

# Minimally Rigid Graphs and their Circle Packings: A Rigid Investigation

Ritwik Rajaram Anand

School of Mathematics and Statistics  
University of St Andrews

## **Abstract**

TBD

## **Declaration**

*I certify that this project report has been written by me, is a record of work carried out by me, and is essentially different from work undertaken for any other purpose or assessment.*

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Rigidity: An Introduction</b>	<b>7</b>
2.1	Graph Theory . . . . .	7
2.2	Frameworks . . . . .	9
2.3	Motions . . . . .	10
2.4	Rigidity . . . . .	12
2.4.1	Infinitesimal Rigidity . . . . .	13
2.4.2	Generic Rigidity . . . . .	17
<b>3</b>	<b>Exploring Rigidity &amp; Circle Packings</b>	<b>19</b>
3.1	Degrees of Freedom . . . . .	19
3.2	The Rigidity Matrix . . . . .	20
3.3	Constructing Rigid Structures . . . . .	22
3.4	Rigidity vs Infinitesimal Rigidity . . . . .	24
3.5	Circle Packings . . . . .	25
<b>4</b>	<b>Rigidity of Minimally Rigid Circle Packings</b>	<b>29</b>
4.1	The Goal . . . . .	29
4.2	<code>Rigidity.py</code> . . . . .	32
4.2.1	Creating a Configuration . . . . .	33
4.2.2	Constructing the Rigidity Matrix . . . . .	33
4.2.3	Rank of the Rigidity Matrix . . . . .	34
4.2.4	Checking rigidity for a list of graphs . . . . .	35
4.2.5	Testing the code . . . . .	35
4.3	<code>Circle_Packing.py</code> . . . . .	36
4.3.1	A Simple Example . . . . .	36
	<b>Bibliography</b>	<b>38</b>

# Chapter One

## Introduction

The circle is arguably the most fundamentally studied mathematical object in history. In fact, the construction of the unit circle dates all the way back to the second century BC [13]. With this elementary object, we are able obtain two trigonometric functions, enabling us to study a myriad of physical processes such as the motion of waves and the oscillatory properties of pendulums. In this project however, we will be looking less at the properties of the circle, and more on various geometries involved when we attempt to group, or *pack* as many as we can together.

As some motivation for what is to come, let's take a stroll down the fruits and vegetables section at the supermarket. While looking around, you may notice that spherical fruit, such as oranges, are stacked high in a pyramidal-like shape that tapers off at the top. A display like this is known as a *face-centered cubic packing*.

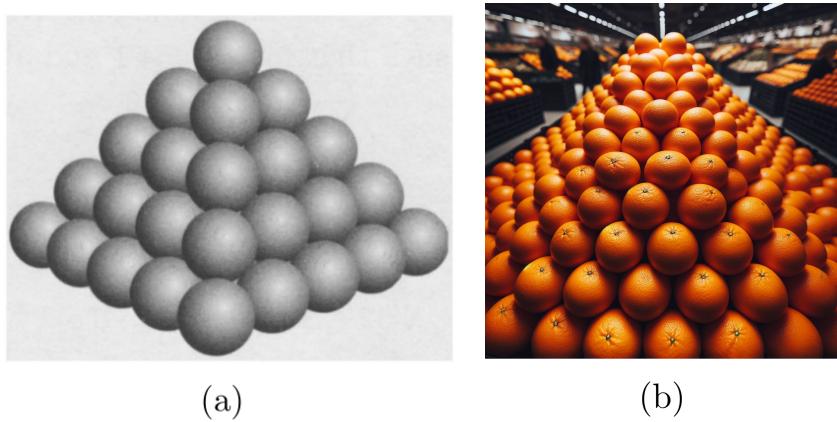


Figure 1.1: (a) A face-centered cubic packing, taken from [7]. (b) An image of oranges stacked in a face-centered cubic packing, generated by DALL·E 3.

This interesting observation was studied by Johannes Kepler, who attempted to find the optimal way to stack cannonballs. In 1611, Kepler conjectured that a face-centered cubic packing is the most efficient method to stack spheres in 3-dimensional space. This was proved by Thomas Hales, at first using computational methods in 1998, and a formal proof was accepted in 2017 [6].

Going back to the 2-dimensional circle, our interest lies in a *circle packing*, which involves arranging several circles in the plane such that they may only touch tangentially, and there is no overlap

between any of the circles. By imposing different constraints, we can generate circle packings where the arrangement formed has circles packed together with a variety of radii, producing an aesthetically pleasing image to regard. They can also be used to study the world around us under a different lens. An example of this involves modelling the arrangement of particles in a crystalline structure as a tight packing of circles [16].

We can study circle packings using similar computational methods, as this project will illustrate. By marking the center of each circle, and joining centers with a line if the circles they inhabit are tangential to each other, we are able to construct the packing's *contact graph*, effectively converting our geometric problem into a graph theoretic one. This allows us to use tools from Graph Theory, increasing the size of our repertoire.

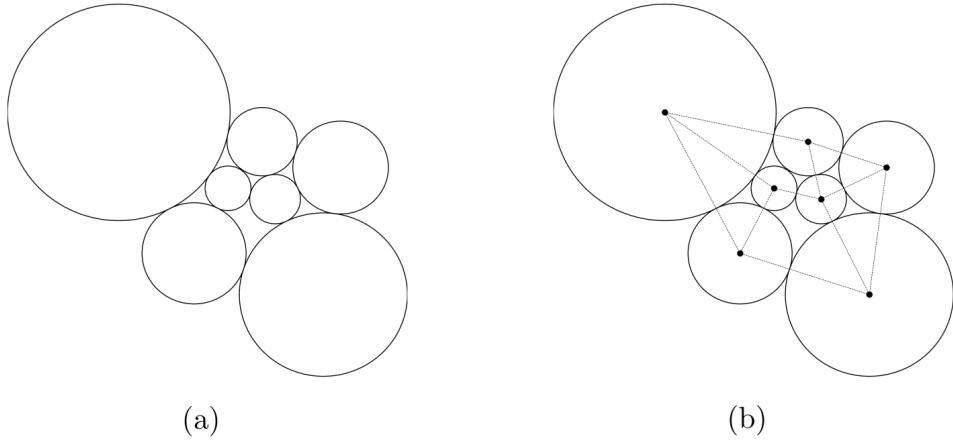


Figure 1.2: (a) A circle packing with 7 circles. (b) The packing with its contact graph visualized.

One such property, and the main focus of this project, is *rigidity*, which was first studied by James Clerk Maxwell. Rigidity is a measure of how 'stiff' something is. Consider structures such as buildings, bridges, sandcastles and so on. Some of them hold up, whereas others don't. This is partly due to the material it is made from, and also because of the fact that some of these structures are rigid, while others are not.

In this project, we will be introduced to the ideas of rigidity, what it means for a *framework* to be rigid, and how we can study the rigidity of circle packings. Along the way, we will try to gain some insight to the question that this thesis focuses on:

*"For every planar and minimally rigid graph, does there exist a circle packing that is also infinitesimally rigid?"*

An exciting aspect of this project is the way in which various fields within mathematics become involved when studying something that seems purely geometric. Ideas from Graph Theory and Linear Algebra come into play and therefore, the intended audience for this project are mathematics students that are accustomed to some fundamental ideas from these two fields, as well as a basic

understanding of how to code. That being said however, everything will be clearly defined and explained as we come across it. Moreover, all the code involved in this project can be viewed on Github at Titwik/Dissertation.

The only pre-requisite required to understand the content of this project involves some fundamental linear algebra. Specifically, the knowledge of how vectors work, and a good understanding of matrices and their properties. The interested reader can explore the flavor of mathematics involved in this project in texts relating to;

- *Graph Theory*: The study of mathematical structures, known as graphs, that represent relationships between objects. It has several applications, some of which include computer science, biology and logistics to name a few.
- *Discrete Mathematics*: The study of countable, discrete objects rather than continuous quantities. Topics involved include Set Theory and Logic, and it plays a big role in the study of computer science.
- *Constrained Optimization*: Attempting to find the maximum or minimum of a certain ‘objective’ function while adhering to specific constraints.

With all that said, let's get into it!

# Chapter Two

## Rigidity: An Introduction

Rigidity can be easily understood as observing a structure and questioning how 'stiff' it is. If some force is applied to it, does it bend or buckle? When we pull the structure across (Euclidean) space, do we move it entirely, or does a part of it bend and move in the direction of the pull? As stiffness can be interpreted in a multitude of ways, this section will be dedicated to formalizing the definition of rigidity that we will use for the remainder of this project.

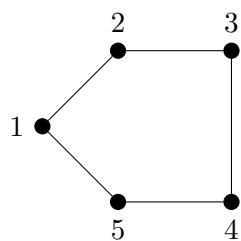
To start, we must first understand what *graphs* are.

### 2.1 Graph Theory

Here, we delve into a small portion of a very large field of mathematics. The definitions presented here will be fundamental for what's to come.

**Definition 2.1.** A *graph*  $G = (V, E)$  consists of a set  $V$  and a set  $E$  of two-element subsets of  $V$ . The elements of  $V$  are called *vertices* and the elements of  $E$  are called *edges*.

**Example 2.2.** The graph  $G = (V, E)$  below has  $|V| = 5$ , and  $|E| = 5$ .



**Definition 2.3.** A graph  $G$  is *connected* if for any two vertices  $u, v$ , there exists a sequence of distinct vertices (called a *path*)  $u_1, \dots, u_k$ , such that  $u_i u_{i+1}$  is an edge of  $G$  for  $0 \leq i \leq k - 1$  that connect  $u$  and  $v$ . The path can be seen as

$$u = u_0, u_1, \dots, u_{k-1}, u_k = v$$

**Definition 2.4.** A *cycle* is a graph  $G = (V, E)$  where  $G$  has  $n$  vertices, and the edges are

$$E(G) = \{12, 23, \dots, (n-1)n\}$$

It can be thought of as a closed path.

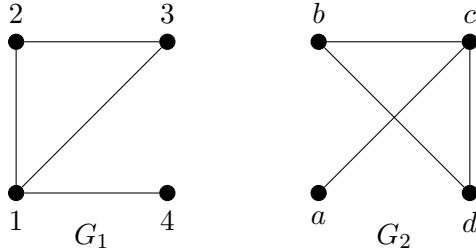
**Example 2.5.** The graph in Example 2.2 is a cycle of length 5.

**Definition 2.6.** The *degree* of a vertex is the number of edges incident to the vertex.

There are several ways of drawing the 'same' graph on the plane. This includes labelling them in a variety of ways as well. Therefore, it is vital that we class these drawings as the same graph.

**Definition 2.7.** Two graphs  $G_1$  and  $G_2$  are said to be *isomorphic* if there is a bijection  $\phi : V(G_1) \rightarrow V(G_2)$  such that  $xy \in E(G_1)$  if and only if  $\phi(x)\phi(y) \in E(G_2)$ . The function  $\phi$  is known as an *isomorphism*.

**Example 2.8.** Consider the two graphs below.

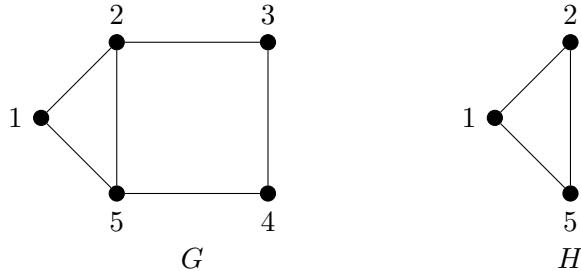


The bijection  $\phi : G_1 \rightarrow G_2$  is given by  $\phi(1) = c$ ,  $\phi(2) = b$ ,  $\phi(3) = c$ , and  $\phi(4) = a$ .

Given a graph  $G$ , we may be interested in a certain subset of vertices and edges that are included in  $G$ . This involves studying a *subgraph* of  $G$ .

**Definition 2.9.** A graph  $H$  is a *subgraph* of  $G$  if there is a graph  $H'$  isomorphic to  $H$  such that  $V(H') \subseteq V(G)$  and  $E(H') \subseteq E(G)$ .

**Example 2.10.** Consider the graphs below.



Here, as  $V(H) \subseteq V(G)$  and  $E(H) \subseteq E(G)$ , it follows that  $H$  is a subgraph of  $G$ .

The last idea we need from graph theory is the notion of *planarity*. The graphs we are interested in for this project are those such that the edges do not cross over each other.

**Definition 2.11.** A graph  $G$  is *planar* if it has a drawing in the plane such that none of edges cross over each other.

**Example 2.12.** Consider Example 2.8. The graph  $G_1$  is a planar drawing of the graph  $G_2$ .

## 2.2 Frameworks

Now that we have a good idea of what graphs are, we can start working on setting up frameworks. The definitions in this section will be defined in the space  $\mathbb{R}^d$ , where  $d \in \mathbb{N}$ . Later, we will find ourselves working exclusively in  $\mathbb{R}^2$ . For the remainder of this chapter, we shall adhere closely to the contents in chapter 2 from the book “Frameworks, Tensegrities, and Symmetry” by Robert Connelly and Simon D. Guest [3].

The first definition we need is that of a *configuration*.

**Definition 2.13.** Suppose we have a collection of  $n$  labelled points in  $\mathbb{R}^d$ , where  $d \in \mathbb{N}$ . Let  $\mathbf{p}_i = (p_{i1}, \dots, p_{id})$  be the position vector of point  $i$ . Then a *configuration* is

$$\mathbf{p} = (\mathbf{p}_1, \dots, \mathbf{p}_n),$$

where  $\mathbf{p}$  is a vector of vectors in  $\mathbb{R}^d$ .

Therefore, a configuration can be thought of as a set of points in the space  $\mathbb{R}^d$ , each point equipped with its own position vector. Henceforth, we will refer to the points in a configuration as *nodes*.

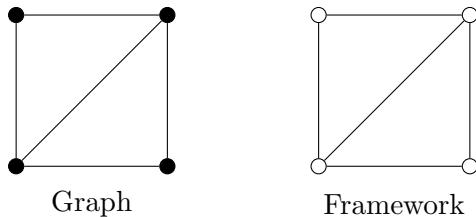
Given a configuration, we can now decide which pairs of nodes to connect with a graph  $G$ . The nodes in  $\mathbf{p}$  correspond to the vertices in  $G$ , and  $G$  is not allowed to have multiple edges between vertices, or loops between the same vertex (such graphs are known to be *simple*).

**Definition 2.14.** A *framework* is a configuration  $\mathbf{p} = (\mathbf{p}_1, \dots, \mathbf{p}_n)$  together with its corresponding graph  $G = (V, E)$ . We denote it as  $(G, \mathbf{p})$ .

Note that as we are working in  $\mathbb{R}^d$ , the *length* of each edge with endpoints  $\mathbf{p}_i$  and  $\mathbf{p}_j$  in a framework  $(G, \mathbf{p})$  is the *Euclidean distance* between the points  $\mathbf{p}_i$  and  $\mathbf{p}_j$ . This is denoted  $|\mathbf{p}_i - \mathbf{p}_j|$ .

In this project, graphs will be drawn with vertices that are filled in, whereas frameworks will have hollow nodes.

**Example 2.15.** In the two diagrams below, the one on the left is a graph, whereas the one on the right is a framework.



At this stage, you may be wondering what exactly the difference between a graph and a framework is. Put simply,

- A graph is an abstract mathematical object that can be visualised in a multitude of ways, as seen in Example 2.8. Varying the edge lengths does not produce a new graph as by Definition 2.7, they are isomorphic, and considered the ‘same’.
- In a framework however, we are fixing points in space. If we consider a framework in  $\mathbb{R}^2$  such as the one in Example 2.15, each node is at a determined distance from each other. Slightly changing the position of a node results in a different framework.

Frameworks are essential when it comes to studying the rigidity of structures. Considering real-world applications, a simple model of a building can be designed by considering what its framework would look like in  $\mathbb{R}^3$ . Investigating this framework allows for insight into how the building in question should be built.

Armed with the notion of a framework, we can begin thinking about how we might be able to change the shape of a framework.

## 2.3 Motions

When it comes to studying the rigidity of frameworks, we begin by understanding what it means for a framework to experience a *motion* (also known as a *flex*).

**Definition 2.16.** Suppose the configuration  $\mathbf{p} \in \mathbb{R}^d$  is on a differentiable smooth path parameterized by time  $t$ , denoted  $\mathbf{p}(t)$ . Then

- The position of node  $\mathbf{p}_i \in \mathbf{p}$  at any time  $t$  is given by  $\mathbf{p}_i(t) \in \mathbb{R}^d$ .
- The *initial position* of a node  $\mathbf{p}_i$  is  $\mathbf{p}_i(0) \in \mathbb{R}^d$
- The *initial configuration* of  $\mathbf{p}$  is

$$\mathbf{p}(0) = (\mathbf{p}_1(0), \dots, \mathbf{p}_n(0)) \in \mathbb{R}^{nd}$$

**Definition 2.17.** A *motion* of the framework  $(G, \mathbf{p})$  is a path  $\mathbf{p}(t)$  such that

$$|\mathbf{p}_i(t) - \mathbf{p}_j(t)| = |\mathbf{p}_i(0) - \mathbf{p}_j(0)|,$$

for all  $t$ , and for all  $ij \in E(G)$ .

In other words, a motion preserves the length of every edge in the framework at any point in time. A motion  $\mathbf{p}(t) = (\mathbf{p}_1(t), \dots, \mathbf{p}_n(t))$  is a *continuous motion* if each of the  $d$  coordinates of  $\mathbf{p}_i(t)$  is continuous in  $t$ . This simply means that the nodes of the configuration should follow an uninterrupted path in  $\mathbb{R}^d$  with respect to time.

There are certain motions that we can apply to a framework such that the distances between all the nodes are preserved as well. Such motions are known as *rigid motions*.

**Definition 2.18.** A *rigid motion* of the framework  $(G, \mathbf{p})$  is a path  $\mathbf{p}(t)$  such that

$$|\mathbf{p}_i(t) - \mathbf{p}_j(t)| = |\mathbf{p}_i(0) - \mathbf{p}_j(0)|,$$

for all  $t$ , and for all  $i, j \in V(G)$

From this definition, we can see that there are only two ways in which a motion can be classed as a rigid motion.

- *Translating* the entire framework across  $\mathbb{R}^d$  preserves the distances between each node, and so it is a rigid motion.
- *Rotating* the entire framework about a point also preserves the distances between each node. Thus, it is a rigid motion as well.

In order to formalize this, we let the matrix  $\mathbf{Q}(t)$  describe a rotation of a node in  $\mathbb{R}^d$ , where  $\mathbf{Q}(0)$  is the identity matrix, and we let  $\mathbf{w}(t)$  be a translation of a node in  $\mathbb{R}^d$ , where  $\mathbf{w}(0)$  is the zero vector. Then we can classify all rigid motions as paths of each node  $\mathbf{p}_i(t)$  that have the form

$$\mathbf{p}_i(t) = \mathbf{Q}(t)\mathbf{p}_i + \mathbf{w}(t)$$

Thus, rotating and translating a framework in any order is always a rigid motion.

**Example 2.19.** Consider the framework with 3 nodes, labelled 1, 2, 3, and two edges, 12, and 23. Apply the motion

$$p_i(t) = \begin{cases} \cos(t) - \sin(t), & \text{if } i = 3 \\ p_i(0), & \text{if } i = 1, 2 \end{cases}$$

to this framework. A diagram of what happens for times  $t = 0$ , and  $t = 1$  is shown below

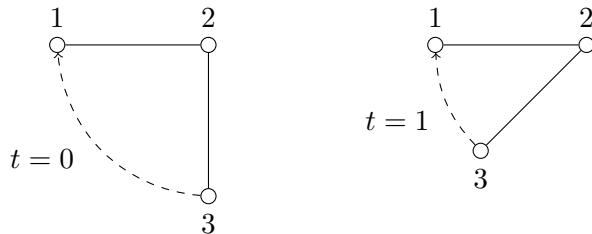


Figure 2.1: A framework where the edge 23 can swing about 2.

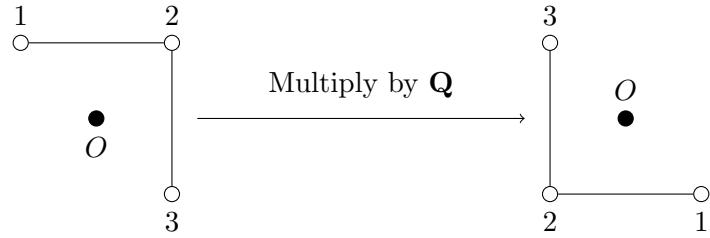
As all the edge lengths are preserved, this is a motion. However, the distance between nodes 1 and 3 changes as  $t$  changes. Therefore, this is not a rigid motion.

**Example 2.20.** Consider the matrix

$$\mathbf{Q} = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$$

This results in a rotation of each node by  $180^\circ$  about the origin  $O$  in  $\mathbb{R}^2$ . Applying this to a framework at  $t = 0$  in Example 2.19, we obtain

As all the edge lengths, as well as the distances between each node are preserved, this is a rigid motion.



## 2.4 Rigidity

As we have seen, rigid motions maintain not only the lengths of every edge in a framework, but they also fix the distance between each node as well. At this stage, we can question what frameworks can only be subject to rigid motions. Is there a framework such that the only way to move it is if we move *all* of it?

The answer is yes, and such frameworks are said to be *rigid*.

**Definition 2.21.** A framework  $(G, \mathbf{p})$  is rigid if every continuous motion  $\mathbf{p}(t) = (\mathbf{p}_1(t), \dots, \mathbf{p}_n(t))$  is a rigid motion. If a framework is not rigid, it is said to be *flexible*.

**Example 2.22.** These two frameworks are rigid.

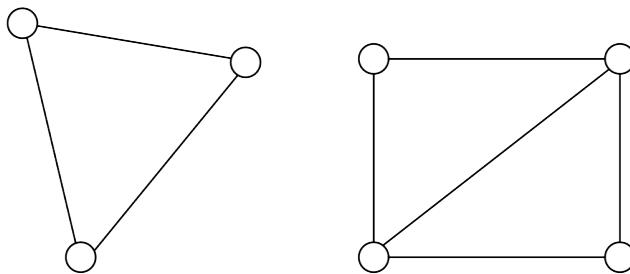


Figure 2.2: Two rigid frameworks in  $\mathbb{R}^2$

However, the following framework is not. We can apply a motion to the two nodes at the top such that the edge lengths are preserved, but the distance between the diagonal nodes change, violating the definition of rigidity.

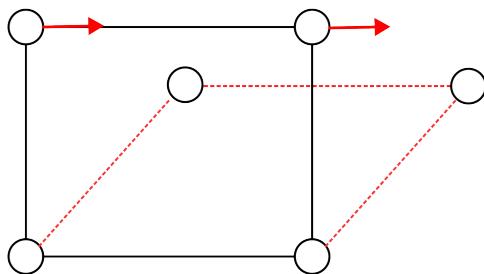


Figure 2.3: A framework that is not rigid in  $\mathbb{R}^2$

Another equivalent definition that was proved by Asimow and Roth [1] that uses ideas of the congruency of frameworks.

**Definition 2.23.** Consider two frameworks  $(G, \mathbf{p})$  and  $(G, \mathbf{q})$  where  $\mathbf{p} = (\mathbf{p}_1, \dots, \mathbf{p}_n)$  and  $\mathbf{q} = (\mathbf{q}_1, \dots, \mathbf{q}_n)$ .

- $\mathbf{q}$  is *equivalent* to  $\mathbf{p}$  if  $|\mathbf{p}_i - \mathbf{p}_j| = |\mathbf{q}_i - \mathbf{q}_j|$  for all  $ij \in E(G)$ .
- $\mathbf{q}$  is *congruent* to  $\mathbf{p}$  if  $|\mathbf{p}_i - \mathbf{p}_j| = |\mathbf{q}_i - \mathbf{q}_j|$  for all  $i, j \in V(G)$ .

To put it another way, two frameworks  $(G, \mathbf{p})$  and  $(G, \mathbf{q})$  are equivalent if the edge lengths are the same for every pair of labelled nodes in  $(G, \mathbf{p})$ .

They are congruent if  $(G, \mathbf{p})$  can be obtained through a series of rotations and translations of  $(G, \mathbf{q})$ .

**Definition 2.24.** A framework  $(G, \mathbf{p})$  on  $n$  nodes is *rigid* if there exists an  $\epsilon > 0$  such that for every configuration  $\mathbf{q}$ , where  $(G, \mathbf{q})$  is equivalent to  $(G, \mathbf{p})$ , satisfies

$$|\mathbf{p}_i - \mathbf{q}_i| < \epsilon$$

for all  $i \in V(G)$ , and is congruent to  $(G, \mathbf{p})$ .

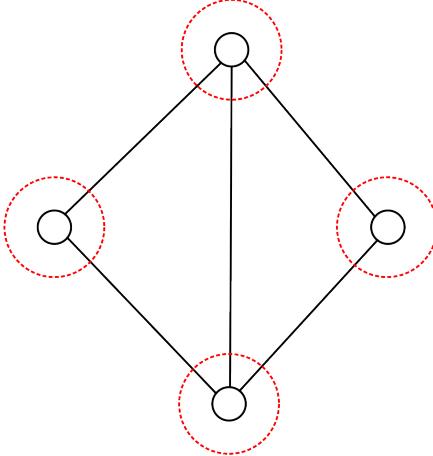


Figure 2.4: A framework  $\mathbf{p}$  with an epsilon bound (red) for every node

In Figure 2.4, we visualise what an epsilon bound for every node looks like. If we create a configuration  $\mathbf{q}$  such that it is equivalent to the framework  $\mathbf{p}$  in the figure, and the nodes of  $\mathbf{q}$  fall within the epsilon bounds of each node in  $\mathbf{p}$ , then we are able to conclude that with some rigid motion, we are able to achieve congruence between  $\mathbf{p}$  and  $\mathbf{q}$ .

In such scenarios, equivalence implies congruence.

#### 2.4.1 Infinitesimal Rigidity

In some sense, the definition of rigidity seems quite loose. Depending on the material used to construct the edges of the framework, we may be able to deform the edges.

Looking at Figure 2.5, if we consider the dotted edge to be a piece of string tied between two fixed nodes, then we can apply some force and deform the structure. Although the initial framework is rigid, it somehow doesn't 'feel' rigid.

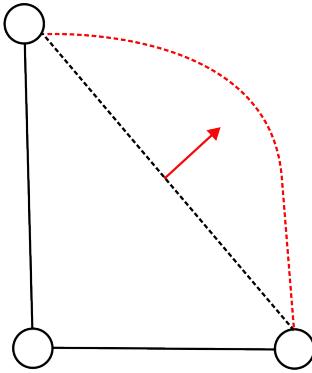


Figure 2.5: A framework with all nodes pinned, and an edge that can be displaced.

In order to strengthen the definition of rigidity, let us first start with an observation described in greater detail by Graver in his book “Counting on Frameworks” [4].

Suppose that the configuration  $\mathbf{p}$  is on a smooth differentiable path parameterized by time  $t$ . Consider a node of the configuration  $\mathbf{p}_i$ . Then its position is given by  $\mathbf{p}_i(t)$  and as  $\mathbf{p}_i$  is on a differentiable path, its derivative is well-defined.

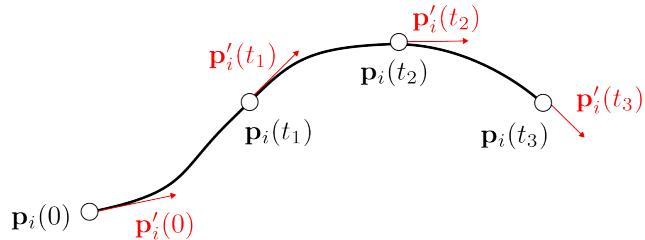


Figure 2.6: Node  $\mathbf{p}_i$  at various positions on path  $\mathbf{p}_i(t)$  for various times  $t$ . The instantaneous velocities at each position  $\mathbf{p}'_i(t)$  are marked in red

Taking the derivative of  $\mathbf{p}_i(t)$  with respect to time gives us the instantaneous velocity  $\mathbf{p}'_i(t)$  of the node  $\mathbf{p}_i$  at time  $t$ .

The velocities  $\mathbf{p}'_i(t)$  become vital when studying infinitesimal rigidity. To motivate this, let us consider a framework in  $\mathbb{R}^2$ .

In Figure 2.7, we have a framework with three nodes, each defined by coordinates  $(x_i(t), y_i(t))$  as well as their corresponding velocities given by  $(x'_i(t), y'_i(t))$  for  $i = 1, 2, 3$ . As each point is fixed in the plane, the distances between each pair of nodes are constants, given as  $l_i$  for  $i = 1, 2, 3$ .

Therefore, we know that

$$\begin{aligned}(x_1(t) - x_2(t))^2 + (y_1(t) - y_2(t))^2 &= l_3^2 \\ (x_1(t) - x_3(t))^2 + (y_1(t) - y_3(t))^2 &= l_1^2 \\ (x_2(t) - x_3(t))^2 + (y_2(t) - y_3(t))^2 &= l_2^2\end{aligned}$$

Now, if we differentiate each equation with respect to  $t$ , we get

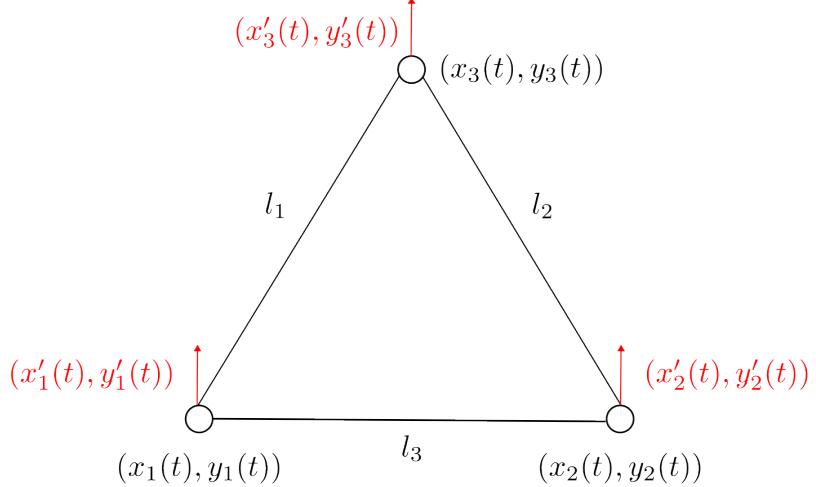


Figure 2.7: A framework in  $\mathbb{R}^2$ , with some instantaneous velocity vectors at each node given in red.

$$\begin{aligned} 2(x_1 - x_2)(x'_1 - x'_2) + 2(y_1 - y_2)(y'_1 - y'_2) &= 0 \\ 2(x_1 - x_3)(x'_1 - x'_3) + 2(y_1 - y_3)(y'_1 - y'_3) &= 0 \\ 2(x_2 - x_3)(x'_2 - x'_3) + 2(y_2 - y_3)(y'_2 - y'_3) &= 0 \end{aligned}$$

where we drop the dependence on  $t$  for notational convenience. By factoring out the 2, this set of equations can be written as

$$\begin{aligned} (x_1 - x_2, y_1 - y_2) \cdot (x'_1 - x'_2, y'_1 - y'_2) &= 0 \\ (x_1 - x_3, y_1 - y_3) \cdot (x'_1 - x'_3, y'_1 - y'_3) &= 0 \\ (x_2 - x_3, y_2 - y_3) \cdot (x'_2 - x'_3, y'_2 - y'_3) &= 0 \end{aligned}$$

Or more simply

$$(x_i - x_j, y_i - y_j) \cdot (x'_i - x'_j, y'_i - y'_j) = 0$$

for all  $i, j = 1, 2, 3$

This observation allows us to impose conditions such that edge lengths and node distances are preserved when velocities are applied to a node, enabling us to study rigidity in even finer detail. With this in mind, let us formalize what we have just seen.

**Definition 2.25.** Let  $(G, \mathbf{p})$  be a framework where each node is on a differentiable smooth path parameterized by  $t$  such that the position of each node is given by  $\mathbf{p}_i(t)$  for all  $i \in V(G)$ . Then *instantaneous velocity* of the node  $\mathbf{p}_i(t)$  is given by

$$\mathbf{p}'_i(t) = \frac{d\mathbf{p}_i(t)}{dt}$$

for all  $t$ .

The instantaneous velocity of a node allows us to consider what happens to a framework as we attempt to deform it by applying some motion or force to every node. If we want a structure to be rigid in the conventional sense, then we enforce a condition such that the edges of the framework do not deform when such velocities are applied to the nodes of a framework.

**Definition 2.26.** Let  $(G, \mathbf{p})$  be a framework parameterized by  $t$ .

- An *infinitesimal motion* of  $(G, \mathbf{p})$  is given by  $(\mathbf{p}_i - \mathbf{p}_j) \cdot (\mathbf{p}'_i - \mathbf{p}'_j) = 0$  for all  $ij \in E(G)$ .
- An infinitesimal motion is an *infinitesimal rigid motion* of  $(G, \mathbf{p})$  if  $(\mathbf{p}_i - \mathbf{p}_j) \cdot (\mathbf{p}'_i - \mathbf{p}'_j) = 0$  for all  $i, j \in V(G)$ .

Therefore, we obtain a system of equations, where the  $(\mathbf{p}'_i - \mathbf{p}'_j)$  terms are unknown, and  $(\mathbf{p}_i - \mathbf{p}_j)$  terms form our coefficients.

As we saw when considering Figure 2.7, an infinitesimal motion of a framework is the assignment of an instantaneous velocities  $\mathbf{p}'_i$  to nodes  $\mathbf{p}_i$  such that the length of the edge  $|\mathbf{p}_i - \mathbf{p}_j|$  remains constant for all edges  $ij \in E(G)$ . Analogously to before, if we can apply infinitesimal motions to the framework such that it preserves the distance between every node, then we have an infinitesimal rigid motion.

We have already seen that for a framework to be rigid in space, all of its motions must be rigid motions. A similar conclusion can be drawn here as well.

**Definition 2.27.** Let  $(G, \mathbf{p})$  be a framework parameterized by  $t$ . Then  $(G, \mathbf{p})$  is *infinitesimally rigid* if all of its infinitesimal motions are infinitesimal rigid motions.

If a framework is not infinitesimally rigid, it is *infinitesimally flexible*.

**Example 2.28.** Let us look at a framework that is infinitesimally flexible.

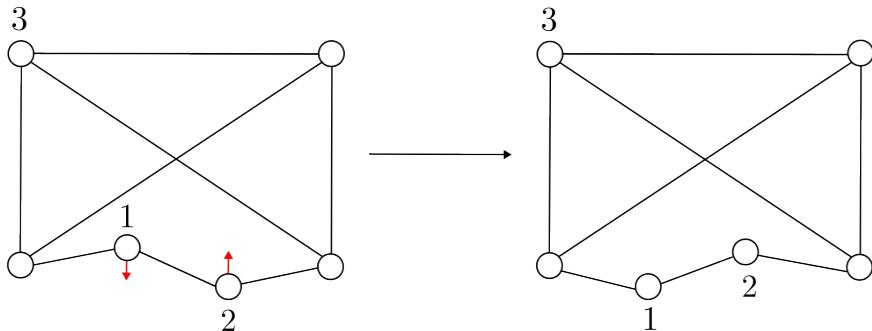


Figure 2.8: A framework that is not infinitesimally rigid

By applying instantaneous velocities to nodes 1 and 2 as shown in the framework on the left of Figure 2.8, we restructure the framework such that all the edge lengths are kept constant. This is shown in the framework on the right. However, the distance between nodes 1 and 3 have been altered. Therefore, this framework is not infinitesimally rigid.

Now consider the framework in Figure 2.9. As we will soon see, triangular frameworks are infinitesimally rigid by a process known as a Henneberg construction of the first kind [9]. So this means that the framework can not be deformed by applying instantaneous velocities to nodes 1, 2, 3 or 5.

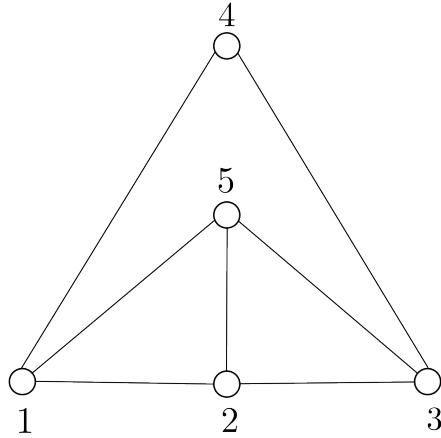


Figure 2.9: An infinitesimally rigid framework

As the only way we can possibly deform this framework is by applying some velocity to 4, let us displace 4 by an infinitesimal distance, say to the left without loss of generality. As our edges are of a fixed length, this will require edge 34 to ‘pull’ node 3 upwards with the motion. As triangles 235 and 125 are known to be rigid, this causes the entire framework to rotate in an anti-clockwise fashion.

We already know that rotations are (infinitesimal) rigid motions, and so any displacement of the node 4 results in an infinitesimal rigid motion. Therefore, we conclude that the framework in Figure 1.5 is infinitesimally rigid.

#### 2.4.2 Generic Rigidity

We close this chapter on one last interesting way to classify rigidity. When studying configurations, we are interested in those that have nothing ‘special’ about the way the points are arranged. Such configurations are known to be *generic*.

To begin, we define a simpler result in order to get a feel for what we mean by not ‘special’.

**Definition 2.29.** Let  $\mathbf{p} \in \mathbb{R}^d$  be a configuration. Then  $\mathbf{p}$  is in *general position* if any  $d + 1$  number of points do not lie in an  $d - 1$  dimensional affine space for  $d > 0$ .

Don’t worry about what an affine space is as it is beyond the scope of this project! The definition essentially says that for a set of points to be in general position, we must **not** have:

- Any two points that coincide at the same point when  $d = 1$ .
- Any three points that lie on the same line when  $d = 2$ .
- Any four points that lie on the same plane when  $d = 3$ .

And the pattern continues. This gives us a set of points that are spread out, making them more interesting to study.

Genericity is a stronger condition than that of general position. The definition is as follows.

**Definition 2.30.** A configuration  $\mathbf{p}$  is *generic* if the only polynomial with coefficients from  $\mathbb{Q}$  that the coordinates of each point in  $\mathbf{p}$  satisfies is the zero polynomial.

A couple of examples of point-sets that are not generic by this definition include four points on a circle and a pair of points on a line that has a rational slope [3].

Finally, we conclude with what it means to be generically rigid.

**Theorem 2.31.** *If  $(G, \mathbf{p})$  is an infinitesimally rigid framework in  $\mathbb{R}^d$ , then  $(G, \mathbf{q})$  is infinitesimally rigid for any generic configuration  $\mathbf{q}$ .*

This is an extremely powerful result. If we can find a configuration  $\mathbf{p} \in \mathbb{R}^d$ , generic or not, such that  $(G, \mathbf{p})$  is infinitesimally rigid, then Definition 2.31 implies that for *any* generic configuration  $\mathbf{q}$ , we can find a framework with the same underlying graph  $G$ , such that it is also infinitesimally rigid.

**Definition 2.32.** A configuration  $(G, \mathbf{p})$  is *generically rigid* in  $\mathbb{R}^d$  if it is infinitesimally rigid for any one configuration  $\mathbf{p}$ .

With that, we conclude this introductory exploration into the definitions of rigidity. These concepts form the basis of everything that is to come in later chapters.

Our current objective is unravelling the multitude of theorems that surround the study of rigid graphs. The goal is to collect as many tools revolving around rigidity and circle packings as possible. By learning and invoking these ideas, we will be ever closer to understanding the structure of the special case of packings this project focuses on.

# Chapter Three

## Exploring Rigidity & Circle Packings

So far, we have seen the various definitions of rigidity that we can use to identify whether a given framework is rigid. However, we have no methods or techniques we can use in order to *test* for the rigidity of a framework. We also don't know how the various definitions of rigidity get along with each other. If a framework is rigid, does it mean its infinitesimally rigid? Can we test for infinitesimal rigidity in some way?

In addition to these questions, we will also introduce circle packings, and discuss how they come about. As rigidity can't be tested merely by observation, testing the rigidity of a circle packing arises from using a few techniques seen earlier. For the remainder of this project, we will be working exclusively in  $\mathbb{R}^2$  unless specified otherwise.

### 3.1 Degrees of Freedom

This section has been adapted from Graver's book [4].

To understand what it means for a framework to have  $x$  number of *degrees of freedom*, let us first go through some simple examples to gain an understanding of what a degree of freedom is.

- Consider a point in  $\mathbb{R}^2$ . It can move right or left along the  $x$ -axis, or up and down along the  $y$ -axis. As it can travel along either axis, we say that this point has two *degrees of freedom*.
- Now, if we consider a line segment  $l$  in the plane with two endpoints, we know that each endpoint has two degrees of freedom, giving us four degrees of freedom, but the line between them prevents the endpoints from moving independently of each other. So,  $l$  can move either along the  $x$  or  $y$ -axes, giving us two degrees of freedom. However, it can also rotate about some point in  $\mathbb{R}^2$ . Therefore, such a segment has three degrees of freedom.
- If we were to consider a point in  $\mathbb{R}^3$ , then as the point can travel along the  $x$ ,  $y$  or  $z$ -axes, it has three degrees of freedom.

At this point, we should be able to see what a degree of freedom means. It is essentially a count of how many ways we can move a structure (or frameworks in our case) around in space. Additionally, we can note that adding an edge between two points reduces the total degrees of freedom in the

system from four to three due to the constraint that the points can't move independently of each other.

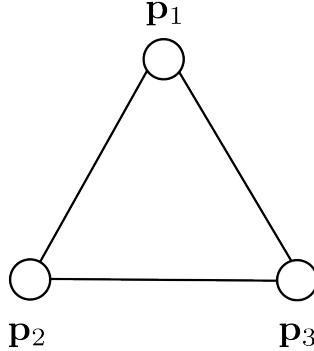


Figure 3.1: A framework on three nodes and three edges

**Example 3.1.** In order to find how many degrees of freedom the framework in Figure 3.1 has, we start with just three nodes, and then build up to the framework given:

- With only three nodes in the plane, and each node having two degrees of freedom, the total degrees of freedom the system has is  $3 \times 2 = 6$ .
- If we add an edge between  $p_1$  and  $p_2$  now, we lose a degree of freedom, giving us 3 degrees of freedom for the segment and 2 for the segment, totalling 5 degrees of freedom in the system.
- Adding the edge between  $p_1$  and  $p_3$ , we further reduce the number of ways the framework can move: 3 degrees of freedom for the first segment and 1 degree of freedom for rotation of the second segment about their common endpoint. At this stage, the structure has 4 degrees of freedom.
- Adding the last edge between  $p_2$  and  $p_3$  gives us the framework in question. We reduce the degrees of freedom by one yet again, giving us 3 degrees of freedom in total now.

Using Definition 2.21, we know that for any (infinitesimally) rigid body in the plane, we must only be allowed to translate it along the two axes, as well as only rotate it about some point.

**Theorem 3.2.** *In  $\mathbb{R}^2$ , any rigid body must have 3 degrees of freedom.*

## 3.2 The Rigidity Matrix

When given a framework, it can be quite difficult to tell whether it is rigid or not. This is where the *rigidity matrix* comes in. By finding a way to encode a framework into a matrix, we can then use tools from linear algebra to study and draw conclusions about the framework itself!

**Definition 3.3.** The *rigidity matrix*  $\mathbf{R}(G, \mathbf{p})$  of a  $d$ -dimensional framework  $(G, \mathbf{p})$  is a matrix that contains the system of equations described in Definition 2.26. It is a  $|E(G)| \times d|V(G)|$  matrix, where the rows are indexed by the edges of  $G$ , and sets of  $d$  consecutive columns correspond to a single node of  $(G, \mathbf{p})$ .

If there is an edge between nodes  $\mathbf{p}_i$  and  $\mathbf{p}_j$  when  $d = 2$ , then the non-zero entries in the corresponding row of the matrix lie in columns corresponding to coordinates of each node. That is;

- Columns  $2i - 1$  and  $2i$  corresponding to the coordinates of node  $p_i$ .
- Columns  $2j - 1$  and  $2j$  corresponding to the coordinates of node  $p_j$ .
- 0 everywhere else.

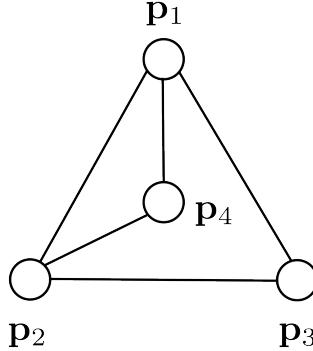


Figure 3.2: A framework on four nodes and five edges.

For example, consider the framework  $(G, \mathbf{p})$  in Figure 3.2.

As we are working in  $\mathbb{R}^2$ , we know that  $\mathbf{p}_i = (x_i, y_i)$ , for each  $i = 1, 2, 3, 4$ . Therefore, we can construct  $\mathbf{R}(G, \mathbf{p})$  as

$$\mathbf{R}(G, \mathbf{p}) = \left[ \begin{array}{ccccccccc} x_1 - x_2 & y_1 - y_2 & x_2 - x_1 & y_2 - y_1 & 0 & 0 & 0 & 0 \\ x_1 - x_3 & y_1 - y_3 & 0 & 0 & x_3 - x_1 & y_3 - y_1 & 0 & 0 \\ x_1 - x_4 & y_1 - y_4 & 0 & 0 & 0 & 0 & x_4 - x_1 & y_4 - y_1 \\ 0 & 0 & x_2 - x_3 & y_2 - y_3 & x_3 - x_2 & y_3 - y_2 & 0 & 0 \\ 0 & 0 & x_2 - x_4 & y_2 - y_4 & 0 & 0 & x_4 - x_2 & y_4 - y_2 \end{array} \right] \quad \begin{array}{l} \text{Edges} \\ \hline \mathbf{p}_1\mathbf{p}_2 \\ \mathbf{p}_1\mathbf{p}_3 \\ \mathbf{p}_1\mathbf{p}_4 \\ \mathbf{p}_2\mathbf{p}_3 \\ \mathbf{p}_2\mathbf{p}_4 \end{array}$$

The rows of  $\mathbf{R}(G, \mathbf{p})$  correspond to the edges in  $(G, \mathbf{p})$ , and each consecutive set of two columns correspond to the  $x$  and  $y$  coordinates of each node in  $\mathbf{p}$ , leading to the creation of  $2 \times 4 = 8$  columns and 5 rows in  $\mathbf{R}(G, \mathbf{p})$ .

**Example 3.4.** Let the framework  $(G, \mathbf{p})$  be the one in Figure 3.2. If we give the nodes of  $(G, \mathbf{p})$  positions in  $\mathbb{R}^2$ , we can compute its numeric rigidity matrix  $\mathbf{R}(G, \mathbf{p})$ .

Suppose that  $\mathbf{p}_1 = (1, 2)$ ,  $\mathbf{p}_2 = (0, 0)$ ,  $\mathbf{p}_3 = (2, 0)$ ,  $\mathbf{p}_4 = (1, 1)$ . Then, the rigidity matrix  $\mathbf{R}(G, \mathbf{p})$  is

$$\mathbf{R}(G, \mathbf{p}) = \left[ \begin{array}{cccccccc} 1 & 2 & -1 & -2 & 0 & 0 & 0 & 0 \\ -1 & 2 & 0 & 0 & 1 & -2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & -2 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & 0 & 0 & 1 & 1 \end{array} \right]$$

Now that we have the rigidity matrix, we can use it to determine whether a given d-dimensional framework is infinitesimally rigid or not. Recalling Definition 2.26, we can see that the space of infinitesimal motions of a framework  $(G, \mathbf{p})$  is equal to the kernel of  $R(G, \mathbf{p})$ , denoted  $\text{Ker}(\mathbf{R})$ . By using a result from Linear Algebra known as the **Rank-Nullity Theorem**, we see that

$$\text{rank}(\mathbf{R}) + \dim(\text{Ker}(\mathbf{R})) = d|V|$$

As the space of infinitesimal rigid motions govern the number of ways a framework can move in space, we can deduce that a useful result.

**Lemma 3.5.** *Let  $(G, \mathbf{p})$  be a framework, and  $\mathbf{R}(G, \mathbf{p})$  be its rigidity matrix. Then  $(G, \mathbf{p})$  has  $\dim(\text{Ker}(\mathbf{R}))$  degrees of freedom.*

As we're concerned with frameworks in  $\mathbb{R}^2$ , using Theorem 3.2, we know that rigid frameworks must have 3 degrees of freedom. From Definition 2.27, we deduce the following theorem.

**Theorem 3.6.** *Let  $(G, \mathbf{p})$  be a framework and  $\mathbf{R}(G, \mathbf{p})$  be its rigidity matrix. Then,  $(G, \mathbf{p})$  is infinitesimally rigid in  $\mathbb{R}^2$  if and only if*

$$\text{rank}(\mathbf{R}) = 2|V| - 3$$

We have now successfully used techniques from linear algebra in order to test for infinitesimal rigidity of a framework! This result was proved by Asimow and Roth in their paper [1], and is going to be a pertinent to this project. Being able to analyze frameworks using matrices enables us to computationally investigate properties of the matrix, allowing for a speedy analysis of a given framework.

### 3.3 Constructing Rigid Structures

Through an algorithmic process of adding nodes and deleting edges, we can create frameworks such that they are rigid in nature. This is done using the Henneberg construction [9]. However, in order to end up with a rigid framework, we must start with a graph known as a *Laman Graph*.

**Definition 3.7.** Let  $G = (V, E)$  be a connected graph, and let  $H$  be a subgraph of  $G$ . Then  $G$  is known as a *Laman graph* if

- The edge set of  $G$  has size  $|E(G)| = 2|V(G)| - 3$ .
- For any subgraph  $H$  defined on  $k \leq |V(G)|$  vertices,  $|E(H)| \leq 2k - 3$ .

as defined in [10].

Such graphs are named after Gerard Laman, who studied the rigidity of graphs in the 1970s. Now, a question we can ask at this stage is if we can find graphs that are rigid on the smallest number of edges possible. That is, are there rigid graphs such that if we take away an edge, it causes the structure to lose its rigidity? Such graphs are known to be *minimally rigid*.

As this project revolves around minimally rigid graphs, it is important to learn what graphs are minimally rigid, and if there's a way to identify which graphs are classed as minimally rigid.

**Theorem 3.8.** *A graph  $G$  is minimally rigid in  $\mathbb{R}^2$  if and only if  $G$  is a Laman graph [12].*

Therefore, a graph  $G$  is minimally rigid if and only if  $G$  has  $2n - 3$  edges, where  $n = |V(G)|$ . This is known as the Geiringer–Laman theorem, as it was first proved by Hilda Pollaczek-Geiringer in 1927 [15], and then independently by Laman in 1970 [12].

Given such a characterization, we can now talk about ways to construct minimally rigid graphs, of which there are two we focus on. They are known as the *Henneberg constructions* [9].

**Definition 3.9.** The *Henneberg construction* in  $\mathbb{R}^2$  involves two processes in order to generate minimally rigid graphs.

- Type I: To an existing Laman graph  $G$ , insert a new vertex  $u$  and attach it to  $G$  by adding two edges between  $u$  and two distinct vertices in  $G$ .
- Type II: To an existing Laman graph  $G$ , delete an edge between two vertices  $v_1, v_2 \in V(G)$ , insert a new vertex  $u$ , and attach it to  $G$  by adding edges between  $u$  and  $v_1, v_2$  and one other vertex in  $G$ .

The resulting graph will be have  $2|V(G)| - 3$  edges, and is therefore minimally rigid by Theorem 3.8.

**Example 3.10.** A sequence of constructions using Henneberg Type I and Type II are shown.

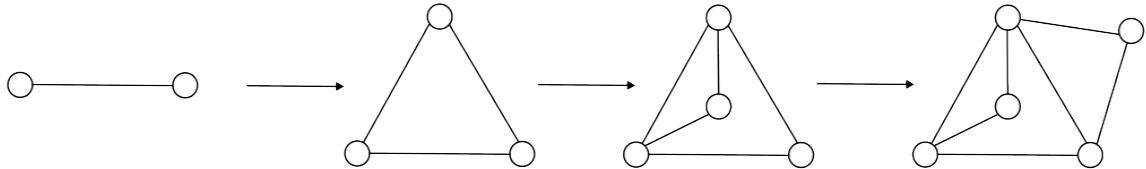


Figure 3.3: A series of constructions created solely using Henneberg Type I

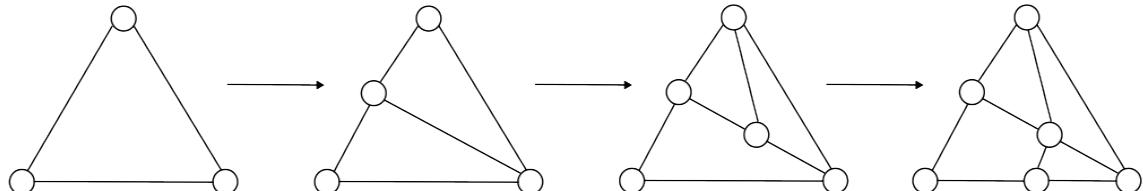


Figure 3.4: A series of constructions created solely using Henneberg Type II

By checking directly, we can verify that the graphs at each stage are indeed Laman graphs.

With the use of the rigidity matrix, or simply by observation, we can see that frameworks at each stage of the Henneberg constructions are infinitesimally rigid.

**Corollary 3.11.** *Let  $(G, \mathbf{p})$  be a minimally rigid framework, where  $G$  is a Laman graph. Then  $(G, \mathbf{p})$  is infinitesimally rigid.*

### 3.4 Rigidity vs Infinitesimal Rigidity

At this stage, we have a good understanding of what rigidity and infinitesimal rigidity are. We have seen ways to check whether a framework is infinitesimally rigid, and learned of a method to construct minimally rigid graphs. In this section, the aim is to learn how one relates with the other.

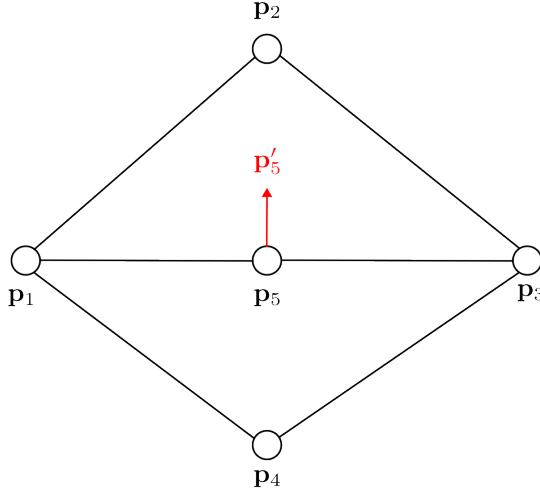


Figure 3.5: A rigid framework with an instantaneous velocity applied to  $p_5$  shown in red.

To motivate this, let us consider the framework  $(G, \mathbf{p})$  in Figure 3.5. As this framework is formed by joining two triangles together, we know it must be rigid as triangles are themselves rigid. The question is whether it is infinitesimally rigid.

Suppose we apply infinitesimal velocities of magnitude 0 to nodes  $p_1$ ,  $p_2$ ,  $p_3$ ,  $p_4$ , and we apply the infinitesimal velocity  $\mathbf{p}'_5 \neq 0$  to node  $p_5$ . We first verify whether these instantaneous velocities make an infinitesimal motion by checking  $(\mathbf{p}_i - \mathbf{p}_j) \cdot (\mathbf{p}'_i - \mathbf{p}'_j) = 0$  for all  $ij \in E(G)$ .

As each velocity vector is 0 except for  $\mathbf{p}'_5$ , it follows that this is true for all the edges in  $(G, \mathbf{p})$ .

Now, consider  $(\mathbf{p}_5 - \mathbf{p}_2) \cdot (\mathbf{p}'_5 - \mathbf{p}'_2)$ . The vector  $(\mathbf{p}_5 - \mathbf{p}_2)$  is simply the line through the two nodes, which is non-zero, and  $(\mathbf{p}'_5 - \mathbf{p}'_2) = \mathbf{p}'_5$  as  $\mathbf{p}'_5$  is non-zero by assumption. As the vectors  $(\mathbf{p}_5 - \mathbf{p}_2)$  and  $\mathbf{p}'_5$  are not perpendicular to each other, it follows that,

$$(\mathbf{p}_5 - \mathbf{p}_2) \cdot (\mathbf{p}'_5 - \mathbf{p}'_2) \neq 0$$

Therefore, this framework is not infinitesimally rigid.

This shows us that even though a framework might be rigid, it may not be infinitesimally rigid. Applying instantaneous velocities at a carefully selected node can very well deform a rigid structure. The converse however is true.

**Theorem 3.12.** *Let  $(G, \mathbf{p})$  be a infinitesimally rigid framework. Then  $(G, \mathbf{p})$  is rigid.*

The proof of this theorem relies on a lot of heavy mathematics which is beyond the scope of this project, and it can be found in Asimow and Roth's paper [1]. Therefore, infinitesimal rigidity is a stronger property than rigidity.

### 3.5 Circle Packings

In this last section, we finally come to circle packings and how they are defined. Additionally, we also learn how we can identify ways in which we can use the methods described earlier in order to test the rigidity of a packing.

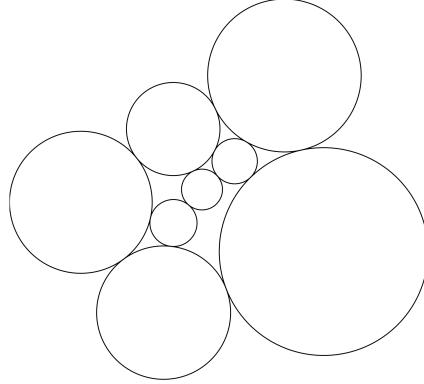


Figure 3.6: A circle packing with 8 circles

Circle packings are configurations of circles satisfying preassigned patterns of tangency. They are designed such that each pair of circles may either touch tangentially, or not at all. Circles are not allowed to overlap or be contained in one another as well. An example of a circle packing is shown in Figure 3.6. As we can see, not all of the circles have the same radius. This is necessary as the circles must be able to adjust their radii in order to fit tightly and satisfy any constraints imposed on them.

**Definition 3.13.** A *circle packing* is a configuration of circles  $P = (C_i)$ , where  $i \in \mathbb{Z}^+$ , in the plane  $\mathbb{R}^2$  such that any two distinct circles in  $P$  have disjoint interiors.

That is, distinct circles in  $P$  may be tangent, but may not overlap.

Given a circle packing, we must now find a way to analyze its properties. Thankfully, there is a natural way to encode the qualities of a packing  $P$  into a graph  $G(P)$ , which we call the circle packing's *contact graph*.

**Definition 3.14.** Let  $P = (C_i)$  be a circle packing, where  $i \in \mathbb{Z}^+$ . Then, the *contact graph*  $G(P)$  of the packing  $P$  is a graph where the vertex set is the set of circles in  $P$ , and an edge joins two vertices  $u, v \in G(P)$  if and only if the circles corresponding to the vertices are tangential in  $P$ .

In other words, an edge is added between two vertices in a contact graph if and only if the corresponding circles make *contact*.

We now have a way to encode a circle packing's properties into a graph! By observation, we can conclude that the contact graph  $G(P)$  is always going to be a planar graph, as if edges were to cross over, this would imply that the corresponding circles in  $P$  overlap each other. Studying a packing's

contact graph allows us to invoke tools and methods we've developed so far in order to investigate the rigidity of the packing in question.

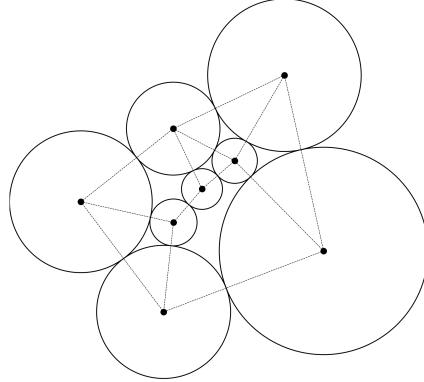


Figure 3.7: The circle packing from Figure 3.6 with its contact graph visualized

For a circle packing  $P$ , we now state that  $P$  is infinitesimally rigid if its contact graph  $G(P)$  is infinitesimally rigid [2]. As rigidity does not imply infinitesimal rigidity, checking for infinitesimal rigidity is sufficient to verify the rigidity of the packing itself.

In Connelly, Gortler, and Theran's paper [2], the following theorem is proved.

**Theorem 3.15.** *Let  $P = (C_i)$  be a circle packing, where  $i = 1, 2, \dots, n$ . Then if  $P$  has  $2n - 3$  contacts, it is rigid and infinitesimally rigid. If the number of contacts is fewer, then  $P$  is flexible and infinitesimally flexible.*

This follows from investigating the contact graph  $G(P)$  of a packing  $P$ . If the packing has  $n$  circles, then there must be  $n$  vertices in  $G(P)$ . Furthermore, if there are  $2n - 3$  contacts in  $P$ , then there must be  $2n - 3$  edges in  $G(P)$ .

By Definition 3.7, we know that  $G(P)$  is a Laman graph, and by invoking Corollary 3.11, we deduce that  $G(P)$  is infinitesimally rigid, and therefore rigid.

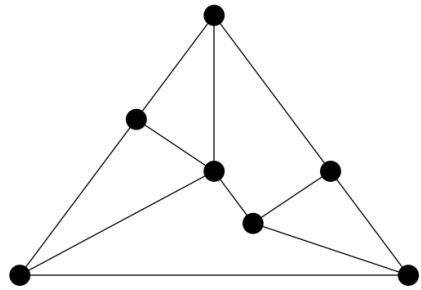
Finally, we need some notion of isomorphism between circle packings and graphs. Packings themselves are not unique, as there are various ways to create a circle packing originating from same graph, illustrated in Figure 3.8. To see this, we can investigate the cycles in the contact graphs of each packing.

Inspecting the graphs in Figure 3.8, we note that the original graph (a) has 3 cycles of length three. Now, comparing the contact graphs (d) and (e) to graph (a), we note that the contact graph in (d) has 3 cycles of length three, while the contact graph in (e) has only 2 cycles of length three. As the number of cycles of length three in (e) don't match those in (a), (a) and (e) must be non-isomorphic.

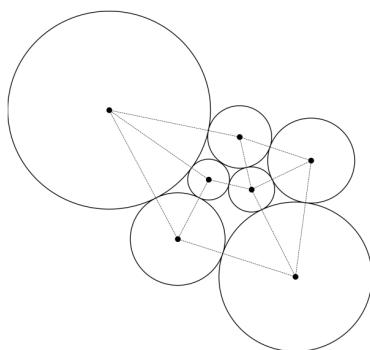
To this end, we use a theorem that was proved by Paul Koebe in 1936, now known as the *Circle Packing Theorem* [11].

**Theorem 3.16.** *Every finite planar graph  $G$ , where  $G$  does not have multiple edges or loops between the same vertex, has a circle packing. That is, there exists a circle packing  $P = (C_i)$  such that  $G(P)$  is isomorphic to  $G$ .*

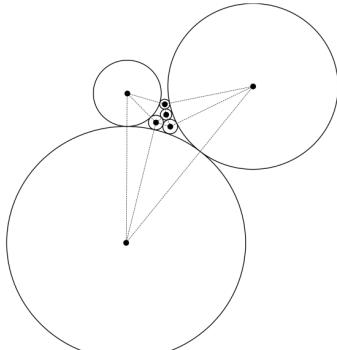
With that, we have everything we need to set up and investigate the problem this project focuses on! By delving deep and exploring the various avenues available to us in order to check the rigidity of a given framework, we've obtained a comprehensive list of techniques at our disposal. Now, equipped with an array of theorems, we can finally get to the heart of the problem at hand!



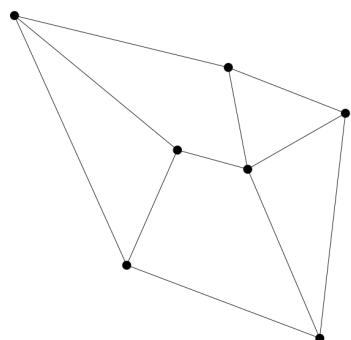
(a)



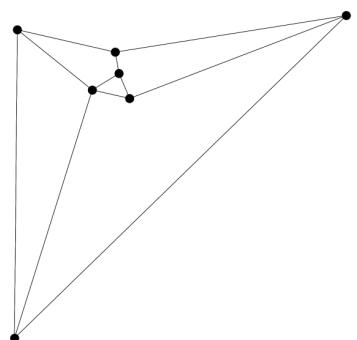
(b)



(c)



(d)



(e)

Figure 3.8: A graph can have various ways in which packings can be derived from it. (a) The initial graph  $G$  used to generate a circle packing. (b) A circle packing  $P_1$  with a contact graph isomorphic to  $G$ . (c) A circle packing  $P_2$  with a contact graph non-isomorphic to  $G$ . (d) The contact graph of  $P_1$ . (e) The contact graph of  $P_2$ .

## Chapter Four

# Rigidity of Minimally Rigid Circle Packings

If we were to relate this thesis to a video game, the previous chapters would be like the levels where we, the player, would need to get through in order to gain skills or powers. These levels would contain small, trivial enemies that we'd easily defeat and gain a set of valuable tools in order to progress the game.

This chapter then, would be the showdown between the main character and the final boss of the game! We'll need to call upon all of the experience gained, use every ounce of the strength developed and employ each weapon attained over the course of this journey in order to win this fight.

### 4.1 The Goal

As it's been stated a few times leading up to this point, the project focuses on figuring out whether every minimally rigid graph has a circle packing that is infinitesimally rigid itself. It is a question for which we (currently) have no answer to, and it is an interesting one in nature because although we know that a packing such that the contact graph is isomorphic to the original graph exists by Theorem 3.16, the contact graph isn't necessarily infinitesimally rigid, see Figure 4.1. Our task in this chapter is to understand whether this is true or not.

If we were to be pedantic, we would need to talk about the framework itself with respect to rigidity, and the graph which gives rise to this framework when talking about isomorphism. To avoid the confusion (and tediousness) this will cause, we shall use the words 'graph' and 'framework' interchangeably from now on.

We try to unpack this question computationally. By finding known, non-trivial examples for which the conjecture holds, we can be fairly confident that it is true. Thus, the two key players here are going to be modules written in Python called `Rigidity.py` and `Circle_Packing.py`.

Titled aptly, the module `Rigidity` takes in a graph and verifies whether it is infinitesimally rigid, and `Circle_Packing` takes a planar graph and attempts to find its circle packing using optimization methods. The graphs we'll be analyzing are all Laman graphs on  $n \in [3, \dots, 10]$  vertices.

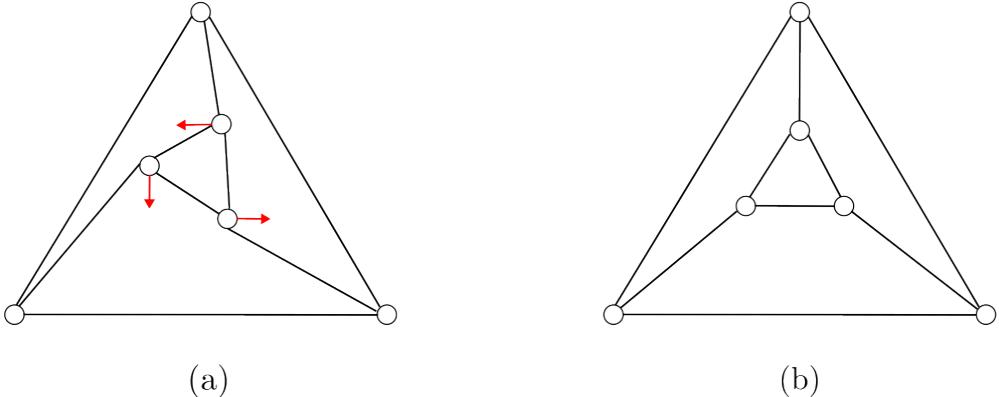


Figure 4.1: Two isomorphic frameworks. (a) Can be deformed by applying instantaneous velocities (marked in red) on the inner nodes. (b) An infinitesimally rigid framework isomorphic to the one in (a).

The generation of minimally rigid graphs on  $n = 3, 4, 5$  vertices can be easily done by hand using the Henneberg Constructions from Theorem 3.9, henceforth abbreviated to  $H_1$  and  $H_2$  for Type I and Type II respectively. Starting with a Laman graph on 3 vertices (a triangle):

- To obtain the Laman graphs on 4 vertices, we can note that applying  $H_1$  yields a graph isomorphic to the one we get when applying  $H_2$ . so there is only one Laman graph on 4 vertices.
- To obtain the Laman graphs on 5 vertices, we apply  $H_1$  and  $H_2$  in four combinations to the triangle;
  - Apply  $H_1$  twice
  - First  $H_1$ , and then  $H_2$
  - First  $H_2$ , and then  $H_1$
  - Apply  $H_2$  twice

However, we can notice that the graph obtained by applying  $H_1$  and then  $H_2$  is isomorphic to the graph we obtain when we apply  $H_2$  followed by  $H_1$ . So we conclude that there are only three Laman graphs on 5 vertices.

This is illustrated in Figures 4.2 and 4.3.

Planar graphs on vertices  $n > 5$  are generated using nauty [14], which allows for the generation of non-isomorphic graphs with a specified number of vertices and edges and saves them as a .g6 file. So for example, if we wanted to generate all planar graphs on 8 vertices with 13 edges, we would input

```
./geng 8 13 | ./planarg | gzip > planar-8-13.g6
```

into the terminal. This produces a file named `planar-8-13.g6`, which contains a list of non-isomorphic planar graphs with the required vertices and edges. At this point, we would invoke the two modules `Rigidity.py` and `Circle_Packing.py` to find the minimally rigid graphs and attempt to pack them.

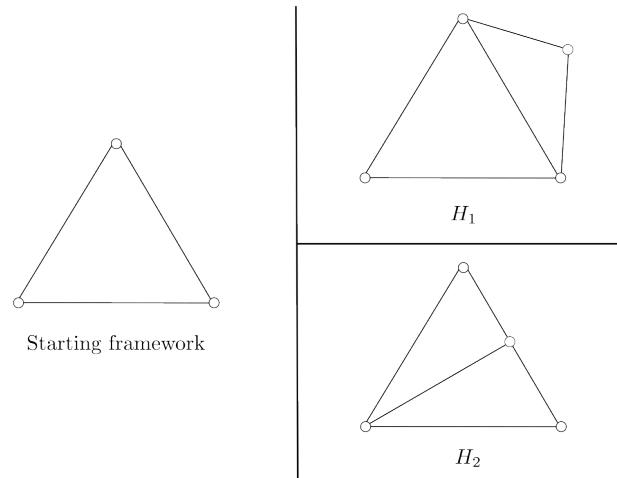


Figure 4.2: Two isomorphic Laman graphs on 4 vertices

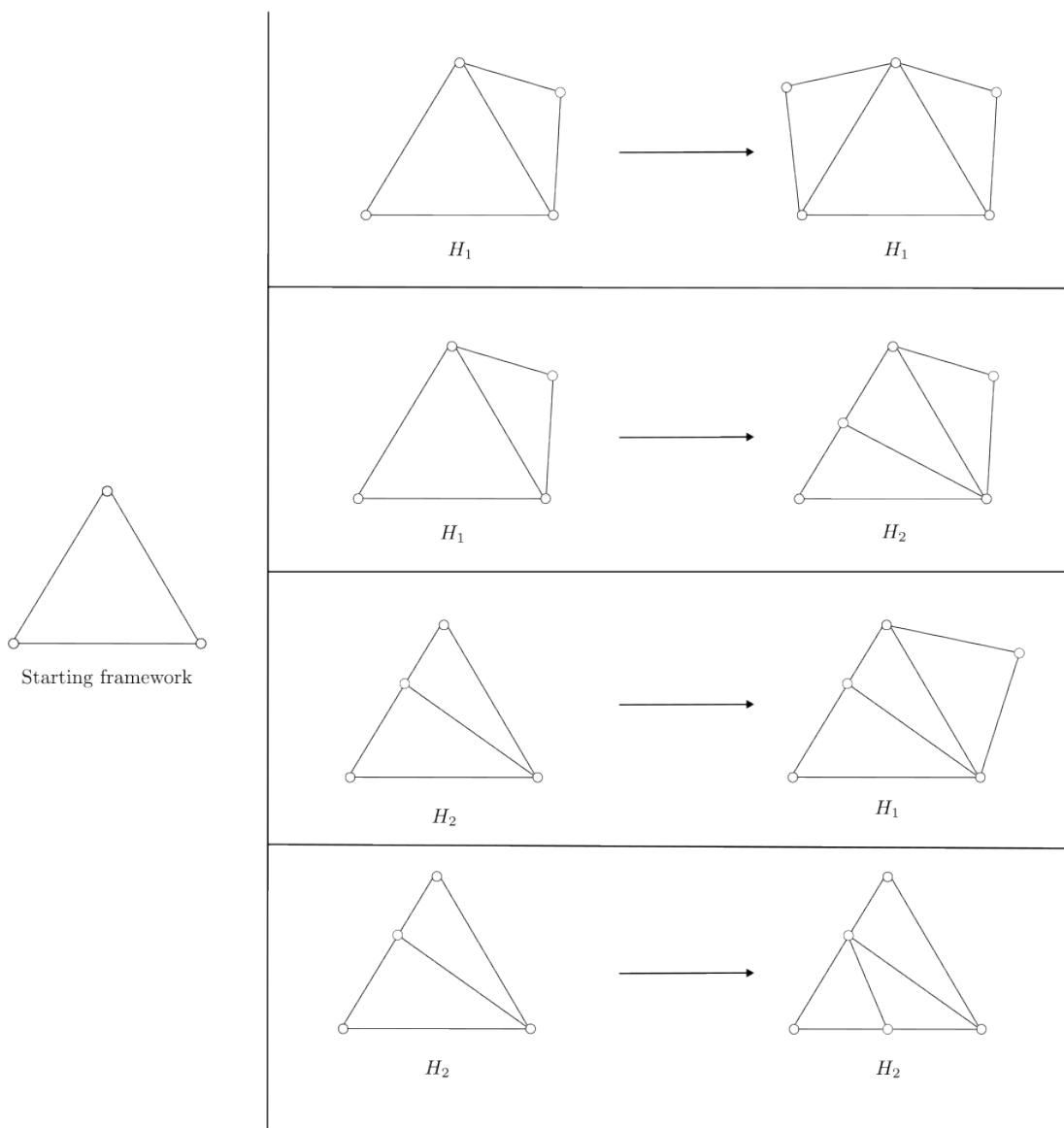


Figure 4.3: Four Laman graphs on 5 vertices. The graphs produced from  $H_1 \rightarrow H_2$  and  $H_2 \rightarrow H_2$  are isomorphic, so there are three non-isomorphic Laman graphs on 5 vertices.

Visually, we can represent what we're trying to do in Figure 4.4.

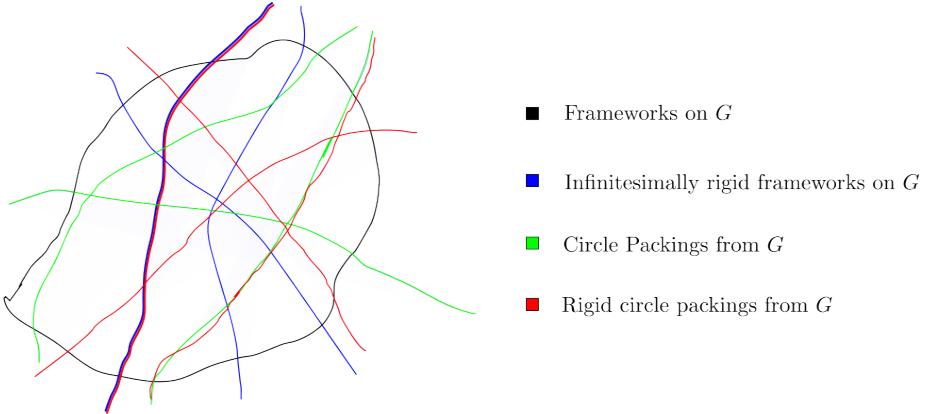


Figure 4.4: The space containing all frameworks on a graph  $G$  drawn as a black outline of the space, the frameworks on  $G$  that are infinitesimally rigid drawn as blue lines, the circle packings obtained from  $G$  drawn as green lines, and the rigid circle packings obtained from  $G$  drawn as red lines.

For a given planar graph  $G$  with  $n$  vertices and  $2n - 3$  edges, there can be a multitude of ways to draw frameworks for it. Referring to Figure 4.4, we can visualize this as the *space* of frameworks on  $G$ . Within this space, we have all the frameworks on  $G$ , some of which are infinitesimally rigid, given as blue lines.

Now, for any circle packing obtained from  $G$ , its underlying contact graph could either be rigid or not, and so we have green lines to represent any circle packing we can get from  $G$ . Red lines are used to denote what we are after, the rigid circle packings.

Thus, the task is to find where the blue and the red lines overlap! That is, we want to find an infinitesimally rigid planar graph such that it produces a circle packing whose contact graph is also infinitesimally rigid (as this implies rigidity). Such an overlap has been made bold in the Figure.

Now that we have an understanding of what it is we're trying to do, let us dive into how we go about doing it! By exploring each module individually, we gain an understanding of what is being done. After that, we can go ahead and use the modules in order to find the packings of interest.

## 4.2 Rigidity.py

The first module we visit is the one we will be using to check for the rigidity of a given graph. To do this, we will need to compute the rigidity matrix from Theorem 3.3, and then check whether its rank satisfies Theorem 3.6. From this point onwards,  $n$  shall denote the number of vertices (or nodes) in a graph (or framework), and  $m$  shall denote the number of edges in the graph.

The graphs we will be considering are made by an existing module in Python known as `networkx` [5]. This module allows for the study and visualization of graphs in Python. Along with this, to

create the matrices of interest, we need to use a module called `numpy` [8]. By using built-in structures and functions, such as arrays and functions related to analyzing arrays, we create our rigidity matrix as a `numpy` array and then use a function to compute its rank.

This module contains several functions designed to do a particular task, so we visit each of them in turn.

#### 4.2.1 Creating a Configuration

As touched upon before, a graph is an abstract mathematical object that can be drawn in a multitude of ways. There is no concept of ‘distance’, or the vertices having ‘coordinates’ in  $\mathbb{R}^2$ . In order to study the rigidity of the graph, we must first take the vertices and give them coordinates, effectively forming a configuration of points. This brings us to the first function in this module, called `G_configuration`.

The function `G_configuration` takes in a single argument `G`, a `networkx` graph, and begins by learning the number of vertices the graph has. For each vertex then, it randomly assigns an integer for  $x$ -coordinate and the  $y$ -coordinate from the interval  $[-2^{40}, 2^{40}]$ . **Sampling for coordinates from such a large range of numbers decreases the risk of bias, and increases the variation of points used**<sup>1</sup>. Finally, it returns two lists, one containing the randomly assigned  $x$ -coordinates and the other containing the randomly assigned  $y$ -coordinates.

Now, we have a number of points, each equipped with their own pair of  $(x, y)$  coordinates. The next step is to construct the rigidity matrix.

#### 4.2.2 Constructing the Rigidity Matrix

We define the function `rigidity_matrix` which takes in a graph `G`, and computes its rigidity matrix with the aid of the function `G_configuration`. Recalling Definition 3.3, the rigidity matrix in  $\mathbb{R}^2$  has  $m$  rows and  $2n$  columns. Knowing this, we can initialise a `numpy` array on  $m$  rows and  $2n$  columns, and have every entry in the array be 0.

The task at this stage is to fill in the appropriate entries of the matrix with the correct values. We use the notation  $x_i$  and  $y_i$  when talking about the  $x$ -coordinate and the  $y$ -coordinate of node  $i$  respectively. The computation of the entries of the rigidity matrix is done by using the `enumerate` function in Python. We keep track of the row index `i`, as well as the edge  $(u, v)$  with each iteration.

1. The element in row `i` and column  $2u$  is the  $x_u - x_v$ .
2. The element in row `i` and column  $2u+1$  is the  $y_u - y_v$ .
3. The element in row `i` and column  $2v$  is the  $x_v - x_u$ .
4. The element in row `i` and column  $2v+1$  is the  $y_v - y_u$ .

The function then returns the rigidity matrix with the modified elements.

---

<sup>1</sup>need a better explanation for why the range is so big

To test whether this function works as expected, we can try to compute the rigidity matrix of the framework given in Example 3.4. This framework has nodes with coordinates  $(1, 2)$ ,  $(0, 0)$ ,  $(2, 0)$  and  $(1, 1)$ .

By defining the graph  $G$  appropriately, and by tweaking the function `rigidity_matrix` slightly in order to take the specified coordinates rather than the random ones obtained from `G_configuration`, we can compute the rigidity matrix for the example.

```
In: print(rigidity_matrix(G))

Out: [[ 1.  2. -1. -2.  0.  0.  0.  0.] # edge (1,2)
      [ 0.  1.  0.  0.  0.  0. -1.] # edge (1,4)
      [-1.  2.  0.  0.  1. -2.  0.  0.] # edge (1,3)
      [ 0.  0. -1. -1.  0.  0.  1.  1.] # edge (2,4)
      [ 0.  0. -2.  0.  2.  0.  0.  0.]] # edge (2,3)
```

Comparing this to the rigidity matrix computed by hand in Example 3.4, we see that while the order of the edges considered has changed, the entries along each corresponding row have not. Thus, we can safely say the function `rigidity_matrix` computes the rigidity matrix for a given graph correctly.

### 4.2.3 Rank of the Rigidity Matrix

From Theorem 3.6, we know that the framework is infinitesimally rigid if and only if the rank of the rigidity matrix is equal to  $2n - 3$ . In order to obtain the rank of a matrix in Python, we make use of the `linalg` library within `numpy`.

To start, we define a function `check_rigidity` which takes a graph  $G$ , and calls on the `rigidity_matrix` function to get the rigidity matrix of  $G$ . From here, we use `np.linalg.matrix_rank` to compute the rank of the matrix. If the rank is equal to  $2n - 3$ , the function returns `True`, and `False` if the condition fails.

Considering the framework in Example 3.4, we know it is infinitesimally rigid by observation as it is a Laman graph on 4 vertices, and this is infinitesimally rigid by Corollary 3.11. By calling this framework  $G$ , let us print the rank of the rigidity matrix of  $G$ .

```
In: print(np.linalg.matrix_rank(rigidity_matrix(G)))

Out: 5
```

As  $(2 \times 4) - 3 = 5$ , we confirm that this framework is indeed infinitesimally rigid. Now, to verify whether `check_rigidity` works as expected, we print its result using  $G$  as the argument.

```
In: print(check_rigidity(G))
```

```
Out: True
```

Thus, we have written a function that now tells us whether a graph is infinitesimally rigid or not.

#### 4.2.4 Checking rigidity for a list of graphs

Using nauty to generate the graphs we need gives us a list containing all the non-isomorphic planar graphs on a specified number of vertices and edges. Therefore, the last function this module contains is one that filters this list for all the minimally rigid graphs that we want to analyze. In addition to this, we also filter out graphs that have a vertex of degree one or two.

If a graph has a vertex with degree one, then it is not rigid and will be discarded when the `check_rigidity` function is called. On the other hand, a vertex of degree two can be constructed using a Henneberg Type I construction, we can pack this vertex trivially. Such graphs are not very interesting to look at.

We start defining the function `find_rigid_graphs` by allowing the argument to be a list of graphs. As the list we are concerned with contains graphs on the same number of vertices, we choose the first graph in the list and let  $n$  be the number of vertices in this graph.

Now, in order to filter out the graphs containing a vertex of degree of two, we first loop through all the graphs in the list, and then loop through each vertex in each graph. If the graph does **not** contain a vertex with degree two, then we add it to a separate list called `graphs_without_degree_2`.

From here, we loop through all the graphs in `graphs_without_degree_2`, check its rigidity using `check_rigidity`, and add it to a new list called `rigid_graphs`. The function finally returns `rigid_graphs`, a list containing all planar minimally rigid graphs of minimum degree atleast three.

#### 4.2.5 Testing the code

The last thing to do is to ensure that the code works for known examples before we start investigating graphs we know nothing about. Graphs that are not infinitesimally rigid can be seen in Figures 2.3 and 2.8. For one more graph that is known to be infinitesimally rigid, we consider the Framework in Figure 2.9. We label these graphs  $G_1$ ,  $G_2$  and  $G_3$  respectively, and construct them using `networkx`.

```
In [1]: print(check_rigidity(G1))
```

```
Out: False
```

```
In [2]: print(check_rigidity(G2))
```

```
Out: False
```

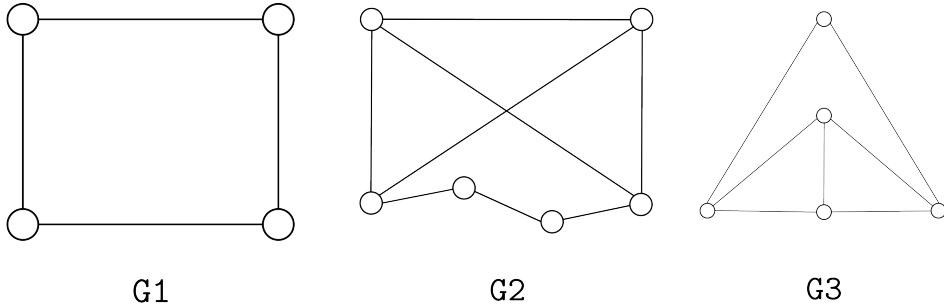


Figure 4.5: The frameworks from Figures 2.3, 2.8 and 2.9 labelled as **G1**, **G2**, and **G3** respectively.

```
In [3]: print(check_rigidity(G3))
Out: True
```

This is reassuring to see! We've successfully verified that the functions work well with each other, and that their outputs are as expected. So, with the confidence we have now when it comes to identifying infinitesimally rigid graphs, we can now move onto creating circle packings for a given planar graph.

### 4.3 Circle\_Packing.py

The process of finding circle packings for a graph involves a process called *numerical minimization*. It is a method in which we try to minimize the parameters involved in a given problem with clearly defined rules or *constraints*. In order to find circle packings, we first have to define the problem carefully, and impose certain constraints so that we end up with exactly what we want. Let us briefly run through exactly what this module covers.

It is made up of four functions, each performing a specific task, similar to how the functions were defined in `Rigidity.py`.

1. The first of these functions defines the *objective function* that we want to minimize.
2. The second function imposes the constraints that the minimization process should obey.
3. The third function generates a list of initial conditions that the minimizer will start minimizing.
4. The very last function is one that uses the three previous functions in order to generate a circle packing for a graph.

The minimizer that we will be using comes from the `scipy` [17] library in Python called `scipy.optimize.minimize`. To illustrate how the minimizer works, we describe a simple scenario in which one may use the minimizer in order to minimize a function that is subject to some constraints.

#### 4.3.1 A Simple Example

<sup>2</sup>Suppose we have a rectangle of length  $x$  and width  $y$ , and we wish to maximize its area  $xy$ , subject to the constraint that its perimeter is 20 units. Translating this into something that we can use in

---

<sup>2</sup>Should I include this example or not? Maybe consider coding it up

the minimizer, the objective function would be  $-xy$  as we are trying to maximize the area with the minimize function. The constraint can be modelled as  $2x + 2y = 20$  and our initial guesses for the values of  $x$  and  $y$  could be 6 and 4 respectively. We would input this into the `scipy.optimize.minimize` function, and obtain the optimal values of  $x$  and  $y$  satisfying these constraints such that the area of the rectangle is minimized. The code for this example is shown below.

```
# define the objective function
def area_rectangle(xy):
    x = xy[0]
    y = xy[1]
    area = xy[0] * xy[1]
    return -area

# define the constraint
cons = ({'type': 'eq', 'fun': lambda xy: 2*xy[0] + 2*xy[1] - 20})

# define the initial condition
initial_guess = [6,4]

# use the minimizer
result = scipy.optimize.minimize(area_rectangle, initial_guess, constraints = cons)

print(f"x = {result.x[0]} and y = {result.x[1]}")

Out: x = 5.0 and y = 5.0
```

Therefore, the values of  $x$  and  $y$  that maximize the area of a rectangle with a perimeter of 20 units are 5 and 5 respectively. Hopefully this provides some insight into how the minimizer works. The details of how `scipy.optimize.minimize` works are not very important for our purposes, but knowing how to use it properly is imperative.

With that in mind, let us dive into the first function in the `Circle_Packing.py` module!

# Bibliography

- [1] Leonard Asimow and Ben Roth. “The rigidity of graphs, II”. In: *Journal of Mathematical Analysis and Applications* 68.1 (1979), pp. 171–190.
- [2] Robert Connelly, Steven J. Gortler, and Louis Theran. “Rigidity for sticky discs”. In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 475.2222 (Feb. 2019).
- [3] Robert Connelly and Simon D. Guest. *Frameworks, Tensegrities, and Symmetry*. Cambridge: Cambridge University Press, 2022. DOI: [10.1017/9780511843297](https://doi.org/10.1017/9780511843297).
- [4] Jack E Graver. *Counting on frameworks: mathematics to aid the design of rigid structures*. 25. Cambridge University Press, 2001.
- [5] Aric Hagberg, Pieter Swart, and Daniel S Chult. *Exploring network structure, dynamics, and function using NetworkX*. Tech. rep. Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.
- [6] Thomas Hales et al. “A formal proof of the Kepler conjecture”. In: *Forum of mathematics, Pi*. Vol. 5. Cambridge University Press. 2017, e2.
- [7] Thomas C Hales. “A proof of the Kepler conjecture”. In: *Annals of mathematics* (2005), pp. 1065–1185.
- [8] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: [10.1038/s41586-020-2649-2](https://doi.org/10.1038/s41586-020-2649-2). URL: <https://doi.org/10.1038/s41586-020-2649-2>.
- [9] Lebrecht Henneberg. *Die graphische Statik der starren Systeme*. Vol. 31. BG Teubner, 1911.
- [10] Sanjeev Khanna. *Proceedings of the 2013 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Ed. by Sanjeev Khanna. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2013. DOI: [10.1137/1.9781611973105](https://doi.org/10.1137/1.9781611973105). eprint: <https://pubs.siam.org/doi/pdf/10.1137/1.9781611973105>. URL: <https://pubs.siam.org/doi/abs/10.1137/1.9781611973105>.
- [11] Paul Koebe. *Kontaktprobleme der konformen Abbildung*. Hirzel Stuttgart, 1936.
- [12] Gerard Laman. “On graphs and rigidity of plane skeletal structures”. In: *Journal of Engineering mathematics* 4.4 (1970), pp. 331–340.
- [13] Christopher M Linton. *From Eudoxus to Einstein: a history of mathematical astronomy*. Cambridge University Press, 2004.
- [14] Brendan D. McKay and Adolfo Piperno. “Practical Graph Isomorphism, II”. In: *Journal of Symbolic Computation* 60 (2014), pp. 94–112. DOI: [10.1016/j.jsc.2013.09.003](https://doi.org/10.1016/j.jsc.2013.09.003). URL: <https://doi.org/10.1016/j.jsc.2013.09.003>.

- [15] Hilda Pollaczek-Geiringer. “Über die gliederung ebener fachwerke”. In: *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik* 7.1 (1927), pp. 58–72.
- [16] Pavan M. V. Raja and Andrew R. Barron. *7.1 Crystal Structure*. [Accessed on: 21 February 2024]. LibreTexts. URL: [https://chem.libretexts.org/Bookshelves/Analytical\\_Chemistry/Physical\\_Methods\\_in\\_Chemistry\\_and\\_Nano\\_Science\\_\(Barron\)/07%3A\\_Molecular\\_and\\_Solid\\_State\\_Structure/7.01%3A\\_Crystal\\_Structure](https://chem.libretexts.org/Bookshelves/Analytical_Chemistry/Physical_Methods_in_Chemistry_and_Nano_Science_(Barron)/07%3A_Molecular_and_Solid_State_Structure/7.01%3A_Crystal_Structure).
- [17] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: [10.1038/s41592-019-0686-2](https://doi.org/10.1038/s41592-019-0686-2).