

# An Introduction to Colour Refinement

Joseph Edwards



University of  
St Andrews

2022 – 2023

### **Acknowledgements**

Firstly, I would like to thank my supervisor, Dr Finn Smith, without whom the completion of this project would not have been possible. Finn's patience and perspective was immeasurably helpful as I traversed the difficulties faced by final-year students, and I complete this project with a great sense of achievement, and an invigorated drive for future work.

Secondly, I would like to thank all of the people that have supported me in the completion of this project, and more broadly in the completion of my degree. From drawing graphs on the bathroom walls to convincing my friends that jellyfish are in fact a concern of pure mathematics, maths has made its way into all aspects of my life, and I am extremely grateful for all the people I have been able to share that with.

Finally, I would like to particularly acknowledge the immense support I have received from my parents. Despite neither of them having any formal training in undergraduate maths, they are always the most forthcoming in offering their support. The encouragement they give has helped me traverse the ups and downs of university life, and for that, I cannot say thank you enough.

## Abstract

The Colour Refinement algorithm is one of the most ubiquitous algorithms in the Graph Isomorphism literature, appearing as a subroutine in many practical Graph Isomorphism solvers, and as a fundamental part of Babai’s quasi-polynomial time Graph Isomorphism algorithm.

In this dissertation, we introduce the Colour Refinement algorithm and discuss those graphs which it can distinguish up to isomorphism: *amenable graphs*. Specifically, we discuss ways in which we can detect amenability, ways in which we can generate new amenable graphs, and equivalent characterisations of amenable graphs. In doing so, we provide an infinite family of graphs that solves an open question regarding the existence of asymmetric, non-regular graphs with certain eigenvalue properties. This is achieved through the process of *jellyfishification*.

## 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 and Preliminaries</b>	<b>1</b>
1.1	Notation . . . . .	2
1.2	Graph theory . . . . .	3
1.2.1	Graph theory basics . . . . .	3
1.2.2	Special graphs . . . . .	4
1.2.3	Isomorphisms . . . . .	6
1.2.4	Colouring . . . . .	7
1.3	Complexity . . . . .	8
1.3.1	Complexity basics . . . . .	8
1.3.2	Complexity classes . . . . .	9
<b>2</b>	<b>Colour Refinement</b>	<b>11</b>
2.1	First definitions . . . . .	11
2.2	A concrete implementation . . . . .	15
2.3	Time complexity . . . . .	20
2.4	More examples . . . . .	21
<b>3</b>	<b>Amenability</b>	<b>23</b>
3.1	Definitions . . . . .	23
3.2	Checking for amenability . . . . .	25
3.3	Types of amenable graphs . . . . .	30
<b>4</b>	<b>Relaxations</b>	<b>33</b>
4.1	The hierarchy . . . . .	35
4.2	Amenable graphs: Round 2 . . . . .	36
4.3	Jellyfishification . . . . .	38
<b>5</b>	<b>Applications and Conclusion</b>	<b>43</b>

# Chapter 1

## Introduction and Preliminaries

The field of graph theory aims to study relations between objects. Whilst these relations are perfectly well defined in an abstract, purely mathematical setting, graph theory is often the perfect language to describe certain applied problems in areas such as physics, chemistry, communication networks, genetics and linguistics. One of the most beautiful aspects of graph theory, therefore, stems from the ability to translate problems between disciplines, provided that we know that the underlying graph theoretical representations are the same. Frustratingly, quickly checking if two graphs are ‘the same’ is quite hard. Perhaps even more frustratingly, quantifying how hard it is to quickly check if two graphs are ‘the same’ is even harder!<sup>1</sup>

Here, when we say ‘the same’, we actually mean isomorphic. As such, the aim of this project is to offer the reader an introduction to the study of the Graph Isomorphism problem by exploring some of the techniques and algorithms we can use to solve it. Along the way, we will motivate and discuss some (hopefully!) natural questions regarding the properties we can recognise with these techniques and algorithms.

We begin our journey in the first chapter by stating concepts and preliminary results that we will use frequently throughout the remainder of the project. These will mostly be graph theoretical, but we briefly touch on complexity too. In the second chapter, we define Colour Refinement—a fundamental algorithm used in most common practical Graph Isomorphism solvers, and an essential part of Babai’s quasi-polynomial time algorithm for Graph Isomorphism [ABK14; Bab16]. With this definition, in the third chapter, we explore the graphs with which the Colour Refinement algorithm behaves particularly nicely, namely *amenable* graphs. Specifically, we explain what amenable graphs are, give some examples, and describe a way in which we can test for them. For the final chapter, having built up our amenability intuition, we relax some of the constraints imposed by Graph Isomorphism and pivot our search to graphs that behave nicely with *fractional isomorphisms*. In doing so, we find an open question and apply our knowledge from the previous chapters to present a solution.


The intended audience for this project is an undergraduate student with a firm grasp of graph theory, and a basic appreciation for complexity theory. With that said, in the remainder of this chapter we provide a quick summary of the key results and definitions from these fields

---

<sup>1</sup>We make this more precise in Section 1.3.

so that everyone is on the same page. Moreover, for the more technical results and definitions, where possible, a holistic overview of the main ideas is provided, so the keen reader without this background shouldn't be discouraged! In what follows, we try and keep the writing lighthearted.

As an example, students that have taken both *MT4514 Graph Theory* and *MT4512 Automata, Languages and Complexity* offered by the University of St Andrews should have an appropriate level of prerequisite knowledge to digest this project.

Throughout this dissertation, we present several pseudo-code 'implementation level' algorithms. Whilst it is intended that these can be performed by hand, this would take a long time if we were to do it often. Therefore, we often appeal to the Python implementation of these algorithms available to view and use on GitHub at <https://github.com/Joseph-Edwards/FinalYearProject>. This repository also hosts a Jupyter Notebook where we verify some of the results we prove in [Chapter 4](#), and a SageMath file we use to enumerate small graphs and select ones with various nice properties. Additionally, the contents of this repository have been made available in the supplementary files to this project. Where we use any code from this repository, we will mark the margin with a little laptop icon .

For the remainder of this project, we will be stating a variety of propositions, lemmas, and theorems. For the most part, we will attempt to prove these results. However, there are several results for which their proof diverges too far from our goals, and doesn't provide any richer understanding. In these cases, the proofs have been omitted in an attempt to preserve readability. The goal, therefore, of this project is not to provide a proof for every result we use; instead, we aim to bring together a collection of results and explore how they interact with each other. Hopefully, by the end of this dissertation, you will agree with me that they do so in a truly beautiful way.

But that's enough about what we *will* do. Let's actually do it! Bring on the maths, graphs and ... *jellyfish*?

## 1.1 Notation

As rational mathematicians, for the most part, we try and stick to standard mathematical notation that would be common in any undergraduate textbook. However, there are some instances where we need slightly more specialist notation, or where we adopt certain conventions, which we describe below.

A set is a collection of elements with neither repeats nor ordering, that is written using curly brackets, such as  $\{a, b, c\}$ . We tend to use uppercase Roman letters such as  $A$  or  $X$  to represent sets, unless that set represents a partition. In such a case we will often choose lowercase Greek letters such as  $\pi$  and  $\rho$  or calligraphic letters  $\mathcal{P}$ , and refer to elements as *cells*.

If  $\pi$  is a partition of some set  $A$ , and  $u$  and  $v$  are elements of  $A$ , we write  $u \approx_\pi v$  if  $u$  and  $v$  appear in the same cell of  $\pi$ . If  $A'$  is a subset of  $A$  formed by taking the union of some cells of  $\pi$ , we say that  $A'$  is  $\pi$  closed. If  $\rho$  is also a partition of  $A$  such that every cell of  $\pi$  is  $\rho$  closed, then we say  $\rho$  *refines*  $\pi$ , and write  $\pi \preceq \rho$ . Informally, if  $\rho$  refines  $\pi$ , we think of obtaining  $\rho$  by breaking down some of the cells of  $\pi$  into smaller cells. It can be shown that  $\preceq$  defines a non-strict partial order on the power-set of  $A$ .

We refer to a collection of objects *with* repeats but still without ordering as a multiset, and denote this with double curly brackets such as  $\{\{a, a, b, c, a, c\}\}$ .<sup>2</sup>

## 1.2 Graph theory

The definitions in this section are predominantly adapted from the 2022 lecture notes for MT4514 Graph Theory [HR22]. For a much more thorough account than we present here, the interested reader is directed to one of the many books on graph theory, such as [Die17].

### 1.2.1 Graph theory basics

**Definition 1.2.1** (Graph, Vertex, Edge). Let  $V$  be a finite set, and  $E$  be a set of two-element subsets  $\{u, v\}$  of  $V$  such that  $u \neq v$ . Then the ordered pair  $G = (V, E)$  is a *graph*. The elements of  $V$  are referred to as *vertices*, and elements of  $E$  are referred to as *edges*.

Our definition above ensures that all of the graphs we consider are finite, have undirected edges, contain no loops, and have no multiple edges. In the literature, these graphs are often referred to as *simple* graphs.

**Definition 1.2.2** (Complement). Let  $G = (V, E)$  be a graph with vertices  $V = \{v_1, v_2, \dots, v_n\}$ . Then the *complement graph*  $\bar{G} = (V, \bar{E})$  is the graph with edges  $\bar{E} = \{\{v_i, v_j\} : \{v_i, v_j\} \notin E\}$ .

In other words, the complement  $\bar{G}$  of a graph  $G$  has an edge between vertices  $u$  and  $v$  if and only if there is *no* edge between  $u$  and  $v$  in  $G$ .

**Definition 1.2.3** (Adjacent, Neighbour, Degree). Let  $G = (V, E)$  be a graph, and  $v \in V$  be a vertex of  $G$ . If  $v \in V$  is a vertex such that  $\{u, v\} \in E$  is an edge, then we say that  $u$  and  $v$  are *adjacent* or *neighbours*. Moreover, the set  $N(u) = \{v \in V : \{u, v\} \in E\}$  of vertices adjacent to  $u$  is called the *neighbourhood* of  $u$ , and the number  $\deg(u) = |N(u)|$  of neighbours of  $u$  is known as the *degree* of  $u$ .

**Definition 1.2.4** (Degree sequence). Let  $G$  be a graph. The *degree sequence* of  $G$  is the list of degrees of each vertex of  $G$  in decreasing order.

**Definition 1.2.5** (Regular). Let  $G$  be a graph. We say that  $G$  is *regular* if all of the vertices of  $G$  have the same degree. If the degree of each vertex is  $d$ , we say that  $G$  is *d-regular*.

**Definition 1.2.6** (Adjacency matrix). Let  $G = (V, E)$  be a graph with  $n$  vertices  $v_1, v_2, \dots, v_n$ . The *adjacency matrix*  $\text{Adj}(G)$  of  $G$  is the  $n \times n$  matrix such that

$$[\text{Adj}(G)]_{i,j} = \begin{cases} 1 & \text{if } \{v_i, v_j\} \in E \\ 0 & \text{otherwise.} \end{cases}$$

**Definition 1.2.7** (Bipartite). Let  $G = (V, E)$  be a graph. We say that  $G$  is *bipartite* if there exists a partition  $\{A, B\}$  of vertices  $V$  such that every edge of  $G$  has one endpoint in  $A$  and one endpoint in  $B$ .

---

<sup>2</sup>Instead of thinking about repeats, we can alternatively regard a multiset as a set where each element has an associated multiplicity.

Alternatively, we could have defined bipartite to be the graphs with a vertex partition  $\{A, B\}$  such that there are no edges with both endpoints in the same cell.

**Definition 1.2.8** (Induced subgraph, Induced bipartite subgraph). Let  $G = (V, E)$  be a graph and  $V' \subseteq V$  be a set of vertices of  $G$ . The *induced subgraph*  $G[V']$  of  $G$  by  $V'$  is the graph with vertex set  $V'$  and edge set  $E' = \{\{u, v\} : u, v \in V' \text{ and } \{u, v\} \in E\}$ .

Furthermore, if  $U' \subseteq V$  is a set of vertices distinct from  $V'$  (i.e.  $U' \cap V' = \emptyset$ ), the *induced bipartite subgraph*  $G[U', V']$  of  $G$  by  $U'$  and  $V'$  is the graph with vertex set  $U' \cup V'$  and edge set  $E'' = \{\{u, v\} : u \in U', v \in V' \text{ and } \{u, v\} \in E\}$ .

We can think of the induced subgraph  $G[V']$  as the graph obtained by removing all edges of  $G$  that do not have an endpoint in  $V'$ . Similarly, we can think of the induced bipartite subgraph  $G[U', V']$  as the graph obtained by removing all edges of  $G$  that do not have exactly one endpoint in  $U'$  and one endpoint in  $V'$ . If we take an induced subgraph, and drop some of the edges, then we simply get a *subgraph*.

Subgraphs are a way of making smaller graphs from a big graph. The next definition gives a way to make large graphs from smaller graphs.

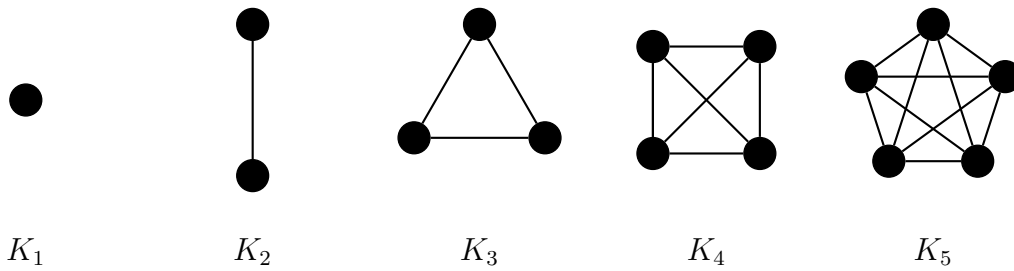
**Definition 1.2.9** (Disjoint union). Let  $G = (V_G, E_G)$  and  $H = (V_H, E_H)$  be graphs. Without loss of generality, assume that  $V_G \cap V_H = \emptyset$ . The *disjoint union*  $G + H$  is the graph  $(V_G \cup V_H, E_G \cup E_H)$ .

If we take the disjoint union of a graph with itself  $k$  times, we write this multiplicatively as  $kG$ .

### 1.2.2 Special graphs

Throughout this project, there are several instances where we reference specific families of graphs. We present some of them here.

**Definition 1.2.10** (Complete graph). The *complete graph*  $K_n$  on  $n$  vertices is the graph with edges between every pair of distinct vertices. The first five are:



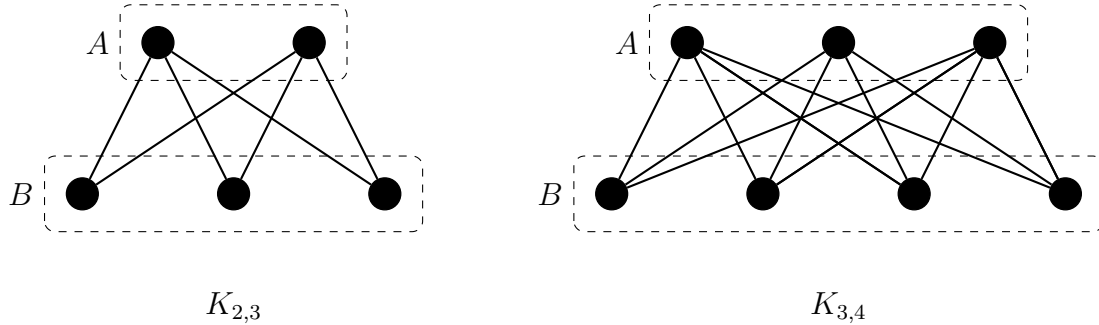
The complete graph on  $n$  vertices is the graph with the maximum number of edges for a given number of vertices. The following definition provides a complement.

**Definition 1.2.11** (Empty graph). The *empty graph*  $E_n$  on  $n$  vertices is the graph with no edges.

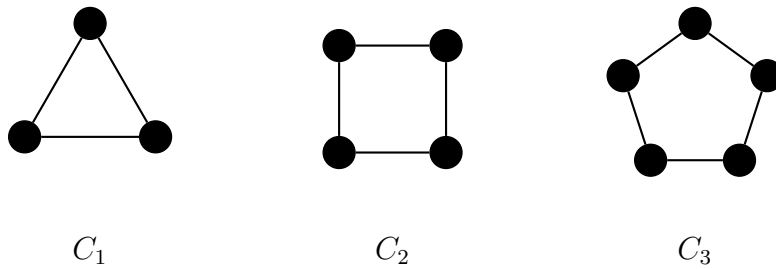
**Definition 1.2.12** (Complete bipartite graph). For natural numbers  $m, n$ , the *complete bipartite graph*  $K_{m,n}$  with cells of size  $m$  and  $n$  is the bipartite graph with vertex bipartition  $\{A, B\}$  such



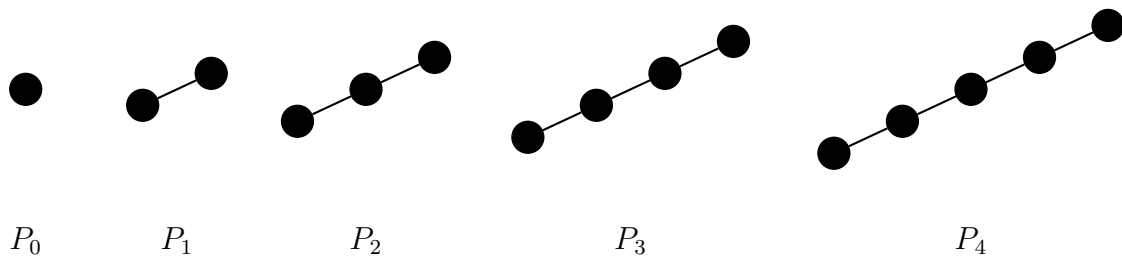
that  $|A| = m$ ,  $|B| = n$ , every vertex in  $A$  is adjacent to every vertex in  $B$ , there are no edges between vertices of  $A$ , and no edges between vertices of  $B$ . Some examples are:



**Definition 1.2.13** (Cycle graph). For a natural number  $n \geq 3$ , the *cycle graph*  $C_n$  on  $n$  vertices is the graph with edges between every consecutive pair of vertices (for any arbitrary ordering on the vertices), and the first and last vertex. We say that  $C_n$  has length  $n$ . The first three are:



**Definition 1.2.14** (Path graph). The *path graph*  $P_n$  of length  $n$  is the graph with  $n + 1$  vertices and edges between every consecutive pair of vertices (for any arbitrary ordering on the vertices). The first five are:



It is important to note that, here, we presented a slightly non-standard definition of  $P_n$ , where  $n$  refers to the *length*, not the number of vertices.

**Definition 1.2.15** (Matching graph). For positive natural number  $k$ , the *kth matching graph*  $M_k$  is  $kP_1$ .

### 1.2.3 Isomorphisms

At the beginning of this chapter, we claimed that this project aims to explore the problem of Graph Isomorphism. As such, we ought to define what an isomorphism actually is!

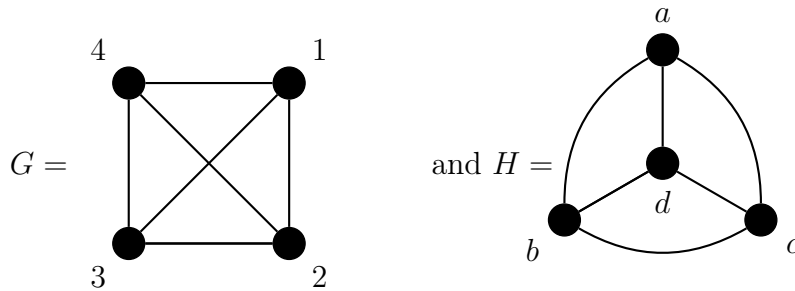
**Definition 1.2.16** (Isomorphism, Automorphism). Let  $G = (V_G, E_G)$  and  $H = (V_H, E_H)$  be graphs. An *isomorphism* between  $G$  and  $H$  is a bijection  $\varphi: V_G \rightarrow V_H$  between the vertices of  $G$  and  $H$  such that  $\{u, v\} \in E_G$  if and only if  $\{\varphi(u), \varphi(v)\} \in E_H$ . If such a bijection exists, we say that  $G$  and  $H$  are *isomorphic*.

An *automorphism* is any isomorphism from  $G$  to itself. The set of all automorphism of  $G$  is denoted  $\text{Aut}(G)$ .

By definition,  $\text{Aut}(G)$  is a set. However, it can be shown that  $\text{Aut}(G)$  is in fact a group under composition. In particular, the set of automorphisms of a graph always contains the identity map  $\text{Id}: V \rightarrow V$  which maps every vertex to itself.

**Definition 1.2.17** (Asymmetric). Let  $G$  be a graph. If  $\text{Aut}(G)$  contains only the identity map, then we say that  $G$  is *asymmetric*.

**Example 1.2.18.** The graphs



are isomorphic, witnessed by the bijection  $\varphi$  with the following mapping:

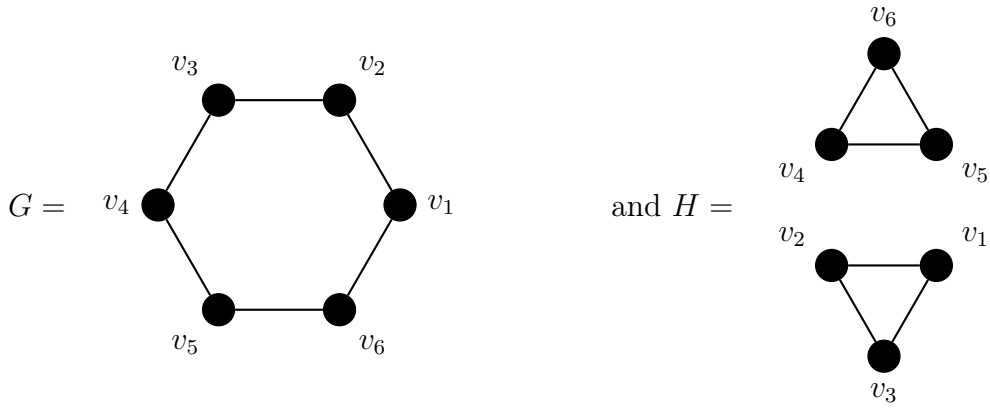
$v$	1	2	3	4
$\varphi(v)$	$a$	$b$	$c$	$d$

**Proposition 1.2.19.** Let  $G$  and  $H$  be two isomorphic graphs. Then  $G$  and  $H$  have the same number of edges and vertices.

*Proof.* This follows immediately from the definition. □

The converse of this statement isn't true; that is, we can find two non-isomorphic graphs that have the same number of edges and nodes.

**Example 1.2.20.** The graphs



have the same number of edges and vertices, but are not isomorphic. This can be seen by observing  $G$  has  $C_6$  as an induced subgraph, but  $H$  does not.

**Definition 1.2.21** (Orbit). Let  $G = (V, E)$  be a graph,  $v \in V$  be a vertex of  $G$ , and  $\Gamma$  be a subgroup of  $\text{Aut}(G)$ . The  $\Gamma$  *orbit* of  $v$  is the set  $\{\varphi(v) : \varphi \in \Gamma\}$  of images of  $v$  under automorphisms in  $\Gamma$ . Where we consider the  $\text{Aut}(G)$  orbit of  $v$ , we simply refer to this as the *orbit*.

**Example 1.2.22.** Let  $G$  be an asymmetric graph. Then the only automorphism of  $G$  is the trivial identity automorphism  $\text{Id}$ . Therefore, for all  $v \in V$ , the orbit of  $v$  is the singleton  $\{v\}$ .

## 1.2.4 Colouring

The reader with a background in graph theory may have a preconceived idea of what a vertex colouring on a graph is.<sup>3</sup> The Colour Refinement literature, however, uses a slightly non-standard definition. We present that definition here, and give some examples of valid colourings.

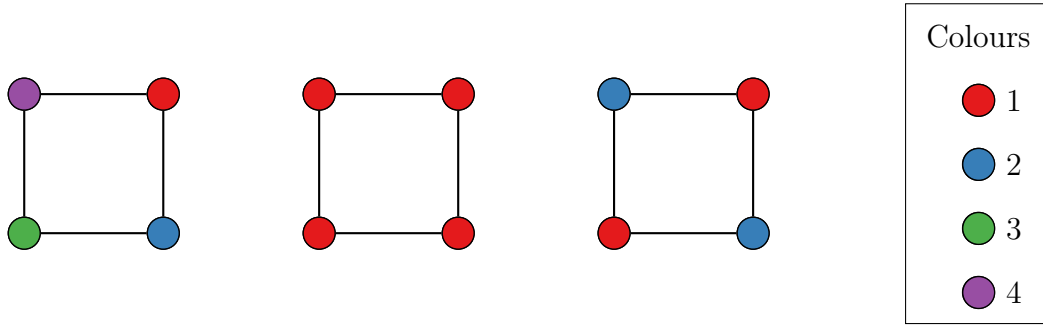
**Definition 1.2.23** (Colouring, Colour class). Let  $G = (V, E)$  be a graph. A *colouring* of  $G$  is any function  $C: V \rightarrow \mathbb{N}$ . For some colour  $c \in \text{Im}(C)$  in the image of  $C$ , the set  $V_c = \{v \in V : C(v) = c\}$  of vertices with colour  $c$  is called the *colour class* of  $c$ .

Throughout this project, we will often use actual colours (think red or blue) when visualising a colouring of a graph, as the following example demonstrates.

**Example 1.2.24.** Three different colourings<sup>4</sup> of the cycle graph  $C_4$ :

<sup>3</sup>Some authors refer to a vertex colouring as an assignment of colours to vertices, with the added constraint that adjacent vertices must not have the same colour.

<sup>4</sup>Notice that the ‘all red’ colouring in this example would *not* be a valid colouring if we forced adjacent vertices to have distinct colours.



**Definition 1.2.25** (Restriction). Let  $G = (V, E)$  be a graph,  $H = (V', E')$  be a subgraph of  $G$ , and  $C$  be a colouring of  $G$ . Then the *restriction of  $C$  to  $H$*  is the colouring  $C': V' \rightarrow \mathbb{N}$  of the vertices of  $H$  inherited from the colouring of the vertices of  $G$ .

## 1.3 Complexity

The main goal of this project, as detailed earlier, is to give the reader a flavour of the Colour Refinement literature with respect to the problem of Graph Isomorphism. For the most part, we aim to do this from a graph theoretic perspective. However, a major goal of the work done in this field is to find efficient algorithms to solve this problem. Therefore, to try and give the reader a more representative idea of the existing literature, we present a very basic introduction to complexity theory. This will hopefully offer an idea of how to quantify the running time, but will not define any formal models of computation.

The definitions in this section are adapted from the 2023 lecture notes for MT4512 Automata, Languages and Complexity [Ron23]. For a more thorough introduction, the interested reader is directed to one of the many books on the theory of computation, such as [Sip13], specifically Chapter 3 for complexity theory.

### 1.3.1 Complexity basics

**Definition 1.3.1** (Running time). Let  $A$  be an algorithm that accepts  $X$  as an input. Then  $A$  has *running time* (or *time complexity*)  $f(n)$  if  $f(n)$  is the maximum number of steps  $A$  takes to terminate when  $X$  has ‘size’  $n$ .

We note here that we are deliberately vague when we say the input has ‘size’  $n$ , because a priori we impose no constraint on what input an algorithm can accept. Therefore, we have no universal way of representing an input that will work for every algorithm.<sup>5</sup>

With that said, in this project, all of our algorithms take graphs as input. Therefore, we can be slightly more precise about what we mean by ‘size’.

**Definition 1.3.2** (Running time for graph algorithms). Let  $A$  be an algorithm that accepts the graph  $G$  as input. Then  $A$  has *running time* (or *time complexity*)  $f(n, m)$  if  $f(n, m)$  is the maximum number of steps  $A$  takes to terminate when  $G$  has  $n$  vertices and  $m$  edges.

<sup>5</sup>If we instead define running time on Turing Machines (rather than algorithms), then the size of the input is the well-defined length of the input word.

If the running time of  $A$  does not depend on  $m$ , we drop the dependence of  $m$  in the input of  $f$  too, and simply write  $f(n)$ .

Oftentimes, the *exact* running time of an algorithm depends on the intricacies of the implementation and can be fiddly to calculate. Therefore, we tend to care more about the *asymptotic running time* of our algorithm for large input. This is encapsulated in the following definitions.

**Definition 1.3.3** (Big-O). Let  $f$  and  $g$  be functions from  $\mathbb{N}$  to  $\mathbb{R}$ . We say that  $f \in \mathcal{O}(g)$  (pronounced “ $f$  is big-O of  $g$ ”) if there exists integers  $c, n_0 \geq 0$  such that, for all  $n \geq n_0$ ,  $f(n) \leq cg(n)$ .

The idea of Big-O is to give an asymptotic upper bound to a function. Therefore, when working with time complexity, we use Big-O to quantify the *worst case* running time of an algorithm. Similarly, we can define terms to encapsulate the *best case* running time of an algorithm.

**Definition 1.3.4** (Big- $\Omega$ ). Let  $f$  and  $g$  be functions from  $\mathbb{N}$  to  $\mathbb{R}$ . We say that  $f \in \Omega(g)$  (pronounced “ $f$  is big-Omega of  $g$ ”) if there exists integers  $c, n_0 > 0$  such that, for all  $n \geq n_0$ ,  $f(n) \geq cg(n)$ .

### 1.3.2 Complexity classes

At the very start of this chapter, we claimed that quantifying how hard it is to quickly check if two graphs are isomorphic is a difficult problem. Without delving too much further into the realm of complexity, we can’t hope to formalise this idea much further. However, we can try and understand the key ideas behind the problem. For a more in-depth discussion, see Chapter 7 in [Sip13].

Informally, the complexity class  $\mathbf{P}$  can be thought of as all of the problems that can be solved by an algorithm with polynomial running time. The complexity class  $\mathbf{NP}$ <sup>6</sup> can be thought of as all of the problems for which a solution can be *verified* by an algorithm with polynomial running time. Deciding whether or not  $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$  is one of the most famous open problems in mathematics, and finding a solution comes with a one million dollar prize from the Clay Mathematics Institute.<sup>7</sup>

One way in which a solution to this problem may be found is by analysing the class of problems that are ‘NP-complete’. We can think of these problems as being the hardest of all of the  $\mathbf{NP}$  problems. Specifically, the existence of a polynomial time algorithm that solves an NP-complete problem implies the existence of a polynomial time algorithm for *every*  $\mathbf{NP}$  problem, proving  $\mathbf{P} = \mathbf{NP}$ .

With these ideas, can we classify the Graph Isomorphism (GI) problem? We can show that  $\mathbf{GI} \in \mathbf{NP}$ , because we can construct a polynomial time algorithm that verifies if a mapping is an isomorphism. Other than this, however, the complexity of GI is unknown. It is one of the few natural problems for which

<sup>6</sup>The abbreviation  $\mathbf{NP}$  stands for *nondeterministic polynomial*, meaning there exists a nondeterministic Turing Machine that recognises the language of the problem in polynomial time.

<sup>7</sup>For more information on this problem, see <http://www.claymath.org/millennium-problems/millennium-prize-problems>.

- it is not known if  $GI \in P$ ; and
- it is not known if  $GI \in NP$ -complete.

It is believed that  $GI \in P$  and  $GI \notin NP$ -complete, however, no polynomial algorithm has been found; the closest we have is a *quasi-polynomial* time algorithm [Bab16; Bab19]. This algorithm relies heavily on Colour Refinement—the motivating algorithm for this dissertation—so it is very important to understand how efficiently it can be implemented!

# Chapter 2

## Colour Refinement

Imagine we have two graphs,  $G$  and  $H$ . There are many questions we could ask about their relationship, however, as alluded to in the previous chapter, there is one question that motivates the rest of the work we do in this project: *are  $G$  and  $H$  isomorphic?*

**Proposition 1.2.19** tells us that a necessary condition for two graphs to be isomorphic is that they must have the same number of vertices and the same number of edges. However, **Example 1.2.20** showed us that this condition is not sufficient. For the remainder of this chapter, we discuss another technique we can employ that can help when testing for isomorphism, namely *Colour Refinement*.

The Colour Refinement algorithm is one of the most ubiquitous algorithms in modern Graph Isomorphism solving. As mentioned earlier, it forms a crucial part of Babai’s quasi-polynomial time algorithm for the Graph Isomorphism problem. Additionally, it is implemented by most modern practical Graph Isomorphism solvers such as **nauty** [MP14].

Whilst being an incredibly rich and interesting algorithm in its own right, the Colour Refinement algorithm is part of a family of algorithms. Specifically, Colour Refinement is the one-dimensional Weisfeiler-Leman algorithm (1-WL algorithm). For a detailed description of the general  $k$ -dimensional Weisfeiler-Leman algorithm, the interested reader is directed to [CFI92].

### 2.1 First definitions

The number of vertices and the number of edges of a graph is a *global* property invariant under isomorphism i.e. a property that all isomorphic graphs share. The goal of this section is to produce an invariant based on the *local* structure of a graph. Specifically, we look to define an invariant based on the neighbourhood of each vertex. So let’s get to it!

**Definition 2.1.1** ( $C$ -equitable, Equitable colouring). Let  $G = (V, E)$  be a graph, and  $C: V \rightarrow \mathbb{N}$  be a colouring. We say that two vertices  $u$  and  $v$  are  *$C$ -equitable* if

- (i)  $C(u) = C(v)$ ;
- (ii)  $\{\{C(x) : x \in N(u)\}\} = \{\{C(y) : y \in N(v)\}\}$ .

We write this as  $u \sim_C v$ .

Let  $X_1, \dots, X_k$  be the colour classes induced by  $C$ . We say that  $C$  is an *equitable colouring* if for all  $i \in \{1, \dots, k\}$ ,  $u \sim_C v \iff u, v \in X_i$ .

In other words, we say that two vertices  $u$  and  $v$  are  $C$ -equitable if they have the same colour and the multi-set of the colours of the neighbours of  $u$  is equal to the multi-set of the colours of the neighbours of  $v$ .

**Proposition 2.1.2.** *Let  $G = (V, E)$  be a graph, and  $C: V \rightarrow \mathbb{N}$  be a colouring. Then  $C$ -equitability defines an equivalence relation on the vertices  $V$ .*

*Proof.* This follows immediately from the definition. □

To develop this idea of equitability further, we define a type of partition that exhibits similar properties to equitable colourings.

**Definition 2.1.3** (Stable). Let  $G = (V, E)$  be a graph, and  $\pi$  be a partition of  $V$ . We say that  $\pi$  is *stable* if for all vertices  $u, v \in V$  such that  $u \approx_\pi v$ , and for all cells  $X \in \pi$ , we have  $|N(u) \cap X| = |N(v) \cap X|$ .

Without much work, we can show that the colour classes of an equitable colouring form a stable partition. With these definitions, we finally have all of the information necessary to state the Colour Refinement algorithm.

---

**Algorithm 2.1.4** Colour Refinement

---

On input graph  $G = (V, E)$ . Let  $C_0: V \rightarrow \mathbb{N}$  be the initial colouring given by  $v \mapsto 1$  for all  $v \in V$ . Now we set  $i = 1$ , and proceed as follows:

1. Define  $\mathcal{P}_i = \{\mathcal{C}_{i,1}, \mathcal{C}_{i,2}, \dots, \mathcal{C}_{i,n_i}\}$  to be the  $C_{i-1}$ -equitable equivalence classes of  $V$  in some order.<sup>1</sup>
2. Define  $C_i: V \rightarrow \mathbb{N}$  to be a colouring that gives each  $C_{i-1}$ -equitable cell its own unique colour. One such colouring has  $C_i(v) = \max_{v \in V} \{C_{i-1}(v)\} + j$  if  $v \in \mathcal{C}_{i-1,j}$  for  $1 \leq j \leq n_i$ .
3. If  $C_i$  and  $C_{i-1}$  partition  $V$  into different colour classes, increment  $i$  by 1 and go back to Step 1. Otherwise, we are done!

Once the algorithm has terminated, we call the final colouring the *stable colouring*. We call the vertex partition induced by this colouring the *stable partition* of  $G$ .<sup>2</sup>

---

*Remark 2.1.5.* Notice that we ensure  $C$ -equitable vertices always have the same colour. Then for vertices  $u, v \in V$  and some colourings  $C_i, C_{i+1}$  obtained from the above process, if  $u \sim_{C_{i+1}} v$  then  $u \sim_{C_i} v$ . In other words, the partition  $\mathcal{P}_{i+1}$  is achieved by taking  $\mathcal{P}_i$  and further breaking—or *refining*—a (potentially empty) collection of its cells into smaller cells. It is for this reason that the algorithm is called *Colour Refinement*.

---

<sup>1</sup>If we just want to obtain the stable partition, then any ordering will do. If, however, we want to use this algorithm practically for isomorphism testing, then we must order the equivalence classes *canonically*. We show a way in which this can be done [Section 2.2](#) when we present a concrete implementation.

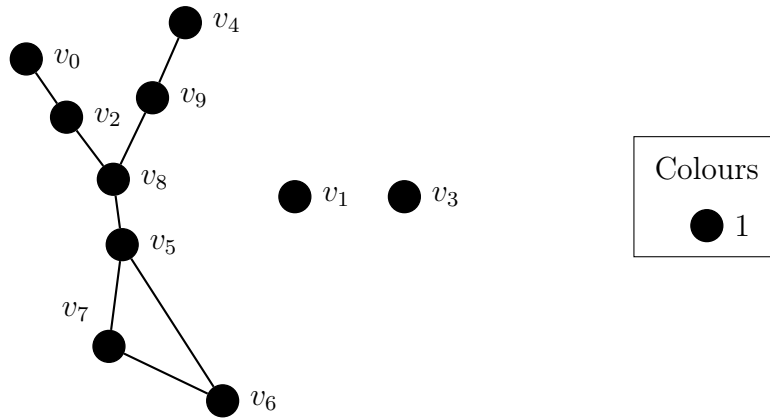
<sup>2</sup>It follows from the definition that the stable partition is in fact stable. In [Section 2.2](#), we show that the stable partition is the *maximal* stable partition (with respect to refinement).



As interesting as the etymology of this algorithm's name is, the seventeen characters necessary to refer to it in full is far too cumbersome for an algorithm we will be seeing so often. As such, when the context is clear, we may refer to this algorithm as “CR”.

To try and build some CR intuition, and to exercise our graph colouring muscles, we present the following example.

**Example 2.1.6.** Consider the following disconnected graph  $G = (V, E)$  with 10 vertices:



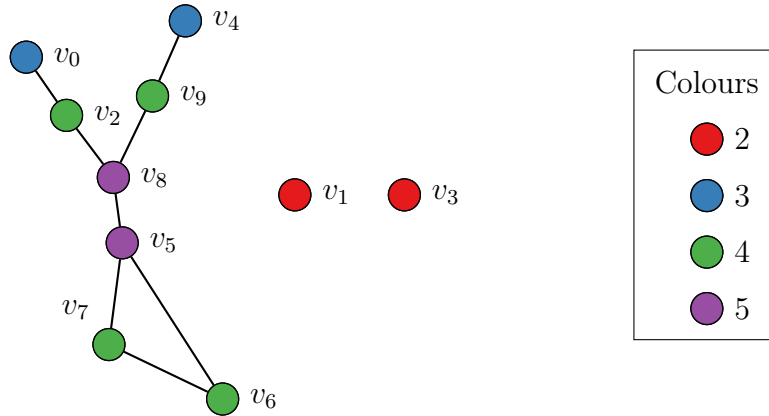
This graph has been coloured with the initial colouring  $C_0 : V \rightarrow \mathbb{N}$  that colours each vertex  $v \in V$  with colour 1 (black). In what follows, we will colour the vertices of  $G$  (with actual colours like red or blue) to show the colour classes (mathematical colours, like 1 or 2). As described in the algorithm, in each iteration, we will use new mathematical colours. However, so we don't run out of actual colours that are visually distinguishable, we reuse actual colours between iterations.

The first step in our algorithm is to find the  $C_0$ -equitable equivalence classes. As all of our vertices currently have the same colour, the classes are defined by the multi-set of the colours of the neighbours. Another consequence of the vertices having the same colour is that the multi-set of colours of neighbours is uniquely determined by the degree of each vertex. Hence for vertices  $u, v \in V$ ,  $u \sim_{C_0} v$  if and only if  $\deg(u) = \deg(v)$ .

Using this fact with  $G$ , we get

$$\mathcal{P}_1 = \{\underbrace{\{v_1, v_3\}}_{\deg 0}, \underbrace{\{v_0, v_4\}}_{\deg 1}, \underbrace{\{v_2, v_6, v_7, v_9\}}_{\deg 2}, \underbrace{\{v_5, v_8\}}_{\deg 3}\} = \{\mathcal{C}_{1,1}, \mathcal{C}_{1,2}, \mathcal{C}_{1,3}, \mathcal{C}_{1,4}\}.$$

With  $\mathcal{P}_1$  defined, we can now construct our new and improved colouring  $C_1$ . Using the suggestion given in Step 1, we define the new colouring  $C_1 : V \rightarrow \mathbb{N}$  as  $C_1(v) = 1 + j$  if  $v \in \mathcal{C}_{1,j}$  for  $1 \leq j \leq 4$ . Hence  $C_1(v_1) = C_1(v_3) = 2$ ,  $C_1(v_0) = C_1(v_4) = 3$ ,  $C_1(v_2) = C_1(v_6) = C_1(v_7) = C_1(v_9) = 4$  and  $C_1(v_5) = C_1(v_8) = 5$ :



The colour classes have changed between iterations, so we go back to Step 1 and compute the  $C_1$ -equitable partition  $\mathcal{P}_2$ .

The vertices with colour 2 have no neighbours, and so the multi-sets of their neighbours are all trivially equal; this cell of vertices doesn't get broken down any further.

We next look at the vertices with colour 3:  $v_0$  and  $v_4$ . The multi-set of colours of the neighbours of both  $v_0$  and  $v_4$  is  $\{\{4\}\}$ , so this cell of vertices doesn't get broken down further either.

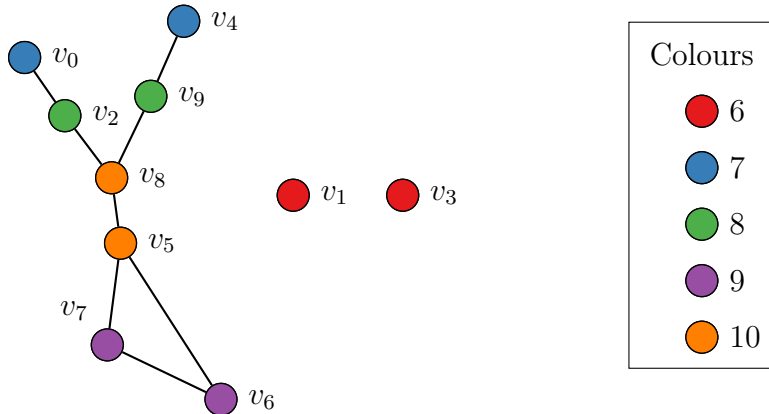
The vertices of colour 4, however, finally do something exciting! The multi-set of colours of the neighbours of both  $v_2$  and  $v_9$  is  $\{\{3, 5\}\}$ , whereas the multi-set of colours of the neighbours of both  $v_6$  and  $v_7$  is  $\{\{4, 5\}\}$ . Therefore, the cell  $\{v_2, v_6, v_7, v_9\}$  gets broken down into the smaller cells  $\{v_2, v_9\}$  and  $\{v_6, v_7\}$ .

The vertices of colour 5 are once again uninteresting; the multi-set of colours of the neighbours of both  $v_5$  and  $v_8$  is  $\{\{4, 4, 5\}\}$ , so this cell of vertices doesn't get broken down further.

To summarise, the above procedure gives

$$\mathcal{P}_2 = \{\{v_1, v_3\}, \{v_0, v_4\}, \{v_2, v_9\}, \{v_6, v_7\}, \{v_5, v_8\}\} = \{\mathcal{C}_{2,1}, \mathcal{C}_{2,2}, \mathcal{C}_{2,3}, \mathcal{C}_{2,4}, \mathcal{C}_{2,5}\}.$$

Once again taking the advice outlined in the definition,  $C_2$  assigns the colours 6, 7, 8, 9 and 10 to the vertices of  $\mathcal{C}_{2,1}, \mathcal{C}_{2,2}, \mathcal{C}_{2,3}, \mathcal{C}_{2,4}$  and  $\mathcal{C}_{2,5}$ , respectively:

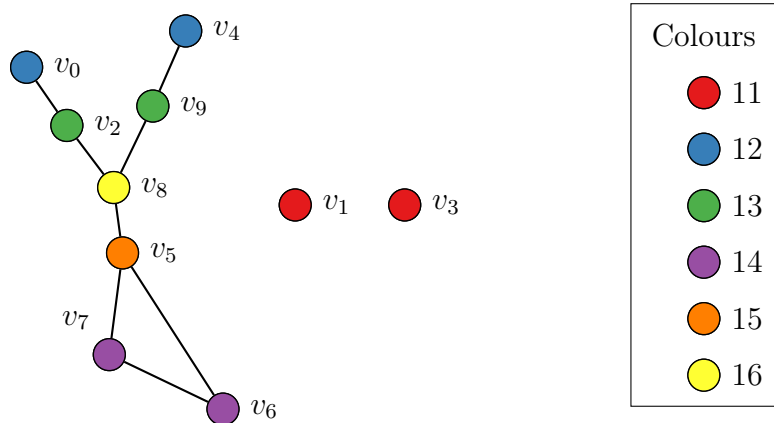


This new colouring partitions the vertices differently than the previous iteration, so we go back to Step 1.

The only interesting case for this colouring is the cell with vertices  $v_5$  and  $v_8$  of colour 10; all of the other  $C_1$ -equitable cells are also  $C_2$ -equitable cells. For  $v_5$ , the multi-set of colours of neighbours is  $\{\{9, 9, 10\}\}$ , whereas for  $v_8$  it is  $\{\{8, 8, 10\}\}$ . Hence the cell  $\{v_5, v_8\}$  gets broken down into the cells  $\{v_5\}$  and  $\{v_8\}$ . This gives the partition

$$\mathcal{P}_3 = \{\{v_1, v_3\}, \{v_0, v_4\}, \{v_2, v_9\}, \{v_6, v_7\}, \{v_5\}, \{v_8\}\} = \{\mathcal{C}_{3,1}, \mathcal{C}_{3,2}, \mathcal{C}_{3,3}, \mathcal{C}_{3,4}, \mathcal{C}_{3,5}, \mathcal{C}_{3,6}\}.$$

The colouring  $C_3$  is constructed to assign the colours 11, 12, 13, 14, 15 and 16 to the vertices of  $\mathcal{C}_{3,1}, \mathcal{C}_{3,2}, \mathcal{C}_{3,3}, \mathcal{C}_{3,4}, \mathcal{C}_{3,5}$  and  $\mathcal{C}_{3,6}$ , respectively:



If you are looking at this graph and thinking “surely we must be finished”, then you would be correct! We can check that performing one more iteration does not change the vertex partition, which is the terminating condition for the algorithm. Therefore the stable colouring of  $G$  is  $C_3$ .

## 2.2 A concrete implementation

**Algorithm 2.1.4** provides us with a high-level description of what we want the Colour Refinement algorithm to do: iteratively make new colourings based on  $C$ -equitability. It does not, however, tell us explicitly how to do this. Moreover, the algorithm offers no insight as to how fast the algorithm will run.

In this section, we present a pseudo-code implementation of CR provided by Grohe et al. in [Gro+21] and prove its validity.

This pseudo-code algorithm, as shown in **Algorithm 2.2.1**, works by maintaining a queue<sup>3</sup> of colours that we need to refine by. Starting with the initial colouring that assigns every vertex the colour 1, the algorithm repeatedly pops a colour, say  $q$ , from the queue  $Q$  of used colours and refines the other colour classes with respect to  $q$ . That’s a lot of  $Q$ s! In other words, given

<sup>3</sup>Here, a *queue* is a first-in-first-out data structure. The process of adding to the back of a queue is called *pushing*, and removing from the front a queue is called *popping*.

**Algorithm 2.2.1** Pseudo-code for practical Colour Refinement [Gro+21]On input graph  $G = (V, E)$ .

---

```

1: procedure COLOURREFINEMENT( $G = (V, E)$ )
2:   for all  $v \in V$  do                                     Define initial colouring
3:      $C(v) \leftarrow 1$ 
4:   end for
5:    $P(1) \leftarrow V$                                        Associate each colour with its colour class in V
6:    $c_{\min} \leftarrow 1$                                      Define min and max colour ranges
7:    $c_{\max} \leftarrow 1$ 
8:   Initialise queue  $Q \leftarrow [1]$ 
9:   while  $Q \neq \emptyset$  do
10:     $q \leftarrow \text{POP}(Q)$ 
11:    for  $v \in V$  do
12:       $D(v) \leftarrow |N_G(v) \cap P(q)|$                  Number of neighbours of v with colour q
13:    end for
14:    Sort  $V$  lexicographically by  $(C(v), D(v))$ 
15:    Define  $(B_1, \dots, B_k)$  to be the sets of vertices with equal  $C$  and  $D$  values
16:    for  $i \in \{c_{\min}, \dots, c_{\max}\}$  do
17:       $k_{i,1}, k_{i,2}$  such that  $k_{i,1} \leq k_{i,2}$  and  $P(i) = B_{k_{i,1}} \cup \dots \cup B_{k_{i,2}}$ 
18:       $i^* \leftarrow \arg \max_{k_{i,1} \leq i \leq k_{i,2}} |B_i|$        Find the largest cell
19:      for  $j \in \{k_{i,1}, \dots, k_{i,2}\} \setminus \{i^*\}$  do   All colours except the one with most vertices
20:        PUSH( $Q, c_{\max} + j$ )
21:      end for
22:    end for
23:     $C_{\min} \leftarrow c_{\max} + 1$                          Define new colour range
24:     $C_{\max} \leftarrow c_{\max} + k$ 
25:    for  $b \in \{c_{\min}, \dots, c_{\max}\}$  do
26:       $P(b) \leftarrow B_{b+1-c_{\min}}$ 
27:      for  $v \in P(b)$  do
28:         $C(v) \leftarrow b$ 
29:      end for
30:    end for
31:  end while
32: end procedure

```

---

a colouring  $C$  and a colour  $q \in \text{Im}(C)$ , for each colour  $i \in \text{Im}(C)$ , the algorithm breaks down the colour class  $P(i)$  of  $V$  into cells  $B_{k_{i,1}}, \dots, B_{k_{i,2}}$  such that, within each cell, all of the vertices  $v$  have the same number  $D_q(v)$  of neighbours with colour  $q$ . This is described between lines 10 and 15.

The remainder of the algorithm works to give each of the new cells described above their own colour. When a colour class  $P(i)$  is broken down into new cells  $B_{k_{i,1}}, \dots, B_{k_{i,2}}$ , the new colours will be  $c_{\max} + i$  for  $k_{i,1} \leq i \leq k_{i,2}$ , where  $c_{\max}$  is the largest colour that was used in the previous refinement round. All of these new colours are pushed to  $Q$ , except the colour  $i^*$ —the colour that corresponds to the largest of the sets in  $B_{k_{i,1}}, \dots, B_{k_{i,2}}$ , as indicated in line 19.

The algorithm terminates with a final colouring  $C^*$  when  $Q$  is empty. This will happen when there is no colour  $c$  for any colour class  $P(i)$  such that there exists vertices  $u$  and  $v$  that have a different number of neighbours with colour  $c$ . In other, more familiar terms, the algorithm terminates with final colouring  $C^*$  when  $C^*$  is an equitable colouring.

This implementation highlights an important property of CR; namely that it is *canonical* and *respects isomorphisms*. That is, if  $C_G$  and  $C_H$  are the stable colourings of isomorphic graphs  $G$  and  $H$ , respectively, and  $\varphi$  is an isomorphism between  $G$  and  $H$ , then  $C_G(v) = C_H(\varphi(v))$  for all vertices  $v$  of  $G$ .

**Proposition 2.2.2.** *Practical Colour Refinement is a canonical algorithm.*

*Proof.* Let  $G$  and  $H$  be isomorphic graphs, and  $\varphi$  be an isomorphism from  $G$  to  $H$ . Further, let  $C_G^i$  be the colouring of  $G$  after  $i$  iterations, and  $C_H^i$  be defined similarly. We prove that the stable colourings of  $G$  and  $H$  respect isomorphisms by showing  $C_G^i(v) = C_H^i(\varphi(v))$  for all  $v$  and for all  $i$ . This is done by induction on  $i$ , the number of refinement rounds.

Let  $v$  be a vertex of  $G$ .

Initially, all vertices get the same colour, so  $C_G^0(v) = 1 = C_H^0(\varphi(v))$ .

Now consider the  $i$ th iteration, and suppose that all previous colourings respect isomorphisms. The colour  $C_G^i(v)$  depends only which  $B_j$  contains  $v$  in line 15. The partitioning process that defines the  $B_j$ s depends only on the current colour of  $v$ , and the number of neighbours of  $v$  of a particular colour. Since the number of neighbours of a vertex of a particular colour and the colour of a vertex all respect isomorphisms by assumption, then so does  $C_G^i$  and  $C_H^i$ .  $\square$

This is all well and good, but does this final colouring  $C^*$  actually correspond to a stable colouring given by CR? It seems that the refining processes in our two implementations differ slightly. The  $C$ -equitability condition in Algorithm 2.1.4 appears to be doing more work than picking one colour and refining with respect to that in Algorithm 2.2.1.

To reconcile this, we define a general refining operation for a graph  $G = (V, E)$  which will encapsulate both scenarios. The idea will be that, for an existing partition  $\pi$  of  $V$ , we choose a union of cells of  $\pi$  to give a refining set  $R$ , and a second union of cells of  $\pi$  to be a splitting set  $S$ . We then break down the cells in  $S$  with respect to how many neighbours in cells of  $R$  each vertex has. The following definition formalises this idea.

**Definition 2.2.3** (Refining operation, [BBG17]). Let  $G = (V, E)$  be a graph, and let  $\pi$  and  $\pi'$  be partitions of  $V$ . For vertex sets  $R, S \subseteq V$  that are  $\pi$ -closed, we say that  $\pi'$  is obtained from  $\pi$  by a *refining operation*  $(R, S)$  if:

- (i) for every  $S' \in \pi$  with  $S' \cap S = \emptyset$ , it holds that  $S' \in \pi'$ , and
- (ii) for every  $u, v \in S$ :  $u \approx_{\pi'} v$  if and only if  $u \approx_{\pi} v$  and for all  $R' \in \pi$  with  $R' \subseteq R$ ,  $|N(u) \cap R'| = |N(v) \cap R'|$  holds.

Point (i) in the above definition says that if  $S'$  is a cell in  $\pi$  that has empty intersection with the splitting set  $S$ , then  $S'$  is also a cell of  $\pi'$ . In other words, cells disjoint from the splitting set don't get split.

Point (ii) in the above definition says that two vertices  $u$  and  $v$  are in the same cell in  $\pi'$  if and only if:

1. they are in the same cell of  $\pi$ ; and,
2. for each cell  $R'$  contained within the refining set  $R$ , the number of neighbours of  $u$  inside of  $R'$  is the same as the number of neighbours of  $v$  inside  $R'$ .

**Lemma 2.2.4.** *Both Algorithm 2.1.4 and Algorithm 2.2.1 implement refinement operations.*

*Proof.* For Algorithm 2.1.4, we observe that  $\mathcal{P}_i$  is obtained from  $\mathcal{P}_{i-1}$  via the refinement operation  $(V, V)$ . To see this, we first note that condition (i) in Definition 2.2.3 is vacuously true when  $S = V$ . For condition (ii), we cast our minds back to Remark 2.1.5, which (in essence) says that the cells of  $\mathcal{P}_i$  are made by splitting cells of  $\mathcal{P}_{i-1}$ . Therefore, it remains to show that for all vertices  $u, v \in V$ :  $u \approx_{\mathcal{P}_i} v$  if and only if for all  $R' \in \mathcal{P}_{i-1}$ ,  $|N(u) \cap R'| = |N(v) \cap R'|$ . But each cell  $R' \in \mathcal{P}_{i-1}$  corresponds to a unique colour in  $C_{i-1}$ , so it suffices to show that  $u \approx_{\mathcal{P}_i} v$  if and only if  $u$  and  $v$  have the same multi-set of colours of neighbours. This, however, is precisely condition (ii) in our definition of  $C$ -equitability! This is one of the conditions that is used to create  $\mathcal{P}_i$ , so we are done.

For Algorithm 2.2.1, we observe that the partition is repeatedly refined in the main while loop beginning on line 9. This is achieved through the refinement operation  $(P(q), V)$ , where  $P(q)$  is a single cell in  $P$  corresponding to the colour class of  $q$ . As above, condition (i) is vacuously true. Condition (ii) is satisfied by the construction of the new partition  $(B_1, \dots, B_k)$  shown in line 15.  $\square$

The above result establishes that both of our algorithms implement refinement operations. The next collection of results, adapted from [BBG17], aims to show that *any* collection of refinement operations that reaches a stable partition in fact finds the maximal stable partition.

**Lemma 2.2.5.** *Let  $\pi'$  be obtained from  $\pi$  by a refining operation  $(R, S)$ , and  $\rho$  be a stable partition that refines  $\pi$ . Then  $\rho$  also refines  $\pi'$*

*Proof.* Consider vertices  $u, v$  such that  $u \approx_{\rho} v$ . We aim to show that  $u \approx_{\pi'} v$ .

Since  $\rho$  refines  $\pi$ , we have  $u \approx_{\pi} v$ , and therefore  $u, v \in S'$  for some  $S' \in \pi$ . If  $S' \cap S = \emptyset$ , then  $S' \in \pi'$  by condition (i) in Definition 2.2.3, and  $u \approx_{\pi'} v$ .

Suppose, then, that  $S' \cap S \neq \emptyset$ . Then by  $\pi$ -closure,  $S' \subseteq S$ . Using the stability of  $\rho$  we observe that, for every cell  $X \in \pi$ ,  $|N(u) \cap X| = |N(v) \cap X|$ , because each cell  $X$  is  $\rho$ -closed and this property holds for all elements of  $\rho$ . In particular,  $|N(u) \cap R'| = |N(v) \cap R'|$  for every  $R' \in \pi$  with  $R' \subseteq R$ . Hence, by condition (ii) in Definition 2.2.3,  $u \approx_{\pi'} v$ .  $\square$

Informally, this result tells us that we can't use refinement operations to jump over stable partitions. That is, if our goal is to find a maximal stable partition, we are sure that refinement operations won't break down our partitions too quickly.

**Lemma 2.2.6.** *Let  $G = (V, E)$  be a graph. Then there is a unique maximal stable partition of  $V$ .*

*Proof adapted from [BBG17].* We first show that maximal (with respect to refinement) stable partitions of  $V$  exist. To see this we observe that the discrete partition  $D = \{\{v\} : v \in V\}$  refines  $V$  and is stable. Because  $\preceq$  is a finite partial order, there exists at least one maximal stable partition that refines  $V$ .

Now we prove uniqueness.

Suppose, for a contradiction, that  $V$  admits two distinct maximal stable partitions  $\rho_1$  and  $\rho_2$ . Choose  $\sigma$  to be a minimal partition of  $V$  such that both  $\rho_1$  and  $\rho_2$  refine  $\sigma$ . Certainly we have  $\rho_1$  and  $\rho_2$  both refine  $\{V\}$ , so  $\sigma$  is well defined.

We first note that  $\sigma$  is not itself stable, for if it was we would have  $\rho_1 = \sigma = \rho_2$  which contradicts distinctness. Therefore, there exists a refining operation  $(R, S)$  that can be applied to  $\sigma$  to yield a distinct  $\sigma'$ .

However, by [Lemma 2.2.5](#), we see that both  $\rho_1$  and  $\rho_2$  refine  $\sigma'$ , which contradicts the minimality of  $\sigma$ . Hence there is exactly one maximal stable partition of  $V$ .  $\square$

**Proposition 2.2.7.** *Both [Algorithm 2.1.4](#) and [Algorithm 2.2.1](#) produce the stable partition of a graph.*

*Proof adapted from [Gro+21].* First observe that, by [Lemma 2.2.4](#) and [Lemma 2.2.6](#), [Algorithm 2.1.4](#) produces the maximal stable partition of  $V$ . We now show that [Algorithm 2.2.1](#) does the same.


We first suppose that the loop defined in line 19 actually iterated over  $\{k_{i,1}, \dots, k_{i,2}\}$ , instead of the algorithm presently presented which removes  $i^*$ . This algorithm finds the maximal stable partition by [Lemma 2.2.4](#) and [Lemma 2.2.6](#).

We now argue why the removal of  $i^*$  is justified.

Suppose that we refine our graph with respect to colour  $q$ . In doing so, all of the vertices that had colour  $c$ , say, get reassigned to have colours  $d_1, d_2, \dots, d_l$ . Without loss of generality, assume that  $d_l$  is the colour assigned to the most vertices. For all  $v \in V$ , define  $n_c(v), n_{d_1}(v), \dots, n_{d_l}(v)$  to be the number of neighbours of  $v$  that have ever been assigned the colours  $c, d_1, \dots, d_l$  respectively. Then push the colours  $d_1, d_2, \dots, d_{l-1}$  to the back of the refinement queue.

Let  $p \in P$  be a cell in the partition of  $V$  after  $G$  has been refined with respect to  $d_1, \dots, d_{l-1}$ . Then for  $u, v \in P$ , we have  $n_c(u) = n_c(v), n_{d_1}(u) = n_{d_1}(v), \dots, n_{d_{l-1}}(u) = n_{d_{l-1}}(v)$ . Moreover, we have  $n_{d_l}(u) = n_c(u) - \sum_{i=1}^{l-1} n_{d_i}(u) = n_c(v) - \sum_{i=1}^{l-1} n_{d_i}(v) = n_{d_l}(v)$ . Hence the cells of  $P$  look as though we have refined  $G$  with respect to  $d_l$  without ever pushing  $d_l$  to the queue!  $\square$

This method of “processing the smaller half” was first considered in Hopcroft’s quasi-linear time algorithm for the minimisation of deterministic finite state automata [[Hop71](#)]. In our context of Colour Refinement, this is the idea that is central to the fastest implementations of CR which we discuss in the next section.

 To supplement this pseudo-code, and to provide an *even more* concrete implementation of the Colour Refinement algorithm, I have implemented a version of [Algorithm 2.2.1](#) in Python which can also be found in the supplementary files of this project and on GitHub.

## 2.3 Time complexity

**Algorithm 2.2.1** provides an explicit implementation of Colour Refinement so, as inquisitive mathematicians, we now want to know how fast the algorithm will run.

**Proposition 2.3.1.** *For a graph  $G$  with  $n$  vertices, **Algorithm 2.2.1** has time complexity  $\mathcal{O}(n^2 \log n)$*

*Proof sketch.* For full details, see [Gro+21].

To calculate the number of neighbours of  $v$  with colour  $q$  in line 12 requires time  $\mathcal{O}(|P(q)|)$ . The size of each colour class  $P(q)$  is at most  $n$ . This is done for each vertex  $v \in V$ , so takes time  $\mathcal{O}(n^2)$  to calculate all of them.

The remainder of the operations in the while loop can be done better than  $\mathcal{O}(n^2)$  by choosing fast sorting methods and data structures.

Thanks to our “processing the smaller half” technique, after each iteration of the while loop, the number of new colours that need to be added to the queue halves each time, so the number of iterations is  $\mathcal{O}(\log n)$ .

Combining these results gives the desired run time.  $\square$

This proposition tells us that, if the number of vertices of a graph grows linearly, the amount of time taken for **Algorithm 2.2.1** to terminate grows (at worst) like  $n^2 \log n$ . This provides a good upper bound as to how fast an algorithm that implements Colour Refinement can be. However, the question we really care about is what is the *fastest* we can hope an algorithm that implements Colour Refinement can be?

An algorithm that performs Colour Refinement in  $\mathcal{O}((n + m) \log n)$  time was first presented by Cardon and Crochemore that uses similar techniques to that of **Algorithm 2.2.1** [CC82]. The main difference is that, when refining a partition with respect to some colour  $q$ , any vertex that is not adjacent to a vertex of colour  $q$  keeps its same colour, so we are more selective about which colours get pushed to the queue. This was later simplified by Paige and Tarjan by refining with respect to old colour classes, not current ones, which leads to even fewer changes in the queue of colours [PT87].

So, can we do better than  $\mathcal{O}((n + m) \log n)$ ?

Under the assumption that a Colour Refinement algorithm starts with the trivial unit partition of vertices and iteratively applies refinement operations until a stable colouring is found, the following result of Berkholz, Bonsma and Grohe provides our answer.

**Theorem 2.3.2.** *For every integer  $k \geq 2$ , there exists a graph  $G_k$  with  $n \in \mathcal{O}(2^k k)$  vertices and  $m \in \mathcal{O}(2^k k^2)$  edges such that any implementation of Colour Refinement has running time in  $\Omega((n + m) \log n)$ .*

*Proof.* Omitted. We direct the reader to [BBG17] for a very thorough discussion of this result, which appears as Theorem 15.  $\square$



**Theorem 2.3.2** therefore guarantees that no implementation of Colour Refinement can have a running time better than  $\mathcal{O}((n + m) \log n)$ !

## 2.4 More examples

Having seen **Example 2.1.6**, we might be asking ourselves about what sort of properties we can infer from the colouring produced by Colour Refinement. Given a stable colouring, can we categorise all graphs that have that colouring? Are graphs that share the same stable colouring isomorphic to each other? Is CR the Holy Grail of all algorithms that solves all of our problems?

Regrettably, no.<sup>4</sup>

By revisiting the graphs we presented in **Example 1.2.20**, we show that there exist non-isomorphic graphs  $G$  and  $H$  that have the same stable colouring.

**Example 2.4.1.** We run **Algorithm 2.1.4** on the cycle graph on six vertices,  $C_6$ , and the disjoint union of two cycle graphs on three vertices,  $C_3 + C_3$ , as shown in **Figure 2.1**. Rather than giving a verbose step-by-step trace through the algorithm, we condense our calculations into the following tables.

For  $C_6$ , CR gives

Vertex $v$	First iteration		Second iteration	
	$C_0(v)$	Neighbour colours	$C_1(v)$	Neighbour colours
$v_1$	1	$\{\{1, 1\}\}$	2	$\{\{2, 2\}\}$
$v_2$	1	$\{\{1, 1\}\}$	2	$\{\{2, 2\}\}$
$v_3$	1	$\{\{1, 1\}\}$	2	$\{\{2, 2\}\}$
$v_4$	1	$\{\{1, 1\}\}$	2	$\{\{2, 2\}\}$
$v_5$	1	$\{\{1, 1\}\}$	2	$\{\{2, 2\}\}$
$v_6$	1	$\{\{1, 1\}\}$	2	$\{\{2, 2\}\}$

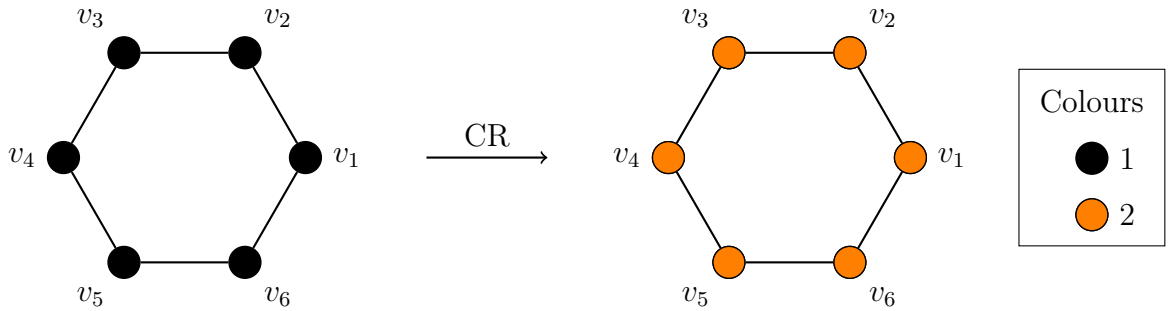
which shows that the partition of the vertices into colour classes is the same at each iteration. Therefore, the colouring  $C_1$  which colours every vertex with the colour 2 is stable, as demonstrated in **Figure 2.1a**.

For  $C_3 + C_3$ , CR gives

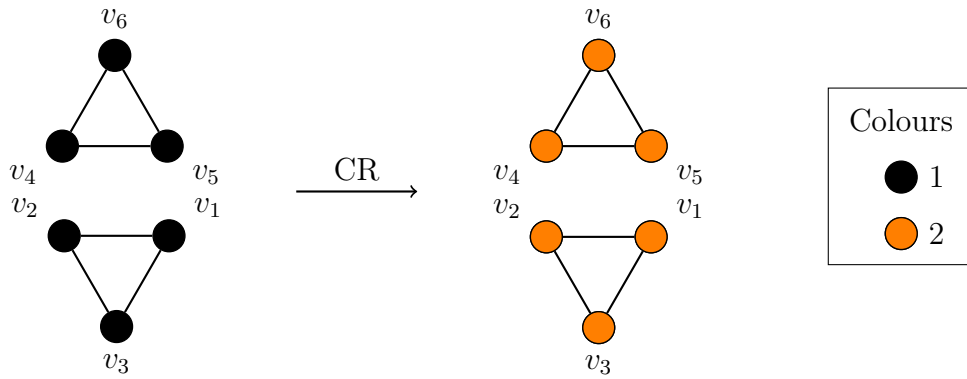
Vertex $v$	First iteration		Second iteration	
	$C_0(v)$	Neighbour colours	$C_1(v)$	Neighbour colours
$v_1$	1	$\{\{1, 1\}\}$	2	$\{\{2, 2\}\}$
$v_2$	1	$\{\{1, 1\}\}$	2	$\{\{2, 2\}\}$
$v_3$	1	$\{\{1, 1\}\}$	2	$\{\{2, 2\}\}$
$v_4$	1	$\{\{1, 1\}\}$	2	$\{\{2, 2\}\}$
$v_5$	1	$\{\{1, 1\}\}$	2	$\{\{2, 2\}\}$
$v_6$	1	$\{\{1, 1\}\}$	2	$\{\{2, 2\}\}$

<sup>4</sup>Incidentally, the graph we presented in **Example 2.1.6** is isomorphic to all graphs that share the stable colouring. If the news excites you, then you're in luck; graphs of this type are discussed extensively in **Chapter 3**!

which is, importantly, the same as that of  $C_6$ .



(a) The input and output of CR on  $C_6$ .



(b) The input and output of CR on  $C_3 + C_3$ .

Figure 2.1: The CR algorithm performed on two non-isomorphic graphs,  $C_6$  and  $C_3 + C_3$ , giving equal stable partitions.

**Proposition 2.4.2.** *The stable colouring of any regular graph is the trivial colouring that gives each vertex the same colour.*

*Proof.* Let  $G = (V, E)$  be a graph. Initially, all vertices have the same colour, say 1. Furthermore, each vertex has the same number of neighbours with colour 1 by regularity of  $G$ , so upon refining with respect to colour 1, the partition of vertices does not get broken down and all vertices get assigned colour 2. As before, all vertices now have the same number of neighbours with colour 2, so the  $C$ -equitable partition remains unchanged. Hence the trivial colouring is stable.  $\square$

**Example 2.4.1** shows that, through the eyes of CR,  $C_6$  and  $C_3 + C_3$  are indistinguishable, despite being non-isomorphic. Hence we have established that Colour Refinement can't distinguish *all* graphs up to isomorphism. Therefore, the next natural question to ask is: are there *any* graphs that Colour Refinement can distinguish up to isomorphism?<sup>5</sup> This is the question we concern ourselves with in the next chapter.

<sup>5</sup>If you read the previous footnote, you will know the answer to this question already. If not, strap yourselves in; it's about to get wild.

# Chapter 3

## Amenability

So far, we have used CR in two ways:

- to find the stable colouring of a single graph, as in [Example 2.1.6](#); and
- to try (and fail) to distinguish between two non-isomorphic graphs, as in [Example 2.4.1](#).

In this chapter, we define some more terminology that appears in [\[Arv+17\]](#) that will help us in our quest to ascertain whether Colour Refinement is a useful tool when checking if two graphs are isomorphic. Specifically, we define those graphs for which Colour Refinement works as an isomorphism test; namely *amenable* graphs. We will then proceed by exploring an algorithm that determines the amenability of graphs, and finish with a discussion about types of amenable graphs. So with that established, let's dive in!

### 3.1 Definitions

The following definitions have been taken from the introduction provided by Arvind et al. in [\[Arv+17\]](#), and serve as a way of identifying those graphs on which Colour Refinement ‘works’.

**Definition 3.1.1** (Succeeds, Amenable). Let  $G = (V_G, E_G)$  be a graph. Then for all graphs  $H = (V_H, E_H)$  non-isomorphic to  $G$ , define  $C_{G,H}: V_G \cup V_H \rightarrow \mathbb{N}$  be the colouring obtained by performing CR on the disjoint union graph  $G + H$  of  $G$  and  $H$ . We say that CR *succeeds* on  $G$  if for all such  $C_{G,H}$ , the restriction of  $C_{G,H}$  to vertices of  $G$  is distinct from the restriction of  $C_{G,H}$  to vertices of  $H$ .

A graph on which Colour Refinement succeeds is called *amenable*.

This definition finally gives us the classification of graphs we have been hoping for! If a graph is amenable, we can algorithmically check whether it is isomorphic to another graph in polynomial time.

**Theorem 3.1.2.** *Let  $G$  be an amenable graph,  $H$  be an arbitrary graph, and  $C_{G,H}$  be the colouring obtained by performing CR on the disjoint union graph  $G + H$  of  $G$  and  $H$ . Then  $G$  and  $H$  are isomorphic if and only if the restriction of  $C_{G,H}$  to vertices of  $G$  is the same as the restriction of  $C_{G,H}$  to vertices of  $H$ .*

*Proof.* This follows from the definition. □

The next thing to check is whether this is a worthwhile definition that actually describes a nonempty collection of graphs.

**Definition 3.1.3** (Discrete). Let  $G$  be a graph.  $G$  is *discrete* if the stable partition consists only of singletons.

**Proposition 3.1.4.** *The following families of graphs are amenable:*

- (i) *Graphs determined up to isomorphism by their degree sequence<sup>1</sup>; and*
- (ii) *Discrete graphs.*

Before we prove this result, we first prove a technical lemma.

**Lemma 3.1.5.** *Let  $G = (V, E)$  be a graph,  $C: V \rightarrow \mathbb{N}$  be the stable CR colouring of  $G$ , and  $u, v \in V$  be vertices such that  $C(u) = C(v)$ . Then for every intermediate colouring  $C_i$  in the CR algorithm,  $C_i(u) = C_i(v)$ .*

This says that each colour of the stable colouring can only be derived in one way, i.e. we can't have two vertices that start off as the same colour, diverge for a bit, and then end up as the same colour.

*Proof.* Let  $G = (V, E)$  be a graph, and  $u, v$  be vertices of  $G$  that have the same stable colouring. Suppose, for a contradiction, that  $u$  and  $v$  do not have the same intermediate colouring at every iteration. Let  $j$  be the smallest integer such that  $C_j(u) = C_j(v)$ , but  $C_{j-1}(u) \neq C_{j-1}(v)$ . Recall that the colouring  $C_j$  is obtained by partitioning  $V$  into  $C_{j-1}$ -equitable cells. In particular, by (i) in [Definition 2.1.1](#), each vertex that has the same  $C_j$  colour must also have the same  $C_{j-1}$  colour, so  $C_{j-1}(u) = C_{j-1}(v)$ . This is a contradiction, so the colour of  $u$  and  $v$  is the same at every iteration. □

We can now prove our proposition.

*Proof of Proposition 3.1.4.* For part (i), let  $G$  be a graph that is determined up to isomorphism by its degree sequence, let  $H$  be a graph non-isomorphic to  $G$ , and consider CR applied to  $G + H$ . First, we observe that  $G$  and  $H$  must have distinct degree sequences. Next, we recall that the first refinement round of CR partitions the set of vertices into cells of constant degree. Therefore, after the first refinement round, the restriction of the colouring to  $G$  and the restriction of the colouring to  $H$  must be distinct. By the above technical lemma, all subsequent colourings—in particular the stable colouring—must also be distinct, so  $G$  is amenable.

For part (ii), let  $G = (V_G, E_G)$  be a discrete graph and suppose, for a contradiction, that  $G$  is not amenable. Therefore, there exists a graph  $H = (V_H, E_H)$  non-isomorphic to  $G$  such that the restriction of the stable colouring  $C$  of  $G + H$  to vertices of  $G$  is the same as the restriction to vertices of  $H$ . Importantly, both of these restrictions use the same colours and give rise to singleton colour classes. Define  $\varphi: V_G \rightarrow V_H$  to be the map that sends the unique vertex in  $V_G$  of colour  $c$  to the unique vertex of  $V_H$  of colour  $c$  for each  $c \in \text{Im}(C)$ .

---

<sup>1</sup>These graphs are sometimes referred to as *unigraphs*.

Let  $u, v \in V_G$  such that  $C(u) = i$  and  $C(v) = j$ . Then  $u$  and  $v$  are adjacent if and only if there is an edge  $e \in E_G$  where one endpoint has colour  $i$ , and the other has colour  $j$ . By properties of Colour Refinement, all vertices of colour  $i$  must be adjacent to the same number of vertices of colour  $j$ . In particular,  $C(\varphi(u)) = C(u) = i$ , and so  $\varphi(u)$  must be adjacent to a vertex of colour  $j$ . The only such choice of vertex is  $\varphi(v)$ . Hence  $\{u, v\} \in E_G$  if and only if  $\{\varphi(u), \varphi(v)\} \in E_H$ , so  $G$  and  $H$  are isomorphic. This contradicts our assumptions, thus  $G$  is amenable  $\square$

**Proposition 3.1.4** addresses some of our biggest fears about CR and, fortunately, shows that amenable graphs exist! Therefore, if we can determine whether an arbitrary graph is amenable, we uncover if CR can be used as a fast test for isomorphism. So now, having defined amenability, we continue our adventure through Isomorphism Land<sup>2</sup> by considering the following question. *For some arbitrary graph  $G$ , can we determine if  $G$  is amenable?*

## 3.2 Checking for amenability

In **Definition 3.1.1**, the condition for a graph  $G$  to be amenable is of the form “for all  $H$  non-isomorphic to  $G \dots$ ”. This might suggest that an algorithm for determining amenability somehow needs a knowledge of all graphs non-isomorphic to  $G$ , which seems like a lot of work! Fortunately, however, [Arv+17] presents a collection of local and global graph properties that are necessary and sufficient to decide amenability. It is these conditions that we concern ourselves with in this section. To start, we present some useful definitions.

**Definition 3.2.1** (Cell graph, Isotropic, Homogeneous). Let  $G = (V, E)$  be a graph with stable colouring  $C$ , and stable partition  $\pi_C$ . We define the *cell graph*  $\text{Cell}(G)$  to be the complete graph with vertices corresponding to cells of  $\pi_C$ .

We call a vertex  $X$  of  $\text{Cell}(G)$  *homogeneous* if the induced subgraph  $G[X]$  of  $G$  is complete or empty, and *heterogeneous* otherwise.

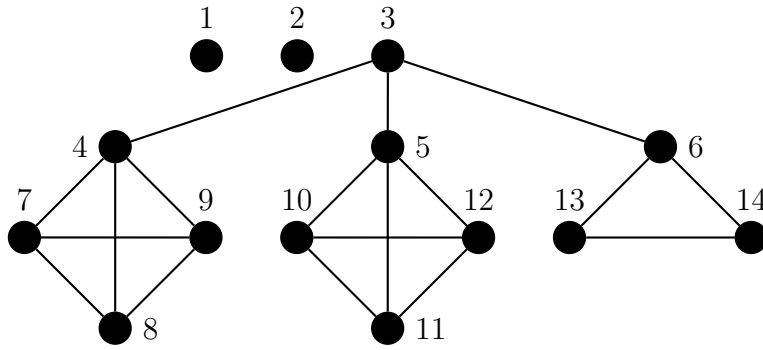
We call an edge  $\{X, Y\}$  of  $\text{Cell}(G)$  *isotropic* if the induced bipartite graph  $G[X, Y]$  of  $G$  is either complete bipartite or empty, and *anisotropic* otherwise. We call a path  $X_1 X_2 \dots X_l$  an *anisotropic path* if each  $\{X_i, X_{i+1}\}$  is anisotropic. We say that vertices  $X$  and  $Y$  are *anisotropically connected* if there exists an anisotropic path between  $X$  and  $Y$ . We call the partition of the vertices of  $\text{Cell}(G)$  induced by anisotropic connectivity *anisotropic components*.

There is a lot going on in this definition, so let’s do an example.

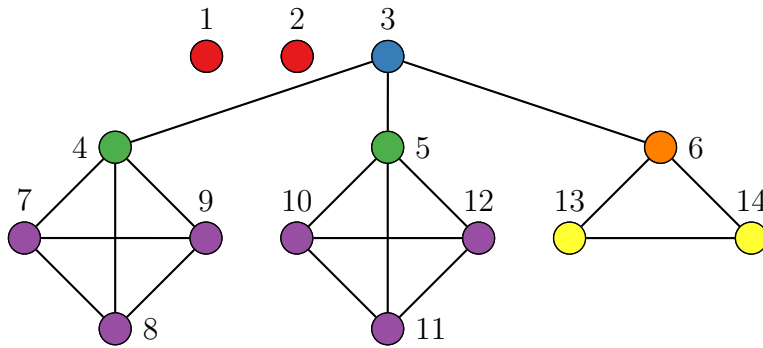
**Example 3.2.2.** We consider the following graph  $G$  and compute its cell graph  $\text{Cell}(G)$ .

---

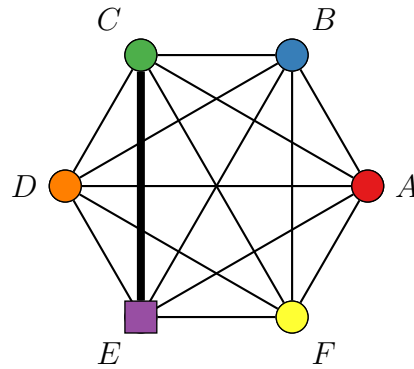
<sup>2</sup>Legend has it that this was the original setting of Lewis Carroll’s best seller, but Wonderland appealed more to the editors.



 The first step is to compute the stable colouring.



Then we create a complete graph with one vertex for each colour class of the stable colouring. In our example, we have six colour classes, so our cell graph will be the complete graph  $K_6$ .



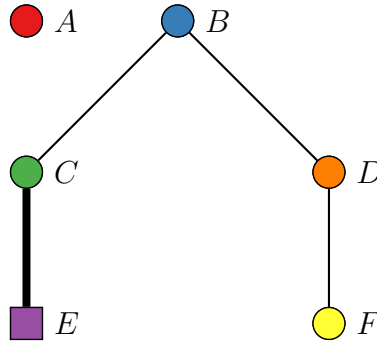
In the image above, the circular vertices represent homogeneous cells. For  $A$ ,  $B$ ,  $C$  and  $D$ , this is because the induced subgraphs  $G[A]$ ,  $G[B]$ ,  $G[C]$  and  $G[D]$  are empty. For  $F$ , this is because the induced subgraph  $G[F]$  is complete. The induced subgraph  $G[E]$  is neither complete nor empty (it is  $2K_3$ ) so  $E$  is heterogeneous, which is represented with a square.

Furthermore, the non-bold edges represent isotropic edges. For example, the edge  $\{A, C\}$  is isotropic because the induced bipartite subgraph  $G[A, C]$  is empty. The edge  $\{B, C\}$  is isotropic because the induced bipartite subgraph  $G[B, C]$  is complete. On the other hand, the induced

bipartite subgraph  $G[C, E]$  is neither complete nor empty (it is  $2K_{1,3}$ ), so the edge  $\{C, E\}$  is anisotropic, which is represented by a bold line.

As there is only one anisotropic edge, there is only one anisotropic path, and the set of anisotropic components is  $\{\{A\}, \{B\}, \{D\}, \{F\}, \{C, E\}\}$ .

Later in this section, we consider the graph obtained by removing all of the empty edges from  $\text{Cell}(G)$ . This offers a more visually intuitive way of seeing the adjacency of different colour classes, as indicated below.



With these definitions established, we have all of the language necessary to describe the conditions that guarantee amenability! For consistency, we label our conditions in the same way as in [Arv+17].

Let  $G = (V, E)$  be a graph with stable colouring  $C$ , and stable partition  $\pi_C$ . The first two conditions will relate amenability to the local structure of the graph. Informally, if I am standing on a vertex, what do my neighbours look like?

- (A) For each cell  $X \in \pi_C$ , the induced subgraph  $G[X]$  of  $G$  is one of:
  - (i) an empty graph;
  - (ii) a complete graph;
  - (iii) a matching graph  $mP_1$ ;
  - (iv) the complement of a matching graph; or
  - (v) the five cycle  $C_5$ .
- (B) For all pairs of cells  $X, Y \in \pi_C$ , the induced bipartite subgraph  $G[X, Y]$  of  $G$  is one of:
  - (i) an empty graph;
  - (ii) a complete bipartite graph;
  - (iii) a disjoint union of stars  $sK_{1,t}$  where  $X$  is the set of  $s$  central vertices, and  $Y$  is the set of  $st$  leaf vertices;
  - (iv) a disjoint union of stars  $sK_{1,t}$  where  $Y$  is the set of  $s$  central vertices, and  $X$  is the set of  $st$  leaf vertices; or,

- (v) the complement of a graph defined in (iii) or (iv).

The next two conditions relate to global graph properties—those properties that describe the relationship *between* cells of  $\pi_C$ . In particular, these conditions describe the properties of each anisotropic component  $A$  of  $\text{Cell}(G)$ .

- (G) Each anisotropic component  $A$  is a tree. Furthermore, let  $M \in A$  be a cell of minimal cardinality, and  $A_M$  be the rooted, directed tree<sup>3</sup> obtained from  $A$  by rooting  $A$  at  $M$ . Then for cells  $X, Y \in A$  with  $Y$  being a child of  $X$  we have  $|X| \leq |Y|$ .
- (H) Each anisotropic component  $A$  contains at most one heterogeneous vertex. If  $T$  is such a vertex, it has minimum cardinality among the cells of  $A$ .

As alluded to at the beginning of this section, these conditions are both necessary and sufficient for a graph to be amenable.

**Theorem 3.2.3.** *Let  $G = (V, E)$  be a graph with stable colouring  $C$ , and stable partition  $\pi_C$ . Then  $G$  is amenable if and only if  $G$  satisfies conditions (A), (B), (G) and (H).*

*Proof.* Omitted. See Theorem 4.1 in [Arv+17]. □

*Exercise.* Using the above result, determine whether or not the graph in Example 3.2.2 is amenable. If not, which of the conditions does it fail to meet?

This result tells us that we can recognise amenable graphs. However, perhaps one of the conditions above is obfuscating lots of hard-to-compute details, and these conditions are very difficult to check. For the remainder of this section, we present some results relating to these conditions, describe an algorithm from [Arv+17] that checks these conditions, and describe the algorithm's time complexity.

Our first result gives a complete characterisation of graphs that satisfy conditions (A) and (B).

**Proposition 3.2.4.** *Let  $G = (V, E)$  be a graph with stable colouring  $C$ , and stable partition  $\pi_C$ . Then the validity of conditions (A) and (B) are determined by the size  $|X_i|$  of each cell  $X_i \in \pi_C$ , and the number  $d_{i,j}$  of neighbours in  $X_j$  a vertex  $u \in X_i$  has, for each  $X_i, X_j \in \pi_V$ .*

*Remark 3.2.5.* In the above proposition,  $d_{i,j}$  is independent of the choice of  $u \in X_i$ , and is therefore well-defined. To see this, recall that the definition of  $C$ -equitability ensures each vertex in  $X_i$  has the same multiset of colours of neighbours. In particular, each vertex in  $X_i$  has the same number of neighbours in  $X_j$ .

*Proof.* Condition (A) is satisfied if and only if for each  $X_i \in \pi_C$ , the values  $d_{i,i}$  and  $|X_i|$  coincide with one of the following.

---

<sup>3</sup>A *rooted directed tree* is a tree where a vertex is identified as a root, and each edge has a direction that points away from the root. For more on rooted directed trees, see [Val21].



$ X_i $	$d_{i,i}$	Corresponding induced subgraph $G[X_i]$
Arbitrary	0	The empty graph
Arbitrary	$ X_i  - 1$	The complete graph $K_{d_{i,i}}$
Arbitrary	1	The matching graph $(\frac{d_{i,i}}{2})P_1$
Arbitrary	$ X_i  - 2$	The complement of the matching graph $(\frac{d_{i,i}}{2})P_1$
5	2	The 5 cycle $C_5$

Condition **(B)** is satisfied if and only if for each  $X_i, X_j \in \pi_C$  with  $i \neq j$ , the values  $d_{i,j}$  coincides with one of the following.

$d_{i,i}$	Corresponding induced bipartite subgraph $G[X_i, X_j]$
0	The empty bipartite graph
$ X_j $	The complete bipartite graph $K_{ X_i ,  X_j }$
1	The disjoint union of stars $ X_i  K_{1,  X_j }$
$ X_j  /  X_i $	The disjoint union of stars $ X_j  K_{1,  X_i }$
$ X_j  - 1$	The complement of the disjoint union of stars $ X_i  K_{1,  X_j }$
$ X_j  -  X_j  /  X_i $	The complement of the disjoint union of stars $ X_j  K_{1,  X_i }$

□

This characterisation allows us to quickly check whether conditions **(A)** and **(B)** hold. In [Algorithm 3.2.6](#), we present an algorithm that uses this result to check whether a given graph is amenable. This algorithm appears as the proof of Corollary 11 in [\[Arv+17\]](#), which shows amenable graphs are recognisable, and therefore we do not prove its validity here.

---

**Algorithm 3.2.6** Check amenability [\[Arv+17\]](#)

---

On input graph  $G = (V, E)$ :

1. Perform CR to find the stable colouring  $C$  and corresponding vertex partition  $\pi_C = \{X_1, \dots, X_l\}$ . Let  $\text{Cell}^*(G)$  be the graph obtained by removing all empty edges from the cell graph  $\text{Cell}(G)$  of  $G$ .
  2. Calculate the adjacency list of each vertex  $X_i$  of  $\text{Cell}^*(G)$  by picking some arbitrary vertex  $u \in X_i \subseteq E$ , looking at the neighbours  $v$  of  $u$ , and returning the cells  $X_j$  that contain a neighbour  $v$ . We are free to choose  $u$  arbitrarily as each vertex  $x \in X_i$  necessarily has the same multiset of colours of neighbours.
  3. As we are constructing the adjacency list, for each pair  $(i, j)$  such that  $i = j$  or  $\{X_i, X_j\}$ , record the number  $d_{i,j}$  of neighbours in  $X_j$  a vertex  $u \in X_i$  has.
  4. For each  $i, j \in \{1, 2, \dots, l\}$ , using  $d_{i,j}$ ,  $|X_i|$  and  $|X_j|$ , determine the structure of the induced subgraphs  $G[X_i]$  and the induced bipartite subgraph  $G[X_i, X_j]$ . Use this to test conditions **(A)** and **(B)** as in [Proposition 3.2.4](#).
  5. Using breadth-first search<sup>4</sup>, find all anisotropic components  $A$  of  $\text{Cell}(G)$  and check that they are trees with at most one heterogeneous cell. Record the cell  $M$  with the minimum cardinality and, if one was found, check  $M$  corresponds to a heterogeneous cell for **(H)**.
  6. Within each anisotropic component, use breadth-first search from  $M$  to check whether the monotonicity condition in **(G)** is satisfied.
-

**Proposition 3.2.7.** *With input graph  $G$  with  $n$  vertices and  $m$  edges, [Algorithm 3.2.6](#) has run time  $\mathcal{O}((n + m) \log n)$ .*

*Proof sketch.* For Step 1, by results from [Section 2.3](#), the stable colouring and partition can be calculated in  $\mathcal{O}((n + m) \log n)$ . Steps 2 and 3 check over all edges of  $G$  which is at worst  $\mathcal{O}(n + m)$ . For Step 4, with the data obtained in Step 3, conditions **(A)** and **(B)** can be checked in  $\mathcal{O}(1)$  thanks to [Proposition 3.2.4](#). For Step 5, finding all anisotropic components with breadth-first search potentially requires looking at all edges, and is  $\mathcal{O}(m)$ . Checking if the anisotropic components are all trees may require looking at all edges and all vertices (depending on the structure of the components) and is at worst  $\mathcal{O}(n + m)$ .<sup>5</sup> Checking heterogeneity for **(H)** is constant time for each component with the use of [Proposition 3.2.4](#), and there are at most  $n$  components;  $\mathcal{O}(n)$  in total. For Step 6, as with checking components are trees, checking monotonicity of each rooted component is  $\mathcal{O}(m + n)$ .

Combining these results gives the required complexity. □



As with the pseudo-code algorithm presented earlier, I have implemented [Algorithm 3.2.6](#) in Python, which is available to view in the supplementary files with this project and on GitHub. Many of the graphs that we discuss in the remainder of this project were found with the help of this implementation.

### 3.3 Types of amenable graphs

We established in [Section 1.3](#) that, in general, we currently have no way of quickly checking if two arbitrary graphs are isomorphic. However, at this point in our journey, we have shown that amenable graphs are precisely the graphs for which Colour Refinement can provide a fast isomorphism test. Therefore, from the perspective of fast isomorphism testing, it would be helpful to have a list of graphs known to be amenable. Finding families of graphs that are amenable will be the motivation for the rest of this dissertation.

So far, we have seen two examples of amenable graphs: unigraphs and discrete graphs. Furthermore, we have developed enough machinery to show that trees and forests are amenable, using [Theorem 3.2.3](#) (see Corollary 5.1 in [\[Arv+17\]](#)). However, up until now, we have been following the ideas presented in [\[Arv+17\]](#) quite closely. Let's try and find some amenable graphs of our own.

One place that might be sensible to look is graphs with trivial automorphism group—or *asymmetric graphs*. Since amenability is so intimately related to Graph Isomorphism, we might expect graphs that have no non-trivial automorphisms to be distinguished by Colour Refinement, and therefore amenable. Unfortunately, this isn't the case, as witnessed by the well-known and well-studied Frucht graph, shown in [Figure 3.1](#).

<sup>4</sup>The breadth-first search algorithm was first presented to find the shortest path through a maze in [\[Moo59\]](#). A modern and thorough discussion of this algorithm can be found in Section 3.3 of [\[Jun13\]](#).

<sup>5</sup>Depending on the model of computation and the data types being used, this may be reduced, but we present  $\mathcal{O}(m + n)$  to be safe.

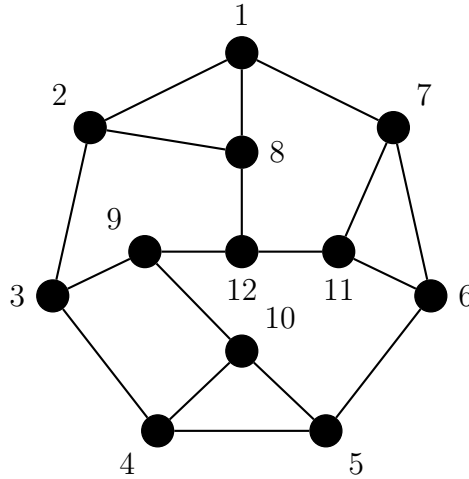


Figure 3.1: The Frucht Graph, first described in [Fru49]. This is the smallest three-regular asymmetric graph. By virtue of its regularity, it has trivial stable colouring by Proposition 2.4.2, so there is only one cell in the cell graph. As the Frucht Graph is not empty, complete, matching, the complement of a matching, or a five cycle, it does not satisfy condition (A). Therefore, it is not amenable, showing that not all asymmetric graphs are amenable.

Okay, so the Frucht graph shows us not all asymmetric graphs work; however, the Frucht graph is regular. As we identified in Proposition 2.4.2, regular graphs always yield a trivial colouring, so perhaps regularity is a special case. Instead, maybe we should restrict our attention to non-regular asymmetric graphs.

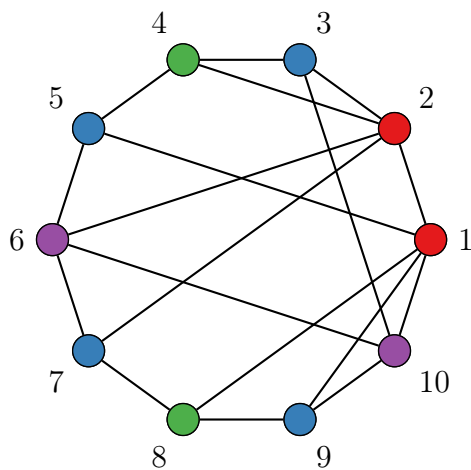
Again, unfortunately, these graphs aren't all amenable either, as witnessed by the less-well-known and presently unnamed graphs shown in Figure 3.2, that have been coloured with their stable colourings.<sup>6</sup> The graph with ten vertices shown in 3.2a, which will henceforth be referred to as  $J_{10}$ , was found by using SageMath's graph generator tool `nauty_geng` and an implementation of Colour Refinement, by enumerating all graphs with ten or fewer vertices [The22; MP14]. The graph with twelve vertices shown in 3.2b, which will henceforth be referred to as  $J_{12}$ , was found similarly but by enumerating graphs with maximum degree at most three and twelve or fewer vertices.

When searching for graphs with these properties, we found that there are no graphs with fewer than ten vertices that are asymmetric, non-regular and non-amenable, hence  $J_{10}$  is one of the joint smallest such examples.<sup>7</sup> We also found that  $J_{12}$  is the *unique* smallest asymmetric, non-regular, non-amenable graph with maximum degree at most three.

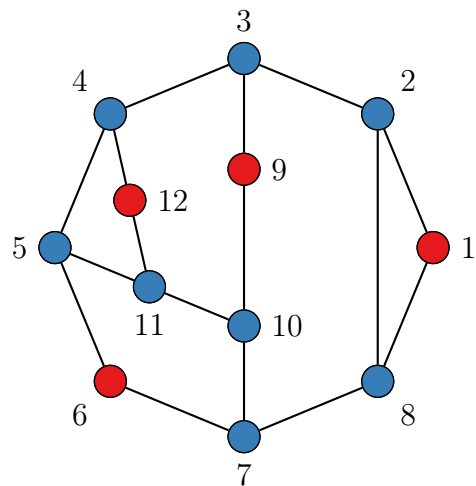
These examples show that, if we want to find any new amenable graphs, we will need to be slightly more creative with where we look. In the next chapter, we discuss an equivalent formulation of the Graph Isomorphism problem that concerns itself with adjacency matrices and explore amenability from this new perspective.

<sup>6</sup>With the results we prove in the next chapter, knowing the graphs are asymmetric but have a non-discrete colouring will be enough to show they are not amenable.

<sup>7</sup>There are 80 in total, all of which are presented in the supplementary files and on GitHub.



(a) The joint smallest asymmetric, non-regular, non-amenable graph;  $J_{10}$ .



(b) The unique smallest asymmetric, non-regular, non-amenable graph with maximum degree at most three;  $J_{12}$ .

Figure 3.2: Two small graphs with their stable colouring that demonstrate not all asymmetric, non-regular graphs are amenable.

# Chapter 4

## Relaxations

In [Section 1.2](#), we defined an isomorphism between two graphs  $G$  and  $H$  to be an edge-preserving bijection between the vertices of  $G$  and the vertices of  $H$ . When thinking in this way, we can informally regard an isomorphism as a map that takes a drawing of  $G$  and ‘geometrically morphs’ it into a different drawing (with potentially different vertex labels).<sup>1</sup>

A different way to think about an isomorphism between graphs  $G = (V_G, E_G)$  and  $H = (V_H, E_H)$  is to consider how it acts on their adjacency matrices  $A$  and  $B$  respectively. Every isomorphism  $\varphi: V_G \rightarrow V_H$  corresponds with some *permutation matrix*  $P$  such that

$$AP = PB, \tag{4.1}$$

where we recall from linear algebra that a permutation matrix is a square matrix of 0s and 1s such that each row and each column contains exactly one 1. In what follows, through a slight abuse of notation, we will identify  $\varphi$  with  $P$  and refer to both as isomorphisms.

With this idea in mind, we can rephrase the problem of finding an isomorphism between  $G$  and  $H$  as the problem of finding permutation matrices  $P$  that satisfy [Equation \(4.1\)](#).

But if we have already shown that testing for isomorphism is hard and slow, why are we considering this equivalent formulation of the problem? That’s a very good question! When dealing with hard problems, a common technique is to relax our constraints to see if we can learn anything from the ‘almost-solutions’. The formulation above lends itself nicely to a relaxation which we discuss now by presenting the key ideas from [\[Tin91\]](#).

**Definition 4.0.1** (Doubly stochastic). Let  $M = [m_{i,j}] \in \mathbb{R}_{n \times n}$  be a real-valued square matrix. We say that  $M$  is *doubly stochastic* if it is a solution to the problem

$$M\mathbf{1} = M^T\mathbf{1} = \mathbf{1}, \quad m_{i,j} \geq 0 \quad \text{for } 1 \leq i, j \leq n \tag{4.2}$$

where  $\mathbf{1} \in \mathbb{R}_{n \times 1}$  is the  $n \times 1$  vector of 1s and  $M^T$  is the transpose of  $M$ .

---

<sup>1</sup>Some really nice visualisations with this idea in mind can be found at <https://sollami.ai/code/2014/12/24/graph-isomorphisms>.

This definition says that  $M$  is doubly stochastic if all of its entries are non-negative, and the sum of the entries in any row or column is 1. Informally, we will think of doubly stochastic matrices as ‘almost-permutation’ matrices. Fortunately, we see that every permutation matrix  $P$  is doubly stochastic. The following result strengthens the relationship between doubly stochastic matrices and permutation matrices.

Recall that a *convex sum* over  $\{z_1, z_2, \dots, z_m\}$  is any sum  $\sum_{i=1}^m \alpha_i z_i$  such that  $\sum_{i=1}^m \alpha_i = 1$ , and that the *convex hull* of a set is the set of all convex sums.

**Theorem 4.0.2** (Birkhoff–von Neuman Theorem). *The convex hull of the set of permutation matrices is the set of doubly stochastic matrices.*

*Proof.* Omitted. See [Hur08]. □

Combining the ideas captured by Equation (4.1) and Equation (4.2), using terminology consistent with [Arv+17], we define our relaxation of the Graph Isomorphism problem as follows.

**Definition 4.0.3** (Fractional isomorphism, Fractional automorphism). Let  $G$  and  $H$  be graphs on  $n$  vertices with adjacency matrices  $A$  and  $B$ , respectively. Then a doubly stochastic matrix  $X \in \mathbb{R}_{n \times n}$  is said to be a *fractional isomorphism* between  $G$  and  $H$  if  $AX = XB$ . Where it is necessary to emphasise the difference between fractional isomorphisms and isomorphisms, we may refer to an isomorphism as a *full isomorphism*.

A *fractional automorphism* is any fractional isomorphism from  $G$  to itself.

Notably, all full isomorphisms are indeed fractional isomorphisms. The following result characterises those graphs where the full and fractional automorphisms coincide.

**Proposition 4.0.4.** *Let  $G$  be a graph. If the fractional and full automorphisms of  $G$  coincide, then  $G$  is asymmetric.*

*Proof.* Let  $G$  be a graph with non-trivial automorphism group and adjacency matrix  $A$ . Then  $G$  has two distinct automorphisms represented by permutation matrices  $P$  and  $Q$ . Observe that  $X = \alpha P + (1 - \alpha)Q$  is a doubly stochastic matrix for all  $\alpha \in [0, 1]$  by the Birkhoff–von Neuman Theorem. Also,

$$\begin{aligned} XA &= (\alpha P + (1 - \alpha)Q)A \\ &= \alpha PA + (1 - \alpha)QA \\ &= \alpha AP + (1 - \alpha)AQ \\ &= A(\alpha P + (1 - \alpha)Q) \\ &= AX, \end{aligned}$$

so  $X$  is a fractional automorphism of  $G$ . For  $\alpha \in (0, 1)$ ,  $X$  is not a permutation matrix, hence the automorphisms of  $G$  do not coincide with the fractional automorphisms of  $G$ . □

In [Tin91], Tinhofer goes on to describe a type of graph that behaves nicely with regard to fractional automorphisms, which we present now.

**Definition 4.0.5** (Compact). Let  $G = (V, E)$  be a graph with  $n$  vertices and adjacency matrix  $A \in \mathbb{R}_{n \times n}$ . Let  $\mathcal{X} = \{X \in \mathbb{R}_{n \times n} : AX = XA, X \text{ doubly stochastic}\}$  be the set of doubly stochastic matrices that commute with  $A$ . If each  $X \in \mathcal{X}$  is expressible as the convex sum of some automorphisms of  $G$ , then we say that  $G$  is *compact*.

In the remainder of this chapter, using further results from [Arv+17] and [FS15], we explore how these compact graphs interact with the notion of amenability. If we're lucky, we might even discover some new amenable graphs!

## 4.1 The hierarchy

At the beginning of Section 3.1, we presented the definition of discrete graphs—those graphs that yield a stable colouring with singleton colour classes. In Proposition 3.1.4 we showed that they were amenable. However, the path graph  $P_2$  of length 2 vertices can be shown to be amenable but not discrete.<sup>2</sup> Hence we have the relation

$$\text{Discrete} \subsetneq \text{Amenable}. \quad (4.3)$$

In an attempt to add more tiers to this hierarchy, we present another family of graphs. This family of graphs is introduced in [Arv+17] and relates the stable colouring of a graph to the orbit partition of its automorphism group  $\text{Aut}(G)$ .

**Definition 4.1.1** (Refinable). Let  $G = (V, E)$  be a graph with stable colouring  $C$ , and stable partition  $\pi_C$ . We say that  $G$  is *refinable* if  $\pi_C$  is equal to the orbit partition of the automorphism group  $\text{Aut}(G)$ .

Refinable graphs behave very nicely with our concept of amenability, as the following result shows.

**Proposition 4.1.2.** *All amenable graphs are refinable.*

To prove this proposition, we first prove a technical lemma.

**Lemma 4.1.3.** *Let  $G = (V, E)$  be a graph with stable colouring  $C$ , and stable partition  $\pi_C$ . Then the orbit partition of  $\text{Aut}(G)$  is a refinement of  $\pi_C$ .*

*Proof.* We show that any two vertices in the same orbit must map to the same colour. Let  $u, v \in V$  be two vertices of  $G$  that are in the same orbit. Then necessarily there exists some automorphism  $\varphi \in \text{Aut}(G)$  such that  $\varphi(u) = v$ . Since Colour Refinement respects isomorphisms by Proposition 2.2.2, and in particular automorphisms, we see that  $C(u) = C(\varphi(u)) = C(v)$ .  $\square$

This result tells us that all vertices of a graph that are in the same orbit must have the same colour. Therefore a graph is not refinable if and only if there are two vertices in different orbits that map to the same colour.

*Proof of Proposition 4.1.2.* We prove the contrapositive. Let  $G = (V_G, E_G)$  be a graph that is not refinable. Then by the above lemma, there exist two vertices  $u, v \in V$  in different orbits

---

<sup>2</sup>In fact, every path graph of length greater than 1 is amenable but not discrete. Can you see why?

that map to the same colour. Define  $H = (V_H, E_H)$  to be the graph obtained by swapping the labels of vertices  $u$  and  $v$  in  $G$ . Note that  $G$  and  $H$  are not isomorphic by assumption. However, the colouring of  $H$  coincides with the colouring of  $G$ , so  $G$  is not amenable.  $\square$

**Proposition 4.1.2** gives us the relation

$$\text{Amenable} \subseteq \text{Refinable}. \quad (4.4)$$

At this point, since we went to the effort of defining them at the start of the chapter, we might be wondering how compact graphs fit into this hierarchy. Maybe that will give us a deeper understanding of amenable graphs. Maybe it doesn't fit in at all. The following theorem, which ties together [Equation \(4.3\)](#) and [Equation \(4.4\)](#), gives the answer.

**Theorem 4.1.4.** *The classes Discrete, Amenable, Compact and Refinable form the inclusion chain*

$$\text{Discrete} \subsetneq \text{Amenable} \subsetneq \text{Compact} \subsetneq \text{Refinable}.$$

*Proof.* Omitted. The interested reader is directed to the proof of Theorem 7.6 in [\[Arv+17\]](#), which is a stronger version of our theorem here.  $\square$

One point to take away from the above theorem is that, in general, compact does *not* imply amenable. However, if a graph has trivial automorphism group, then the orbit partition of  $\text{Aut}(G)$  consists purely of singletons. Therefore, if an asymmetric graph is discrete, then it is also refinable, and the whole hierarchy collapses. Hence, in the setting of asymmetric graphs, we can use methods of constructing and/or recognising compact graphs as a way to construct and/or recognise amenable graphs.

**Corollary 4.1.5.** *For asymmetric graphs*

$$\text{Discrete} = \text{Amenable} = \text{Compact} = \text{Refinable}.$$

We are now at the stage where we can continue the search for amenable graphs that we began in [Section 3.3](#). In the next section, we try and understand what it means to be an asymmetric compact graph and attempt to find some examples.

## 4.2 Amenable graphs: Round 2

To begin this section, we explore what the added requirement of asymmetry does to our definition of compactness.

Let  $G = (V, E)$  be an asymmetric graph with  $n$  vertices and adjacency matrix  $A \in \mathbb{R}_{n \times n}$ . Then the only automorphism of  $G$  is represented by the  $n \times n$  identity matrix  $I_n$ . By substituting this fact into the definition of compactness, we next observe that  $G$  is compact if and only if each doubly stochastic matrix  $X$  that commutes with  $A$  can be expressed as a convex sum over  $\{I_n\}$ .

---

<sup>3</sup>Stronger here refers to the fact that the result in [\[Arv+17\]](#) includes two further classifications of graphs: Tinhofer and Godsil.



In other words,  $G$  is compact if and only if the only doubly stochastic matrix that commutes with  $A$  is  $I_n$ . This line of reasoning is encapsulated in the following result.

**Theorem 4.2.1.** *Let  $G$  be a graph with  $n$  vertices. Then the only fractional automorphism of  $G$  is  $I_n$  if and only if  $G$  is asymmetric and compact.*

*Proof.* If the only fractional automorphism of  $G$  is  $I_n$ , then the only full automorphism of  $G$  is  $I_n$ , as all full automorphisms are also fractional automorphisms. Therefore, the full and fractional automorphisms of  $G$  coincide, so by [Proposition 4.0.4](#), we see that  $G$  is asymmetric. From the discussion above, an asymmetric graph is compact if and only if its full and fractional automorphisms coincide.

The proof of the reverse direction is precisely the discussion above.  $\square$

With this result established, in order to find some examples of compact graphs with trivial automorphism groups, we instead search for graphs that only have  $I_n$  as a fractional automorphism. Such graphs have already been considered in the literature, though (to the best of my knowledge) not from the perspective of Colour Refinement or amenability [[ABK14](#); [FS15](#)]. For the remainder of this section, we state some of the main results from [[FS15](#)], and provide some examples.

**Definition 4.2.2** (Simple spectrum). Let  $G$  be a graph with adjacency matrix  $A$ . We say that  $G$  is *simple spectrum* if each eigenvalue of  $A$  has multiplicity one.

**Theorem 4.2.3.** *Let  $G$  be a simple spectrum graph with  $n$  vertices, adjacency matrix  $A$ , and having  $k \geq 0$  eigenvectors  $u_i$  such that  $u_i^T \mathbf{1} = 0$ . If each  $u_i$  has at least  $2k + 1$  non-zero entries, then the only fractional automorphism of  $G$  is  $I_n$ .*

*Proof.* Omitted. This appears as Theorem 2.1 in [[FS15](#)].  $\square$

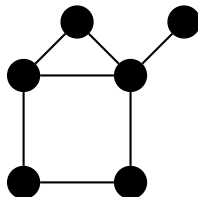
Notably, combining [Theorem 4.2.3](#) with [Theorem 4.2.1](#) says that simple spectrum graphs that satisfy the above orthogonality condition are compact. One such family of graphs satisfying these conditions is the family of *friendly* graphs.

**Definition 4.2.4** (Friendly). Let  $G$  be a simple spectrum graph with adjacency matrix  $A$ . If  $A$  has no eigenvectors  $u$  such that  $u^T \mathbf{1} = 0$ , we say that  $G$  is *friendly*.

**Corollary 4.2.5.** *Friendly graphs are amenable.*

We finish this section by presenting an example of a friendly graph, and discussing a comment at the end of [[FS15](#)].

**Example 4.2.6** (From [[FS15](#)]). The following graph  $G$  can be shown to be simple spectrum and have no eigenvectors orthogonal to  $\mathbf{1}$ :




This shows that  $G$  is a friendly graph.

It has been a common theme throughout this project to abstractly define a class of graphs that satisfy some collection of properties, and then try to find some concrete examples within that class. At the end of [FS15], the authors think similarly. Specifically, the authors ask if there exists any non-regular, asymmetric, simple spectrum graph that has non-trivial fractional automorphisms. Unfortunately, they were not able to find any examples.

Fortunately, we were.

### 4.3 Jellyfishification

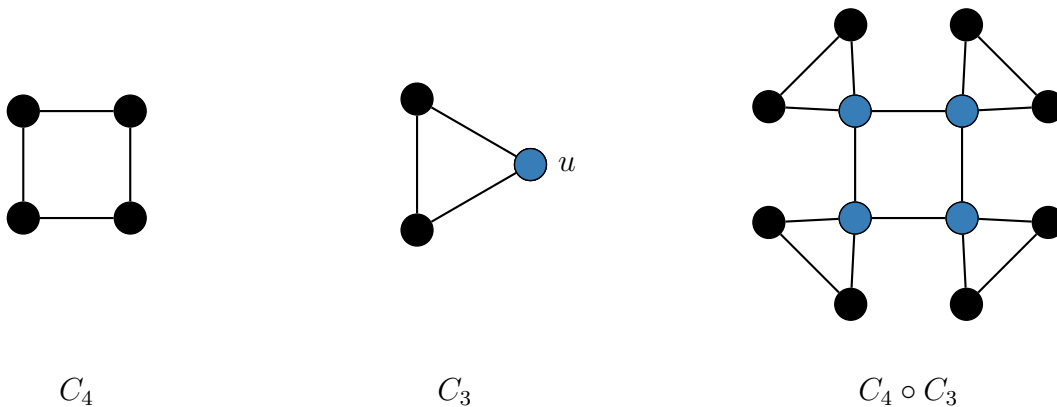
In Figure 3.2 we presented  $J_{10}$  and  $J_{12}$ ; asymmetric non-regular graphs that are not amenable. Since then, we have established that if a graph  $G$  is not amenable, then it isn't compact (Theorem 4.1.4), so the set of fractional automorphisms of  $G$  does not coincide with the set of full automorphisms (Theorem 4.2.1). Therefore, if either of  $J_{10}$  or  $J_{12}$  are simple spectrum, then they are precisely the graphs that Fiori and Sapiro were hoping to find in [FS15].

 It can be checked using a computational algebra package that the adjacency matrices of both  $J_{10}$  and  $J_{12}$  do in fact have eigenvalues all of multiplicity one and therefore provide an answer to Fiori and Sapiro's question! For the remainder of this section, we will describe a process of turning these graphs into an infinite family of non-regular, asymmetric, simple spectrum graphs that have non-trivial fractional automorphisms; namely, the process of *jellyfishification*.

We first introduce a method in which we can combine two graphs [LHH17; GM78].

**Definition 4.3.1** (Rooted product graph). Let  $G$  be a graph with  $n$  vertices, and  $H$  be a graph in which we identify  $u$  as a root. Then the *rooted product graph*  $G \circ H$  is the graph obtained by identifying each vertex of  $G$  with a copy of  $H$  rooted at  $u$ .

**Example 4.3.2.** The rooted product  $C_4 \circ C_3$ <sup>4</sup>:



This construction describes a way in which we can combine many varieties of graphs. In what follows, however, we restrict our attention to taking the rooted product with path graphs  $P_n$ .

<sup>4</sup>Many people claim that Star Wars creator George Lucas based the design of the TIE Interceptor on the graph  $C_4 \circ C_3$ . Can you see the resemblance?

**Definition 4.3.3** (Jellyfishification). Let  $G$  be a graph, and  $n$  be a non-negative integer. Then the  *$n$ th order jellyfishification*  $J_n(G)$  of  $G$  is the rooted product graph  $G \circ P_n$ , where  $P_n$  is the path graph with length  $n$ , rooted at one of its vertices of degree 1.

In order to use this procedure to generate our infinite family of graphs, we must show that jellyfishification preserves simple spectrum, non-regularity, asymmetry, and non-triviality of fractional automorphisms. We do so in the following lemmas.

**Lemma 4.3.4.** *Let  $G$  be a graph with  $n$  vertices and  $d \leq n$  distinct eigenvalues. Then  $J_k(G)$  has exactly  $(k + 1)d$  distinct eigenvalues. Moreover, if  $G$  is simple spectrum, then  $J_k(G)$  is simple spectrum.*

*Proof.* Omitted. This result is adapted from Corollary 3.1 in [FS15]; a corollary of Theorem 3.2. The proof follows in the same way.  $\square$

**Lemma 4.3.5.** *Let  $G$  be a graph. If  $G$  is non-regular then for all  $k$ ,  $J_k(G)$  is non-regular.*

*Proof.* Observe that jellyfishification increases the degree of each vertex in  $G$  by exactly one. Therefore, if there are two vertices in  $G$  of different degree, then for all  $k$  there will be two vertices in  $J_k(G)$  of different degree.  $\square$

**Lemma 4.3.6.** *Let  $G$  be a graph with no isolated vertices<sup>5</sup>. Then  $G$  is asymmetric if and only if for all  $k$ ,  $J_k(G)$  is asymmetric.*

*Proof.* The backwards direction is immediate by observing that  $G = J_0(G)$ .

For the forward direction, suppose that  $G = (V, E)$  is asymmetric, and that  $\varphi \in \text{Aut}(J_k(G))$  is an automorphism of  $J_k(G) = (V_J, E_J)$ . We show that  $\varphi(v) = (v)$  for all  $v \in V_J$ .

For ease of explanation (and to reinforce that these graphs do in fact resemble jellyfish) we refer to the copy of  $G$  inside of  $J_k(G)$  as the ‘body’, and the added copies of  $P_k$  as the ‘legs’.

First suppose that  $v$  is one of the body vertices. The assumption that  $G$  has no isolated vertices ensures that there is no body vertex that has degree 1. Then the distance between  $v$  and the nearest degree 1 vertex—the vertex at the end of the leg that extends from  $v$ —is  $k$ . This implies that the vertex  $\varphi(v)$  must be a distance of  $k$  from the nearest vertex of degree 1. The only vertices that are a distance of  $k$  from a vertex of degree 1 are the body vertices, hence  $\varphi(v)$  must be a body vertex. By asymmetry of  $G$ , we must have  $\varphi(v) = v$ .

Next suppose that  $v$  corresponds to one of the leg vertices that extends from body vertex  $u$ . Further suppose that the distance between  $u$  and  $v$  is  $d$ . There is precisely one vertex in  $V_J$  that can be reached from  $u$  by always traversing down the leg that extends from it, namely  $v$ . By the earlier argument,  $\varphi(u) = u$ , hence  $\varphi(v) = v$ .  $\square$

**Lemma 4.3.7.** *Let  $G$  be a graph. Then  $G$  has non-trivial fractional automorphisms if and only if for all  $k$ ,  $J_k(G)$  has non-trivial fractional automorphisms.*

---

<sup>5</sup>An isolated vertex is a vertex with degree 0.

*Proof.* As with the previous lemma, the backwards direction is immediate with  $k = 0$ .

For the forward direction, suppose that  $G$  is a graph with  $n$  vertices, adjacency matrix  $A \in \mathbb{R}_{n \times n}$ , and at least one non-trivial fractional automorphism  $X \in \mathbb{R}_{n \times n}$ . Then the vertices in  $J_k(G)$  can be arranged so that each collection of  $n$  rows in the adjacency matrix corresponds to a layer<sup>6</sup> in  $J_k(G)$ . In particular

$$A_k = \begin{pmatrix} A & I_n & & 0 \\ I_n & 0_n & I_n & \\ & I_n & 0_n & \ddots \\ & & \ddots & \ddots & I_n \\ 0 & & & I_n & 0_n \end{pmatrix} \in \mathbb{R}_{nk \times nk},$$

where  $0_n \in \mathbb{R}_{n \times n}$  is the  $n \times n$  matrix of all zeros, and the large 0s indicate that the non-labelled parts of the matrix are all zeros.

We construct a new doubly stochastic matrix  $X_k$  such that  $A_k X_k = X_k A_k$ . Specifically, define

$$X_k = \begin{pmatrix} X & & & 0 \\ & X & & \\ & & X & \\ & & & \ddots \\ 0 & & & & X \end{pmatrix} \in \mathbb{R}_{nk \times nk}.$$

Then

$$\begin{aligned} A_k X_k &= \begin{pmatrix} AX & X & & 0 \\ X & 0_n & X & \\ & X & 0_n & \ddots \\ & & \ddots & \ddots & X \\ 0 & & & X & 0_n \end{pmatrix} \\ &= \begin{pmatrix} XA & X & & 0 \\ X & 0_n & X & \\ & X & 0_n & \ddots \\ & & \ddots & \ddots & X \\ 0 & & & X & 0_n \end{pmatrix} \\ &= X_k A_k. \end{aligned}$$

Hence we have shown that  $X_k$  is a non-trivial fractional automorphism for  $J_k(G)$ . □

---

<sup>6</sup>Here, a *layer* refers to the vertices in  $J_k(G)$  that are a constant distance from the root graph. [Figure 4.1](#) is drawn in such a way to make the layers clear.

The following theorem summarises these results.

**Theorem 4.3.8.** *Let  $G$  be a graph with no isolated vertices. If  $G$  is non-regular, asymmetric, and simple spectrum with non-trivial fractional automorphisms, then so is  $J_k(G)$  for all non-negative integers  $k$ .*

This result, together with the existence of  $J_{10}$  and  $J_{12}$  shows that there are infinitely many non-regular, asymmetric, simple spectrum graphs with non-trivial fractional automorphisms.<sup>7</sup>

To put this result to use, and to give us the chance to draw some fun pictures, Figure 4.1 shows the first and second-order jellyfishification applied to  $J_{12}$ .<sup>8</sup>

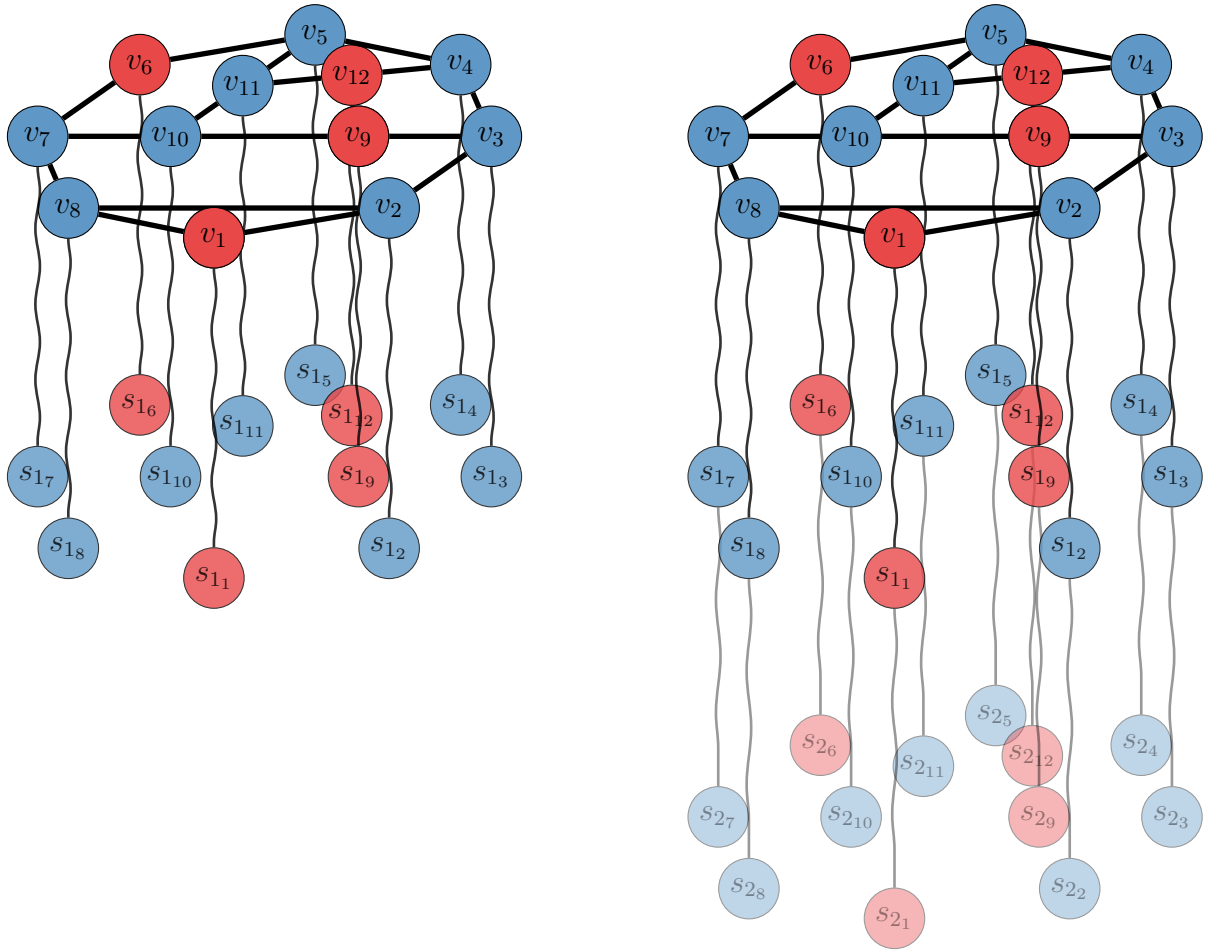


Figure 4.1: The first and second order jellyfishification process demonstrated on  $J_{12}$ .

We remember that the reason we considered these graphs in the first place was that we were trying to better understand amenability, and in doing so we discovered a new equivalent problem to work with. Now that we have established some exciting results, we can leverage this equivalence in the other direction to produce a result about amenability.



<sup>7</sup>In the supplementary files and on GitHub, we verify some of these properties directly.

<sup>8</sup>The graph  $J_{12}$  was chosen here over  $J_{10}$ , due to its planarity, which lends itself nicely to drawing. Can you show that the jellyfishification of a planar graph is always planar?

**Theorem 4.3.9.** *Let  $G$  be an asymmetric graph with no isolated vertices. Then  $G$  is amenable if and only if  $J_k(G)$  is amenable for all non-negative integers  $k$ .*

*Proof.* Let  $G$  be an asymmetric graph with  $n$  vertices, none of which are isolated. Then  $J_k(G)$  is also asymmetric by [Lemma 4.3.6](#). Furthermore,  $G$  is amenable if and only if  $G$  is compact ([Corollary 4.1.5](#)).  $G$  is compact if and only if the only fractional automorphism of  $G$  is  $I_n$  ([Theorem 4.2.1](#)).  $I_n$  is the only fractional automorphism of  $G$  if and only if  $I_n$  is the only fractional automorphism of  $J_k(G)$  ([Lemma 4.3.7](#)). Applying [Theorem 4.2.1](#) and [Corollary 4.1.5](#) to  $J_k(G)$  gives the result.  $\square$

*Remark 4.3.10.* We could have proved this result directly by arguing about the colour of each leg vertex, but I hope you will agree that the proof provided is a truly beautiful exposition of how some of the results we have proved interact with each other.

# Chapter 5

## Applications and Conclusion

Our goal at the beginning of this project was to explore the Graph Isomorphism problem. Whilst we acknowledge that it is a difficult problem with no known polynomial time algorithm, the Colour Refinement algorithm has served us well as an entry point into this field. Let's recap what we have learnt about it.

Firstly, we defined the Colour Refinement algorithm as a way of computing the stable partition of a graph; a partition that groups together vertices that look similar on a local scale. We were initially hopeful that CR might distinguish all graphs up to isomorphism, as it can be implemented quickly, but [Example 2.4.1](#) showed us this isn't always the case. Having seen this example, we embarked on our journey to explore those graphs that Colour Refinement *can* distinguish up to isomorphism; amenable graphs.

In [Chapter 3](#), after giving a formal definition of amenability, we began exploring how we can check for amenable graphs. This started by working with concrete graphs, where we proved the amenability of unigraphs and discrete graphs in [Proposition 3.1.4](#). We then moved more into the abstract and presented [Algorithm 3.2.6](#); a fast algorithm for checking the amenability of graphs based on Colour Refinement. Wanting to characterise more amenable graphs, at the end of this chapter we discussed other families we believed to be good amenable candidates, namely asymmetric graphs and non-regular asymmetric graphs. These didn't turn out to be amenable, as shown by the Frucht Graph ([Figure 3.1](#)), and  $J_{10}$  and  $J_{12}$  ([Figure 3.2](#)), but these examples proved to be useful in our later work.

Not wanting to give up on our search for amenable graphs, we employed two widely applicable mathematical techniques: we translated our problem so it can be studied from a new perspective, and relaxed our conditions to learn about the 'almost solutions'. This resulted in our consideration of *fractional isomorphisms*, and *compact graphs*. With these compact graphs, we were able to establish a hierarchy of graphs ([Theorem 4.1.4](#)) that collapsed in the asymmetric setting. Whilst studying this collapse, we came across an open question that asked if there exists any asymmetric, non-regular, simple spectrum graph with non-trivial doubly stochastic matrix that commutes with the adjacency matrix. The examples we gave in [Figure 3.2](#) answered this question, and we proved in [Theorem 4.3.9](#) that the jellyfishification process gives an infinite family of such solutions.

If, having read this dissertation, you have found yourself growing very fond of the Colour Refinement algorithm, a natural question you might ask at this point is *where do we go from here?* Fortunately, there are many different avenues we can explore.

A key theme of this project was finding connections between mathematical objects by exploring equivalent representations of our definitions. One connection that we didn't explore is the connection between Colour Refinement and first order logic. It can be shown that the Colour Refinement algorithm acts as an equivalence test for first order logic sentences with at most two variables and counting quantifiers. Moreover, this result can be generalised to the family of algorithms we described at the beginning of [Chapter 2](#); the Weisfeiler-Leman algorithms. This idea was first presented in [\[IL90\]](#), and is discussed further in [\[CFI92\]](#) and Chapter 3 of [\[Gro17\]](#).

Whilst the results we presented in this project very much relate to graph theory and isomorphisms, the *techniques* we established are more widely applicable. For example, consider the process of creating the cell graph defined in [Section 3.2](#). We can think of this as compressing a graph by associating suitably similar vertices—those vertices that get the same stable colour. In this way, we have used Colour Refinement to reduce a graph to a smaller object that we can more easily work with. In an analogous way, Colour Refinement can be used as a way to reduce the problem instances when solving problems in linear programming or probabilistic inference. This idea is discussed in Section 15.6 of [\[Gro+21\]](#).

In a world where everything seems to be “powered by AI”, it is perhaps not all that surprising that Colour Refinement has found uses in the land of machine learning; specifically in *graph kernels*. The goal of a graph kernel on input graphs  $G$  and  $H$  is to quickly associate a score representing how ‘similar’  $G$  and  $H$  are. When CR is performed on  $G + H$ ,  $G$  and  $H$  are deemed to be ‘similar’ if there are many vertices of  $G$  that get given the same colour as many vertices of  $H$ . This idea holds for the intermediate colourings as well as the stable colouring, and a graph kernel can be built by formalising this. For a thorough introduction to this idea, with lots of references that can take you even deeper into the machine learning literature, the interested reader is directed to Section 15.7 of [\[Gro+21\]](#).



# Bibliography

- [ABK14] Yonathan Aflalo, Alex Bronstein, and Ron Kimmel. *Graph matching: relax or not?* Oct. 12, 2014. DOI: [10.48550/arXiv.1401.7623](https://doi.org/10.48550/arXiv.1401.7623) (cit. on pp. [1](#), [37](#)).
- [Arv+17] Vikraman Arvind et al. “Graph Isomorphism, Color Refinement, and Compactness”. In: *computational complexity* 26.3 (Sept. 1, 2017), pp. 627–685. DOI: [10.1007/s00037-016-0147-6](https://doi.org/10.1007/s00037-016-0147-6) (cit. on pp. [23](#), [25](#), [27–30](#), [34–36](#)).
- [Bab16] László Babai. “Graph isomorphism in quasipolynomial time [extended abstract]”. In: *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*. STOC ’16: Symposium on Theory of Computing. STOC ’16. Cambridge MA USA: Association for Computing Machinery, June 19, 2016, pp. 684–697. DOI: [10.1145/2897518.2897542](https://doi.org/10.1145/2897518.2897542) (cit. on pp. [1](#), [10](#)).
- [Bab19] László Babai. “GROUP, GRAPHS, ALGORITHMS: THE GRAPH ISOMORPHISM PROBLEM”. In: *Proceedings of the International Congress of Mathematicians (ICM 2018)*. International Congress of Mathematicians 2018. Rio de Janeiro, Brazil: WORLD SCIENTIFIC, May 2019, pp. 3319–3336. DOI: [10.1142/9789813272880\\_0183](https://doi.org/10.1142/9789813272880_0183) (cit. on p. [10](#)).
- [BBG17] Christoph Berkholz, Paul Bonsma, and Martin Grohe. “Tight Lower and Upper Bounds for the Complexity of Canonical Colour Refinement”. In: *Theory of Computing Systems* 60.4 (May 1, 2017), pp. 581–614. DOI: [10.1007/s00224-016-9686-0](https://doi.org/10.1007/s00224-016-9686-0) (cit. on pp. [17–20](#)).
- [CFI92] Jin-Yi Cai, Martin Fürer, and Neil Immerman. “An optimal lower bound on the number of variables for graph identification”. In: *Combinatorica* 12.4 (Dec. 1992), pp. 389–410. DOI: [10.1007/BF01305232](https://doi.org/10.1007/BF01305232) (cit. on pp. [11](#), [44](#)).
- [CC82] A. Cardon and M. Crochemore. “Partitioning a graph in  $O(|A|\log^2|V|)$ ”. In: *Theoretical Computer Science* 19.1 (July 1982), pp. 85–98. DOI: [10.1016/0304-3975\(82\)90016-0](https://doi.org/10.1016/0304-3975(82)90016-0) (cit. on p. [20](#)).
- [Die17] Reinhard Diestel. *Graph theory*. 5th ed. Vol. 173. Graduate Texts in Mathematics. New York, NY: Springer Berlin Heidelberg, 2017 (cit. on p. [3](#)).
- [FS15] M. Fiori and G. Sapiro. “On spectral properties for graph matching and graph isomorphism problems”. In: *Information and Inference* 4.1 (Mar. 1, 2015), pp. 63–76. DOI: [10.1093/imaiai/iav002](https://doi.org/10.1093/imaiai/iav002) (cit. on pp. [35](#), [37–39](#)).
- [Fru49] Robert Frucht. “Graphs of Degree Three with a Given Abstract Group”. In: *Canadian Journal of Mathematics* 1.4 (Aug. 1949), pp. 365–378. DOI: [10.4153/CJM-1949-033-6](https://doi.org/10.4153/CJM-1949-033-6) (cit. on p. [31](#)).

- [GM78] C.D. Godsil and B.D. McKay. “A new graph product and its spectrum”. In: *Bulletin of the Australian Mathematical Society* 18.1 (Feb. 1978), pp. 21–28. DOI: [10.1017/S0004972700007760](https://doi.org/10.1017/S0004972700007760) (cit. on p. 38).
- [Gro17] Martin Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Lecture Notes in Logic. Cambridge: Cambridge University Press, 2017. DOI: [10.1017/9781139028868](https://doi.org/10.1017/9781139028868) (cit. on p. 44).
- [Gro+21] Martin Grohe et al. “Color Refinement and Its Applications”. In: *An Introduction to Lifted Probabilistic Inference*. Ed. by Guy Van den Broeck, Kristian Kersting, and Sriraam Natarajan. The MIT Press, Aug. 17, 2021, pp. 349–372. DOI: [10.7551/mitpress/10548.003.0023](https://doi.org/10.7551/mitpress/10548.003.0023) (cit. on pp. 15, 16, 19, 20, 44).
- [Hop71] John Hopcroft. “AN  $n \log n$  ALGORITHM FOR MINIMIZING STATES IN A FINITE AUTOMATON”. In: *International Symposium on the Theory of Machines and Computations*. Theory of Machines and Computations. Technion Haifa, Israel: Academic Press, 1971, pp. 189–196. DOI: [10.1016/B978-0-12-417750-5.50022-1](https://doi.org/10.1016/B978-0-12-417750-5.50022-1) (cit. on p. 19).
- [HR22] Sophie Huczynska and Colva Roney-Dougal. “MT4514 Graph Theory Lecture Notes”. University of St Andrews, 2022 (cit. on p. 3).
- [Hur08] Glenn Hurlbert. *A SHORT PROOF OF THE BIRKHOFF-VON NEUMAN THEOREM*. Arizona State University, 2008 (cit. on p. 34).
- [IL90] Neil Immerman and Eric Lander. “Describing Graphs: A First-Order Approach to Graph Canonization”. In: *Complexity Theory Retrospective: In Honor of Juris Hartmanis on the Occasion of His Sixtieth Birthday, July 5, 1988*. Ed. by Alan L. Selman. New York, NY: Springer, 1990, pp. 59–81. DOI: [10.1007/978-1-4612-4478-3\\_5](https://doi.org/10.1007/978-1-4612-4478-3_5) (cit. on p. 44).
- [Jun13] Dieter Jungnickel. *Graphs, Networks and Algorithms*. Vol. 5. Algorithms and Computation in Mathematics. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. DOI: [10.1007/978-3-642-32278-5](https://doi.org/10.1007/978-3-642-32278-5) (cit. on p. 30).
- [LHH17] Zhenzhen Lou, Qiongxiang Huang, and Xueyi Huang. “Construction of graphs with distinct eigenvalues”. In: *Discrete Mathematics* 340.4 (Apr. 2017), pp. 607–616. DOI: [10.1016/j.disc.2016.11.033](https://doi.org/10.1016/j.disc.2016.11.033) (cit. on p. 38).
- [MP14] Brendan D. McKay and Adolfo Piperno. “Practical graph isomorphism, II”. In: *Journal of Symbolic Computation* 60 (Jan. 1, 2014), pp. 94–112. DOI: [10.1016/j.jsc.2013.09.003](https://doi.org/10.1016/j.jsc.2013.09.003) (cit. on pp. 11, 31).
- [Moo59] Edward F Moore. “The shortest path through a maze”. In: *Int. Symp. on Theory of Switching Part II*. Cambridge: Harvard University Press, 1959, pp. 285–292 (cit. on p. 30).
- [PT87] Robert Paige and Robert E. Tarjan. “Three Partition Refinement Algorithms”. In: *SIAM Journal on Computing* 16.6 (Dec. 1987), p. 982. DOI: [10.1137/0216062](https://doi.org/10.1137/0216062) (cit. on p. 20).
- [Ron23] Colva Roney-Dougal. “MT4512 Automata, Languages and Complexity Lecture Notes”. University of St Andrews, 2023 (cit. on p. 8).
- [The22] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.5)*. <https://www.sagemath.org>. 2022 (cit. on p. 31).

- [Sip13] Michael Sipser. *Introduction to the theory of computation*. Third edition, international edition. Australia Brazil Japan Korea Mexiko Singapore Spain United Kingdom United States: Cengage Learning, 2013. 458 pp. (cit. on pp. 8, 9).
- [Tin91] G. Tinhofer. “A note on compact graphs”. In: *Discrete Applied Mathematics* 30.2 (Feb. 28, 1991), pp. 253–264. DOI: [10.1016/0166-218X\(91\)90049-3](https://doi.org/10.1016/0166-218X(91)90049-3) (cit. on pp. 33, 34).
- [Val21] Gabriel Valiente. *Algorithms on Trees and Graphs: With Python Code*. Texts in Computer Science. Cham: Springer International Publishing, 2021. DOI: [10.1007/978-3-030-81885-2](https://doi.org/10.1007/978-3-030-81885-2) (cit. on p. 28).