

and is defined as follows:

$$z_\Phi(e) = \mathbf{X}_{v_h,i}^{-1} \mathbf{X}_{v_h,j} \tag{6.20}$$

with $e \in \mathcal{E}_\Phi$ such that $(s_\Phi(e), t_\Phi(e)) = (i, j)$ and $v_h \in (\mathcal{V}_i' \cap \mathcal{V}_j')$. In other words, there is an edge in $\mathcal{E}_\Phi$ connecting two global states for every vertex shared between the subgraphs represented by such states, each with its own edge label deriving from the estimate of the transformation between their ambiguity factors.

Let us solve the group synchronization problem for the patch multi-graph Φ by applying the *constrained synchronization* technique introduced in (5.2.4). We obtain a matrix $\mathbf{X}_\Phi$ containing the vertex labels that represent the solution to the synchronization problem for Φ:

$$\mathbf{X}_\Phi = \begin{pmatrix} \mathbf{X}_{\Phi,1} \\ \vdots \\ \mathbf{X}_{\Phi,k} \end{pmatrix} \tag{6.21}$$

6.6.5 Solution merging

The last step is to make sure that the solutions to the k sub-problems are defined up to a global (right) product with the same group element. In order to achieve this, for every matrix $\mathbf{X}_{\Phi,i}$ in $\mathbf{X}_\Phi$, let us apply the following transformation to the vertex labels recovered from the solution to the i-th subgraph $\mathbf{X}_i$:

$$\mathbf{X}_i = \begin{pmatrix} \mathbf{X}_{i,1}\mathbf{X}_{\Phi,i}^{-1} \\ \vdots \\ \mathbf{X}_{i,|\mathcal{V}_i'|}\mathbf{X}_{\Phi,i}^{-1} \end{pmatrix} \tag{6.22}$$

After all transformations are applied and the problem of ambiguity is solved, we need to build the matrix $\mathbf{X}$ containing the estimated vertex labels for the input measurement graph Γ. As usual, $\mathbf{X}$ is a matrix of matrices $\mathbf{X}_i$, each representing the estimated label for vertex $v \in \mathcal{V}$. These estimates can be recovered from the solutions to the sub-problems, given that they are now defined up to the same group element.

In the case in which more estimates for the label of a vertex $v \in \mathcal{V}$ are available, we can choose to pick the one which results in the lowest consistency error or combine them if an aggregation function can be defined for group Σ.

6.6.6 Advantages and disadvantages

In our experiments, which are reported in section (6.8), partitioned synchronization with subgraph augmentation performs very well, sometimes even outperforming traditional synchronization techniques on the whole measurement graph when data sets are noisy. The choice to estimate the transformations between sub-problems by considering shared vertices in different subgraphs provides reliable measurements that allow us to merge the solutions in an accurate manner. Augmenting the subgraphs has also

the implicit effect of making them larger, thus increasing the amount of information available for solving the synchronization sub-problems.

This approach has two clear downsides, though. Firstly, augmenting subgraphs makes the independent problems harder to solve, given that the number of vertices in a graph determines the complexity of the synchronization problem. Secondly, and perhaps more importantly, the last step of the algorithm, that is solution merging, requires the averaging of multiple estimates for the label of a vertex shared between subgraphs. This may be a challenge that is easy to overcome if synchronization is performed on a group for which an aggregation function exists, e.g. $SO(3)$, but may lead to sub-optimal results in other scenarios. Discarding some labels instead of aggregating them also constitutes a sub-optimal approach and may lead to worse results, especially if the data sets are noisy.

6.7 Partitioned synchronization w/o subgraph augmentation

In this section we introduce partitioned synchronization without subgraph augmentation, a variant of partitioned synchronization that exploits relative measurements between vertices in disjoint clusters in order to estimate the pairwise transformations for the patch multigraph.

Graph partitioning remains unchanged compared to the algorithm in section (6.6), with the only exception that clusters are not augmented. Synchronization on subgraphs is also unchanged: each subgraph (or patch) Γ_i is synchronized independently using standard synchronization techniques, obtaining a total of k different labelings $\mathbf{X}_i$, where k is the number of subgraphs induced by the partition.

6.7.1 Patch multigraph building

The last step is to remove the local ambiguities and lift the multiple labeling $\mathbf{X}_i$ to a single consistent labeling on the original graph. To this end we build a *patch multigraph* defined as follows. The vertices of the patch multigraph correspond to the patches $\Gamma_1, \dots \Gamma_k$, the multi-edge connecting Γ_u and Γ_v consist of all the *cut edges*, *i.e.*, the edges that have one end in Γ_u and the other end in Γ_v.

The vertex labels in each patch must be multiplied by an unknown group element in order to bring them in a common frame. Let these two transformations be $\mathbf{M}_u \in \Sigma$ and $\mathbf{M}_v \in \Sigma$, respectively, then the edge (u, v) in the patch multigraph should be labeled with $\mathbf{M}_u\mathbf{M}_v^{-1}$. In order to find this label, let us consider one edge that connects vertex $i \in \Gamma_u$ and vertex $j \in \Gamma_v$, and let $\mathbf{X}_{i,u}$ and $\mathbf{X}_{j,v}$ be the respective labels. The edge label is:

$$\mathbf{Z}_{i,j} = (\mathbf{X}_{i,u}\mathbf{M}_u)(\mathbf{X}_{j,v}\mathbf{M}_v)^{-1} = \mathbf{X}_{i,u}\mathbf{M}_u\mathbf{M}_v^{-1}\mathbf{X}_{j,v}^{-1} \tag{6.23}$$

Hence:

$$\mathbf{M}_u\mathbf{M}_v^{-1} = \mathbf{X}_{i,u}^{-1}\mathbf{Z}_{i,j}\mathbf{X}_{j,v} \tag{6.24}$$

The patch multigraph for partitioned synchronization without subgraph augmentation is defined as follows:

Definition 6.7.1. ***(Patch multigraph – without subgraph augmentation)*** *Let us consider a Σ-labeled measurement graph $\Gamma = (\mathcal{V}, \mathcal{E}, z)$ and a set $\mathcal{S} = \{\Gamma_1, \dots, \Gamma_k\}$ for which every element Γ_u is a subgraph of Γ.*
The patch multigraph for Γ and its subgraph set $\mathcal{S}$ is defined as a Σ-labeled multigraph $\Phi = (\mathcal{V}_\Phi, \mathcal{E}_\Phi, s_\Phi, t_\Phi, z_\Phi)$, where $\mathcal{V}_\Phi$ contains a vertex for each $\Gamma_u \in \mathcal{S}$ and $\mathcal{E}_\Phi$ contains, for every pair of subgraphs Γ_u and Γ_v, a multi-edge from $u \in \mathcal{V}_\Phi$ to $v \in \mathcal{V}_\Phi$ of multiplicity m_{uv} equal to the number of

cut-edges between Γ_u and Γ_v. In addition,

$$z : \mathcal{E} \to \Sigma \tag{6.25}$$

and, given vertices i and j in Γ_u and Γ_v respectively, is defined as follows:

$$z(e) = \mathbf{X}_{i,u}^{-1} z(i,j) \mathbf{X}_{j,v} \tag{6.26}$$

with $\mathbf{X}_{i,u}$ being the label for vertex i estimated in Γ_u, $\mathbf{X}_{j,v}$ being the label for vertex j estimated in Γ_j, $s_\Phi(e) = u$, $t_\Phi(e) = v$ and $z(i,j)$ being the consistency constraint between vertices (i,j) in Γ. s_Φ, t_Φ are the source and target function respectively, as in the definition of the generic group-labeled multigraph.

6.7.2 Synchronization on the patch graph

The constrained synchronization technique on multigraphs introduced in section (5.2) is applied to the patch multigraph to synchronize all the patch transformations $\mathbf{M}_1 \dots \mathbf{M}_k$ and eventually transform the local label assignments $\mathbf{X}_u$ into a unique, globally consistent labeling.

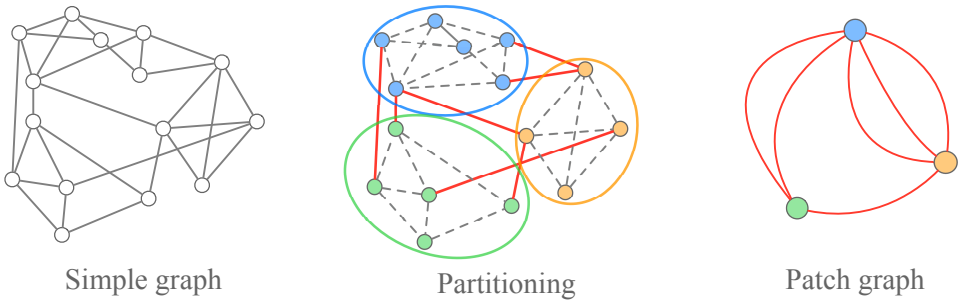

Figure 6.12: *Partitioned synchronization: A simple graph (on the left) is partitioned in three patches $\Gamma_1, \Gamma_2, \Gamma_3$. Each patch is synchronized individually. To solve for the local ambiguities, a patch multigraph is built (on the right), whose nodes correspond to patches and multi-edges contain the cut edges (drawn in red). The patch multigraph is synchronized with our multigraph approach.*

6.7.3 Advantages and disadvantages

This technique overcomes the issues of partitioned synchronization with subgraph augmentation discussed in section (6.6.6). Firstly, subgraphs do not need to be augmented, therefore the complexity of the sub-problems does not increase. Secondly, the algorithm does not output multiple labels for the same vertex, meaning that we are not required to define an aggregation function or to discard estimates.

On the other hand, this approach usually results in less accurate results, since the estimates for the transformations between patches are based on relative measurements which may be very noisy or even outliers. In addition, the resulting patch multigraph can have multi-edges with very high multiplicity values, thus making multigraph expansion very computationally intensive and negating the advantages of partitioning. A solution to this problem is to sample a subset of the cut-edges when estimating the transformations, thus increasing the efficiency of the algorithm at a small cost in accuracy.

6.8 Experimental validation

In this section, we evaluate partitioned synchronization with subgraph augmentation introduced in section (6.6) and compare it to the *edge averaging* technique in the context of partitioned synchronization. For the purposes of this section, we call our method MULTISYNC. We define *edge averaging* as the simplification of a patch multigraph, obtained following the procedure introduced in (6.6), into a simple graph by averaging the labels of the individual edges in each multi-edge. We also compared our approach to partitioned synchronization with the baseline performance provided by the synchronization problem defined on the entire measurement graph. We refer to this problem as *full synchronization*.

For the experimental validation, we considered partitioned synchronization problems in SO(3).We used large-scale image data sets taken from [42] which provide the input graph and estimates of relative rotations. The output of Bundler [35] was taken as ground-truth, as customarily done in the literature. All the experiments were performed in MATLAB on an Intel Core i9 9900k at stock speeds coupled with 16 GB of 3200 MHz RAM.

For every dataset, we perform 50 independent runs, each time sampling a random subset of vertices to share between sub-graphs. Results are averaged and reported in the following sections. The number of clusters c for every data set is computed from the number of vertices n in the measurement graph according to this formula:

$$c = \left\lceil 0.54\sqrt{n} \right\rceil \tag{6.27}$$

where the coefficient 0.54 has been manually tuned on the whole data set.

Analogously to section (5.3), we compare the estimated labels $\tilde{x}$ with the ground-truth ones x as the angular distance between rotations [13], which is defined as:

$$\epsilon_i = || \log(x(i), \tilde{x}(i)^\intercal) || \tag{6.28}$$

6.8.1 Non robust experiments

The first batch of experiments is performed using non robust techniques for synchronization. The synchronization sub-problems are solved independently by spectral decomposition (EIG) and the solutions are merged by solving the synchronization problem on the patch multigraph as follows:

- In MULTISYNC, we use the constrained synchronization algorithm introduced in (5.2.4)

- In edge averaging we average the edge labels and project them in SO(3) and use spectral decomposition (EIG) on the resulting graph

For reference we also report the results of full synchronization performed on the whole measurement graph using spectral decomposition (EIG).

Table (6.6) reports the mean/median angular error and the execution time of all the analysed methods. Results show that MULTISYNC consistently outperforms edge averaging in terms of accuracy on all the cases. In some instances, the median error obtained with MULTISYNC is close or outperforms full synchronization. We interpret this improvement in performance compared to full synchronization as the result of partitioning the problem into subgraphs that mitigate the effect of erroneous measurements or outliers.

MULTISYNC has the important advantage of being significantly less computationally expensive with respect to full synchronization. In Table (6.2) we observe a significant reduction in the execution time for MULTISYNC especially for data sets that are very large. For smaller datasets we observe a regression in performance due to the added overhead of multigraph synchronization. Nonetheless, larger data sets show up to a 2x improvement in performance. Of course, partitioned synchronization trades increased performance for a relatively small loss in accuracy. In Table (6.6), results show that MULTISYNC is generally a bit slower when compared to edge averaging due to the added overhead of the multigraph expansion, though it generally retains a significant lead against full synchronization.

Table 6.1: *Partitioned synchronization with subgraph augmentation in $SO(3)$ on real data sets from [42]. The average error $\bar{\epsilon}$, median error $\hat{\epsilon}$, and computational time t are reported for constrained synchronization and edge averaging. Partitioned synchronization on the sub-problems is solved with spectral decomposition (EIG) and edge averaging is also not robust. Full synchronization performances are included but are intended only as an ideal reference as it works on different assumptions, having at disposal the full graph instead of partitioning it.*

			Edge averaging			Multisync			Full sync		
Data set	n	c	$\bar{\epsilon}$	$\hat{\epsilon}$	t	$\bar{\epsilon}$	$\hat{\epsilon}$	t	$\bar{\epsilon}$	$\hat{\epsilon}$	t
Ellis Island	247	9	12.6	3.9	0.27	11.3	3.1	0.31	6.5	2.8	0.17
Piazza del Popolo	345	11	16.6	3.7	0.26	15.5	3.4	0.32	12.9	3.3	0.24
NYC Library	376	11	29.9	8.3	0.26	29	5.9	0.29	8.4	5.5	0.41
Madrid Metropolis	394	11	39.8	13.4	0.35	32.9	10.9	0.39	20.8	7.5	0.53
Yorkminster	458	12	15.3	6.2	0.25	15.2	5.9	0.29	8.1	5.8	0.82
Montreal N. Dame	474	12	11.4	2.7	0.31	11.4	2.6	0.35	6.6	2.2	0.75
Tower of London	489	13	18.6	5.6	0.24	17.9	4.9	0.28	6.8	3.9	0.82
Notre Dame	553	13	7.9	3.4	0.72	7.7	3	0.76	7.7	3.4	1.12
Alamo	627	14	11.1	3.2	1.1	11	2.9	1.18	7.7	3.2	1.27
Gendarmenmarkt	742	15	72.18	74.88	0.89	71.90	73.27	0.94	66.13	76.52	1.06
Vienna Cathedral	918	17	22.3	8.2	0.58	21	7.8	0.62	20.7	6	2.59
Union Square	930	17	16.1	9.7	0.46	13.8	7.2	0.52	10.7	8.6	1.17
Roman Forum	1102	17	44.9	24.2	0.61	35.8	22.7	0.75	65.8	21.5	1.69
Piccadilly	2508	21	83.1	77	8.38	80.8	62.5	8.43	63.9	35	14.4
Cornell Arts Quad	5530	41	77.4	48.7	6.23	75.9	46.1	6.3	72.3	43.8	58.6

Table 6.2: *Comparison between non robust partitioned synchronization with constrained synchronization for the patch multigraph and non robust full synchronization. Percent changes for the median error $\hat{\epsilon}$ and execution time t are reported.*

		Partitioned sync vs Full sync		
Data set	n	$\bar{\epsilon}$ (% change)	$\hat{\epsilon}$ (% change)	t (% change)
Ellis Island	247	74%	11%	82%
Piazza del Popolo	345	20%	3%	33%
NYC Library	376	245%	7%	-29%
Madrid Metropolis	394	58%	45%	-36%
Yorkminster	458	88%	2%	-65%
Montreal N. Dame	474	73%	18%	-53%
Tower of London	489	163%	26%	-66%
Notre Dame	553	0%	-12%	-32%
Alamo	627	43%	-9%	-7%
Gendarmenmarkt	742	9%	-4%	-11%
Vienna Cathedral	918	1%	30%	-76%
Union Square	930	29%	-16%	-56%
Roman Forum	1102	-46%	6%	-56%
Piccadilly	2508	131%	79%	-41%
Cornell Arts Quad	5530	5%	5%	-89%

6.8.2 Robust experiments

The second batch of experiments is performed using robust techniques for synchronization. The synchronization sub-problems are solved independently by applying MPLS (Message Passing Least Squares in [33]), which represents the current state of the art for robust group synchronization on the datasets in [42]. The solutions are merged by solving the synchronization problem on the patch multigraph as follows:

- In MULTISYNC we use the constrained synchronization method augmented with IRLS, as in [1], to provide robustness to outliers
- In edge averaging the labels of the multi-edges are averaged using the Weiszfeld algorithm [21], thus collapsing the multi-graph into a *simple* graph; then, the resulting graph is synchronized with MPLS [33].

Table (6.3) reports the mean/median angular error and the execution time of all the analysed methods. MULTISYNC consistently outperforms

edge averaging in terms of accuracy on all the cases. The worse accuracy on the Gendarmenmarkt data set is caused by the fact that the graph is relatively sparse and lacks cycle information, as already observed in [32]. The great advantage of MULTISYNC is that it trades relatively little accuracy in exchange for a major speed up in the execution time of the algorithm. Some of the larger data sets, such as Piccadilly, show up to a 13x improvement in this regard. As a matter of fact, In Table (6.4), results for MULTISYNC are even more favorable compared to full synchronization compared to the results for non robust techniques. In some of the larger datasets, we trade less than 10% accuracy for a very significant speed up in performance.

Table 6.3: *Partitioned synchronization with subgraph augmentation in $SO(3)$ on real data sets from [42]. The average error $\bar{\epsilon}$, median error $\hat{\epsilon}$, and computational time t are reported for constrained synchronization and edge averaging. Partitioned synchronization on the sub-problems is solved robustly with MPLS [33] and edge averaging is performed with the Weiszfeld algorithm [21]. Full synchronization is also solved with MPLS. Full synchronization performances are included but are intended only as an ideal reference as it works on different assumptions, having at disposal the full graph instead of partitioning it.*

			Edge averaging			Multisync			Full sync		
Data set	n	c	$\bar{\epsilon}$	$\hat{\epsilon}$	t	$\bar{\epsilon}$	$\hat{\epsilon}$	t	$\bar{\epsilon}$	$\hat{\epsilon}$	t
Ellis Island	247	9	3.9	0.59	0.77	3.9	0.58	0.81	3	0.47	3.4
Piazza del Popolo	345	11	3.9	1.01	0.83	3.7	1.02	0.97	3.3	0.86	3.47
NYC Library	376	11	3.7	1.43	2.79	3.6	1.34	2.88	3.16	1.27	4.8
Madrid Metropolis	394	11	8.6	2.41	2.75	8	2.23	2.82	6.6	1.12	1.18
Yorkminster	458	12	6.91	2.75	1.66	6.23	2.26	1.84	3.5	1.58	4.83
Montreal N. Dame	474	12	1.29	0.53	8.32	1.27	0.5	8.44	1.12	0.5	10.1
Tower of London	489	13	4.56	2.3	1.27	4.49	2.05	1.35	4.21	2.33	3.91
Notre Dame	553	13	3.12	0.68	2.81	3.12	0.66	2.88	2.7	0.65	22
Alamo	627	14	5.9	1.1	4.08	5.88	1.07	4.12	3.7	1.02	26.5
Gendarmenmarkt	742	15	38.13	20.61	2.38	34.12	12.53	2.49	40.82	6.09	39.9
Vienna Cathedral	918	17	21.4	1.45	3.07	21.3	1.31	3.11	6.2	1.27	50.2
Union Square	930	17	15.9	6.62	2.49	15.6	6.3	2.59	6.18	3.6	6.61
Roman Forum	1102	17	6.28	3.37	3.83	5.15	2.52	3.94	2.81	1.4	12.1
Piccadilly	2508	21	7.68	3.01	17.6	7.3	2.76	17.7	4.42	1.94	241
Cornell Arts Quad	5530	41	3.6	1.98	18.4	3.4	1.88	18.5	3.2	1.71	191

Table 6.4: *Comparison between robust partitioned synchronization with constrained synchronization for the patch multigraph and robust full synchronization with MPLS. Percent changes for the median error $\hat{\epsilon}$ and execution time t are reported.*

		Partitioned sync vs Full sync		
Data set	n	$\bar{\epsilon}$ (% change)	$\hat{\epsilon}$ (% change)	t (% change)
Ellis Island	247	30%	23%	-76%
Piazza del Popolo	345	12%	19%	-72%
NYC Library	376	14%	6%	-40%
Madrid Metropolis	394	21%	99%	139%
Yorkminster	458	78%	43%	-62%
Montreal N. Dame	474	13%	0%	-16%
Tower of London	489	7%	-12%	-65%
Notre Dame	553	16%	2%	-87%
Alamo	627	59%	5%	-84%
Gendarmenmarkt	742	-16%	106%	-94%
Vienna Cathedral	918	244%	3%	-94%
Union Square	930	152%	75%	-61%
Roman Forum	1102	83%	80%	-67%
Piccadilly	2508	65%	42%	-93%
Cornell Arts Quad	5530	6%	10%	-90%

6.8.3 Additional testing

Partitioned synchronization without subgraph augmentation: We evaluated the performance of partitioned synchronization *without* subgraph augmentation and compared it to its counterpart *with* subgraph augmentation. Table (6.5) shows the performance of this method under the same conditions and parameters as the robust experiments conducted in section (6.8.2).

In the majority of test cases, this variant of partitioned synchronization trades accuracy for marginal improvements in execution time compared to synchronization *with* subgraph augmentation. Although these improvements in efficiency can definitely make a difference in certain scenarios, these results give credit to our recommendation of using subgraph augmentation whenever possible, given that the small efficiency gains do not justify the loss in accuracy.

Other synchronization methods: As already noted, MULTISYNC can be seen as a general framework that is agnostic about the synchronization technique used on subgraphs. This aspect is investigated in Table (6.6), which

Table 6.5: *Partitioned synchronization without subgraph augmentation in $SO(3)$ on real data sets from [42]. The average error $\bar{\epsilon}$, median error $\hat{\epsilon}$, and computational time t are reported for constrained synchronization and edge averaging. Partitioned synchronization on the sub-problems is solved robustly with MPLS [33] and edge averaging is performed with the Weiszfeld algorithm [21]. Full synchronization is also solved with MPLS. Full synchronization performances are included but are intended only as an ideal reference as it works on different assumptions, having at disposal the full graph instead of partitioning it.*

			Edge averaging			MULTISYNC			Full sync		
Data set	n	c	$\bar{\epsilon}$	$\hat{\epsilon}$	t	$\bar{\epsilon}$	$\hat{\epsilon}$	t	$\bar{\epsilon}$	$\hat{\epsilon}$	t
Ellis Island	247	9	3.56	0.73	1.02	3.49	0.68	1.07	3	0.47	3.4
Piazza del Popolo	345	11	5.62	1.86	1.27	5.22	1.41	1.38	3.3	0.86	3.47
NYC Library	376	11	4.91	3.35	0.93	4.24	2.32	1.01	3.16	1.27	4.8
Madrid Metropolis	394	11	7.84	3.19	1.13	7.14	2.52	1.18	6.6	1.12	1.18
Yorkminster	458	12	5.96	3.98	1.33	4.98	2.91	1.37	3.5	1.58	4.83
Montreal N. Dame	474	12	2.67	1.02	1.32	2.11	0.87	1.41	1.12	0.5	10.1
Tower of London	489	13	6.43	3.51	1.07	5.54	2.74	1.12	4.21	2.33	3.91
Notre Dame	553	13	4.25	1.92	3.82	3.43	0.85	3.87	2.7	0.65	22
Alamo	627	14	6.89	1.63	4.21	6.42	1.57	4.28	3.7	1.02	26.5
Gendarmenmarkt	742	15	39.54	21.18	2.29	34.32	12.68	2.34	40.82	6.09	39.9
Vienna Cathedral	918	17	15.73	4.87	3.88	11.01	3.73	3.92	6.2	1.27	50.2
Union Square	930	17	7.71	3.88	3.65	7.25	3.67	3.71	6.18	3.6	6.61
Roman Forum	1102	17	10.03	9.12	4.95	6.91	3.39	5.01	2.81	1.4	12.1
Piccadilly	2508	21	19.89	10.21	17.45	17.47	8.21	17.61	4.42	1.94	241
Cornell Arts Quad	5530	41	8.64	3.29	17.88	6.25	2.71	18.02	3.2	1.71	191

shows the performance of MULTISYNC coupled with different synchronization methods, namely EIG-IRLS [1], L1-IRLS [12] and R-GODEC [2]. The state-of-the-art MPLS [32] is only reported in Table 6.3. EIG-IRLS results are not reported on Cornell Arts Quad because it did not reach convergence.

It turns out that different performances of the various methods on the full graph reflect on the partitioned case: for instance, R-GoDec is the fastest solution whereas L1-IRLS is the most accurate. With respect to full synchronization, again we see that MULTISYNC strikes a good balance between accuracy and computational burden.

Table 6.6: *Performances of* MULTISYNC *leveraging on different techniques on subgraphs, for synchronization in $SO(3)$ on real data sets [42]. The median error $\hat{\epsilon}$, and computational time t are reported.*

			EIG-IRLS				L1-IRLS				R-GoDec			
			MULTISYNC		Full sync		MULTISYNC		Full sync		MULTISYNC		Full sync	
Data set	n	c	$\hat{\epsilon}$	t	$\hat{\epsilon}$	t	$\hat{\epsilon}$	t	$\hat{\epsilon}$	t	$\hat{\epsilon}$	t	$\hat{\epsilon}$	t
Ellis Island	247	9	1.15	0.43	1.18	0.82	1.05	0.41	0.57	2.35	1.48	0.23	1.00	0.23
Piazza del Popolo	345	11	1.78	1.18	1.02	2.24	1.84	0.53	0.98	3.55	1.86	0.24	1.48	0.49
NYC Library	376	11	3.24	3.15	1.98	1.65	2.02	0.39	1.33	2.36	2.82	0.47	3.20	1.35
Madrid Metropolis	394	11	4.01	3.83	4.43	1.79	2.68	0.57	1.01	4.20	3.94	0.29	4.07	0.49
Yorkminster	458	12	2.91	2.85	1.81	3.18	2.86	1.07	1.69	2.29	2.85	0.43	2.69	2.03
Montreal N. Dame	474	12	2.12	6.92	0.59	4.09	1.01	3.67	0.58	7.10	1.02	0.52	0.85	1.05
Tower of London	489	13	2.98	3.74	2.79	2.43	2.83	1.20	2.63	1.94	2.89	0.46	3.28	2.11
Notre Dame	553	13	1.23	5.22	0.74	7.46	1.22	25.94	0.65	29.11	1.57	1.79	1.05	1.10
Alamo	627	14	1.99	2.01	1.19	11.05	1.87	1.84	1.09	32.22	1.61	0.79	1.48	1.67
Gendarmenmarkt	742	15	26.88	8.19	76.97	11.30	14.81	2.12	28.85	12.01	37.36	1.01	28.70	5.83
Vienna Cathedral	918	17	4.72	2.88	1.62	18.23	3.92	1.68	1.37	56.80	2.53	1.59	2.08	9.26
Union Square	930	17	18.95	4.13	4.93	6.48	4.33	1.05	3.97	4.82	20.17	1.57	7.16	10.58
Roman Forum	1102	17	7.89	6.11	1.86	15.46	3.85	2.20	2.27	12.46	5.54	1.12	7.54	14.77
Piccadilly	2508	21	39.55	54.05	24.87	284.87	9.01	8.93	1.89	287.23	11.79	12.24	13.36	47.13
Cornell Arts Quad	5530	41	-	-	-	-	5.49	30.10	1.98	73.51	17.13	28.24	13.21	586.6

CHAPTER 7

Conclusions and future work

In this work, we studied – for the first time – the task of synchronization on group-labeled multigraphs. Traditionally, synchronization problems are defined on simple graphs and do not cope with multiple measurements between unknown states. We instead explored this possibility by introducing a novel and general multigraph synchronization framework that allows us to account for multiple measurements between states.

After formally introducing the theoretical framework (section 4.2), we designed MULTIGRAPHEXPAND, an iterative greedy algorithm for expanding any labeled multigraph into a simple graph (section 4.5). The algorithm is designed for efficiency and preserves the pairwise measurements as well as the underlying structure of the original multigraph, while replicating only the vertices that are strictly required.

Then, we solved the challenges that naturally arise when synchronizing expanded multigraphs (section 5.1). We formally introduced a new synchronization technique based on a constrained optimization for which we defined a principled closed-form spectral solution (section 5.2). Our approach exploits the redundancy encapsulated in the multi-edges and has better statistical properties with respect to its competitor edge averaging, improving its performance both in terms of accuracy and precision as demon-

strated by our synthetic experiments (section 5.3).

In addition, we discussed a prominent application of synchronization on multigraphs, *i.e.* partitioned synchronization (chapter 6). The measurement graph is partitioned into smaller subgraphs that are synchronized independently, exploiting parallelism to greatly speed up the execution time. Our multigraph formulation is used to remove the local ambiguities and lift the multiple labelings for the subgraphs to a single consistent labeling on the original graph. We introduced two variants: i) with subgraph augmentation and ii) without subgraph augmentation, with the first being more accurate but slightly less efficient (sections 6.6 and 6.7). Our experiments on real data sets indicate that our approach to partitioned synchronization is very efficient and outperforms its other partitioned competitors in terms of accuracy and precision.

To summarize, our contribution is three-fold:

- We presented, for the first time, a formal definition of the synchronization of group-labeled multigraphs, which is a significant extension of the synchronization of simple graphs
- We derived a practical algorithm for solving a synchronization problem on a multigraph, which is based on an expansion algorithm coupled with a constrained spectral solution to deal with replicated vertices
- We demonstrated how the multigraph framework can be conveniently used to partition classical synchronization tasks, achieving a good trade-off between accuracy and complexity.

Applications of multigraph synchronization are countless, including partitioned synchronization, measurements taken from multiple sensors and SLAM. This aspect provides the main motivation for future research. Future research should focus on understanding which scenarios are best suited to adopt the multigraph formulation and on evaluating the impact of our contribution. Our framework allows us to account for multiple measurements from different sensors. In SLAM, multiple sensors (cameras, IMU, GPS, ...) can estimate the 6 d.o.f motion of a vehicle, therefore it represents a promising field for future research. In addition, we focused on partitioned synchronization and tested our method on well-known data sets in the literature [42]. Another interesting possibility for future research is to evaluate the performance of our contribution on additional data sets and different real-world scenarios. To summarize, future work will be focused on

looking into more applications in which multiple pairwise measurements between states can be obtained and on evaluating the impact of our contribution in additional real-world scenarios.

Bibliography

[1] F. Arrigoni, B. Rossi, and A. Fusiello. Spectral synchronization of multiple views in SE(3). *SIAM Journal on Imaging Sciences*, 9(4):1963 – 1990, 2016.

[2] Federica Arrigoni, Beatrice Rossi, Pasqualina Fragneto, and Andrea Fusiello. Robust synchronization in SO(3) and SE(3) via low-rank and sparse matrix decomposition. *Computer Vision and Image Understanding*, 174:95–113, 2018.

[3] Arrigoni, Fusiello. Synchronization problems in computer vision with closed-form solutions. *IJCV*, 128:26–52, 2020.

[4] Arrigoni, Magri, Pajdla. On the usage of the trifocal tensor in motion segmentation. *ECCV*, 2020.

[5] A. Bartoli and P. Sturm. Constrained structure and motion from multiple uncalibrated views of a piecewise planar scene. 52(1):45–64, 2003.

[6] Florian Bernard, Johan Thunberg, Paul Swoboda, and Christian Theobalt. HiPPI: Higher-order projected power iterations for scalable multi-matching. 2019.

[7] Brojeshwar Bhowmick, Suvam Patra, Avishek Chatterjee, Venu Madhav Govindu, and Subhashis Banerjee. Divide and conquer: Efficient large-scale structure from motion using graph partitioning. In *12th Asian Conference on Computer Vision (ACCV 2014)*, 2014.

[8] Simone Bianco, Gianluigi Ciocca, and Davide Marelli. Evaluating the performance of structure from motion pipelines. *Journal of Imaging*, 4(8), 2018.

[9] T. Birdal, M. Arbel, U. Simsekli, and L. J. Guibas. Synchronizing probability measures on rotations via optimal transport. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1566–1576, 2020.

[10] Tolga Birdal and Umut Simsekli. Probabilistic permutation synchronization using the riemannian structure of the birkhoff polytope. pages 11105–11116, 2019.

[11] Tolga Birdal, Umut Simsekli, Mustafa Onur Eken, and Slobodan Ilic. Bayesian pose graph optimization via bingham distributions and tempered geodesic mcmc. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[12] A. Chatterjee and V. M. Govindu. Efficient and robust large-scale rotation averaging. 2013.

[13] Du Q. Huynh. Metrics for 3d rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 35(2):155–164, 2009.

[14] Meiling Fang, Thomas Pollok, and Chengchao Qu. Merge-sfm: Merging partial reconstructions. page 29, 2019.

[15] Gene H. Golub. Some modified eigenvalue problems. *SIAM*, 15(2):318–334, 1973.

[16] Marcel Geppert, Viktor Larsson, Pablo Speciale, Johannes L Schönberger, and Marc Pollefeys. Privacy preserving structure-from-motion. In *European Conference on Computer Vision*, pages 333–350. Springer, 2020.

[17] Ghojogh, Karray, Crowley. Eigenvalue and generalized eigenvalue problems: Tutorial. pages 1–8, 2019.

[18] David Gibson, Ravi Kumar, and Andrew Tomkins. Discovering large dense subgraphs in massive graphs. In *Proceedings of the 31st international conference on Very large data bases*, pages 721–732. Citeseer, 2005.

[19] V. M. Govindu. Lie-algebraic averaging for globally consistent motion estimation. pages 684–691, 2004.

[20] V. M. Govindu and A. Pooja. On averaging multiview relations for 3D scan registration. 23(3):1289–1302, 2014.

[21] R. Hartley, K. Aftab, and J. Trumpf. L1 rotation averaging using the Weiszfeld algorithm. pages 3041–3048, 2011.

[22] R. I. Hartley, J. Trumpf, Y. Dai, and H. Li. Rotation averaging. *International Journal of Computer Vision*, 2013.

[23] P. W. Holland and R. E. Welsch. Robust regression using iteratively reweighted least-squares. *Communications in Statistics - Theory and Methods*, 6(9):813–827, 1977.

[24] Xiangru Huang, Zhenxiao Liang, Xiaowei Zhou, Yao Xie, Leonidas J. Guibas, and Qixing Huang. Learning transformation synchronization. June 2019.

[25] Samir Khuller and Barna Saha. On finding dense subgraphs. In *International Colloquium on Automata, Languages, and Programming*, pages 597–608. Springer, 2009.

[26] S. Leonardos, X. Zhou, and K. Daniilidis. Distributed consistent data association via permutation synchronization. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2645–2652, 2017.

[27] X. Li and H. Ling. Hybrid camera pose estimation with online partitioning for slam. *IEEE Robotics and Automation Letters*, 5(2):1453–1460, 2020.

[28] Pulak Purkait, Tat-Jun Chin, and Ian Reid. Neurora: Neural robust rotation averaging. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 137–154. Springer International Publishing, 2020.

[29] D. M. Rosen, C. DuHadway, and J. J. Leonard. A convex relaxation for approximate global optimization in simultaneous localization and mapping. pages 5822 – 5829, 2015.

[30] E. Santellani, E. Maset, and A. Fusiello. Seamless image mosaicking via synchronization. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-2:247–254, 2018.

[31] J. Shi and J. Malik. Normalized Cuts and Image Segmentation. 22(8):888–905, 2000.

[32] Yunpeng Shi and Gilad Lerman. Message passing least squares framework and its application to rotation synchronization. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 8796–8806. PMLR, 2020.

[33] Yunpeng Shi and Gilad Lerman. Message passing least squares framework and its application to rotation synchronization. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 8796–8806. PMLR, 13–18 Jul 2020.

[34] A. Singer. Angular synchronization by eigenvectors and semidefinite programming. *Applied and Computational Harmonic Analysis*, 30(1):20 – 36, 2011.

[35] Snavely, Seitz, Szelisk. Photo tourism: Exploring photo collections in 3d. *SIGGRAPH*, pages 835–846, 2006.

[36] J. Thunberg, F. Bernard, and J. Gonçalves. Distributed synchronization of euclidean transformations with guaranteed convergence. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 3757–3762, 2017.

[37] A. Torsello, E. Rodolà, and A. Albarelli. Multiview registration via graph diffusion of dual quaternions. pages 2441 – 2448, 2011.

[38] R. Tron and K. Danilidis. Statistical pose averaging with varying and non-isotropic covariances. 2014.

[39] R. Tron and R. Vidal. Distributed 3-D localization of camera sensor networks from 2-D image measurements. *IEEE Transactions on Automatic Control*, 59(12):3325–3340, 2014.

[40] von Luxburg. A tutorial on spectral clustering. pages 1–32, 2007.

[41] L. Wang and A. Singer. Exact and stable recovery of rotations for robust synchronization. *Information and Inference: a Journal of the IMA*, 2(2):145–193, 2013.

[42] Wilson, Snavely. Robust global translations with 1dsfm. *European Conference on Computer Vision*.

[43] Jin-Gang Yu, Gui-Song Xia, Ashok Samal, and Jinwen Tian. Globally consistent correspondence of multiple feature sets using proximal Gauss–Seidel relaxation. *Pattern Recognition*, 51:255 – 267, 2016.