UCLouvain

École polytechnique de Louvain

Matrix Completion and Graph Rigidity: Exploiting Surprising Similarities

Author: **Marie THIBEAU**
Supervisor: **Julien HENDRICKX**
Readers: **Jean-Charles DELVENNE, Simon VARY**
Academic year 2021–2022
Master [120] in Mathematical Engineering

ABSTRACT

This master thesis addresses the problem of uniqueness of matrix completion by means of the graph rigidity results. The state of the art studies the existing results in both rigidity theory and matrix completion problem. It allows to determine that no author who addressed the question of the uniqueness of matrix completion using the results of graph rigidity, tried to directly reproduce the combinatorial results of Laman in two dimensions. A summary of Laman's results allows their good understanding in order to reproduce them in the context of matrix completion. A necessary and sufficient condition is found for the uniqueness of matrix completion. Finally, a pebble game algorithm allows to show whether a given matrix is uniquely completable.

Key words: Unique matrix completion, Graph rigidity, Pebble game algorithm, Combinatorial characterization.

ACKNOWLEDGMENTS

First of all, I would like to thank my supervisor Prof. Julien Hendrickx for his availability and guidance. He always made himself available to answer my questions and give me ideas, despite the very tight schedule I sometimes imposed.

I would also express my great gratitude to my Mum, Valérie Goffin, for her careful proofreading. She helped me approach things differently and more serenely. I will remember 3 learnings from these shared moments:

1. As long as the end is not reached, the universe of possibilities is open to exploration;

2. The impossible is only an undiscovered possibility or just a reason to look elsewhere;

3. Even if being alone allows you to build yourself up, a woman is never alone in the world (nor is a man)... it is worth seeking at others what allows you to move forward!

I don't forget my Dad, François Thibeau, for checking my English, thank you very much for your time.

Finally, my immense gratitude to my friends for their support in this challenge and in everyday life.

CONTENTS

CHAPTER 1

INTRODUCTION

Exploiting the mathematical similarities between two problems that are *a priori* completely unrelated is a wonderful goal for a master thesis. Even more wonderful when, after having studied the two problems, we move from the status of "trying to progress on an **issue**" to the status of "replicating, in the context of **problem 2**, the developments made in the context of **problem 1**".

Problem 1: Graph rigidity

Rigidity of a graph characterizes the ability of a structure composed of joints and beams to maintain its shape. See the example below where the left graph is rigid and the right one is flexible, both in the plane.

Indeed, the graph on the right can be deformed while keeping the length of its edges constant, which is not the case for the one on the left.

Rigidity is very useful in civil and mechanical engineering as well as in autonomous agent systems, scene analysis or in chemistry and biology.

Problem 2: Matrix completion

Low-rank matrix completion consists in completing a low-rank matrix A such that its known entries are preserved. The matrix A is for example as follows.

$$\begin{pmatrix} a_{11} & ? & ? & a_{14} & ? \\ a_{21} & ? & a_{23} & ? & a_{25} \\ ? & ? & a_{33} & a_{34} & ? \\ ? & a_{42} & ? & ? & a_{45} \end{pmatrix}$$

The question arises as to when a mask Ω of known entries a_{ij} is sufficient to uniquely complete a rank-d matrix A.

This problem is of great interest in various applications such as image compression, recommendation systems or localization within a network.

Issue: Uniqueness of matrix completion

These two problems seem completely unrelated but they present surprising mathematical similarities. These similarities have been noticed but not entirely exploited and they could help to solve the question of uniqueness of matrix completion.

The question is whether the known entries of the matrix imply a unique completion, i.e. a unique value for the missing entries. Most results addressing the uniqueness of matrix completion assume that the mask of the known entries is randomly chosen and that the matrix has incoherence property, i.e. the singular vectors of the matrix A must be sufficiently spread out.

No result seems to address the problem of uniqueness when considering a given mask, i.e. a non-randomly chosen mask.

In the rigidity problem, a full characterization using graph theory has been provided for the two-dimensional case. Given the similarities between the two problems, the results developed for the rigidity theory could be used to solve the problem of uniqueness of matrix completion.

Overview

This paper aims to give a complete characterization of the problem of uniqueness of matrix completion using graph rigidity results. Here are the different parts that are discussed.

The Chapter 2 studies the state of the art of the two problems of graph rigidity and matrix completion. It also gives a complete overview of papers addressing the matrix completion problem from a graph rigidity perspective. Finally, it provides the contribution of this master thesis to the literature.

In Chapter 3, the whole theory of graph rigidity used in this master thesis is described in detail. It is based on the results that fully characterize the rigidity problem in the two-dimensional case.

The Chapter 4 gives a full characterization of the matrix completion problem in the case of a partially filled symmetric matrix of rank 2. The results reproduce those of graph rigidity but in the case of matrix completion. At the end, a necessary and sufficient condition is found for the uniqueness of the matrix completion.

Finally, the Chapter 5 describes algorithms that allow to show whether a given graph is rigid and whether a given matrix is completable. An implementation of the algorithm for completable matrix is also provided.

Discussion, perspectives and conclusion are presented in Chapter 6.

CHAPTER 2

STATE OF THE ART

The objective of this chapter is to position this master thesis in the literature as well as to give the necessary background to understand all the concepts used. The main two topics are rigidity of graphs and matrix completion.

First, the Section 2.1 summarizes the rigidity problem statement developed in Chapter 3 and gives applications and a brief description of the different results obtained in the literature. Then, the Section 2.2 describes the matrix completion problem, its fields of application and gives a set of existing results. Moreover, the Section 2.3 shows the similarities between the rigidity and completion problems and presents previous works that have addressed matrix completion from a rigidity point of view. This section is important and will shed light on already existing results on the issue. Finally, in Section 2.4, the contribution of this master thesis to the literature is provided.

2.1 Rigidity of graphs

The problem of knowing if a graph is rigid is briefly posed in this section because we need notations for comparison with the context of matrix completion. More details and illustrations are available in the chapter about rigidity theory, in Section 3.1. Then some applications of rigidity theory are given as well as a summary of existing results.

2.1.1 Problem statement

The rigidity problem consists in saying whether a given graph $G(V, E)$ is rigid or flexible. If it is rigid, it can be either locally rigid or globally rigid.

We consider a graph $G(V, E)$ with n vertices and m edges. A realization of this graph in $\mathbb{R}^d$ is given by $p = (p_1, p_2, ..., p_n) \in \mathbb{R}^{dn}$. We called p_i the position of the vertex i and the length of the edge (i, j) is given by $||p_i - p_j||$. The association of the graph G with a realization p is called a skeletal structure and noted (G, p).

A graph with a given realization (G, p) is said to be rigid in $\mathbb{R}^d$ if every continuous path in $\mathbb{R}^{dn}$, preserving the edge lengths of (G, p) and beginning at p, terminates at a point $q \in \mathbb{R}^{dn}$ which is congruent to p. Two realizations are said

to be congruent if they can be obtained one from each other by a composition of Euclidean transformations, i.e. translations, rotations and reflections.

It is important to notice that rigidity is a property defined for realizations of graphs. However, rigidity is a generic property of a graph, meaning that either almost all its realizations are rigid or all of them are not rigid.

Another way of approaching rigidity is through the notion of infinitesimal rigidity which is a kind of first order approach. Consider a motion of the realization $p \in \mathbb{R}^{dn}$ with $p_i(t)$ being the motion vector of the vertex i at time t. Any infinitesimal motion that instantaneously preserves the lengths of the edges must satisfy $\frac{d}{dt}||p_i - p_j||^2 = 0$ for all $(i,j) \in E$. If we denote the infinitesimal motion by $\delta \in \mathbb{R}^{dn}$:

$$\delta = \begin{pmatrix} \delta_1 \\ \delta_2 \\ \vdots \\ \delta_n \end{pmatrix},$$

where $\delta_i \in \mathbb{R}^d$, for each edge we have the following equation:

$$(p_i - p_j)^T(\delta_i - \delta_j) = 0, \quad \text{for all } (i,j) \in E.$$

This gives a system of m linear equations with dn unknowns that can be written in matrix form: $R_{G,p}\delta = 0$, where the matrix $R_{G,p} \in \mathbb{R}^{m \times dn}$ is called the *rigidity matrix* and is sparse because it has only at most $2d$ nonzero elements per row.

In this case, a skeletal structure (G,p) is said to be infinitesimally rigid if all infinitesimal motions are trivial, i.e. δ is a motion that is a combination of some orthogonal transformations and some translations, [29].

2.1.2 Applications

The theory of rigidity is of great use in many areas such as civil and mechanical engineering. The application that seems most logical is the rigidity of structures and especially frames. *Tay*, in his paper on "Generating Isostatic Frameworks" [33], discusses a way to generate graphs of all rigid frames in space and in particular in the plane. He generates such a graph using a Henneberg sequence from a single edge.

Another field of application where rigidity is very useful is autonomous agent systems. An autonomous agent is a system that attempts to fulfill a set of objectives in a complex and dynamic environment by making its own decisions. Autonomous robots, artificial life agents or computer viruses are examples of autonomous agents. Autonomous agents can interact with each other when they are

part of a multi-agent system. In this case, it is often important that the individual agents keep the distance to their neighbors constant and that the shape of the formation is maintained, so rigidity theory is used. *Zhang et al.* [36] give an algorithm to keep an "As-Rigid-As-Possible" structure. *Singer* [28] proposes an algorithm to find a global positioning of points in Euclidean space from a set of pairwise distances. *Hendrickx* [11] also uses rigidity theory to study autonomous agent systems. Finally, in [37], *Zhao and Zelazo* drew a parallel between bearing rigidity theory and distance rigidity theory, the former being the case where each agent in a network can only perceive the relative bearings to their nearest neighbors.

As in *Whiteley*'s paper [35], rigidity theory can also be used for scene analysis and geometry. Scene analysis is a useful scene model for re-identifying people from one camera to another.

Finally, rigidity theory can be used in chemistry and biology. For example, it is used for determining molecular conformations [9], in the field of non-crystalline solids [25] or to identify the heterogeneous composition of flexible and rigid regions of biomolecules [13].

2.1.3 Existing results

This section briefly describes the theoretical results obtained in the literature. The theory used in this master thesis is described in more detail in Chapter 3.

In 1911, in the plane, *Henneberg* in [12] was the first to propose operations consisting in adding:

- a vertex of degree 2, called *vertex addition*;
- a vertex of degree 3, called *edge splitting*.

Under certain conditions, these operations preserve the rigid character of the structure.

Laman was one of the first to propose results in 1970 in his paper "On Graphs and Rigidity of Plane Skeletal Structures" [19]. He gave new definitions of skeletal structure and of rigidity. *Laman* limited himself to the study of rigidity in the Euclidean plane. His criteria for rigidity are combinatorial. Indeed, it uses graph theory and the counting of edges in the graph to determine whether the graph is rigid or not. He also defined a new class of graphs which have edges distributed in such a way as to have a limited number of edges on each subset of vertices, they are called E-graphs. He proved that every graph belonging to this new class has

rigid realizations. An exact combinatorial condition for rigid graphs is currently not available in dimensions higher than the plane ($d \geq 3$). In higher dimension, the condition on the distribution of edges in the graph is necessary but not sufficient.

Later, in the late 1970s, *Asimow and Roth* gave criteria for determining whether a graph is flexible or rigid in $\mathbb{R}^d$. They also defined the notion of generic rigidity, which means that a graph is rigid in $\mathbb{R}^d$ for almost all locations of its vertices. They used the edge function of the graph G, f_G, defined as

$$f_G(t_1, ..., t_n) = (..., ||t_i - t_j||^2, ...)$$

where $(i, j) \in E$ and $t_k \in \mathbb{R}^d$ for $1 \leq k \leq n$. The rank of the derivative of f_G in point p must satisfy certain conditions for the graph G to be rigid for its realization p. They were interested in both continuous rigidity [2] and infinitesimal rigidity [3]. In their second paper, they also adapted Laman's combinatorial criteria using the edge function.

In the article "Recent advances in the generic rigidity of structures" [32] published in 1984, *Tay and Whiteley* synthesized results describing rigid graphs in an d-dimensional space.

Whiteley, in his 1988 paper [34], showed that results characterizing graphs of minimal infinitely rigid structures can be obtained from the model of their representative rigidity matrices and the connection to unions of the cycle matroids of a graph.

In 1990, *Crapo* [8] characterized the generic rigidity of bar frameworks in the plane in terms of tree decompositions. Then *Tay* [31] gave a direct proof of this characterization and uses it to give a short proof of the characterization of the generic rigidity of $(d-1, 2)$-frames in d-space, it is a structure consisting of a set of $(d-2)$-dimensional panels where some pairs are connected by one or more rods by means of ball joints.

In 1992, *Hendrickson* [10] used rigidity to identify necessary conditions for a graph to have a unique realization in any dimension. He also proposed an algorithm for testing rigidity.

Jacobs and Thorpe [17] and *Jacobs and Hendrickson* [16] built a simple pebble game algorithm based on the rigidity algorithm proposed by *Hendrickson* [10]. The algorithm is constructed around Laman's theorem [19] saying that in a rigid graph $G(V, E)$ with $2n - 3$ edges in two dimensions, there is no subgraph G' with more than $2n' - 3$ edges, see Chapter 3. Laman's theorem gives a poor algorithm because it requires counting the edges in every subgraph. The goal is to apply the theorem of Laman recursively by building the network up one bond at a time.

Lee and Streinu [21] generalized the notion and give pebble games algorithms for sparse graphs and not only rigid graphs. A multi-graph G on n vertices is (k, l)-sparse if every subset of $n' \leq n$ vertices spans at most $kn' - l$ edges. G is tight if, in addition, it has exactly $kn - l$ edges. A (k, l)-spanning graph is one containing a tight subgraph that spans the entire vertex set V. In consequence, the $(2, 3)$-tight graphs are the Laman graphs (generic minimally rigid) and the spanning ones are rigid graphs.

The pebble game algorithms of interest for this master thesis is described in detail in Chapter 5.

All this exploration of the literature allowed to highlight a limited number of results which proved to be useful to us: the theorem of Laman and the Pebble Game algorithm.

2.2 Matrix completion

The matrix completion problem is widely used in many applications. In this section, the problem is stated, some applications are given and existing results are summarized.

At the end of this section, we will realize that there is still a lot to discover in this field. For example, the case where the known entries of the matrix are given in a deterministic way is not yet much explored, although it could be useful for some applications. It is precisely this point of view that interests us in this master thesis.

2.2.1 Problem statement

Matrix completion and more specifically low-rank matrix completion consists in completing a low-rank matrix A with n_1 rows and n_2 columns from m partial observations. The number m of observed entries is usually much smaller than $n_1 n_2$, the total number of entries. Some entries a_{ij} are given together with a value of rank d and we look for the matrix $X \in \mathbb{R}^{n_1 \times n_2}$ with all entries x_{ij} known and such that $x_{ij} = a_{ij}$ for all known entries a_{ij}. In many applications, the rank d of the matrix A is supposed to be low.

$$\begin{pmatrix} a_{11} & ? & ? & a_{14} \\ a_{21} & ? & a_{23} & ? \\ ? & ? & a_{33} & a_{34} \\ ? & a_{42} & ? & ? \end{pmatrix}$$

Figure 2.1: The mask of this matrix is $\Omega = \{(1,1), (1,4), (2,1), (2,3), (3,3), (3,4), (4,2)\}$.

Definition 2.2.1. *The* mask Ω *is the set of indices of the observed entries:* $(i, j) \in \Omega$ *if* a_{ij} *is known, see Figure 2.1.*

Definition 2.2.2. *The* sampling operator $S_\Omega(A)$ *of a matrix* A *is:*

$$[S_\Omega(A)]_{ij} = \begin{cases} a_{ij} & \text{if } (i,j) \in \Omega \\ 0 & \text{otherwise} \end{cases}$$

The basic problem to complete the low-rank matrix A is the rank minimization problem:

$$\begin{array}{ll} \min\limits_X & rank(X) \\ \text{s.t.} & x_{ij} = a_{ij}, (i,j) \in \Omega \end{array} \quad \Leftrightarrow \quad \begin{array}{ll} \min\limits_X & rank(X) \\ \text{s.t.} & S_\Omega(X) = S_\Omega(A) \end{array} \tag{2.1}$$

where X is the matrix recovered from the partially filled matrix A.

2.2.2 Applications

Matrix completion is used in many applications [27]. A first well-known application is the Netflix problem in the field of recommendation systems [4]. The data is represented in a matrix where each column corresponds to a user's ratings for the movies. Users do not rate many films and therefore the matrix is sparsely filled. The aim is to estimate the interests of the users by filling in the matrix. Only a few factors contribute to an individual's tastes and preferences such as type, price and appearance of the product. For this reason, user ratings form a low-rank column space and the rating matrix is considered low-rank.

It is also possible to perform localization within a network [26] using matrix completion for applications such as a network of wireless sensors randomly dispersed in an area for environmental monitoring (temperature, pressure, humidity), healthcare or surveillance. It is of great importance to know where the data is collected in order to analyze it properly. For this purpose, a distance matrix is considered. Its observed entries are the distance estimates between the sensors that are near enough. Then the true distance matrix is estimated, its rank is at most $d+2$ in the d-dimensional space ($d = 2$ or $d = 3$).

Another application is phase retrieval [6] where the aim is to recover a signal from the observation of the amplitude which is of great interest in X-ray crystallography and quantum mechanics, i.e. applications where only the amplitude of the Fourier transform is measured. In this case, we consider a matrix of rank 1, $M = mm^H$, composed of the unknown signal of the time domain $m = [m_0 \dots m_{n-1}]$, acquired as the measured magnitude of the Fourier transform.

Matrix completion can be used for image compression and restoration. The entries in the matrix correspond to the value of each pixel in the image. Some

pixels are considered clean (uncontaminated) and constitute the set of observed entries of the matrix. An original image can therefore be recovered by completing the low-rank matrix.

Numerous other applications exist in signal processing [22], system identification [24], multi-task learning [1] and so on.

The purpose of this master thesis is not to complete matrices for a particular application, nor the completion itself. The aim is to determine whether completing the matrix is feasible or not, according to certain criteria.

2.2.3 Existing results

This section is based on the work of *Nguyen et al.* [27] who collected all the techniques used to solve the matrix completion problem (2.1). We also use the article of *Candès and Recht* [7] on exact matrix completion.

There are two main types of algorithms, those where the value of the rank is not available and those that use this information. All techniques are listed in the Table 2.1 and briefly described below.

Table 2.1: Matrix completion techniques.

When the rank is known	**When the rank is unknown**
I. Rank minimization technique II. Nuclear norm minimization (NNM) 1. NNM via convex optimization 2. Singular value thresholding (SVT) 3. Iteratively reweighted least squares (IRLS) minimization	I. Frobenius norm minimization (FNM) 1. Heuristic greedy algorithm 2. Alternating minimization technique 3. Optimization over smooth Riemannian manifold 4. Truncated NNM

The easiest case for solving the rank minimization problem (2.1) is when we consider to have $rank(A) = 1$. In this case, the solution can be found by combinatorial search because any pair of columns of the matrix A are linearly dependent and we have a system of equations $a_i = c_{ij}a_j$ for some $c_{ij} \in \mathbb{R}$. If there is no solution we consider that $rank(A) = 2$ and the system of equations is $a_i = c_{ij}a_j + c_{ik}a_k$.

The rank is increased progressively until a solution is found. This technique is not feasible in most scenarios because of its exponential complexity.

One technique consists in replacing the non-convex objective function by a convex one using nuclear norm, $||X||_*$, the sum of the singular values of X. The main advantage is that there are many efficient solver for convex optimization. Another way is to put a regularization term into the objective function which becomes $\tau||X||_* + \frac{1}{2}||X||_F^2$, this is the singular value thresholding technique and it allows to alleviate the computation when the matrix is large. The last technique based on nuclear norm is the minimization by iteratively reweighted least squares. It is computationally efficient and simple and the objective function is replaced with $||W^{\frac{1}{2}}X||_F^2$ where $W = (XX^T)^{-\frac{1}{2}}$.

When we have information on the rank, either its exact value or its maximum value is known. The problem (2.1) is therefore formulated as a Frobenius norm minimization (FNM) problem:

$$\begin{array}{ll} \min\limits_{X} & \frac{1}{2}||S_\Omega(A) - S_\Omega(X)||_F^2 \\ \text{s.t.} & rank(X) \leq d \end{array} \tag{2.2}$$

The two main advantages of this technique is that the problem can be easily considered in noisy scenario and that the objective function is differentiable. In consequence, many gradient-based optimization solvers can be used.

The FNM problem can be solved by using greedy algorithm such as atomic decomposition for minimum rank approximation. We can also use alternating minimization techniques, in this case we consider that the low-rank matrix $A \in \mathbb{R}^{n_1 \times n_1}$ can be written as $A = P^T Q$ where $P \in \mathbb{R}^{d \times n_1}$, $Q \in \mathbb{R}^{d \times n_2}$ and d is the rank of A. The solution is found by updating P and Q alternately. Yet another way to solve problem (2.2) is by truncated NNM. It is a variation of the NNM techniques when the rank is known. In this case, we consider as objective function only the $n - d$ smallest singular values, $||X||_d = \sum_{i=d+1}^{n} \sigma_i(X)$.

These techniques can be simplified when the matrix has a specific structure, e.g. a positive semidefinite matrix, a distance matrix, etc.

A final important remark is about the conditions on the matrix and on the mask. The first one is the incoherence of the matrix: the singular vectors of the matrix A must be sufficiently spread out, in order to minimize the number of observations needed to recover a low-rank matrix. One might not recover a low-rank matrix if its nonzero entries are concentrated in a certain area, e.g. in the upper corner. The second condition is the sparsity of the observed entries: only a small number of

observed entries is needed to recover a low-rank matrix and there is a lower bound on this number in order to have exact recovery. This condition states that the mask Ω must be a set of cardinality m sampled uniformly randomly. For more details on these conditions see [27] and [7].

When considering a given mask, i.e. a deterministic one, one is not sure to meet these two properties. It is therefore a challenge to be able to determine whether a given mask allows the completion of the matrix or not.

Liu et al. addressed the case of deterministic sampling of the mask in [23]. They propose two deterministic conditions, isomeric condition and relative well-conditionedness, for guaranteeing an arbitrary matrix to be recoverable from a sampling of its entries.

In view of the necessary conditions (both in random and deterministic sampling cases), one may ask whether for a given mask, not necessarily random, it is possible to complete the matrix. One can also ask what technique could be used to answer this question.

2.3 Matrix completion using rigidity theory of graphs

Graph rigidity and matrix completion are used in very different application areas. It is therefore not surprising that there are not yet many results using rigidity theory to solve the completion problem. This section addresses this issue. The matrix completion problem is very large. It is necessary to narrow-down its scope in order to compare it with rigidity. This comparative analysis is made in a second step before summarizing the existing results in a last step.

2.3.1 Specific completion problem

In order to make a parallel with rigidity, we consider a square low-rank matrix $A \in \mathbb{R}^{n\times n}$ and the specific case where the low-rank matrix can be written as $A = P^T P$ with $P \in \mathbb{R}^{d\times n}$. If the matrix P is considered as the collection of n column vectors $p_i \in \mathbb{R}^d$, we have the following equality: $p_i^T p_j = a_{ij}$ for all known input a_{ij}. The matrix A is symmetric because $p_i^T p_j = p_j^T p_i$.

$$P = \begin{bmatrix} p_1 & p_2 & \dots & p_n \end{bmatrix} \quad \text{and} \quad A = \begin{bmatrix} p_1^T p_1 & p_1^T p_2 & \dots & p_1^T p_n \\ p_2^T p_1 & p_2^T p_2 & \dots & p_2^T p_n \\ \vdots & \vdots & \ddots & \vdots \\ p_n^T p_1 & p_n^T p_2 & \dots & p_n^T p_n \end{bmatrix}$$

The problem can be linearized by considering small variation δ_i around the value p_i that preserve the scalar products for all $(i,j) \in \Omega$ in order to preserve the known entries of the matrix A. To this end, differentiation of the scalar product should be equal to zero, denote $\delta_i := \frac{d}{dt}p_i$:

$$p_i^T \frac{d}{dt}p_j + p_j^T \frac{d}{dt}p_i = p_i^T \delta_j + p_j^T \delta_i = 0 \quad \text{for all } (i,j) \in \Omega.$$

This corresponds to a system of m equations that can be rewritten in matrix form: $C_p\delta = 0$, where $C_p \in \mathbb{R}^{m \times nd}$ and $\delta \in \mathbb{R}^{nd}$. One line of this matrix is given by:

$$\begin{bmatrix} \dots & 0 & p_j^T & 0 & \dots & 0 & p_i^T & 0 & \dots \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} \dots & 0 & 2p_i^T & 0 & \dots & 0 & 0 & 0 & \dots \end{bmatrix}$$

depending if $i \neq j$ or $i = j$. The vector δ is given by:

$$\delta = \begin{pmatrix} \delta_1 \\ \delta_2 \\ \vdots \\ \delta_n \end{pmatrix}.$$

The matrix C_p is called the *completion matrix* and is sparse because it has only at most $2d$ nonzero elements per row. More precisely, if $i \neq j$ there are at most $2d$ nonzero elements and if $i = j$ there are at most d nonzero elements.

2.3.2 Similarities between the two problems

The Table 2.2 is showing the comparison between the linearized matrix completion problem and the infinitesimal rigidity problem.

Table 2.2: Comparison of the linearized matrix completion problem and the infinitesimal rigidity problem.

	Linearized Matrix Completion	**Infinitesimal Rigidity**
d	Rank of the low-rank matrix	Dimension of the problem
m	Number of known entries	Number of edges
n	Size of the matrix	Number of vertices
Vector p	$p_i \in \mathbb{R}^d$	$p_i \in \mathbb{R}^d$, position of vertices
Matrix	Completion matrix $C_p \in \mathbb{R}^{m \times dn}$ $\begin{bmatrix} \dots & 0 & p_j^T & 0 & \dots & 0 & p_i^T & 0 & \dots \end{bmatrix}$ or $\begin{bmatrix} \dots & 0 & 2p_i^T & 0 & \dots & 0 & 0 & 0 & \dots \end{bmatrix}$	Rigidity matrix $R_{G,p} \in \mathbb{R}^{m \times dn}$ $\begin{bmatrix} \dots & 0 & (p_i - p_j)^T & 0 & \dots \\ \dots & 0 & (p_j - p_i)^T & 0 & \dots \end{bmatrix}$
δ	Small variations around p	Infinitesimal displacements of p

We observe many similarities between the two problems, which leads us believe that rigidity theory could be used to solve the specific matrix completion problem described in Section 2.3.1.

2.3.3 Existing results

The first to address the issue of matrix completion through rigidity theory were *Singer and Cucuringu* in their article "Uniqueness of low-rank matrix completion by rigidity theory" [29]. Indeed, they observed that the scalar products in matrix completion can play the role of the distances in rigidity theory.

Later, others built on *Singer and Cucuringu*'s observations and findings to propose new results. All these findings are briefly described in this section.

From Singer and Cucuringu

Singer and Cucuringu, in [29], proposed randomized algorithms that verify the necessary and sufficient conditions for local completion and that verify the sufficient conditions for global completion. They were interested in the case of the completion of positive semidefinite matrices, called Gram matrices, as well as in the more general case of the completion of rectangular matrices.

In the case of the $n \times n$ rank-d positive semidefinite Gram matrix with some entries missing, we consider the following matrix, $A = P^T P$ where $P = \begin{bmatrix} p_1 & p_2 & \dots & p_n \end{bmatrix} \in \mathbb{R}^{d \times n}$. So each known entry is given by the scalar product $a_{ij} = p_i^T p_j$. The set of observed entries of A defines the graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, the number of observed entries. There is an edge $(i, j) \in E$ if and only if a_{ij} is observed. *Singer and Cucuringu* introduced the *completion matrix*, C_p. It allows to write in matrix notation the following system of equations:

$$p_i^T \delta_j + p_j^T \delta_i = 0 \quad \Leftrightarrow \quad C_p \delta = 0.$$

This system of m equations corresponds to the small variations δ_i around the p_i that preserve the scalar products $a_{ij} = p_i^T p_j$. The completion matrix, of size $m \times dn$, is sparse and its rank helps to determine if a unique completion of the Gram matrix A is possible. If

$$\dim \ker(C_p) = dn - \operatorname{rank}(C_p) > \frac{d(d-1)}{2},$$

then there exist non-trivial solutions to the system of equations, which means that there are non-trivial infinitesimal motions that preserve the scalar product and the matrix is **not locally completable**. The last observation made by *Singer and Cucuringu* is that almost all the completion matrices of a given graph have the same rank and thus it does **not depend on the realization**. We can thus construct a completion matrix from a random realization and its rank will determine the completability of the Gram matrix. They proposed a first randomized algorithm for testing local completion which consists in checking the existence of non-trivial infinitesimal motions δ. If a non-trivial infinitesimal motion exists then G cannot

be locally completable, otherwise G is locally completable. For global completion, we must check that the only deformations that preserve the scalar products are the trivial orthogonal transformations. It will give us a **unique completion**. Indeed, local completion allows a finite number of different completions because it does not exclude the possibility of having a non-trivial discontinuous deformation that satisfies the scalar product constraints. *Singer and Cucuringu* introduced the *completion stress*, ω, which is an assignment of weights ω_{ij} on the edges of the graph such that

$$\sum_{j:(i,j)\in E} \omega_{ij} p_j = 0 \quad \forall i \in V.$$

The completion stress therefore depends on the realization and satisfies $C_p^T \omega = 0$. We can construct the stress matrix, Ω, as a $n \times n$ symmetric matrix where $\Omega_{ij} = \omega_{ij}$ if $(i,j) \in E$ and $\Omega_{ij} = 0$ if $(i,j) \notin E$. They only succeeded in proposing an algorithm to determine sufficient condition for global completion by checking if the kernel of Ω contains vectors which are not linear combinations of the d coordinate vectors. If no such other kernel vectors exist (i.e. if $\dim \ker(\Omega) = d$) then G is **globally completable**; otherwise global completion of G is undecided. *Singer and Cucuringu* commented that when the diagonal entries of the Gram matrix are known, then both local and global completion problems are equivalent to the standard rigidity and global rigidity problem.

In the case of a $n_1 \times n_2$ rectangular low rank matrix, $A = P^T Q$ where $P \in \mathbb{R}^{d \times n_1}$ and $Q \in \mathbb{R}^{d \times n_2}$ and are given by $P = \begin{bmatrix} p_1 & p_2 & \dots & p_{n_1} \end{bmatrix}$ and $Q = \begin{bmatrix} q_1 & q_2 & \dots & q_{n_2} \end{bmatrix}$. The entries of A are given by

$$a_{ij} = p_i^T q_j,$$

and its observed entries define a bipartite graph $G = (V, E)$ with $|V| = n_1 + n_2$ and $|E| = m$, the number of observed entries. The reasoning is similar to the Gram matrix case. Randomized algorithms can be obtained for testing local completion as well for testing sufficient conditions for global completion of rectangular matrices.

To summarize the results obtained by applying the algorithms given by *Singer and Cucuringu*, we end up with three categories of matrices with missing entries. The **first class** contains matrices that are not globally completable, i.e. those that did not pass the local completion test. The **second class** contains globally completable matrices, i.e. those that passed both the local and global completion tests. The **third class** contains matrices that passed the local completion test but did not pass the global completion test. Since the global completion test only checks a sufficient condition, it is not known whether these matrices are globally completable or not.

What interests us more than randomized algorithms is the combinatorial characterization of locally/globally completable graphs. *Singer and Cucuringu* found

conditions for local and global completion in one dimension [29]. The results are set out in the propositions below.

Proposition 2.3.1. *A graph G is minimally Gram locally completable in one dimension if and only if each connected component of G is a tree plus one edge that contains an odd cycle.*

Proposition 2.3.2. *A graph is minimally Gram globally completable in one dimension if and only if it is a tree plus an edge that forms an odd cycle.*

Proposition 2.3.3. *A rectangular graph is minimally locally completable in one dimension if and only if it is a forest (a disjoint union of trees, or equivalently, a $(1,1)$-tight sparse bipartite graph).*

Proposition 2.3.4. *A rectangular graph is minimally globally completable in one dimension if and only if it is a tree.*

From the others

Jackson et al. also proposed combinatorial characterization of local and global completability in [14]. They focused on special families of graphs (cluster graphs, planar bipartite graph). Their results are based on observations that certain graph operations preserve local or global completability, as well as on a further connection between rigidity and completability. Finally they also proved that a rank condition on the completability stress matrix of a graph is a sufficient condition for global completability.

They built on some of Singer and Cucuringu's results to find out **under which circumstances completion can be unique**. Firstly, they were interested in the case of the completion of a low-rank Gram matrix and they introduced the definition of *c-independence*, a framework (G,p) is c-independent if rank $C_p = |E|$, where C_p is the completion matrix. They also recalled that a realization $p : V \to \mathbb{R}^d$ is called generic if the set of coordinates of p is algebraically independent over the rational field and thus the rank of C_p is the same for all generic realizations of G. Since it has been proven that infinitesimal completion is a necessary condition for local completion, and that it is equivalent when the framework (G,p) is generic, the graph G is locally completable or c-independent in $\mathbb{R}^d$ if some generic realization of G in $\mathbb{R}^d$ is locally completable or c-independent. They concluded that in the **generic case**, the **local uniqueness** of a completion of a Gram matrix with missing entries depends only on the underlying graph G, which is determined by the positions on the known entries in the matrix. For the case of **global completion**, we cannot say that the uniqueness of the completion depends only on the positions of the known entries when $d \geq 2$. We can say that a graph G

is globally completable in $\mathbb{R}^d$ if every generic realization of G in $\mathbb{R}^d$ is globally completable.

In addition to the theory on rigidity *Jackson et al.* used propositions linking completion and rigidity. However, they were interested in the specific cases of cone graphs and cluster graphs. They therefore did not provide a more general case which is of less interest to us.

In another paper [15], *Jackson et al.* looked at candidates for completion and they provided new results on global completion based on geometric observations. They also provided sufficient conditions for local and global unique low-rank completion of a partially filled $n \times n$ matrix in terms of either the minimum number of known entries per row, or the total number of known entries, as functions of n and d.

Firstly, they introduced the following definition, a graph G is *vertex redundantly completable* if $G - v$ is locally completable for all $v \in V$. Using this definition, they gave and proved the following theorem on global completion.

Theorem 2.3.5. *Let $G = (V, E)$ be a vertex redundantly completable graph in $\mathbb{R}^d$ with $|V| \geq d + 1$ for some $d \geq 2$. Then G is globally completable in $\mathbb{R}^d$.*

This implies that a partially filled positive semidefinite matrix of order n is globally rank d completable if every $n - 1$ principal submatrix is locally rank d completable.

Then, they looked for conditions on the minimum degree (i.e. the minimum number of known entries per row), $\delta(G)$, of the graph G for local and global completion. They gave the following theorems.

Theorem 2.3.6. *Let $G = (V, E)$ be a semi-simple graph on n vertices with $\delta(G) \geq \lceil n/2 \rceil + 2$. Then G is locally completable in $\mathbb{R}^2$.*

Theorem 2.3.7. *Let $G = (V, E)$ be a semi-simple graph on n vertices. Suppose that $\delta(G) \geq \lceil n/2 \rceil + 3$. Then G is globally completable in $\mathbb{R}^2$.*

Theorem 2.3.8. *For all $d \geq 1$ and $\epsilon > 0$ there exists an integer $N = N_{d,\epsilon}$ such that every semi-simple graph G on $n > N$ vertices with $\delta(G) \geq n(1 + \epsilon)/2$ is locally completable in $\mathbb{R}^d$.*

The other characteristic they investigated was the number of missing edges (i.e. the number of missing entries). They gave theorems on local and global completion for simple and semi-simple graphs. Here are the theorems for semi-simple graphs.

Theorem 2.3.9. *Let $G = (V, E)$ be a semi-simple graph on $n \geq d$ vertices. Suppose that G has at most $n - d$ missing edges. Then G is locally completable in $\mathbb{R}^d$.*

Theorem 2.3.10. *Let $G = (V, E)$ be a semi-simple graph on $n \geq d$ vertices. Suppose that G has at most $n - d - 1$ missing edges. Then G is globally completable in $\mathbb{R}^d$.*

These results imply that, if sufficient entries are known in each row/column of the given partially filled matrix (or if the number of unknown entries is sufficiently small), then, in the generic case, the completion is locally/globally unique.

We can also mention the work of *Stark* [30], *Krone and Kubjas* [18] and *Laurent and Varvitsiotis* [20] who were respectively interested in the case of a 2×2 block Gram matrix, a non-negative matrix and a semidefinite programming formulation. However, this is not what we are most interested in.

In conclusion, after going through the literature where people attack the uniqueness of local/global completability of the matrix by looking at the rigidity theory, we notice that none of them has directly used Laman's proofs. The question of whether it is possible to reproduce Laman's proofs in the field of matrix completion is therefore still open.

2.4 Contribution

In this thesis, we succeed in reproducing Laman's proofs in the case of matrix completion. This provides a complete characterization of the matrix completion problem in the case of a partially filled symmetric matrix of rank 2.

We also provide an algorithm for recognizing matrices that are completable using the equivalence of the completability of the matrix and the fact that the graph corresponding to this matrix has well distributed edges.

CHAPTER 3

GRAPH RIGIDITY THEORY

As developed in the state of the art in Chapter 2, the theory of rigidity is the basis of the theoretical developments in this master thesis. It is summarized in this chapter in order to have the necessary theory for the understanding of this thesis. The theoretical results used are those proposed by *Laman* in his paper "On Graphs and Rigidity of Plane Skeletal Structures" [19].

First, the problem is stated in general terms in Section 3.1. Reminder of graph theory is given as well as some useful definitions. Then the results of Laman are summarized in Section 3.2. The following are discussed: skeletal structures in the plane, the various existing displacements, rigidity criteria and a family of graphs called E-graphs which have good rigidity properties. At the end of this section, it is possible to determine whether a given graph is (minimally) rigid or flexible in $\mathbb{R}^2$.

No graph theoretical criterion such as Laman's has been found for higher dimensional rigidity. Other interesting results in rigidity theory are given in the state of the art in Section 2.1.

3.1 Problem statement

In this section, the basic notions of graph theory are first defined based on the course of *Blondel and Delvenne*, "Théorie et Algorithmique des Graphes" [5]. Then the definition of a skeletal structure is given as well as the condition to be rigid.

3.1.1 Reminder on graph theory

Definition 3.1.1. *A* graph *G consists of a duet (V, E), where:*

- *V is a finite set whose elements are called vertices;*
- *E is a finite set whose elements are called edges.*

The set of edges E is a subset of $\mathcal{P}(V)$, i.e. the set having as elements subsets of V containing exactly two elements of V. The graph G is noted $G(V, E)$.

Definition 3.1.2. *A* loop *is an edge incident to a single vertex. Two edges are said to be* multiple *if they are both incident to the same two vertices.*

Definition 3.1.3. *A* simple graph *is a graph without loop and without multiple edges. A* semi-simple graph *is a graph without multiple edges, it could have loops.*

Definition 3.1.4. *The* degree of a vertex *is the number of edges incident to it. Loops counting as double incident edges.*

Definition 3.1.5. *A* subgraph *of the graph $G(V, E)$ is a graph $G'(V', E')$ with:*

- $V' \subset V$;
- E' *is the restriction of* E *to* V', *noted* $E' = E(V')$.

For example, let $V = \{1, 2, 3, 4\}$ and $E = \{(1, 1), (1, 3), (2, 2), (2, 3), (2, 4), (3, 4)\}$. $G(V, E)$ is a graph and can be represented as below on the left. In this graph, the edges $(1, 1)$ and $(2, 2)$ are loops and the degrees of each vertex are $(3, 4, 3, 2)$ respectively. Let $V' = \{2, 3, 4\}$, then $E' = E(V') = \{(2, 2), (2, 3), (2, 4), (3, 4)\}$ and the graph $G'(V', E')$ is a subgraph of $G(V, E)$, represented below on the right.

In the rigidity theory, we do not take loops and multiple edges into account because they do not influence the rigidity property of graphs.

3.1.2 Definition of skeletal structures

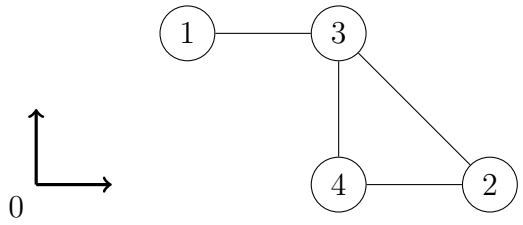
Definition 3.1.6. *A* realization *p in $\mathbb{R}^d$ of a graph $G(V, E)$ with $n = |V|$ vertices and $m = |E|$ edges is a mapping $p : \{1, ..., n\} \to \{p_1, ..., p_n\}$ where $p_i \in \mathbb{R}^d$, the d-dimensional Cartesian space. p_i is called the position of the node i and the length of the edge (i, j) is given by $d_{ij} = ||p_i - p_j||$ where $|| \cdot ||$ is the Euclidean norm.*

The d-dimensional Cartesian space is the d-dimensional Euclidean space (finite-dimensional scalar product space over the real numbers) described by Cartesian coordinates. The Euclidean norm is given by $||x|| = ||x||_2 := \sqrt{x_1^2 + x_2^2 + ... + x_d^2}$ where $x = (x_1, x_2, ..., x_d)$ is a vector on the d-dimensional Cartesian plane $\mathbb{R}^d$.

The realization is denoted $p = (p_1, p_2, ..., p_n)$ and is thus a finite list of n elements of $\mathbb{R}^d$, called a n-tuple of d-tuples. From this, we can say that the dimension

of p is dn, the dimension of the space, times the number of nodes, $p \in \mathbb{R}^{dn}$.

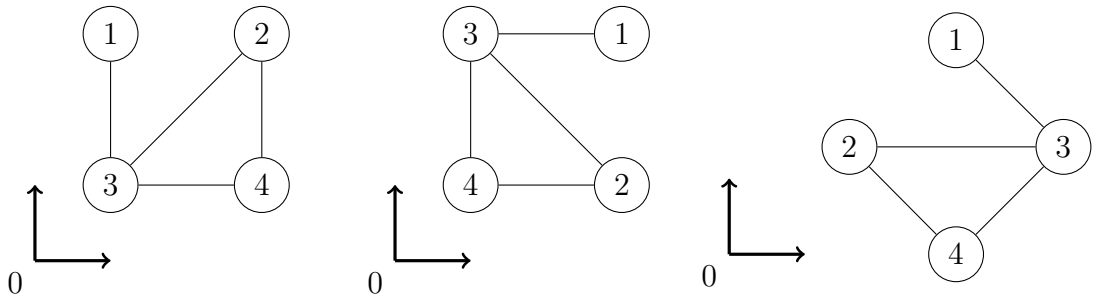
For example, a realization in the Cartesian plane $\mathbb{R}^2$ ($d = 2$) of the graph $G(V, E)$ with $V = \{1, 2, 3, 4\}$ and $E = \{(1, 3), (2, 3), (2, 4), (3, 4)\}$ is given by $p = ((2, 2), (6, 0), (4, 2), (4, 0))$ which is represented below.



Definition 3.1.7. *An* Euclidean transformation *is a geometric transformation of space that preserves the distance between each pair of points. These transformations include rotations, translations, reflections or any combination of these.*

Definition 3.1.8. *Two realizations are* congruent *if they are images of each other by Euclidean transformations.*

Three congruent realizations of the graph G are given below. The graph is $G(V, E)$ with $V = \{1, 2, 3, 4\}$ and $E = \{(1, 3), (2, 3), (2, 4), (3, 4)\}$. The realizations are respectively $p^1 = ((1, 3), (3, 3), (1, 1), (3, 1))$, $p^2 = ((3, 3), (3, 1), (1, 3), (1, 1))$ and $p^3 = (3, 2\sqrt{2}), (3 - \sqrt{2}, \sqrt{2}), (3 + \sqrt{2}, \sqrt{2}), (3, 0))$. These realizations are congruent because p^2 is obtained by a rotation of $-90°$ of center $(2, 2)$ from p^1 and p^3 is obtained from p^1 by a combination of a rotation of $-45°$ of center $(2, 2)$, a reflection with respect to the axis passing through nodes 1 and 4 and a translation of $+1$ in x and $+(1 - \sqrt{2})$ in y.



Definition 3.1.9. *A* skeletal structure *is the association of a graph G with a realization p in the Cartesian space $\mathbb{R}^d$ for each of the vertices of the graph that satisfies $p_i \neq p_j$ if (i, j) is an edge of the graph. It is denoted (G, p).*

Two nodes linked by an edge cannot be found at the same place in $\mathbb{R}^d$ to form a skeletal structure.

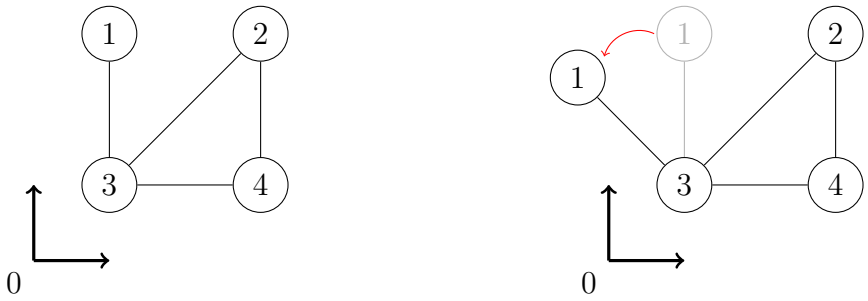
3.1.3 Rigidity property of skeletal structures

The rigidity problem consists in showing whether a given graph $G(V, E)$ is rigid or flexible. If it is rigid, it can be either locally rigid or globally rigid.

Definition 3.1.10. *A skeletal structure* (G, p) *is* rigid *in* $\mathbb{R}^d$ *if every continuous path* $\Gamma \in \mathbb{R}^{dn}$ *that preserve the edge lengths of* (G, p) *and begin at* p *terminates at a position* $q \in \mathbb{R}^{dn}$ *which is congruent to* p.

A continuous path between two points of the space $\mathbb{R}^d$ is a continuous function $\gamma_i : \mathbb{R}^d \to \mathbb{R}^d : p_i \to q_i$. The notation Γ contains all the continuous paths between each node of the two realizations, $\Gamma = (\gamma_1, ..., \gamma_n) : \mathbb{R}^{dn} \to \mathbb{R}^{dn} : (p_1, ..., p_n) \to (q_1, ..., q_n)$.

For example, the graph $G(V, E)$ given as an example so far is flexible because we can find a continuous path preserving the length of its edges that ends in a realization q not congruent to p, as illustrated below. On the left, the realization is $p = ((1, 3), (3, 3), (1, 1), (3, 1))$ and on the right it is $q = ((1 - \sqrt{2}, 1 + \sqrt{2}), (3, 3), (1, 1), (3, 1))$.



Definition 3.1.11. *A* generic property *of a graph is a property that is true for almost all realizations of the graph.*

It is important to note that rigidity is a property defined for realizations of graphs. However, if a graph is rigid for one of its realizations, it is rigid for almost all its realizations. We say that rigidity is a generic property.

Another way of approaching rigidity is through the notion of infinitesimal rigidity which is a kind of first order approach. Moreover, it is a stronger approach because any infinitesimally rigid skeletal structure is rigid [11]. Infinitesimal rigidity is described by *Singer and Cucuringu* [29]. Consider a motion of the realization $p \in \mathbb{R}^{dn}$ with $p_i(t)$ being the motion vector of the vertex i at time t. Any infinitesimal motion that instantaneously preserves the lengths of the edges must

satisfy $\frac{d}{dt}||p_i(t) - p_j(t)||^2 = 0$ for all $(i,j) \in E$. Denote the infinitesimal motion of the vertex i, $\dot{p}_i(t) = \frac{d}{dt}p_i(t)$, by $\delta_i \in \mathbb{R}^n$ and $\delta \in \mathbb{R}^{dn}$ such that:

$$\delta = \begin{pmatrix} \delta_1 \\ \delta_2 \\ \vdots \\ \delta_n \end{pmatrix},$$

then, for each edge the following equation holds:

$$(p_i - p_j)^T(\delta_i - \delta_j) = 0, \quad \text{for all } (i,j) \in E.$$

Remembering that there are m edges in the graph, this gives a system of m linear equations with dn unknowns that can be written in matrix form: $R_{G,p}\delta = 0$, where the matrix $R_{G,p} \in \mathbb{R}^{m \times dn}$ is called the *rigidity matrix* and is sparse because it has only at most $2d$ nonzero elements per row.

For example, the skeletal structure described by the graph $G(V,E)$ with $V = \{1,2,3,4\}$ and $E = \{(1,3),(2,3),(2,4),(3,4)\}$ and the realization $p = ((1,3),(3,3),(1,1),(3,1))$ has the following rigidity matrix:

$$\begin{pmatrix} (p_1 - p_3)^T & 0 & (p_3 - p_1)^T & 0 \\ 0 & (p_2 - p_3)^T & (p_3 - p_2)^T & 0 \\ 0 & (p_2 - p_4)^T & 0 & (p_4 - p_2)^T \\ 0 & 0 & (p_3 - p_4)^T & (p_4 - p_3)^T \end{pmatrix} = \begin{pmatrix} 0 & 2 & 0 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & 2 & 2 & -2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & -2 \\ 0 & 0 & 0 & 0 & -2 & 0 & 2 & 0 \end{pmatrix}$$

Definition 3.1.12. *A skeletal structure (G,p) is* infinitesimally rigid *in $\mathbb{R}^d$ if all infinitesimal motions that preserve the edge lengths of (G,p) are trivial.*

Definition 3.1.13. *A* trivial infinitesimal motion *is a motion that is a combination of some orthogonal transformations and some translations.*

3.2 Rigidity of planar skeletal structures

Laman, in [19], gave criteria for the rigidity of skeletal structures in two dimensions, called plane skeletal structures.

Before giving the definition of a rigid plane skeletal structure, a set of displacements is defined: length-preserving, infinitesimal, small, admissible, trivial, etc. All these definitions can be found in the second section of [19]. Two of these definitions are presented in simplified form below:

Definition 3.2.1. *A* small displacement *is* admissible *if it preserves the distance between the points connected by an edge.*

Definition 3.2.2. *An* infinitesimal displacement δ *of* (G, p) *is* trivial *if it consists of Euclidean transformations in the plane* $\mathbb{R}^2$*.*

The definition of rigid plane skeletal structure is then given.

Definition 3.2.3. *A plane skeletal structure is* rigid *if every admissible small displacement has a trivial infinitesimal displacement.*

It is therefore necessary to be able to distinguish which infinitesimal displacements are trivial. To do so, some propositions are given in Section 3.2.1. The Section 3.2.2 gives some results about rigidity and an interesting family of graphs called E-graphs is presented in Section 3.2.3.

3.2.1 Trivial and non-trivial infinitesimal displacements

Thanks to other propositions and lemmas, we have the following proposition which gives a characteristic of a trivial infinitesimal displacement. First, in general, the point x is written $x = (\xi, \eta)$, so let the position of the node i be $p_i = (\xi_i, \eta_i)$. We also define the notation $\langle u, v \rangle$ as the scalar product between the vectors u and v.

Proposition 3.2.4. *An infinitesimal displacement* δ *of* (G, p) *is trivial if and only if the matrix*

$$\begin{pmatrix} \xi - \xi_1 & \eta - \eta_1 & \langle \delta_1, p_1 - x \rangle \\ \xi - \xi_2 & \eta - \eta_2 & \langle \delta_2, p_2 - x \rangle \\ \vdots & \vdots & \vdots \\ \xi - \xi_n & \eta - \eta_n & \langle \delta_n, p_n - x \rangle \end{pmatrix}$$

has rank 2 *identically in* x*. Furthermore, the trivial infinitesimal displacements constitute a* 3*-dimensional subspace of the* $2n$*-dimensional vector space of infinitesimal displacements.*

This matrix is linked to the system of equations of the rigidity matrix. Indeed, considering a point x that is added in the graph and that there is an edge between

x and all the other node in V $(1, ..., n)$, then we have the following:

$$\begin{aligned}
& (x-p_i)^T(\delta_x-\delta_i)=0 \quad \text{for all } i\in V \\
\Leftrightarrow \quad & (x-p_i)^T\delta_x+(p_i-x)^T\delta_i=0 \quad \text{for all } i\in V \\
\Leftrightarrow \quad & \begin{pmatrix} (x-p_1)^T & (p_1-x)^T\delta_1 \\ (x-p_2)^T & (p_2-x)^T\delta_2 \\ \vdots & \vdots \\ (x-p_n)^T & (p_n-x)^T\delta_n \end{pmatrix} \begin{pmatrix} \delta_x \\ 1 \end{pmatrix} = 0 \\
\Leftrightarrow \quad & \begin{pmatrix} \xi-\xi_1 & \eta-\eta_1 & \langle\delta_1, p_1-x\rangle \\ \xi-\xi_2 & \eta-\eta_2 & \langle\delta_2, p_2-x\rangle \\ \vdots & \vdots & \vdots \\ \xi-\xi_n & \eta-\eta_n & \langle\delta_n, p_n-x\rangle \end{pmatrix} \begin{pmatrix} \delta_{x,1} \\ \delta_{x,2} \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}
\end{aligned}$$

This matrix is the same as the one in the above proposition and it allows to find the infinitesimal displacement linked to the new node x.

3.2.2 Criteria for rigidity

This section gives some results about rigidity. The first theorem gives statements that are equivalent.

Theorem 3.2.5. *The following assertions are equivalent:*
A. The plane skeletal structure (G, p) is rigid.
B. If $\langle p_i - p_j, \delta_i - \delta_j\rangle = 0$ for all $(i, j) \in E$, then δ is trivial.
C. If $\langle p_i - p_j, \delta_i - \delta_j\rangle = 0$ for all $(i, j) \in E$,
then $\langle p_i - p_j, \delta_i - \delta_j\rangle = 0$ for all $(i, j) \in \mathcal{P}(V)$.
D. The rigidity matrix, i.e. the matrix of the system of equations:

$$\langle p_i - p_j, \delta_i - \delta_j\rangle = 0 \quad \text{for all } (i, j) \in E$$

has rank $2|V| - 3$.

From the last assertion, we can say that every rigid plane skeletal structure has a rigid substructure with $|V|$ vertices and $2|V| - 3$ edges.

We therefore have a new property for rigid plane skeletal structures with a certain number of edges.

Theorem 3.2.6. *Any rigid plane skeletal structure (G, p) with $|E| = 2|V| - 3$ edges has the following property:*
If $V' \subset V$ and $|V'| \geq 2$ then $|E \cap E(V')| \leq 2|V'| - 3$.

Next, two rigidity-preserving operations are described: vertex addition and edge splitting.

To add a vertex a to a rigid plane skeletal structure, it must be connected by an edge to two distinct vertices b_1 and b_2 of the graph such that p_a is non-collinear with p_{b_1} and p_{b_2}, see Figure 3.1

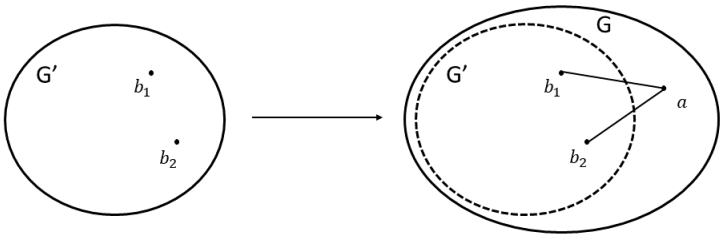


Figure 3.1: Vertex addition operation: adding a new node a and adding the edges (a, b_1) and (a, b_2).

To split an edge from a rigid plane skeletal structure, e.g. the edge (b_1, b_2), we need to delete this edge, add a vertex a and connect it to b_1 and b_2 and to a third vertex b_3, see Figure 3.2.

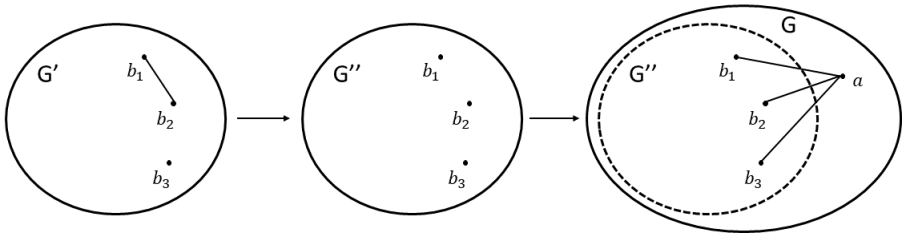


Figure 3.2: Edge splitting operation: deleting the edge (b_1, b_2), adding a node a and adding the edges (a, b_1), (a, b_2) and (a, b_3).

The new plane skeletal structure formed by adding a vertex or splitting an edge from a rigid plane skeletal structure is also rigid.

Laman calls an E-graph a graph having $|V| \geq 3$, $|E| = 2|V| - 3$ and respecting the property given in Theorem 3.2.6. In summary, each rigid plane skeletal structure contains an E-graph.

We would now like to prove that every E-graph has rigid realizations, which is done in the next section.

3.2.3 A new family of rigid graphs: E-graphs

A first important observation is that an E-graph necessarily contains a vertex of degree 2 or a vertex of degree 3. Then, Laman found two operations preserving the property of an E-graph.

The first operation is that if a vertex of degree 2 is found in an E-graph and deleted, the resulting graph is also an E-graph and vice versa. This operation is called *inverse vertex addition* and is described in Figure 3.3.

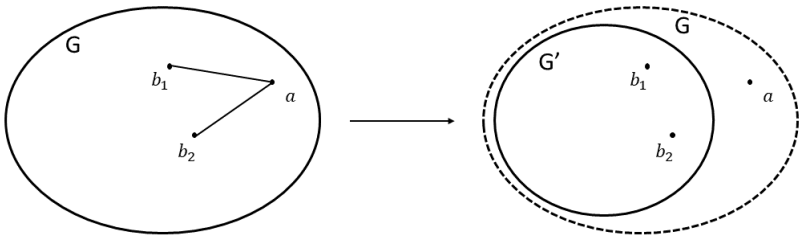


Figure 3.3: Inverse vertex addition operation: a has degree 2, deleting a and the edges (a, b_1) and (a, b_2).

The second operation is that if a vertex a of degree 3 is found in an E-graph, it can be deleted. Then by adding an edge between one of the three pairs of neighbors of a, the resulting graph is also an E-graph. This operation is called *inverse edge splitting* and is described in Figure 3.4.

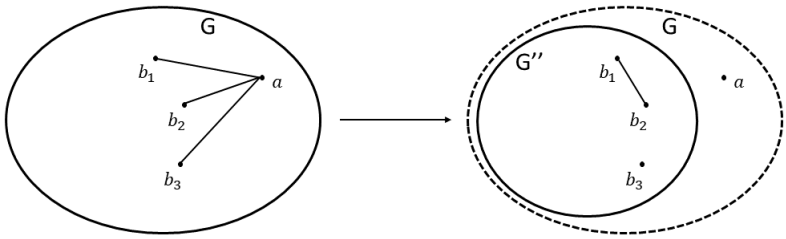


Figure 3.4: Inverse edge splitting operation: a has degree 3, deleting a and the edges (a, b_1), (a, b_2) and (a, b_3), adding the edge between one of the couples of neighbors of a, e.g. the edge (b_1, b_2).

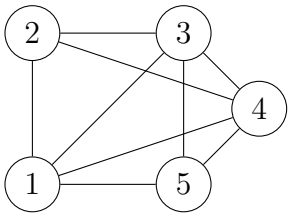
The last theorem can now be stated.

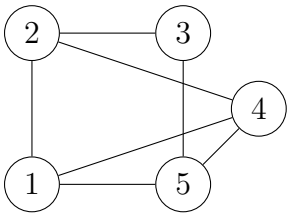
Theorem 3.2.7. *Every E-graph has rigid realizations.*

This can be proven inductively by starting from the triangular E-graph which has an obvious rigid realization and by using the rigidity-preserving operations and

the E-graph property-preserving operations.

In conclusion, if a given graph contains a E-subgraph on its vertex set then it is rigid. For example, consider the graph $G(V, E)$ with $V = \{1, 2, 3, 4, 5\}$ and $E = \{(1, 2), (1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (3, 4), (3, 5), (4, 5)\}$ that is represented below left. This graph has a subgraph $G'(V, E')$ with $E' = E \backslash \{(1, 3), (3, 4)\}$ which is an E-graph and is represented below right. The graph $G(V, E)$ is therefore rigid.





Now that we have laid down the theoretical foundations and contextualized what was useful about rigidity, we can tackle the problem of matrix completion. The next chapter takes Laman's proofs and adapts them to the problem of matrix completion.

CHAPTER 4

MATRIX COMPLETION FOR A GIVEN MASK

This chapter attempts to answer the question of whether a given partially filled matrix can be uniquely completed. It is based on *Laman*'s work on rigidity [19] which is put in parallel for matrix completion. The theory of rigidity on which this chapter is based on was summarized in Chapter 3. Since rigidity is fully characterized only in the plane, the problem of matrix completion is addressed in the Cartesian plane. This means that we consider a partially filled matrix of rank two, rank one being a trivial problem. Moreover, the matrices are considered as symmetric.

The basic definitions are given in Section 4.1. Trivial infinitesimal displacements are defined in Section 4.2 and completion criteria are given in Section 4.3. Some examples to illustrate the concepts of completion are provided in Section 4.4. Moreover, the case of C-graphs, a set of graphs with a condition on the number and distribution of their edges, is of great interest and is described in Section 4.5. Finally, the final completion theorem for an arbitrary matrix is stated in Section 4.6.

At the end of this chapter, it will be possible to determine whether a given partially filled symmetric matrix is uniquely completable. If it is, this means that its missing entries can be uniquely found so that the matrix is of rank two.

4.1 Definitions

This section defines all the concepts used in this chapter. We can find notably the definition of a plane structure, the possible displacements to be made on the vertices of this structure and the definition of completable structure.

Definition 4.1.1. *A* graph G corresponding to a partially filled symmetric matrix $A \in \mathbb{R}^{n\times n}$ *consists of a finite set* $V = \{1, 2, ..., n\}$ *(set of vertices of* G*) and a finite set* E *(set of edges of* G*). Notation:* $G(V, E)$*. There is an edge between the vertices* i *and* j *if the entry* (i, j) *of the matrix* A *is known. Self-loops are therefore allowed but no double edges. Thus, if* A *has* m *known entries, there is* m *edges in the corresponding graph* G*.*

The set of edges of the graph, E, corresponds to the mask of the matrix, Ω, the set of indices of the known entries.

Definition 4.1.2. *A* realization *p in the Cartesian plane $\mathbb{R}^2$ of a graph $G(V, E)$ is a mapping $p = (p_1, p_2, ..., p_n) : V \to \mathbb{R}^{2n} : \{1, ..., n\} \to \{p_1, ..., p_n\}$ where $p_i \in \mathbb{R}^2$, assigning a position to each vertex. For each edge (i, j) the associated scalar product is $s_{ij} = \langle p_i, p_j \rangle = p_i^T p_j$ with $p_i \in \mathbb{R}^2$ and $p_j \in \mathbb{R}^2$.*

Definition 4.1.3. *A* plane structure *is the association of a graph G with a realization $p \in \mathbb{R}^{2n}$ satisfying $\langle p_i, p_j \rangle \neq 0$ if $(i, j) \in E$. It is denoted (G, p).*

Each known entry of the matrix is given by the following scalar product, $a_{ij} = p_i^T p_j$ for every $(i, j) \in \Omega$, Ω being the mask of the matrix. Thus, for each edge in the graph G corresponding to the partially filled matrix A, the scalar product must be preserved. We are therefore interested in the definition of displacements preserving the scalar product.

Definition 4.1.4. *A* scalar product-preserving displacement *of a plane structure (G, p) consists in:*
a) A segment $[\beta, \gamma]$ of real numbers with $\beta \in \mathbb{R}^-$ and $\gamma \in \mathbb{R}^+$;
b) for every $i \in V$ and for every $\tau \in [\beta, \gamma]$, a point $p_i(\tau) \in \mathbb{R}^2$, satisfying the following conditions:

1. *the function $p_i(\tau)$ is differentiable for every i;*
2. *$p_i(0) = p_i$ for every i;*
3. *$(G, p(\tau))$ is a plane structure for every τ;*
4. *$\langle p_i(\tau), p_j(\tau) \rangle = \langle p_i, p_j \rangle$ for every τ and every $(i, j) \in E$.*

In the plane, only the rotation around the origin preserves the scalar product. The following definition describes this motion.

Definition 4.1.5. *A* rotation around the origin *of $\mathbb{R}^2$ is a function π of time τ and of $x \in \mathbb{R}^2$, differentiable in τ and taking values $\pi_x(\tau)$ in $\mathbb{R}^2$ in such a way that for every x: $\pi_x(0) = x$ and for every τ and for every x_1 and x_2: $||\pi_{x_1}(\tau) - \pi_{x_2}(\tau)|| = ||x_1 - x_2||$ and $||\pi_x(\tau)|| = ||x||$.*

The notation $||x||$ corresponds to the Euclidean norm and is equal to the length of the vector x.

Definition 4.1.6. *A scalar product-preserving displacement is* trivial *if it results from some rotation around the origin of $\mathbb{R}^2$ in itself, i.e. if there is a π satisfying $\pi(p(\tau)) = p(\tau)$.*

The trivial displacement for the problem of completability is only the rotation around the origin. In Laman's problem of rigidity, the trivial displacements are two translations (along the two directions of the plane) and one rotation (not necessarily around the origin). To only have rotation around the origin in the completion problem, only the condition $||\pi_x(\tau)|| = ||x||$ is added for every x.

We are interested in small variation δ_i around the value p_i that preserves the scalar products for all $(i,j) \in \Omega$ in order to preserve the known entries of the matrix. These small variations can be considered by applying an infinitesimal displacement to each node position.

Definition 4.1.7. *An* infinitesimal displacement *of a plane structure is a map δ of V into the Cartesian plane $\mathbb{R}^2$.*

An infinitesimal displacement of a plane structure is the assignment (δ) of a vector (δ_i) to every position (p_i). This vector can therefore be considered as the velocity vector.

Definition 4.1.8. *A* small displacement *of a plane structure consists in:*

1. *An infinitesimal displacement δ;*
2. *a real number $\alpha > 0$;*
3. *for every real $\tau \in [-\alpha, \alpha]$, a realization $p(\tau) : V \to \mathbb{R}^2$ satisfying $p_i(\tau) = p_i + \tau \cdot \delta_i + o(\tau)$ for every $i \in V$.*

In the above definition, $o(\tau)$ has the usual meaning: $\lim_{\tau \to 0} \frac{o(\tau)}{\tau} = 0$.

Definition 4.1.9. *A small displacement is* admissible *if for every $(i,j) \in E$:*

$$\langle p_i(\tau), p_j(\tau) \rangle - \langle p_i, p_j \rangle = o(\tau).$$

Definition 4.1.10. *An* infinitesimal rotation around the origin *of $\mathbb{R}^2$ is a map $\psi : \mathbb{R}^2 \to \mathbb{R}^2$ satisfying $\langle \psi_{x_1} - \psi_{x_2}, x_1 - x_2 \rangle = 0$ for every pair x_1, x_2 of $\mathbb{R}^2$ and $\langle \psi_x, x \rangle = 0$ for every x of $\mathbb{R}^2$. Putting the two conditions together, the mapping satisfies $\langle \psi_{x_1}, x_2 \rangle + \langle \psi_{x_2}, x_1 \rangle = 0$ for every pair x_1, x_2 of $\mathbb{R}^2$.*

Definition 4.1.11. *An infinitesimal displacement δ of (G,p) is* trivial *if there exists an infinitesimal rotation around the origin ψ satisfying $\psi(p) = \delta$.*

Definition 4.1.12. *A plane structure (G,p) is* completable *if every admissible small displacement has a trivial infinitesimal displacement.*

In other words, the plane structure (G,p) is completable if any possible displacement that preserves the scalar product for every edge is a rotation around the origin.

4.2 Trivial and non-trivial infinitesimal displacements

This section gives propositions for determining whether an infinitesimal displacement is trivial or not. In a first time, Proposition 4.2.1 and Lemma 4.2.2 give general results useful to prove the Proposition 4.2.3. The latter gives a characterization of trivial infinitesimal displacements.

First of all, some notations are defined. Let $x_i = (\xi_i, \eta_i)$ be points of $\mathbb{R}^2$ and u_i be any vectors of $\mathbb{R}^2$ for $i = 0, 1, 2, ..., n$. The following matrix and determinant

$$\begin{pmatrix} \xi_1 & \eta_1 & \langle u_1, x\rangle \\ \xi_2 & \eta_2 & \langle u_2, x\rangle \\ \vdots & \vdots & \vdots \\ \xi_n & \eta_n & \langle u_n, x\rangle \end{pmatrix} \quad \text{and} \quad \begin{vmatrix} \xi_k & \eta_k & \langle u_k, x\rangle \\ \xi_l & \eta_l & \langle u_l, x\rangle \\ \xi_m & \eta_m & \langle u_m, x\rangle \end{vmatrix}$$

are respectively denoted by $(x_i \quad \langle u_i, x\rangle)_{i=1,2,...,n}$ and $|x_i \quad \langle u_i, x\rangle|_{i=k,l,m}$.

Proposition 4.2.1. *If $x_i = (\xi_i, \eta_i)$ ($i = 1, 2, 3$) are points of $\mathbb{R}^2$ such that $x_i \neq c \cdot x_j$ for every i and j ($c \in \mathbb{R}$) and if u_i are any three vectors of $\mathbb{R}^2$ then the following assertions are equivalent:*

A. $\langle x_i, u_j\rangle + \langle x_j, u_i\rangle = 0$ for every i and j;

B. $|x_i \quad \langle u_i, x\rangle|_{i=1,2,3} = 0$ identically in x.

Proof. Suppose that assertion A is true. The expression $|x_i \quad \langle u_i, x\rangle|_{i=1,2,3}$ can be considered as a straight line in the coordinates ξ, η of x. It can be rewritten as follows:

$$\begin{aligned} \begin{vmatrix} \xi_1 & \eta_1 & \langle u_1, x\rangle \\ \xi_2 & \eta_2 & \langle u_2, x\rangle \\ \xi_3 & \eta_3 & \langle u_3, x\rangle \end{vmatrix} &= (\eta_3\xi_2 - \eta_2\xi_3)\langle u_1, x\rangle + (\eta_1\xi_3 - \eta_3\xi_1)\langle u_2, x\rangle + (\eta_2\xi_1 - \eta_1\xi_2)\langle u_3, x\rangle \\ &= A\langle u_1, x\rangle + B\langle u_2, x\rangle + C\langle u_3, x\rangle \end{aligned}$$

where A, B and $C \in \mathbb{R}$ are nonzero because $x_i \neq c \cdot x_j$ for every i and j. It is possible to prove that this expression of a straight line is zero at the points $x = 0$ and $x = x_1 + x_2 + x_3$ and therefore vanishes identically in x. For $x = 0$, the result is trivial and for $x = x_1 + x_2 + x_3$ the expression becomes:

$$\begin{aligned} A\left[\langle u_1, x_1\rangle + \langle u_1, x_2\rangle + \langle u_1, x_3\rangle\right] + B\left[\langle u_2, x_1\rangle + \langle u_2, x_2\rangle + \langle u_2, x_3\rangle\right] \\ + C\left[\langle u_3, x_1\rangle + \langle u_3, x_2\rangle + \langle u_3, x_3\rangle\right] \\ = A\left[\langle u_1, x_1\rangle - \langle u_2, x_1\rangle - \langle u_3, x_1\rangle\right] + B\left[-\langle u_1, x_2\rangle + \langle u_2, x_2\rangle - \langle u_3, x_2\rangle\right] \\ + C\left[-\langle u_1, x_3\rangle - \langle u_2, x_3\rangle + \langle u_3, x_3\rangle\right] \end{aligned}$$

where we have used the assertion A. By developing this expression, we notice that it is equal to zero. Hence assertion A implies assertion B.

Now suppose that assertion B is true. Substitution of $x = x_1 + x_2 + x_3$ into $|x_i \quad \langle u_i, x\rangle|_{i=1,2,3} = 0$ gives:

$$\begin{aligned}
& A\left[\langle u_1, x_1\rangle + \langle u_1, x_2\rangle + \langle u_1, x_3\rangle\right] + B\left[\langle u_2, x_1\rangle + \langle u_2, x_2\rangle + \langle u_2, x_3\rangle\right] \\
& \qquad + C\left[\langle u_3, x_1\rangle + \langle u_3, x_2\rangle + \langle u_3, x_3\rangle\right] = 0 \\
\Leftrightarrow \quad & A[w_{12} + w_{13}] + A\left[\langle u_1, x_1\rangle - \langle u_2, x_1\rangle - \langle u_3, x_1\rangle\right] \\
& B[w_{23} + w_{21}] + B\left[-\langle u_1, x_2\rangle + \langle u_2, x_2\rangle - \langle u_3, x_2\rangle\right] \\
& C[w_{32} + w_{31}] \underbrace{+C\left[-\langle u_1, x_3\rangle - \langle u_2, x_3\rangle + \langle u_3, x_3\rangle\right]}_{=0 \text{ by developing}} = 0 \\
\Leftrightarrow \quad & A[w_{12} + w_{13}] + B[w_{23} + w_{21}] + C[w_{32} + w_{31}] = 0
\end{aligned}$$

where A, B and $C \in \mathbb{R}$ are the same as above and $w_{ij} = \langle u_i, x_j\rangle + \langle u_j, x_i\rangle$. The last expression implies that $w_{ij} = \langle u_i, x_j\rangle + \langle u_j, x_i\rangle = 0$ for every i and j. Hence assertion B implies assertion A. □

Lemma 4.2.2. *If x_i $(i = 1, 2, 3)$ are points of $\mathbb{R}^2$ such that $x_1 \neq c \cdot x_2$ $(c \in \mathbb{R})$ and x_3 is arbitrary and if u_1, u_2 are any vectors of $\mathbb{R}^2$ then the system of equations*

$$\begin{aligned}
\langle x_3, u_1\rangle + \langle x_1, u\rangle = 0 \\
\langle x_3, u_2\rangle + \langle x_2, u\rangle = 0
\end{aligned}$$

has exactly one solution.

Proof. The determinant of the system is non-zero. Indeed, the system can be rewritten as:

$$\begin{pmatrix} x_1^T \\ x_2^T \end{pmatrix} u = \begin{pmatrix} -x_3^T u_1 \\ -x_3^T u_2 \end{pmatrix}$$

for which the determinant is $\begin{vmatrix} x_1^T \\ x_2^T \end{vmatrix} = \begin{vmatrix} \xi_1 & \eta_1 \\ \xi_2 & \eta_2 \end{vmatrix}$ which is indeed non-zero if $x_1 \neq c \cdot x_2$. □

Proposition 4.2.3. *An infinitesimal displacement δ of (G, p) is trivial if and only if the matrix*

$$(p_i \quad \langle \delta_i, x\rangle)_{i \in V}$$

has rank 2 identically in x. The trivial infinitesimal displacements constitute a 1-dimensional subspace of the $2|V|$-dimensional vector space of infinitesimal displacements.

Proof. First suppose the rank of the matrix $(p_i \quad \langle \delta_i, x \rangle)_{i \in V}$ to be 2 identically in x. Choose two positions p_{k_1} and p_{k_2} such that $p_{k_1} \neq c \cdot p_{k_2}$ $(c \in \mathbb{R})$; and an arbitrary third position p_{k_3}. Let $\psi(p_{k_1}) = \delta_{k_1}$ and $\psi(p_{k_2}) = \delta_{k_2}$, the infinitesimal rotation ψ is clearly unique. Then by Proposition 4.2.1 it follows that both $u = \delta_{k_3}$ and $u = \psi(p_{k_3})$ satisfy

$$\begin{cases} \langle p_{k_3}, \delta_{k_1} \rangle + \langle p_{k_1}, u \rangle = 0 \\ \langle p_{k_3}, \delta_{k_2} \rangle + \langle p_{k_2}, u \rangle = 0 \end{cases}$$

Lemma 4.2.2 says that this system has a unique solution and $\psi(p_{k_3}) = \delta_{k_3}$. For every other $k \in V$, we repeat the above argument and find $\psi(p_k) = \delta_k$ for all $k \in V$. So δ is trivial.

If, on the other hand, δ is trivial then, it follows easily from the existence of an infinitesimal rotation ψ and from Proposition 4.2.1 that the matrix $(p_i \quad \langle \delta_i, x \rangle)_{i \in V}$ has rank 2 for every x.

There is a one-to-one correspondence between the infinitesimal rotation around the origin ψ and the trivial infinitesimal displacements. So the trivial infinitesimal displacements constitute a 1-dimensional subspace of the $2|V|$-dimensional vector space of infinitesimal displacements. □

.

4.3 Criteria for completability

This section leads to assertions equivalent to the completability of a plane structure. In addition, two operations preserving the completability of a plane structure are described: vertex addition and edge splitting.

Proposition 4.3.1. *A small displacement is admissible if and only if the corresponding infinitesimal displacement satisfies* $\langle p_i, \delta_j \rangle + \langle p_j, \delta_i \rangle = 0$ *for every* $(i, j) \in E$.

Proof.

$$\begin{aligned} \langle p_i(\tau), p_j(\tau \rangle - \langle p_i, p_j \rangle &= \langle p_i + \tau \cdot \delta_i + o(\tau), p_j + \tau \cdot \delta_j + o(\tau) \rangle - \langle p_i, p_j \rangle \\ &= \langle p_i, p_j \rangle + \tau \langle p_i, \delta_j \rangle + \tau \langle p_j, \delta_i \rangle + o(\tau) - \langle p_i, p_j \rangle \\ &= \tau [\langle p_i, \delta_j \rangle + \langle p_j, \delta_i \rangle] + o(\tau) \end{aligned}$$

From this and Definition 4.1.9 the proposition follows. □

The condition $\langle p_i, \delta_j \rangle + \langle p_j, \delta_i \rangle = 0$ for every $(i,j) \in E$ corresponds to a system of equations that can be written in matrix form $C_p \delta = 0$ where C_p is called the completion matrix. For example, the following system,

$$\begin{pmatrix} 2p_1^T & 0 & 0 \\ p_2^T & p_1^T & 0 \\ p_3 & {}^T0 & p_1^T \\ 0 & p_3^T & p_2^T \end{pmatrix} \begin{pmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \end{pmatrix} = 0,$$

corresponds to the condition of admissible small displacements of a plane structure with three vertices and the following edges: $(1,1), (1,2), (1,3), (2,3)$.

The condition to be satisfied in Proposition 4.3.1 is equivalent to having $\langle p_i - p_j, \delta_i - \delta_j \rangle = 0$ for each $(i,j) \in E$ and $\langle p_i, \delta_i \rangle = 0$ for each $i \in V$. Remember that in the rigidity problem, the only condition is $\langle p_i - p_j, \delta_i - \delta_j \rangle = 0$ for each $(i,j) \in E$. In the completion problem, there is an additional condition on each vertex.

Theorem 4.3.2. *The assertions A, B, C, D are equivalent:*
A. The plane structure (G,p) is completable.
B. If $\langle p_i, \delta_j \rangle + \langle p_j, \delta_i \rangle = 0$ for all $(i,j) \in E$, then δ is trivial.
C. If $\langle p_i, \delta_j \rangle + \langle p_j, \delta_i \rangle = 0$ for all $(i,j) \in E$,
then $\langle p_i, \delta_j \rangle + \langle p_j, \delta_i \rangle = 0$ for all $(i,j) \in \mathcal{P}(V)$.
D. The completion matrix, i.e. the matrix of the system of equations:
$\langle p_i, \delta_j \rangle + \langle p_j, \delta_i \rangle = 0$ for all $(i,j) \in E$ has rank $2|V| - 1$.

Proof. The equivalence of A and B is a consequence of the definition of completability, Definition 4.1.12, and Proposition 4.3.1.

The equivalence of B and C is a consequence of Proposition 4.2.3. Indeed, if the matrix $(p_i \quad \langle \delta_i, x \rangle)_{i \in V}$ has rank 2 identically in x, then for every three nodes $k_1, k_2, k_3 \in V$, we have $|p_{k_i} \quad \langle \delta_{k_i}, x \rangle|_{i=1,2,3} = 0$ and by Proposition 4.2.1 this is equivalent to have $\langle p_{k_i}, \delta_{k_j} \rangle + \langle p_{k_j}, \delta_{k_i} \rangle = 0$ for every $i, j = 1, 2, 3$.

The equivalence of B and D is a consequence of the second part of Proposition 4.2.3. □

Remember that the notation $\mathcal{P}(V)$ corresponds to the set of pairs of V, i.e. the set having as elements subsets of V containing exactly two elements of V.

Proposition 4.3.3 (Vertex addition)**.** *Suppose $G'(V', E')$ to be a graph. Let $V = V' \cup \{a\}$, $E = E' \cup \{(a, b_1), (a, b_2)\}$ and $G = G(V, E)$, where b_1 and b_2 both belong to V' or one of the two is the node a and the other belongs to V'. Let moreover (G', p') be completable. Then (G, p) is completable where p satisfies*

$p_k = p'_k$ for every $k \in V'$ and $p_{b_1} \neq c \cdot p_{b_2}$ for every $c \in \mathbb{R}$ in both cases ($b_1, b_2 \neq a$ and $b_2 = a$). In the first case where $b_1, b_2 \neq a$, p_a can take any value (it can even be collinear with p_{b_1} and p_{b_2}).

Proof. By adding the two new edges, we add two rows to the completion matrix. This matrix has a rank of $2|V'| - 1$ since it is completable. We want the rank after adding the two rows to be $2|V| - 1$ in order to remain completable. Since $|V| = |V'| + 1$, the new rank must be equal to $2|V'| + 2 - 1$. So we want the addition of the two rows to increase the rank of the matrix by 2. Let's consider the two cases separately:

1. Adding (a, b_1) and (a, b_2) with $b_1, b_2 \neq a$. The system corresponding to the two new edges is:

$$\begin{aligned}
&\begin{cases} \langle p_a, \delta_{b_1} \rangle + \langle p_{b_1}, \delta_a \rangle = 0 \\ \langle p_a, \delta_{b_2} \rangle + \langle p_{b_2}, \delta_a \rangle = 0 \end{cases} \\
\Leftrightarrow &\begin{cases} \langle p_{b_1}, \delta_a \rangle = -\langle p_a, \delta_{b_1} \rangle \\ \langle p_{b_2}, \delta_a \rangle = -\langle p_a, \delta_{b_2} \rangle \end{cases} \\
\Leftrightarrow &\begin{pmatrix} p_{b_1}^T \\ p_{b_2}^T \end{pmatrix} \delta_a = \begin{pmatrix} -p_a^T \delta_{b_1} \\ -p_a^T \delta_{b_2} \end{pmatrix}
\end{aligned}$$

   where δ_a is the unknown. If $p_{b_1} \neq c \cdot p_{b_2}$ the determinant of the system is non-zero and the system of equations has exactly one solution.

2. Adding (a, b_1) and (a, a). The system corresponding to the two new edges is:

$$\begin{aligned}
&\begin{cases} \langle p_a, \delta_{b_1} \rangle + \langle p_{b_1}, \delta_a \rangle = 0 \\ \langle 2p_a, \delta_a \rangle = 0 \end{cases} \\
\Leftrightarrow &\begin{cases} \langle p_{b_1}, \delta_a \rangle = -\langle p_a, \delta_{b_1} \rangle \\ 2\langle p_a, \delta_a \rangle = 0 \end{cases} \\
\Leftrightarrow &\begin{pmatrix} p_{b_1}^T \\ 2p_a^T \end{pmatrix} \delta_a = \begin{pmatrix} -p_a^T \delta_{b_1} \\ 0 \end{pmatrix}
\end{aligned}$$

   where δ_a is the unknown. If $p_{b_1} \neq c \cdot p_a$ the determinant of the system is non-zero and the system of equations has exactly one solution.

Since in both cases the determinant is nonzero, the two new rows are not linear combinations of each other. Moreover, they are not linear combinations with the other rows either. The rank is therefore increased by 2 and the plane structure (G, p) is completable. □

An illustration of the vertex addition operation is shown on Figure 4.1.

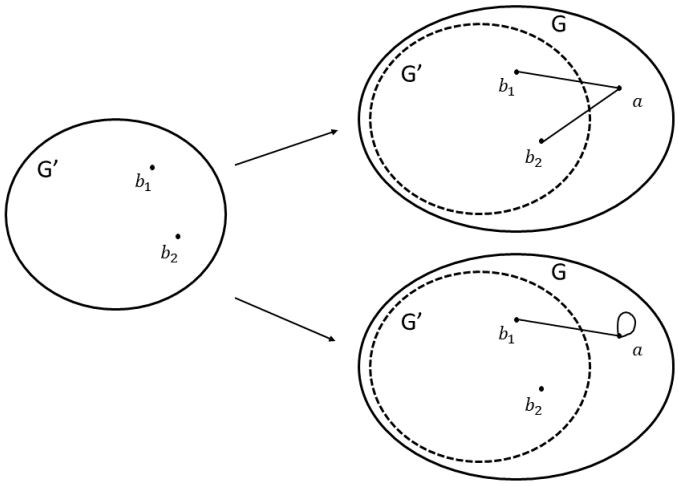


Figure 4.1: Vertex addition operation: adding a new node a and adding the edges (a, b_1) and (a, b_2) or (a, b_1) and (a, a).

Proposition 4.3.4 (Edge splitting)**.** *Suppose $G'(V', E')$ to be a graph. Suppose that there is two nodes $b_1, b_2 \in V'$ such that $(b_1, b_2) \in E'$ is not a self-loop. Let $V = V' \cup \{a\}$, $E'' = E' \backslash \{(b_1, b_2)\}$, $E = E'' \cup \{(a, b_1), (a, b_2), (a, b_3)\}$, $G'' = G''(V', E'')$ and $G = G(V, E)$, where b_3 belongs to V'. Let moreover (G', p') be completable. Then there is a $p : V \to \mathbb{R}^2$ satisfying $p_k = p'_k$ for every $k \in V'$ and such that (G, p) is completable.*

Proof. Assume (G'', p') to be completable. Then, if one of the couples of neighbors of a (e.g. the edge (b_1, b_2)) is such that $p_{b_1} \neq c \cdot p_{b_2}$, by Proposition 4.3.3, (G, p) is completable. Indeed, the operation consists in a vertex addition with $b_1, b_2 \neq a$ and adding one more edge (b_3, a), which preserve completability.

Assume (G'', p') to be non-completable. The admissible infinitesimal displacements of (G', p') are $\alpha\lambda^1$ with $\alpha \in \mathbb{R}$ and λ^1 trivial because (G', p') is completable. Therefore, the admissible infinitesimal displacements of (G'', p') are $\beta_1\lambda^1 + \beta_2\lambda^2$ with λ^2 non-trivial because (G'', p') is not completable. So, $\left| p_{b_j} \quad \langle \lambda^2_{b_j}, x \rangle \right|_{j=1,2,3} = 0$ does not hold identically in x.

For any choice of p_a, any admissible displacement δ of (G, p) should satisfy $\delta = \beta_1\delta^1 + \beta_2\delta^2$ with $\delta^i_k = \lambda^i_k$ for $k \in V'$ and

$$\langle p_a, \delta_{b_j} \rangle + \langle p_{b_j}, \delta_a \rangle = 0 \text{ for } j = 1, 2, 3.$$

The necessary and sufficient condition for solving this system is given in Proposition 4.2.1 by

$$\left|p_{b_j} \quad \langle \delta_{b_j}, p_a \rangle\right|_{j=1,2,3} = 0.$$

We have that $\delta_{b_j} = \beta_1 \lambda^1_{b_j} + \beta_2 \lambda^2_{b_j}$ and $\left|p_{b_j} \quad \langle \lambda^1_{b_j}, p_a \rangle\right|_{j=1,2,3} = 0$ identically in p_a because λ^1 is trivial. So the condition of solvability of the system is equivalent to

$$\beta_2 \left|p_{b_j} \quad \langle \lambda^2_{b_j}, p_a \rangle\right|_{j=1,2,3} = 0.$$

Choose p_a such that $\left|p_{b_j} \quad \langle \lambda^2_{b_j}, p_a \rangle\right|_{j=1,2,3} \neq 0$. Then β_2 is zero, $\delta = \beta_1 \lambda^1$ is a trivial admissible infinitesimal displacement and (G, p) is completable. □

Remark. The edge $(b_1, b_2) \in E'$ cannot be a self-loop because this would imply the addition of two edges between a and b_1 and we cannot have double edges in the graphs, indeed there are semi-simple.

An illustration of the edge splitting operation is shown on Figure 4.2.

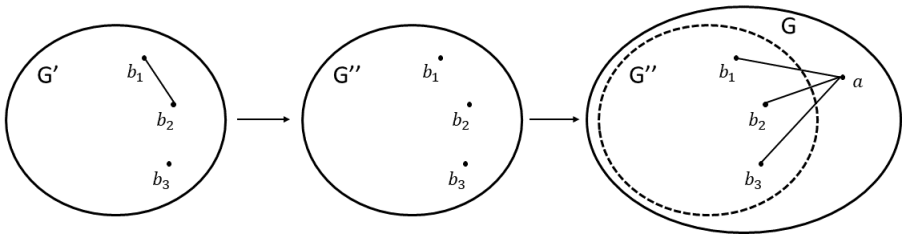


Figure 4.2: Edge splitting operation: deleting the edge (b_1, b_2), adding a node a and adding the edges (a, b_1), (a, b_2) and (a, b_3).

The following two results allow us to define a new type of graph that turns out to have completable realizations.

Proposition 4.3.5. *Every completable plane structure (G, p) has a completable subgraph with $|V|$ nodes and $2|V| - 1$ edges.*

Proof. This follows from Theorem 4.3.2, D; the matrix of $\langle p_i, \delta_j \rangle + \langle p_j, \delta_i \rangle = 0$ for all $(i, j) \in E$ has rank $2|V| - 1$. We can drop equations (and corresponding edge $(i, j) \in E$) until only $2|V| - 1$ independent equations are left corresponding to $2|V| - 1$ edges. □

Theorem 4.3.6. *Any completable plane structure (G, p) with $|E| = 2|V| - 1$ has the following property:*
If $V' \subset V$ and $|V'| \geq 1$ then $|E(V')| \leq 2|V'| - 1$.

Proof. To every $(i,j) \in E$ there corresponds an equation $\langle p_i, \delta_j \rangle + \langle p_j, \delta_i \rangle = 0$; because of completability and Proposition 4.3.5 those equations are independent. If for some $V' \subset V$ with $|V'| \geq 1$ we should have $|E \cap E(V')| > 2|V'| - 1$ then by Proposition 4.3.5 there would be dependence among the corresponding $2|V'| - 1$ equations, which is a contradiction. □

Remember that the notation $E(V')$ corresponds to the restriction of E to V'.

A graph with the property describes in Theorem 4.3.6, $|V| \geq 2$ and $|E| = 2|V|-1$ is called a C-graph. The Section 4.5 allows to prove that every C-graph has completable realizations. And we can say that every completable plane structure contains a completable plane structure of a C-graph, see Section 4.6.

4.4 Examples

This section presents two examples of matrices. A first one for which it is sometimes possible to find a complete realization and a second for which it is always impossible to find a complete realization.

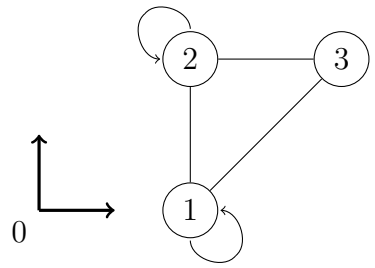
A first basic example is the following partially filled matrix and its corresponding graph:

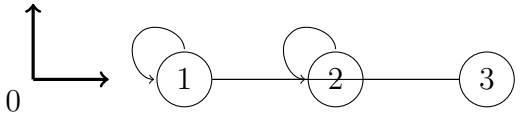
$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & ? \end{pmatrix} \quad \text{and} \quad \begin{array}{l} G(V,E) \text{ with} \\ V = \{1,2,3\} \\ E = \{(1,1),(1,2),(1,3),(2,2),(2,3)\} \end{array}$$

The system to solve is the following:

$$\begin{pmatrix} 2p_1^T & 0 & 0 \\ p_2^T & p_1^T & 0 \\ p_3^T & 0 & p_1^T \\ 0 & 2p_2^T & 0 \\ 0 & p_3^T & p_2^T \end{pmatrix} \begin{pmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \end{pmatrix} = 0$$

We can consider two realizations $p^1 = ((2,0),(2,2),(4,2))$ and $p^2 = ((2,0),(4,0),(6,0))$ for which the two planar structures (G,p^1) and (G,p^2) and their systems are respectively represented below:





$$\underbrace{\begin{pmatrix} 4 & 0 & 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 0 & 0 & 0 \\ 4 & 2 & 0 & 0 & 2 & 0 \\ 0 & 0 & 4 & 4 & 0 & 0 \\ 0 & 0 & 4 & 2 & 2 & 2 \end{pmatrix}}_{rank=5} \begin{pmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \end{pmatrix} = 0 \qquad \underbrace{\begin{pmatrix} 4 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 2 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 8 & 0 & 0 & 0 \\ 0 & 0 & 6 & 0 & 4 & 0 \end{pmatrix}}_{rank=3} \begin{pmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \end{pmatrix} = 0$$

The plane structure (G, p^1) is completable whereas (G, p^2) is not. Indeed, in the first case the only infinitesimal displacements that give admissible small displacements are $\delta^1 = ((0, c), (-c, c), (-c, 2c))$ where $c \in \mathbb{R}$. These infinitesimal displacements are trivial because we can find $\psi(p^1) = \delta^1$ with $\psi(x, y) = (\frac{-c}{2}y, \frac{c}{2}x)$ which is an infinitesimal rotation around the origin of $\mathbb{R}^2$, see Definition 4.1.10. In the second case, the infinitesimal displacements that give admissible small displacements are $\delta^2 = ((0, c), (0, d), (0, e))$ $(c, d, e \in \mathbb{R})$. Take for example $\delta^2 = ((0, 1), (0, 2), (0, 4))$. In this case, we cannot find an infinitesimal rotation around the origin ψ such that $\psi(p^2) = \delta^2$ and thus δ^2 is non-trivial.

This example shows that for the same partially filled matrix and therefore the same graph, one can have a completable and a non-completable plane structure depending on the realization p.

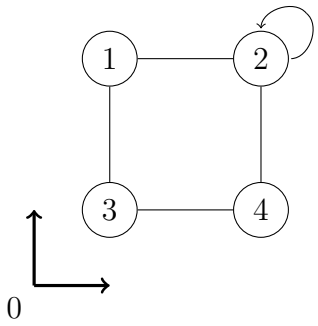
In a second example, consider the following partially filled matrix and its corresponding graph:

$$\begin{pmatrix} ? & a_{12} & a_{13} & ? \\ a_{12} & a_{22} & ? & a_{24} \\ a_{13} & ? & ? & a_{34} \\ ? & a_{24} & a_{34} & ? \end{pmatrix} \quad \text{and} \quad \begin{array}{l} G(V, E) \text{ with} \\ V = \{1, 2, 3, 4\} \\ E = \{(1, 2), (1, 3), (2, 2), (2, 4), (3, 4)\} \end{array}$$

The system to solve is the following:

$$\underbrace{\begin{pmatrix} p_2^T & p_1^T & 0 & 0 \\ p_3^T & 0 & p_1^T & 0 \\ 0 & 2p_2^T & 0 & 0 \\ 0 & p_4^T & 0 & p_2^T \\ 0 & 0 & p_4^T & p_3^T \end{pmatrix}}_{\max rank=5} \begin{pmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ \delta_4 \end{pmatrix} = 0$$

The plane structure (G, p) is never completable whatever the realization p because it is always possible to find an admissible small displacement with non-trivial infinitesimal displacement. For example, the realization $p = ((1,3),(3,3),(1,1),(3,1))$ gives the following representation and system of equations.



$$\underbrace{\begin{pmatrix} 3 & 3 & 1 & 3 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 3 & 0 & 0 \\ 0 & 0 & 6 & 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 1 & 0 & 0 & 3 & 3 \\ 0 & 0 & 0 & 0 & 3 & 1 & 1 & 1 \end{pmatrix}}_{rank=5} \begin{pmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ \delta_4 \end{pmatrix} = 0$$

The infinitesimal displacements that give admissible small displacements are $\delta = ((a,b),(\frac{3(a+b)}{2},\frac{-3(a+b)}{2}),(\frac{a+b}{2},\frac{-(a+b)}{2}),(c,-(a+b+c)))$ where $a, b, c \in \mathbb{R}$. Take for example $\delta = ((1,1),(3,-3),(1,-1),(1,-3))$. In this case, we cannot find an infinitesimal rotation around the origin ψ such that $\psi(p) = \delta$ and thus δ is non-trivial. In conclusion, the plane structure (G, p) is not completable. This could have been seen directly because the number of edges is insufficient. Indeed, Theorem 4.3.2 gives a necessary condition for a plane structure to be rigid: $|E| \geq 2|V| - 1$. Here we have $5 < 2.4 - 1 = 7$.

4.5 C-graphs

The aim of this section is to prove that every C-graph has completable realizations. To do so, two operations preserving the edge distribution property of C-graphs are described: inverse vertex addition and inverse edge splitting.

Let us first give a proposition and a lemma which will be very useful.

Proposition 4.5.1. *A C-graph does not contain nodes of degree 1 and has at least one node of degree ≤ 3. It also does not have isolated node.*

Proof. Suppose $G(V, E)$ is a C-graph and the node a has degree 1. Then $V' = V\backslash\{a\}$ and $|E(V')| = |E| - 1 = (2|V| - 1) - 1 = 2|V| - 2$. But $|V'| = |V| - 1$ and thus $|E(V')| = 2(|V'| + 1) - 2 = 2|V'| > 2|V'| - 1$. This contradicts the property of C-graphs.

The sum of degrees of all the nodes is given by $\sum_i d_i = 2|E| = 4|V| - 2$. In consequence, all nodes cannot have degree ≥ 4 and there is at least one node of degree ≤ 3.

Finally suppose the existence of an isolated node a. There are two possibilities:

1. a has no self-loop and thus has degree 0. In this case, the number of edges of the graph $G'(V', E(V'))$ with $V' = V\backslash\{a\}$ is $|E'| = |E| = 2|V| - 1 = 2(|V'| + 1) - 1 = 2|V'| + 1 > 2|V'| - 1$.

2. a has a self-loop and thus has degree 2. In this case, the number of edges of the graph $G'(V', E(V'))$ with $V' = V\backslash\{a\}$ is $|E'| = |E| - 1 = 2|V| - 2 = 2(|V'| + 1) - 2 = 2|V'| > 2|V'| - 1$.

In both cases, the property of C-graphs does not hold. □

Lemma 4.5.2. *Let $G(V, E)$ be a C-graph, $L \subset V$, $M \subset V$, $|L \cap M| \geq 1$, $|E(L)| = 2|L| - 1$ and $|E(M)| = 2|M| - 1$. Then $|E(L \cup M)| = 2|L \cup M| - 1$.*

Proof.

$$\begin{aligned} |E(L \cup M)| &\geq |E(L)| + |E(M)| - |E(L \cap M)| \quad (\text{where } |E(L \cap M)| \leq 2|L \cap M| - 1) \\ &\geq 2|L| - 1 + 2|M| - 1 - (2|L \cap M| - 1) \\ &= 2|L \cup M| - 1 \end{aligned}$$

By property of C-graphs we also have $|E(L \cup M)| \leq 2|L \cup M| - 1$. Therefore equality holds. □

The following two theorems are about operations that preserve the property of C-graphs.

Theorem 4.5.3 (Inverse vertex addition)**.** *Let $G(V, E)$ be a graph and $a \in V$ a node of degree 2 or of degree 3 with a self-loop, i.e. a is a node with two neighbors. Then $G'(V', E(V'))$, where $V' = V\backslash\{a\}$, is a C-graph if and only if $G(V, E)$ is a C-graph.*

Proof. If $G(V, E)$ is an C-graph we have

$$|E(V')| = |E| - 2 = 2|V| - 1 - 2 = 2|V'| - 1$$

and for every $L \subset V'$ with $|L| \geq 1$

$$|E(V') \cap E(L)| = |E(L)| \leq 2|L| - 1 \qquad \text{since } L \subset V.$$

So G' is a C-graph.

If on the other hand $G'(V', E(V'))$ is a C-graph, then

$$|E| = |E(V')| + 2 = 2|V'| - 1 + 2 = 2|V| - 1$$

and for $L \subset V$ with $|L| \geq 1$

$$|E(L)| = \begin{cases} |E(V') \cap E(L)| \leq 2|L| - 1 & \text{if } a \notin L; \\ |E(V') \cap E(L)| + 2 \leq 2|L \backslash \{a\}| - 1 + 2 = 2|L| - 1 & \text{if } a \in L. \end{cases}$$

So G is a C-graph. □

An illustration of the inverse vertex addition operation is shown on Figure 4.3.

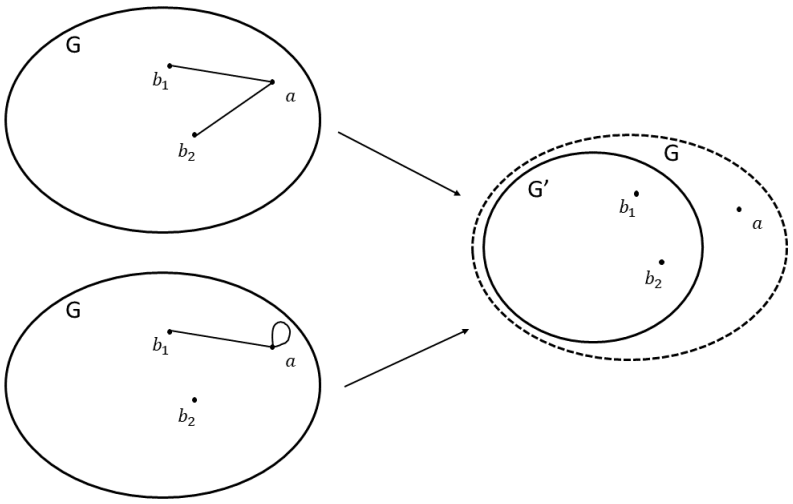


Figure 4.3: Inverse vertex addition operation: if a has degree 2, deleting a and the edges (a, b_1) and (a, b_2); if a has degree 3 with a self-loop, deleting a and the edges (a, b_1) and (a, a).

Theorem 4.5.4 (Inverse edge splitting)**.** *Let $G(V, E)$ be a graph and $a \in V$ a node of degree 3 without self-loop, the neighbors of a being b_1, b_2 and b_3. Then to at least one of the couples of neighbors of a (e.g. the edge (b_1, b_2)), there does not exist a $L \subset V'$ with $b_1 \in L$, $b_2 \in L$ and $|E(L)| = 2|L| - 1$. And if $(b_1, b_2) \in E$, $(b_2, b_3) \in E$ and $(b_1, b_3) \in E$, to at least one of the neighbors of a, b_1 or b_2 (for which $|E(L)| < 2|L| - 1$) (e.g. the node b_1), there does not exist a $M \subset V'$ with*

$b_1 \in M$ and $|E(M)| = 2|M| - 1$. And $G'(V', E')$ is a C-graph if and only if $G(V, E)$ is a C-graph where $V' = V \backslash \{a\}$ and $E' = E(V') \cup \{(b_1, b_2)\}$ if $(b_1, b_2) \notin E$ and $E' = E(V') \cup \{(b_1, b_1)\}$ if $(b_1, b_2) \in E$, $(b_2, b_3) \in E$ and $(b_1, b_3) \in E$.

Proof. **I.** Suppose $G(V, E)$ is a C-graph. Suppose there exist $L_i \subset V'$ $(i = 1, 2, 3)$ such that $b_2, b_3 \in L_1$, $b_1, b_3 \in L_2$ and $b_1, b_2 \in L_3$ and $|E(L_i)| = 2|L_i| - 1$ $(i = 1, 2, 3)$. We have that $|L_1 \cap L_2| \geq 1$, both contain b_3, and by using Lemma 4.5.2 we have $|E(L_1 \cup L_2)| = 2|L_1 \cup L_2| - 1$. Moreover $|(L_1 \cup L_2) \cap L_3| \geq 2$, both contain b_1 and b_2. Applying Lemma 4.5.2 once again we find

$$|E(L_1 \cup L_2 \cup L_3)| = 2|L_1 \cup L_2 \cup L_3| - 1.$$

Finally we find

$$\begin{aligned} |E(L_1 \cup L_2 \cup L_3 \cup \{a\})| &= 2|L_1 \cup L_2 \cup L_3| - 1 + 3 \\ &= 2|L_1 \cup L_2 \cup L_3 \cup \{a\}| \\ &> 2|L_1 \cup L_2 \cup L_3 \cup \{a\}| - 1 \end{aligned}$$

It contradicts the fact that G is a C-graph and thus for one of the L_i, e.g. the set L_3, we have $|E(L_3)| < 2|L_3| - 1$.

If $(b_1, b_2) \in E$, $(b_2, b_3) \in E$ and $(b_1, b_3) \in E$, suppose there exist $M_i \subset V'$ $(i = 1, 2)$ such that $b_1 \in M_1$, $b_2 \in M_2$ and $|E(M_i)| = 2|M_i| - 1$ $(i = 1, 2)$. There are two cases:

- $|M_1 \cap M_2| \geq 1$. Then by Lemma 4.5.2 we have $|E(M_1 \cup M_2)| = 2|M_1 \cup M_2| - 1$.
- $|M_1 \cap M_2| = 0$. Then $|M_1| + |M_2| = |M_1 \cup M_2|$, so

  $$\begin{aligned} |E(M_1 \cup M_2)| &\geq \sum_i |E(M_i)| + 1 \quad \text{because } (b_1, b_2) \in M_1 \cup M_2, \notin M_1, \notin M_2. \\ &= 2|M_1| + 2|M_2| - 2 + 1 \\ &= 2|M_1 \cup M_2| - 1 \end{aligned}$$

  Since G is a C-graph we also have $|E(M_1 \cup M_2)| \leq 2|M_1 \cup M_2| - 1$ and therefore equality holds.

Since $M_1 \cup M_2$ contains both b_1 and b_2, by the result already proven, we must have $|E(M_1 \cup M_2)| < 2|M_1 \cup M_2| - 1$ and thus we have a contradiction in both cases. So at least one of the sets (e.g. the set M_1) has $|E(M_1)| < 2|M_1| - 1$ if $(b_1, b_2) \in E$, $(b_2, b_3) \in E$ and $(b_1, b_3) \in E$.

Now let's prove that G' is also a C-graph. We have

$$|E'| = |E| - 3 + 1 = 2|V| - 1 - 2 = 2|V'| + 2 - 3 = 2|V'| - 1.$$

Now let $N \subset V'$ and $|N| \geq 1$. There are two cases:

- $(b_1, b_2) \notin E$. Then we have proven in the above that the edge (b_1, b_2) can be added. If N does not contain both b_1 and b_2 then $|E(N) \cap E'| = |E(N)| \leq 2|N| - 1$. If $b_1 \in N$ and $b_2 \in N$ then $|E(N)| < 2|N| - 1$ by the result already proven, so $|E(N) \cap E'| = |E(N)| + 1 \leq 2|N| - 1$.

- $(b_1, b_2) \in E$, $(b_2, b_3) \in E$ and $(b_1, b_3) \in E$. Then we have proven in the above that the self-loop (b_1, b_1) can be added. If N does not contain b_1, then $|E(N) \cap E'| = |E(N)| \leq 2|N| - 1$. If $b_1 \in N$ then $|E(N)| < 2|N| - 1$ by the result already proven, so $|E(N) \cap E'| = |E(N)| + 1 \leq 2|N| - 1$.

Therefore, in both cases, G' is a C-graph.

II. Suppose $G'(V', E')$ is a C-graph, then

$$|E| = |E(V')| + 3 = |E(V') \cup \{(b_1, b_2)\}| + 2 = 2|V'| - 1 + 2 = 2|V| - 2 + 1 = 2|V| - 1.$$

And for $L \subset V$ with $|L| \geq 1$,

- if $E' = E(V') \cup \{(b_1, b_2)\}$, we have three cases:

  1. if $a \notin L$, we have

  $$\begin{aligned} |E(L)| &= |E(V') \cap E(L)| \\ &\leq |(E(V') \cup \{(b_1, b_2)\}) \cap E(L)| \\ &= |E' \cap E(L)| \\ &\leq 2|L| - 1 \end{aligned}$$

  2. if $a \in L$ and either $b_1 \notin L$ or $b_2 \notin L$, we have

  $$\begin{aligned} |E(L)| &\leq |E(L \backslash \{a\})| + 2 \\ &= |E' \cap E(L \backslash \{a\})| + 2 \\ &\leq 2|L \backslash \{a\}| - 1 + 2 \\ &= 2|L| - 1 \end{aligned}$$

  3. if $a \in L$, $b_1 \in L$ and $b_2 \in L$, we have

  $$\begin{aligned} |E(L)| &\leq |E(L \backslash \{a\})| + 3 \\ &= |E' \cap E(L \backslash \{a\})| + 3 \\ &< 2|L \backslash \{a\}| - 1 + 3 \quad \text{because } L \backslash \{a\} \text{ contains both } b_1 \text{ and } b_2. \\ &= 2|L \backslash \{a\}| + 2 \\ |E(L)| &\leq 2|L \backslash \{a\}| + 1 \\ &= 2|L| - 1 \end{aligned}$$

- if $E' = E(V') \cup \{(b_1, b_1)\}$, we have three cases:

  1. if $a \notin L$, we have

$$\begin{aligned} |E(L)| &= |E(V') \cap E(L)| \\ &\leq |(E(V') \cup \{(b_1, b_1)\}) \cap E(L)| \\ &= |E' \cap E(L)| \\ &\leq 2|L| - 1 \end{aligned}$$

  2. if $a \in L$ and $b_1 \notin L$, we have

$$\begin{aligned} |E(L)| &\leq |E(L\backslash\{a\})| + 2 \\ &= |E' \cap E(L\backslash\{a\})| + 2 \\ &\leq 2|L\backslash\{a\}| - 1 + 2 \\ &= 2|L| - 1 \end{aligned}$$

  3. if $a \in L$ and $b_1 \in L$, we have

$$\begin{aligned} |E(L)| &\leq |E(L\backslash\{a\})| + 3 \\ &= |E' \cap E(L\backslash\{a\})| + 3 \\ &< 2|L\backslash\{a\}| - 1 + 3 \quad \text{because } L\backslash\{a\} \text{ contains } b_1. \\ &= 2|L\backslash\{a\}| + 2 \\ |E(L)| &\leq 2|L\backslash\{a\}| + 1 \\ &= 2|L| - 1 \end{aligned}$$

So, in every case, $G(V, E)$ is a C-graph. □

An illustration of the inverse edge splitting operation is shown on Figure 4.4.

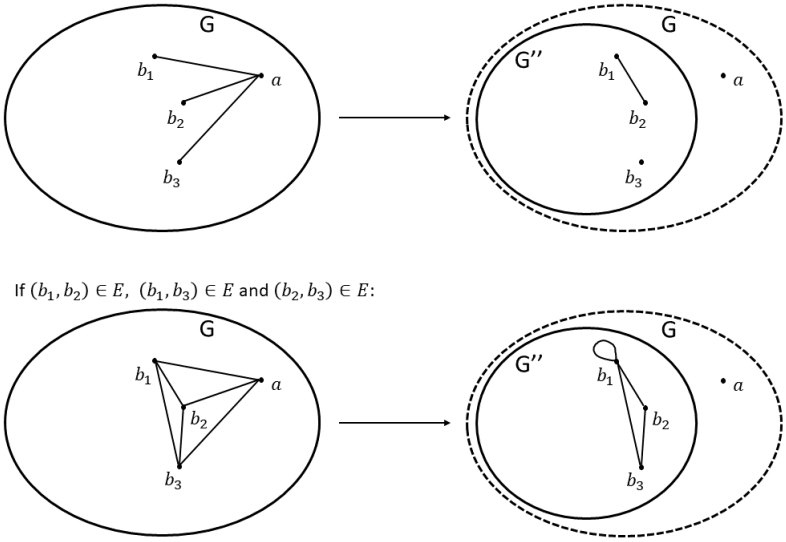
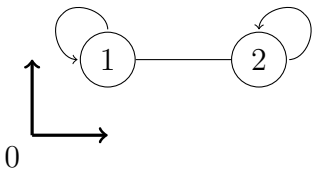


Figure 4.4: Inverse edge splitting operation: a has degree 3 without self-loop, deleting a and the edges (a, b_1), (a, b_2) and (a, b_3), adding the edge between one of the couples of neighbors of a, (b_1, b_2). In the particular case where $(b_1, b_2) \in E$, $(b_1, b_3) \in E$ and $(b_2, b_3) \in E$, adding a self-loop on one of the neighbors of a, (b_1, b_1), as last step.

The final theorem about C-graphs is stated and proven.

Theorem 4.5.5. *Every C-graph has completable realizations.*

Proof. Proceed by induction:
I. The (unique) C-graph with $|V| = 2$ obviously has complete realizations. Indeed, it corresponds to a 2×2 matrix whose entries are all known as shown below.



$$\begin{pmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{pmatrix}$$

II. Assume all C-graphs with k nodes to have completable realizations and let $G(V, E)$ be an arbitrary C-graph with $|V| = k + 1$. By Proposition 4.5.1 there is either a node of degree 2 or a node of degree 3 (with or without a self-loop).

If a is a node of degree 2 or a node of degree 3 with a self-loop then $G'(V \backslash \{a\}, E \cap E(V \backslash \{a\}))$ is a C-graph with $|V \backslash \{a\}| = k$ by Theorem 4.5.3. By assumption G' has completable realization and by Proposition 4.3.3 (G', p') is extendable to a completable realization (G, p) of G.

If a is a node of degree 3 without self-loop then Theorem 4.5.4 yields a C-graph $G'(V', E')$ with $|V'| = k$. By assumption G' has a completable realization and by Proposition 4.3.4 (G', p') is extendable to a completable realization (G, p) of G.

By induction it follows that every C-graph has a completable realization. □

4.6 Arbitrary matrices

This last section states a theorem for determining whether a given matrix is completable. An illustrative example is also given.

Theorem 4.6.1. *A given graph $G(V, E)$ has a completable realization if and only if it contains a C-subgraph that spans the entire vertex set V.*

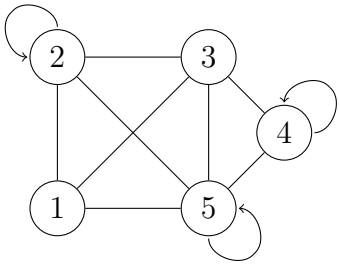
Proof. Suppose the graph $G(V, E)$ contains a C-subgraph that spans the entire vertex set V, denoted $G'(V, E')$ where $E' \subset E$. By Theorem 4.5.5 $G'(V, E')$ has a completable realization. By adding additional edges to G' between the vertices of V, we keep a graph that has completable realization. Indeed, by Theorem 4.3.2, we have that completability of (G', p) is equivalent to the condition that if $\langle p_i, \delta_j\rangle + \langle p_j, \delta_i\rangle = 0$ for all $(i, j) \in E'$, then $\langle p_i, \delta_j\rangle + \langle p_j, \delta_i\rangle = 0$ for all $(i, j) \in \mathcal{P}(V)$. We can thus add edges from $\mathcal{P}(V)$ while keeping the completability property and then obtain the graph $G(V, E)$ which admits a completable realization.

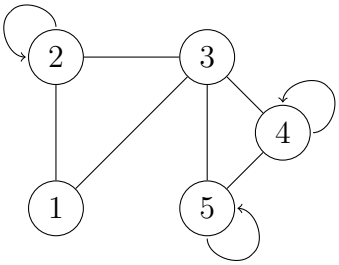
Suppose $G(V, E)$ has a completable realization, then by Proposition 4.3.5, the completable plane structure (G, p) has a completable substructure (G', p) with $G' = G'(V, E')$ where $|E'| = 2|V| - 1$. This substructure has the property of Theorem 4.3.6 and its underlying graph G' corresponds to a C-graph with $|V|$ vertices. □

For example, consider the following matrix and its corresponding graph.

$$\begin{pmatrix} ? & a_{12} & a_{13} & ? & a_{15} \\ a_{12} & a_{22} & a_{23} & ? & a_{25} \\ a_{13} & a_{23} & ? & a_{34} & a_{35} \\ ? & ? & a_{34} & a_{44} & a_{45} \\ a_{15} & a_{25} & a_{35} & a_{45} & a_{55} \end{pmatrix} \quad \text{and} \quad \begin{array}{l} G(V, E) \text{ with} \\ V = \{1, 2, 3, 4, 5\} \\ E = \{(1, 2), (1, 3), (1, 5), (2, 2), (2, 3), (2, 5), \\ \qquad (3, 4), (3, 5), (4, 4), (4, 5), (5, 5)\} \end{array}$$

This can be represented as below left.





This graph has a subgraph, $G'(V, E')$ where $E' = E \backslash \{(1,5),(2,5)\}$, with the property of C-graphs and thus $G(V, E)$ is completable. The subgraph G' is represented as above right.

The next chapter presents an algorithm to tell whether a given graph is a C-graph or contains a C-subgraph. This algorithm can therefore show whether a given matrix is completable. The algorithm is called the Pebble Game and is described in Section 5.3.

CHAPTER 5

PEBBLE GAME ALGORITHM

In the previous chapters, we identified very specific families of graphs, the E-graphs (generic minimally rigid or Laman's graphs) and the C-graphs (generic minimally completable graphs). In this chapter, we propose algorithms for recognizing these graphs, pebble game algorithms, as first mentioned at the end of Section 2.1 of the state of the art chapter.

The Section 5.1 describes the algorithm for recognizing rigid graphs given in [17, 16]. Then, a general pebble game algorithm for (k, l)-sparse graphs is given in Section 5.2 based on [21]. Finally, we adapt the algorithm to the specific case of matrix completion in Section 5.3 and an implementation is provided, see Appendix A.

At the end of this chapter it will be possible to use a pebble game algorithm to determine whether a partially filled symmetric matrix is uniquely completable. Indeed, the provided algorithm takes as argument the graph corresponding to this matrix and shows if it is a C-graph, if it contains one or if it is not the case. If the graph is a C-graph or contains one, its corresponding matrix is uniquely completable as demonstrated in Chapter 4.

5.1 Test for rigid graphs

As said in the state of the art, at the end of Section 2.1.3, *Jacobs and Thorpe* [17] and *Jacobs and Hendrickson* [16] built a simple pebble game algorithm based on the rigidity algorithm proposed by *Hendrickson* [10]. The rest of this section explains the principle of this algorithm.

Remember that a E-graph $G(V, E)$ is a graph with $|V| \geq 3$, $|E| = 2|V| - 3$ and such that $|E \cap E(V')| \leq 2|V'| - 3$ for $V' \subset V$ and $|V'| \geq 2$, this condition is called *Laman's condition* in the following. Laman has proven that every E-graph has rigid realizations.

The goal of the pebble game algorithm is to apply the Laman's theorem recursively by building the graph up one edge at a time. Then only the subgraphs

counting the newly added edge have to be checked for Laman's condition. If all subgraphs satisfy this condition, the considered edge is independent and can be added to the graph, otherwise it is redundant. Redundancy means that the distance between the two vertices of the edge is already fixed by the edges already considered. Searching for the subgraphs is done by constructing a pebble game.

At the beginning, two free pebbles are assigned to each vertex. A free pebble can be used for covering an edge incident to the vertex where the pebble is located. One free pebble represents one independent motion that a vertex can undertake. A vertex with two free pebbles has two translation motions. If another vertex has also two free pebbles, the distance between these two vertices is not fixed and thus placing an edge between these two vertices constrains their distance of separation. To do so, one of the four free pebbles is used to cover this new edge. This edge must always be covered during all the algorithm.

When considering a new edge, the pebbles need to be rearranged in order to bring two pebbles at each vertex incident to the edge. To find a pebble, a depth-first search is performed in a directed graph. If an edge $e = (u, v)$ is covered by a pebble from vertex u, then the edge is directed from u to v. If a pebble is found at vertex w by a depth-first search from vertex v, the pebble is brought from w to v by reversing all the directed edge along the path between v and w. It is possible that no free pebble can be found. A rigid cluster with at least two vertices cannot have more than three free pebbles, this corresponds to the three degrees of freedom of a rigid body in two dimensions (two translations and one rotation). If four pebbles cannot be gathered around an edge, this edge is redundant, it is not covered by a pebble and it creates an over-constrained region.

5.2 Generalization: test for (k, l)-sparse graphs

Lee and Streinu [21] generalized the notion and give pebble games algorithms for sparse graphs, which are described in the following definition.

Definition 5.2.1. *Let $G(V, E)$ be a multigraph with n vertices and m edges. G is a* (k, l)-sparse graph *if each subset of $n' \leq n$ vertices contains at most $kn' - l$ edges. G is called* tight *if in addition to this condition, we have $m = kn - l$. Finally, a* (k, l)-spanning graph *is a graph that contains a tight subgraph spanning the entire vertex set V.*

In their paper, *Lee and Streinu* described the (k, l)-pebble game algorithms and prove that these algorithms recognize exactly the (k, l)-sparse graphs for the entire

range $l \in [0, 2k)$.

They gave lots of properties of sparse graphs about the degrees of their vertices, the presence of multiple edges and loops or the existence of a tight graph, etc. They also defined what a component is. In the rigidity case, components correspond to rigid clusters.

Definition 5.2.2. *A* block *is a subgraph $G'(V', E')$ of a sparse graph $G(V, E)$ that spans exactly $k|V'| - l$ edges with $V' \subset V$. A* component *is a maximal block with respect to the set of vertices.*

Lee and Streinu proposed a theorem for the decomposition of a sparse graph into components. This decomposition allows to speed up the algorithm.

The basic (k, l)-pebble game algorithm is then described in the Figure 4 of their paper. At the beginning of the algorithm, k pebbles are assigned to each vertex. An edge is accepted if we can gather at least $l + 1$ pebbles on the two vertices of the considered edge. All the edges of the graph are considered in an arbitrary order. At the end, the graph is put in one of the four categories: well constrained, under-constrained, over-constrained and other; depending if some edges were rejected during the game and on the number of pebbles remaining in the game when all edges have been considered.

The authors also managed to prove the following correspondence between the result of the algorithm and the type of the graph.

Theorem 5.2.3. *The categories given by the pebble game algorithm, well constrained, under-constrained, over-constrained and other, coincide respectively with tight graphs, sparse graphs, spanning graphs and graphs that are neither sparse nor spanning.*

Finally, the basic algorithm is improved by taking into account the components. This algorithm is very similar to the first one, there is just an additional condition to check for each edge considered. If the two vertices incident to the edge do not belong to a common component, the search of the required number of vertices is guaranteed to succeed and the edge is accepted; otherwise the edge is immediately discarded because it corresponds to a redundant edge. The detection of new components formed by the insertion of an accepted edge must also be done, a technique is described in Figure 9 of the paper.

The time complexity of the basic algorithm is $O(n^3)$ and the time complexity of the improved algorithm is $O(n^2)$ because the redundant edges are rejected in constant time.

5.3 Test for completable matrices

In Chapter 4 we described a type of graphs called C-graphs. A graph $G(V, E)$ with $|E| = 2|V| - 1$ edges is a C-graph if each subset of $|V'| \leq |V|$ vertices contains at most $|E'| = 2|V'| - 1$ edges. To draw a parallel with Section 5.2, a C-graph corresponds to a $(2, 1)$-tight graph and we can therefore develop a pebble game algorithm to recognize whether a given graph is a C-graph, i.e. a minimally completable graph. It will also be possible to tell whether a partially filled matrix and its corresponding graph is completable or not in a general way.

We give, in Figure 5.1, the pebble game algorithm adapted from [21] for the specific case of $(2, 1)$-sparse graph. Figures 5.2 and 5.3 show sub-algorithms used in the pebble game algorithm.

Algorithm 1: Pebble Game Algorithm for $(2, 1)$**-sparse graphs.**
Input: A graph $G = G(V, E)$, possibly with loops.
Output: Well constrained, Under-constrained, Over-constrained, Other.
Setup: Maintain, as an additional data structure, a directed graph D, on which the game is played. Initialized D to be the empty graph on V, and place 2 pebbles on each vertex.
Rules:

1. **Pebbles.** No more than 2 pebbles may be present on a vertex at any time.

2. **Edge acceptance.** An edge between two vertices u and v is accepted for insertion in D when a total of at least 2 pebbles are present on the two endpoints u and v.

Allowable moves:

1. **Pebble collection.** An additional pebble may be collected on a vertex w by searching the directed graph D, e.g., via depth-first search. If a pebble is found, the edges along the directed path leading to it are reversed and the pebble is moved along the path until it reaches w.

2. **Edge insertion.** If an edge between two vertices u and v is accepted, then at least one of the vertices (say, u) contains a pebble. The edge is inserted in D as a directed edge $u \to v$ and a pebble is removed from u.

Algorithm: The edges of G are considered in arbitrary order and the edge acceptance condition is checked. Components are maintained throughout the game. Let $e = (u, v)$ be the current edge.

1. First check if u and v are in some common component. If so, **reject** and discard e.
2. Else, the edge e is guaranteed to be **accepted**. If the acceptance condition is not met for e, the algorithm attempts to collect the required number of pebbles (2) on its two endpoints u and v using the following strategy:
   (a) Mark vertices u and v as visited for the depth-first search algorithm (so they will not be searched, and their pebbles are protected from being moved);
   (b) Perform *Pebble collection* using depth-first search.
3. If the edge e is accepted:
   (a) The edge e is immediately inserted into D as specified by the *Edge insertion* move.
   (b) Detect new component via *Component Detection*, if one is formed, and perform necessary *Component Maintenance*.

The algorithm ends when all the edges have been considered. If exactly 1 pebble remains in the game at the end, the output is **Well constrained** if no edge was rejected, and **Over-constrained** otherwise. If more than 1 pebble remains, the output is **Under-constrained** if there was no edge rejection, and **Other** otherwise.

Figure 5.1: Pebble game algorithm for $(2, 1)$-sparse graphs, [21].

Algorithm 2: Component Detection.
Input: Directed graph $D = D(V, E')$, into which edge $e = (u, v)$ has just been inserted. At least 1 pebble is present on u and v.
Output: The vertex set V' of the new component induced by e, if one was formed; $\emptyset$, otherwise.

Method:

1. If more than 1 pebble is present on u and v, **return** $\emptyset$: the new edge is free.
2. Otherwise, compute $Reach(u, v) = Reach(u) \cup Reach(v)$.
   (a) If any $w \in Reach(u, v)$ has at least one free pebble, **return** $\emptyset$.
   (b) Otherwise, let D' be the directed graph obtained from D by reversing the direction of every edge. For all vertices $w \in V \backslash Reach(u, v)$ with at least one free pebble, perform a depth-first search in D' from w. **Return** V', the set of non-visited vertices from all these searches.

Figure 5.2: Component detection algorithm, [21].

Algorithm 3: Component Maintenance.
Method: The components are vertex disjoint in our case, their maintenance is accomplished with a simple labeling scheme: each vertex is labeled with an id of the component to which it belongs. So when a new component V' has been detected, we simply update the marks or labels of vertices in V'.

Figure 5.3: Component maintenance algorithm, [21].

By the Theorem 5.2.3 of *Lee and Streinu*, the categories given as the result of the pebble game algorithm are equivalent to the graph type and thus to the completability property of the matrix. We have the following correspondences:

- Well constrained $\Leftrightarrow$ C-graph $\Leftrightarrow$ minimally completable matrix;
- Over-constrained $\Leftrightarrow$ graph spanning a C-graph $\Leftrightarrow$ completable matrix;
- Under-constrained $\Leftrightarrow$ sparse graph $\Leftrightarrow$ non-completable matrix;
- Other $\Leftrightarrow$ neither sparse nor spanning graph $\Leftrightarrow$ non-completable matrix.

This algorithm has been implemented in *Python* and is listed in Appendix A.

To test whether a matrix is completable, the algorithm must be told the size of the matrix and which entries of the matrix are known. In the graph associated with this matrix, these data are respectively the number of nodes and the list of edges. You can modify the variables `ex_n` and `ex_edges` as you wish. The variable `ex_edges` is an `numpy array` of dimension $|E| \times 2$ where, for example, the edges $E = \{(0, 1), (0, 2), (0, 3), (1, 1), (1, 2), (3, 3)\}$ are encoded as follows:

`ex_edges = np.array([[0,1],[0,2],[0,3],[1,1],[1,2],[3,3]])`.

Since this is *Python*, we start counting at zero for the entries in the matrix. So the first node must be 0. In the above example we thus have $V = \{0, 1, 2, 3\}$ and `ex_n = 4`.

A basic example is the following:

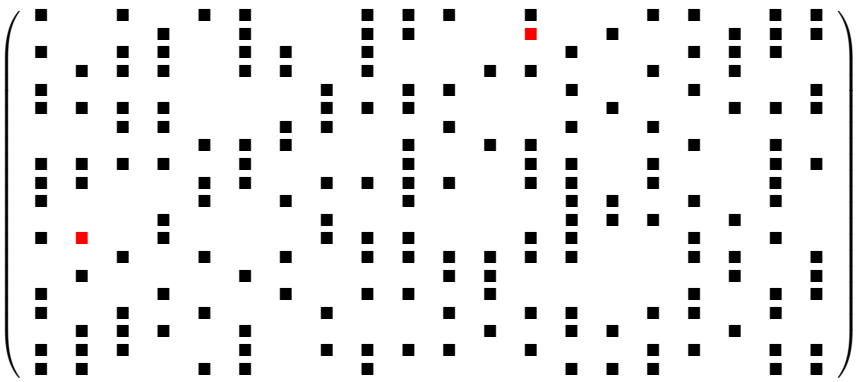
$$\begin{pmatrix} ? & a_{01} & a_{02} & a_{03} \\ a_{01} & a_{11} & a_{12} & ? \\ a_{02} & a_{12} & ? & ? \\ a_{03} & ? & ? & a_{33} \end{pmatrix} \quad \text{and} \quad \begin{array}{l} G(V, E) \text{ with} \\ V = \{0, 1, 2, 3\} \\ E = \{(0,1), (0,2), (0,3), (1,1), (1,2), (3,3)\} \end{array}$$

In this case all the edges are accepted and there are two pebbles left in the graph. The algorithm thus returns the result "Under-constrained" and the considered graph is a $(2, 1)$-sparse graph. The matrix is therefore not completable, there are not enough known entries.

If we add the edge $(1, 3)$ to the above example, all the edges are accepted by the algorithm and there is only one pebble left in the graph. The result is "Well constrained" and the graph is a tight $(2, 1)$-sparse graph, i.e. a C-graph. The matrix is therefore a minimally completable matrix. By adding more edges, these would be rejected by the algorithm, giving an "Over-constrained" result. The graph is therefore a spanning $(2, 1)$-sparse graph and the matrix is completable.

If instead the edge $(0, 0)$ had been added to the basic example, the result is "Other" because edges are rejected while there are still enough pebbles in the game. This means that, despite the sufficient number of known entries in the matrix, they are not positioned sparsely and therefore the matrix is not completable.

So far we have considered a simple example with few nodes. The algorithm can also handle much larger matrices, for example with hundreds of nodes in the corresponding graph. However, the encoding of the edges can be very time consuming. So here is an example with 20 nodes where the black boxes represents known entries.



The graph corresponding to this matrix has 20 node and 100 edges. The algorithm returns the "Over-constrained" result and thus this matrix is completable.

However, if we remove only the $(1, 12)$ entry (in red in the above matrix), the result is "Other" and the matrix is no longer completable.

We now have a handy tool to recognize whether a matrix is uniquely completable.

CHAPTER 6

CONCLUSION

The aim of this master thesis is to progress with the issue of uniqueness of matrix completion with the help of graph rigidity results and, if possible, reproduce these results for completion.

With this in mind, we established a complete and detailed state of the art of both graph rigidity and matrix completion. This allowed us to see that there are not yet many results talking about a deterministic mask for matrix completion. Moreover, authors interested in this problem by using rigidity theory have not tried to directly reproduce Laman's proofs about combinatorial characterization of rigid graph in the plane. The question of whether it is possible to reproduce Laman's proofs in the field of matrix completion was therefore still open.

Before starting to reproduce the result of graph rigidity for matrix completion, we gave a summary of the results of Laman. It allows to have the necessary background for the understanding of the content of the master thesis.

The **main goal** of characterizing the matrix completion has been done. The main conclusion is that we obtained a sufficient and necessary condition to specify completable matrix:

A partially filled matrix A is uniquely completable if
and only if its corresponding graph $G(V, E)$ contains a C-subgraph.

A C-graph $G'(V', E')$ being a graph with $|V'| \geq 2$, $|E'| = 2|V'| - 1$ and if $V'' \subset V'$ and $|V''| \geq 1$ then $|E'(V'')| \leq 2|V''| - 1$.

Finally, we provided an algorithm that allows to recognize matrix that are uniquely completable.

Discussion and perspectives

We achieved to reproduce the development having led to a characterization of graph rigidity in the context of matrix completion. We found a sufficient and necessary

condition to assert whether a partially filled symmetric matrix of rank 2 is uniquely completable.

Starting again from this work, one could be interested in the $d \geq 3$ case in order to obtain a characterization in the d-dimensional space. Unfortunately, no combinatorial characterization of rigid graphs exists for dimensions $d \geq 3$. Only a necessary condition exists but it is not sufficient. It may therefore be difficult to find one for matrix completion. One can hope to find only one necessary condition.

Another way of taking this work further is to consider a matrix A that is no longer symmetric but can be written in the form $A = P^T Q$ where $P \in \mathbb{R}^{d \times n_1}$ and $Q \in \mathbb{R}^{d \times n_2}$. In this case the corresponding graph is a bipartite graph $G(V, E)$ with $|V| = n_1 + n_2$ and $|E| = m$, m being the number of observed entries. To do this, one needs to learn about the characterization of rigid bipartite graphs.

Finally, another possible approach is noise sensitivity. For a given Ω mask, is it possible to recover the matrix if its known inputs are subject to noise ? In particular, what should be the structure of this mask to reduce the sensitivity to noise ?

APPENDIX A

PEBBLE GAME ALGORITHM

This appendix gives the implementation of the pebble game algorithm for checking if a matrix is completable in *Python*, see the description of the algorithm in Section 5.3.

To test whether a matrix is completable, the algorithm must be told the size of the matrix and which entries of the matrix are known. In the graph associated with this matrix, these data are respectively the number of nodes and the list of edges. You can modify the variables `ex_n` and `ex_edges` as you wish. The variable `ex_edges` is an `numpy array` of dimension $|E| \times 2$ where, for example, the edges $E = \{(0,1),(0,2),(0,3),(1,1),(1,2),(3,3)\}$ are encoded as follows:

```
ex_edges = np.array([[0,1],[0,2],[0,3],[1,1],[1,2],[3,3]]).
```

Since this is *Python*, we start counting at zero for the entries in the matrix. So the first node must be 0. In the above example we thus have $V = \{0,1,2,3\}$ and `ex_n = 4`.

```
"""
Master thesis: Matrix Completion and Graph Rigidity: Exploiting
    Surprising Similarities

Title: Pebble game algorithm for (2,1)-sparse graphs, i.e. graphs
    corresponding to completable matrices

Author: Marie Thibeau

Year 2021 - 2022
"""

import numpy as np

def FindPebble(visited,node,pebbles,edges_in_D,peb):
    """
    FindPebble is a recursive function that tries to find a pebble in
     the directed graph D by depth-first search starting at the node '
    node'.

    Parameters
```

```
18      ----------
19      visited : np array
20          Array containing the nodes indices that are already been
        visited by the depth-first search.
21      node : int
22          Current node index for starting the depth-first search.
23      pebbles : np array
24          Index of the node where there is a pebble.
25      edges_in_D : np array
26          Set of directed edges in the graph D.
27      peb : boolean
28          True if a pebble has been found, False otherwise.
29
30      Returns
31      -------
32      peb : boolean
33          True if a pebble has been found, False otherwise.
34      visited : np array
35          Array containing the nodes indices that are already been
        visited by the depth-first search.
36      """
37      edges,index = np.where(edges_in_D==node)
38      edges = edges[np.where(index==0)]
39      if len(edges)!=0:
40          for e in edges:
41              if peb:
42                  return peb,visited
43              new_node = edges_in_D[e,1]
44              if new_node not in visited:
45                  if len(np.where(pebbles==new_node)[0])>0:
46                      peb = True
47                      visited = np.append(visited,new_node)
48                      return peb,visited
49                  visited = np.append(visited,new_node)
50                  peb, visited = FindPebble(visited,new_node,pebbles,
        edges_in_D,peb)
51      return peb,visited
52
53  def FindPath(start,end,edges_in_D,visited=[],path=[]):
54      """
55      FindPath is a recursive function that finds a path in the
        directed graph D that starts at node 'start' ans end at node 'end
        '.
56
57      Parameters
58      ----------
59      start : int
60          Index of the node for the start of the path.
61      end : int
```

```
62          Index of the node for the end of the path.
63      edges_in_D : np array
64          Set of directed edges in the graph D.
65      visited : np.array, optional
66          Array containing the nodes indices that have already been
        visited by the algorithm FindPath. The default is [].
67      path : np array, optional
68          Array containing the nodes indices that are in the path. The
        default is [].
69
70      Returns
71      -------
72      path : np array
73          Array containing the nodes indices that are in the path.
74      """
75      visited = np.append(visited,start)
76      path = np.append(path,start)
77      if start == end:
78          return path
79      else:
80          edges,index = np.where(edges_in_D==start)
81          edges = edges[np.where(index==0)]
82          for i in edges:
83              node = edges_in_D[i,1]
84              if node not in visited:
85                  path = FindPath(node, end, edges_in_D,visited, path)
86
87      path = path[0:-1]
88      visited = np.delete(visited,np.where(visited==start)[0][0])
89
90      return path
91
92  def RearrangePebble(start,end,edges_in_D,pebbles):
93      """
94      RearrangePebble finds a path using the function FindPath then
        rearrange the pebbles by putting the pebble at node 'end' in node
        'start' and inverse the edges along the path between 'start' and '
        end'.
95
96      Parameters
97      ----------
98      start : int
99          Index of the node for the start of the path.
100     end : int
101         Index of the node for the end of the path.
102     edges_in_D : np array
103         Set of directed edges in the graph D.
104     pebbles : np array
105         Index of the node where there is a pebble.
```

```
106
107     Returns
108     ---------
109     pebbles : np array
110         Index of the node where there is a pebble after rearrange the
        pebbles.
111     edges_in_D : np array
112         Set of directed edges in the graph D after rearrange the
        pebbles.
113     """
114     path = FindPath(start,end,edges_in_D)
115     path = np.append(path,end)
116
117     edges_index = np.array([])
118     ind = -1
119     for i in range(len(path)-1):
120         s = path[i]
121         e = path[i+1]
122         edge_start,index = np.where(edges_in_D==s)
123         edge_start = edge_start[np.where(index==0)]
124         for i in edge_start:
125             if edges_in_D[i,1]==e:
126                 ind = i
127         edges_index = np.append(edges_index,ind)
128
129     for i in edges_index:
130         i = int(i)
131         edges_in_D[i,:] = [edges_in_D[i,1],edges_in_D[i,0]]
132
133     pebbles = np.delete(pebbles,np.where(pebbles==end)[0][0])
134     pebbles = np.append(pebbles,start)#path[0])
135
136     return pebbles, edges_in_D
137
138 def PebbleCollection(pebbles,edges_in_D,u,v):
139     """
140     PebbleCollection attempts to find a pebble on a node w by depth-
        first search of the directed graph D and bring it back to the node
        u or v.
141
142     Parameters
143     ----------
144     pebbles : np array
145         Index of the node where there is a pebble.
146     edges_in_D : np array
147         Set of directed edges in the graph D.
148     u : int
149         Index of the first node incident to the edge e.
150     v : int
```

```
151         Index of the second node incident ti the edge e.
152
153     Returns
154     -------
155     found : boolean
156         True if a pebble has been found. False otherwise.
157     pebbles : np array
158         Index of the node where there is a pebble after performing
        PebbleCollection.
159     edges_in_D : np array
160         Set of directed edges in the graph D after performing
        PebbleCollection.
161     """
162     found, visited = FindPebble(np.array([u,v]),u,pebbles,edges_in_D,
        False)
163     if found:
164         start_node = u
165         end_node = visited[-1]
166     else:
167         found, visited = FindPebble(np.array([u,v]),v,pebbles,
        edges_in_D,False)
168         if found:
169             start_node = v
170             end_node = visited[-1]
171         else:
172             return found,pebbles,edges_in_D
173
174     pebbles,edges_in_D = RearrangePebble(start_node,end_node,
        edges_in_D,pebbles)
175     return found,pebbles,edges_in_D
176
177 def EdgeInsertion(pebbles,edges_in_D,u,v):
178     """
179     EdgeInsertion insert the accepted edge e=uv in D.
180
181     Parameters
182     ----------
183     pebbles : np array
184         Index of the node where there is a pebble.
185     edges_in_D : np array
186         Set of directed edges in the graph D.
187     u : int
188         Index of the first node incident to the edge e.
189     v : int
190         Index of the second node incident ti the edge e.
191
192     Returns
193     -------
194     pebbles : np array
```

```
195         Index of the node where there is a pebble after inserted edge
        e.
196     edges_in_D : np array
197         Set of directed edges in the graph D after inserted edge e.
198     """
199     if len(np.where(pebbles==u)[0])!=0:
200         edges_in_D = np.append(edges_in_D,[[u,v]],axis=0)
201         pebbles = np.delete(pebbles,np.where(pebbles==u)[0][0])
202         return pebbles,edges_in_D
203     else:
204         edges_in_D = np.append(edges_in_D,[[v,u]],axis=0)
205         pebbles = np.delete(pebbles,np.where(pebbles==v)[0][0])
206         return pebbles,edges_in_D
207
208 def ReachOneNode(reached,node,edges_in_D):
209     """
210     ReachOneNode is a recursive function that finds all the nodes
        that can be reached from node 'node' in the directed graph D by
        depth-first search.
211
212     Parameters
213     ----------
214     reached : np array
215         Nodes indices that can be reached from node 'node'.
216     node : int
217         Index of the node from which we start depth-first search.
218     edges_in_D : np array
219         Set of directed edges in the graph D.
220
221     Returns
222     -------
223     reached : np array
224         Nodes indices that can be reached from node 'node'.
225     """
226     edges,index = np.where(edges_in_D==node)
227     edges = edges[np.where(index==0)]
228     if len(edges)!=0:
229         for e in edges:
230             new_node = edges_in_D[e,1]
231             if new_node not in reached:
232                 reached = np.append(reached,new_node)
233                 reached = ReachOneNode(reached,new_node,edges_in_D)
234     return reached
235
236 def Reach(u,v,edges_in_D):
237     """
238     Reach finds all the nodes that can be reached from nodes 'u' and
        'v' in the directed graph D by using ReachOneNode.
239
```

```
240     Parameters
241     ----------
242     u : int
243         Index of the first node from which we look for reachable
        nodes.
244     v : int
245         Index of the second node from which we look for reachable
        nodes.
246     edges_in_D : np array
247         Set of directed edges in the graph D.
248
249     Returns
250     -------
251     reached : np array
252         Nodes indices that can be reached from nodes 'u' and 'v'.
253     """
254     reach_u = ReachOneNode([u],u,edges_in_D)
255     reach_v = ReachOneNode([v],v,edges_in_D)
256     reached = np.unique(np.concatenate((reach_u,reach_v)))
257     return reached
258
259 def ReverseD(edges_in_D):
260     """
261     ReverseD inverses all the directed edges in the graph D.
262
263     Parameters
264     ----------
265     edges_in_D : np array
266         Set of directed edges in the graph D.
267
268     Returns
269     -------
270     D_prime : np array
271         Set of directed edges in the graph D', the reverse graph of D
        .
272     """
273     D_prime = np.zeros(np.shape(edges_in_D))
274     for i in range(np.shape(edges_in_D)[0]):
275         D_prime[i,:] = [edges_in_D[i,1],edges_in_D[i,0]]
276     return D_prime
277
278 def ComponentDetection(pebbles,edges_in_D,u,v,n):
279     """
280     ComponentDetection detects the possible new component created by
        adding the edge e=uv in D.
281
282     Parameters
283     ----------
284     pebbles : np array
```

```
285         Index of the node where there is a pebble.
286     edges_in_D : np array
287         Set of directed edges in the graph D.
288     u : int
289         Index of the first node incident to the edge e.
290     v : int
291         Index of the second node incident ti the edge e.
292 
293     Returns
294     ---------
295     new_comp : np array
296         Array containing the nodes that are part of the new component
        formed by adding the edge e=uv in D.
297     """
298     new_comp = np.array([])
299     if (len(np.where(pebbles==u)[0])+len(np.where(pebbles==v)[0])>1)
      & (u!=v):
300         return []
301     elif (len(np.where(pebbles==u)[0])>1) & (u==v):
302         return []
303     else:
304         reach = Reach(u,v,edges_in_D)
305         for w in reach:
306             if (w!=u)&(w!=v):
307                 if len(np.where(pebbles==w)[0])!=0:
308                     return []
309         D_prime = ReverseD(edges_in_D)
310         nodes = np.delete(np.arange(n),reach.astype(int))
311         nodes_with_pebbles = np.array([i for i in nodes if i in
      pebbles])
312         visited = np.array([],dtype='int')
313         for w in nodes_with_pebbles:
314             r = ReachOneNode([w],w,D_prime)
315             visited = np.unique(np.append(visited,r))
316 
317         new_comp = np.delete(np.arange(n),visited.astype(int))
318     return new_comp
319 
320 def ComponentMaintenance(comp_id,new_comp):
321     """
322     ComponentMaintenance allows a same number to vertices that are in
        the same component.
323 
324     Parameters
325     ----------
326     comp_id : np array
327         Array containing the id of the component for each vertex. '0'
        means that the vertex is not part of any component.
328     new_comp : np array
```

```
329         New component detected by ComponentDetection. Array
        containing the nodes indices that are part of this new component.
330
331     Returns
332     -------
333     comp_id : np array
334         Array containing the id of the component for each vertex
        taking into account the new component.
335     """
336     new_id = max(comp_id)+1
337     comp_id[new_comp] = new_id
338     return comp_id
339
340 def PebbleGame(edges,n):
341     """
342     PebbleGame performs the pebble game algorithm given by Lee and
        Streinu using components for the specific case k=2 and l=1.
343
344     Parameters
345     ----------
346     edges : np array
347         Set of edges of the input graph..
348     n : int
349         Number of nodes in the input graph.
350
351     Returns
352     -------
353     category : string
354         Category to which the input graph belongs: Well constrained,
        Under-constrained, Over-constrained, Other.
355     pebbles : np array
356         Index of the node where there is a pebble at the end of the
        algorithm.
357     edges_in_D : np array
358         Set of directed edges in the graph D at the end of the
        algorithm. It is thus the set of all accepted edges by the
        algorithm.
359     rejected_edges : np array
360         Set of rejected edges by the algorithm.
361     """
362     edges.sort()
363     pebbles = np.repeat(np.arange(n),2)
364     rejected_edges = np.empty((0,2))
365     comp_id = np.zeros(n)
366     edges_in_D = np.empty((0,2))
367
368     for u,v in edges:
369         print("The edge considered is",[u,v])
370         if (comp_id[u]!=0) & (comp_id[u]==comp_id[v]):
```

```
371             rejected_edges = np.append(rejected_edges,[[u,v]],axis=0)
372             print("    The edge is rejected")
373         else:
374             f = True
375             if (len(np.where(pebbles==u)[0])+len(np.where(pebbles==v)
    [0])<2) & (u!=v):
376                 f,pebbles,edges_in_D = PebbleCollection(pebbles,
    edges_in_D,u,v)
377             elif (len(np.where(pebbles==u)[0])<2) & (u==v):
378                 f,pebbles,edges_in_D = PebbleCollection(pebbles,
    edges_in_D,u,v)
379
380             if f:
381                 print("    The edge is accepted")
382                 pebbles,edges_in_D = EdgeInsertion(pebbles,edges_in_D
    ,u,v)
383                 new_comp = ComponentDetection(pebbles,edges_in_D,u,v,
    n)
384                 if len(new_comp)>0:
385                     comp_id = ComponentMaintenance(comp_id,new_comp)
386             else:
387                 rejected_edges = np.append(rejected_edges,[[u,v]],
    axis=0)
388                 print("    The edge is rejected !!!")
389     if len(pebbles)==1:
390         if len(rejected_edges)==0:
391             category = 'WELL CONSTRAINED'
392         else:
393             category = 'OVER-CONSTRAINED'
394     else:
395         if len(rejected_edges)==0:
396             category = 'UNDER-CONSTRAINED'
397         else:
398             category = 'OTHER'
399     return category, pebbles, edges_in_D, rejected_edges
400
401 print('\n------ START of the algorithm ------')
402 ex_edges = np.array
    ([[0,1],[0,2],[3,3],[0,3],[1,1],[1,2],[1,3],[2,2]])
403 ex_n = 4
404 c,p,a,r = PebbleGame(ex_edges, ex_n)
405 print('------ END of the algorithm ------\n')
406 print('The graph is',c)
407 print('The final number of pebbles is',len(p),'\nThe pebbles are in
    nodes: ',p)
408 print('The accepted edges are:\n',a)
409 print('The rejected edges are:\n',r)
```

BIBLIOGRAPHY

[1] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Convex multi-task feature learning. *Machine learning*, 73(3):243–272, 2008.

[2] Leonard Asimow and Ben Roth. The rigidity of graphs. *Transactions of the American Mathematical Society*, 245:279–289, 1978.

[3] Leonard Asimow and Ben Roth. The rigidity of graphs, ii. *Journal of Mathematical Analysis and Applications*, 68(1):171–190, 1979.

[4] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, 2007.

[5] V Blondel and J-C Delvenne. *INMA1691 - Théorie et Algorithmique des Graphes*. Catholic University of Louvain, Louvain-la-Neuve, Belgium, 2019.

[6] Emmanuel J Candes, Yonina C Eldar, Thomas Strohmer, and Vladislav Voroninski. Phase retrieval via matrix completion. *SIAM review*, 57(2):225–251, 2015.

[7] Emmanuel J Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717–772, 2009.

[8] Henry Crapo. *On the generic rigidity of plane frameworks*. PhD thesis, INRIA, 1990.

[9] Gordon M Crippen, Timothy F Havel, et al. *Distance geometry and molecular conformation*, volume 74. Research Studies Press Taunton, 1988.

[10] Bruce Hendrickson. Conditions for unique graph realizations. *SIAM journal on computing*, 21(1):65–84, 1992.

[11] Julien M Hendrickx and V Blondel. *Graphs and networks for the analysis of autonomous agent systems.* PhD thesis, Catholic University of Louvain, Louvain-la-Neuve, Belgium, 2008.

[12] Lebrecht Henneberg. *Die graphische Statik der starren Systeme*, volume 31. BG Teubner, 1911.

[13] Susanne MA Hermans, Christopher Pfleger, Christina Nutschel, Christian A Hanke, and Holger Gohlke. Rigidity theory for biomolecules: concepts, software, and applications. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 7(4):e1311, 2017.

[14] Bill Jackson, Tibor Jordán, and Shin-ichi Tanigawa. Combinatorial conditions for the unique completability of low-rank matrices. *SIAM Journal on Discrete Mathematics*, 28(4):1797–1819, 2014.

[15] Bill Jackson, Tibor Jordán, and Shin-ichi Tanigawa. Unique low rank completability of partially filled matrices. *Journal of Combinatorial Theory, Series B*, 121:432–462, 2016.

[16] Donald J Jacobs and Bruce Hendrickson. An algorithm for two-dimensional rigidity percolation: the pebble game. *Journal of Computational Physics*, 137(2):346–365, 1997.

[17] Donald J Jacobs and Michael F Thorpe. Generic rigidity percolation: the pebble game. *Physical review letters*, 75(22):4051, 1995.

[18] Robert Krone and Kaie Kubjas. Uniqueness of nonnegative matrix factorizations by rigidity theory. *SIAM Journal on Matrix Analysis and Applications*, 42(1):134–164, 2021.

[19] Gerard Laman. On graphs and rigidity of plane skeletal structures. *Journal of Engineering mathematics*, 4(4):331–340, 1970.

[20] Monique Laurent and Antonios Varvitsiotis. Positive semidefinite matrix completion, universal rigidity and the strong arnold property. *Linear Algebra and its Applications*, 452:292–317, 2014.

[21] Audrey Lee and Ileana Streinu. Pebble game algorithms and sparse graphs. *Discrete Mathematics*, 308(8):1425–1437, 2008.

[22] Xiao Peng Li, Lei Huang, Hing Cheung So, and Bo Zhao. A survey on matrix completion: Perspective of signal processing. *arXiv preprint arXiv:1901.10885*, 2019.

[23] Guangcan Liu, Qingshan Liu, Xiao-Tong Yuan, and Meng Wang. Matrix completion with deterministic sampling: Theories and methods. *IEEE transactions on pattern analysis and machine intelligence*, 43(2):549–566, 2019.

[24] Zhang Liu, Anders Hansson, and Lieven Vandenberghe. Nuclear norm system identification with missing inputs and outputs. *Systems & Control Letters*, 62(8):605–612, 2013.

[25] M Micoulaut. Concepts and applications of rigidity in non-crystalline solids: a review on new developments and directions. *Advances in Physics: X*, 1(2):147–175, 2016.

[26] Luong Nguyen and Byonghyo Shim. Localization of internet of things network via euclidean distance matrix completion. In *2016 IEEE/CIC International Conference on Communications in China (ICCC)*, pages 1–4. IEEE, 2016.

[27] Luong Trung Nguyen, Junhan Kim, and Byonghyo Shim. Low-rank matrix completion: A contemporary survey. *IEEE Access*, 7:94215–94237, 2019.

[28] Amit Singer. A remark on global positioning from local distances. *Proceedings of the National Academy of Sciences*, 105(28):9507–9511, 2008.

[29] Amit Singer and Mihai Cucuringu. Uniqueness of low-rank matrix completion by rigidity theory. *SIAM Journal on Matrix Analysis and Applications*, 31(4):1621–1641, 2010.

[30] Cyril Stark. Global completability with applications to self-consistent quantum tomography. *arXiv preprint arXiv:1209.6499*, 2012.

[31] Tiong-Seng Tay. A new proof of laman's theorem. *Graphs Comb.*, 9(2-4):365–370, 1993.

[32] Tiong-Seng Tay and Walter Whiteley. Recent advances in the generic ridigity of structures. *Structural Topology, 1984, núm. 9*, 1984.

[33] Tiong-Seng Tay and Walter Whiteley. Generating isostatic frameworks. *Structural Topology 1985 Núm 11*, 1985.

[34] Walter Whiteley. The union of matroids and the rigidity of frameworks. *SIAM Journal on Discrete Mathematics*, 1(2):237–255, 1988.

[35] Walter Whiteley. A matroid on hypergraphs, with applications in scene analysis and geometry. *Discrete & Computational Geometry*, 4(1):75–95, 1989.

[36] Lei Zhang, Ligang Liu, Craig Gotsman, and Steven J Gortler. An as-rigid-as-possible approach to sensor network localization. *ACM Transactions on Sensor Networks (TOSN)*, 6(4):1–21, 2010.

[37] Shiyu Zhao and Daniel Zelazo. Bearing rigidity theory and its applications for control and estimation of network systems: Life beyond distance rigidity. *IEEE Control Systems Magazine*, 39(2):66–83, 2019.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**