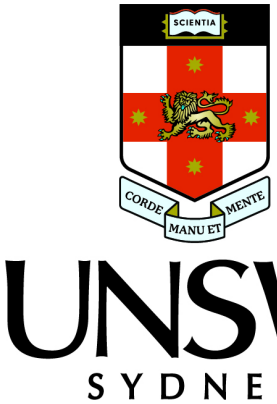


THE COMPUTATIONAL COMPLEXITY OF COUNTING PROBLEMS

Laura Margarete Stemmler Hermida

Supervisor: Professor Catherine Greenhill

School of Mathematics and Statistics
UNSW Sydney

October 2025

Submitted in partial fulfilment of the requirements of the degree of
Master of Mathematics

Plagiarism statement

I declare that this thesis is my own work, except where acknowledged, and has not been submitted for academic credit elsewhere.

I acknowledge that the assessor of this thesis may, for the purpose of assessing it:

- Reproduce it and provide a copy to another member of the University; and/or,
- Communicate a copy of it to a plagiarism checking service (which may then retain a copy of it on its database for the purpose of future plagiarism checking).

I certify that I have read and understood the University Rules in respect of Student Academic Misconduct, and am aware of any potential plagiarism penalties which may apply.

By signing this declaration I am agreeing to the statements and conditions above.

Signed: _____ Date: _____

Acknowledgements

I would like to sincerely thank my supervisor Catherine Greenhill, for her kindness, constant support, and the many insightful discussions we've shared throughout the course of this thesis. I've learned a great deal since starting this thesis, and the experience has been both fun and rewarding. Thank you.

Thank you to all the friends I have made over the past two years at UNSW. I would especially like to thank Nathan, Dominic, Wentao, and Yilyu for their support and many good memories we've shared. I have had the best time learning mathematics with you all.

I am also grateful to my uncle Robin, for always showing interest in my work and our fun phone calls where I try (and often fail) to explain things. And to my brother Oliver, thank you for your constant encouragement and inspiration.

Lastly, I want to thank my parents. If I had to quantify my parents' love and support, it would surely be #P-complete – easy to verify but immensely difficult to fully capture.

Laura Margarete Stemmler Hermida

Abstract

Counting certain combinatorial structures in graphs, such as independent sets or k-colourings, appear to only admit algorithms that take an exponential amount of time. This begs the question: "Why is counting so hard?", which is at the heart of complexity theory. This thesis will illustrate the main techniques of complexity proofs for counting and will prove the intractability of several problems in graph theory. The main proof is in Chapter 6, which shows that counting the number of homomorphisms to a fixed target graph is #P-complete for almost all graphs.

Contents

Chapter 1

Introduction

Let us start with a fun, seemingly unrelated problem (adapted from [33]) that will introduce the subject of this thesis.

Suppose you are employed at the mathematics department of a university. This mathematics department has a single floor made up of convoluted connected hallways dreamt up by a topologist. One of the professors at the university has recently achieved a major breakthrough on the problem of P vs NP. Word gets out that they might have even found a constructive proof for P = NP, which would have enormous implications! There are a few kinks to be worked out, though, before the paper is posted. The dean, who dreams of creating the world's greatest mathematics department, calls for an Internet firewall and a complete physical lockdown. He has tasked you with creating a high-security system to ensure the research is not stolen before publication. As a reader of mathematics, you are aware of a great lower bound on the minimum number of guards needed, so that the whole mathematics floor is under guard. However, your plan must also mitigate risks. You ask yourself: "Given a small chance of one of the guards falling asleep, how can I place the guards, so that the mathematics floor is fully observed, with high probability, at all times?". You consider placing guards at junctions and ends of hallways, so a hallway is always under watch by at least two guards, provided they are both awake (see Figure 1.1).

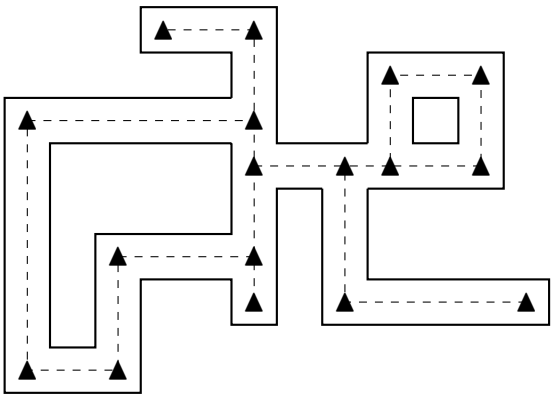


Figure 1.1: The mathematics floor plan with guards ▲.

You model the mathematics floor as a graph G where vertices mark the positions of security guards. Given the current arrangement, it is clear that, even if some guards fall asleep, the mathematics floor will still be fully guarded. In graph theory,

this would correspond to a *vertex cover* of G. Now, given a probability p of each guard falling asleep, you want to calculate the probability that all the hallways remain under guard, assuming guards cannot see past other guards. You want to be extra cautious and suppose that $p = 1/2$. A simple calculation shows that, for any subset V' of guards V, the probability that exactly that subset of guards is awake is independent of the chosen subset V':

$$\Pr(\text{Only the guards in } V' \text{ are awake}) = \left(\frac{1}{2}\right)^{|V'|}\left(\frac{1}{2}\right)^{(|V|-|V'|)} = 2^{-|V|}.$$

Hence, the probability that the guards in V' are awake and alert is the same for all subsets V' of V. Thus, the probability the whole mathematics floor is under surveillance at any given time is

$$\Pr(\text{The mathematics floor is fully observed}) = \frac{\#\text{ Vertex Covers of } G}{2^{|V|}}.$$

And so, determining the reliability of such a surveillance system boils down to counting vertex covers in a graph. These two problems are, in fact, computationally equivalent.

This concept of turning a problem into another one is central to this thesis. If we have an efficient algorithm for counting vertex covers, then we would have an efficient algorithm for the surveillance reliability problem, as well as vice versa (unless FP = #P, the counting analogues of P and NP, respectively). However, as we will see in Chapter 4, it is highly unlikely an efficient algorithm for counting vertex covers exists, thus, the same is true for determining the reliability of the surveillance system.

Chapters 2 and 3 introduce definitions from graph theory needed for this thesis and cover the complexity theory of counting problems.

The vertex cover problem is equivalent to another problem of graph theory: counting independent sets in graphs. In Chapter 4, we discuss the #P-completeness of counting independent sets. This means that the independent sets of a graph is one of the hardest problems to solve efficiently in the class of problems #P. We prove that this problem remains #P-complete even when we restrict instances to graphs of maximum degree three. This implies that determining the reliability of the surveillance system is computationally difficult even when the floor of the mathematics department has a simple layout.

In Chapter 5, we introduce graph homomorphisms. Many graph properties can be described in terms of homomorphism counts to a fixed target graph; this explains why determining the complexity of homomorphisms is valuable.

In Chapter 6, we prove a dichotomy theorem for counting homomorphisms to fixed graphs. From this, we conclude that counting k-colourings, partition functions to certain models of statistical physics, and generalised colourings is hard.

Chapter 2

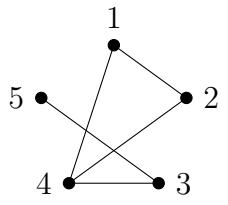
Preliminaries

Graphs are the basic object of this thesis. This chapter describes the basic notions of graph theory, for which we loosely follow the definitions and notation from Diestel's standard text [9].

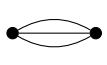
A **graph** $G = (V, E)$ is a set of vertices V and a set of edges E, where every edge is a pair of unordered vertices $\{v_i, v_j\}$ in V. We will always assume that all graphs are finite with $|V| < \infty$. This standard definition of a graph is called **simple**. The definition of a graph can be modified in many ways:

- A **multigraph** is a graph with possibly multiple edges between the same pair of vertices.
- A **directed graph** $G = (V, E)$ is a graph in which the edges of E are *ordered*.
- A **looped-simple** graph permits edges that connects a vertex to itself in a loop.
- A **weighted graph** assigns values from a field F to its vertices and/or edges.

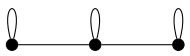
Throughout the thesis, graphs by default are presumed to be **simple**, meaning they have no multiple edges, self-loops, undirected and unweighted. Some examples of graphs are shown in Figure 2.1.



(a) A simple graph with vertex set $V = \{1, 2, 3, 4, 5\}$ and edge set $E = \{\{1, 2\}, \{1, 4\}, \{2, 4\}, \{3, 4\}, \{3, 5\}\}$.



(b) A multigraph.



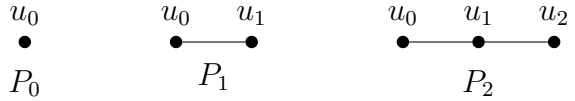
(c) A looped-simple graph.

Figure 2.1: Different types of graphs.

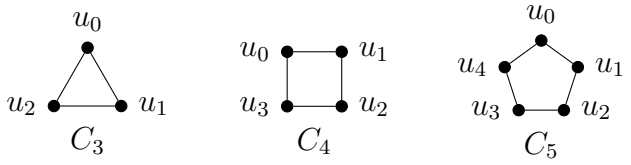
Some graphs have special names and notation:

- A **complete graph** has edges between every pair of vertices. The complete graph on n vertices is denoted K_n.
- A **path** of length n, where $n \geq 0$, is a graph with n edges in which the vertices can be arranged in a linear order with exactly one edge between each pair of

consecutive vertices. For $n \geq 0$, we let P_n denote the path of length n with vertices $\{u_0, \ldots, u_n\}$, and edges $\{u_i, u_{i+1}\}$ for $0 \leq i < n$.



- A **cycle** on n vertices, where $n \geq 3$, is the closure of a path graph P_{n-1} by introducing a vertex between the first and the last vertex. We let C_n denote the cycle on n vertices with vertex set $\{u_0, \ldots, u_n\}$ and edge set $\{u_0, u_{n-1}\} \cup \{\{u_i, u_{i+1}\} \mid 0 \leq i < n-1\}$.



We adopt the terminology of Chen, Curticapean, and Dell [6] and call a graph **reflexive** it all its vertices have self-loops and **irreflexive** if none of its vertices have loops.

A **subgraph** of a graph $G = (V, E)$ is another $H = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$.

Given a subset of the vertices $S \subseteq V$ of some graph $G = (V, E)$, the **induced subgraph** on S is the graph with vertex set S and edge set $\{\{v, w\} \in E \mid v \in S \text{ and } w \in S\}$.

The **complement** of the graph $G = (V, E)$ is the graph $\overline{G}$ with vertex set V and edge set $\overline{E} = \{\{u, v\} \mid \{u, v\} \notin E\}$.

A graph is **planar** if it can be drawn on the plane in such a way that the edges only intersect at their endpoints.

A **bipartite graph** is one in which the vertices may be partitioned into two sets such that the edges of the graph connect only vertices in distinct partitions.

A **matching** in a graph is a subset of the edges such that no two edges share an endpoint. A **perfect matching** includes every vertex in the matching.

A ***k*-colouring** of a graph G is a mapping $c : V \to S$ where S is a set of size k. The elements of S are referred to as *colours*. A colouring c is said to be proper if $c(v) \neq c(w)$ for all $\{v, w\} \in E$. The set of all proper k-colourings of G is denoted by $\Omega_k(G)$.

An **independent set** of a graph $G = (V, E)$ is a subset $X \subseteq V$ such that $\{u, v\} \not\subseteq X$ for all $\{u, v\} \in E$. We denote the set of all the independent sets of a graph G by $\mathcal{I}(G)$.

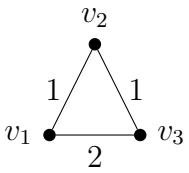
A **clique** is of a graph $G = (V, E)$ is a subset of vertices $X \subseteq V$ such that the graph induced on X is a complete.

A **vertex cover** on a graph $G = (V, E)$ is a subset $X \subseteq V$ such that every edge in E has at least one endpoint in X.

A **graph parameter** is a function defined on isomorphism types of multigraphs with loops to the real numbers. For instance, the number of perfect matchings in the graph is a graph parameter.

Two vertices u and v are said to be **adjacent** or **neighbours** if there exists an edge connecting u and v. The **neighbourhood** of a vertex v is the set of all neighbours of v and is denoted by $N(v)$. The **degree** of a vertex v is denoted by $d(v)$ and equals the size of its neighbourhood. We denote the minimum degree of a graph G by δ and the maximum degree by Δ.

The **adjacency matrix** of a graph G is the $|V| \times |V|$ matrix A with $A_{ij} = 1$ if and only if v_i is adjacent to v_j. If a graph G is undirected, then its adjacency matrix is symmetric. If a graph has weights associated to the edges, then ijth entry of the adjacency matrix equals the weight of edge connected the ith and jth vertices.



$$A = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix}$$

Figure 2.2: A weighted graph and its adjacency matrix.

Lastly, if A is an $m \times n$ matrix and B is a $p \times q$ matrix, the **Kronecker product** of A and B, denoted $A \otimes B$, is the $pm \times qn$ block matrix:

$$A \otimes B = \begin{pmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{pmatrix}.$$

Graphs constructed from the Kronecker product of their adjacency matrices will be considered in Chapter 6.

Chapter 3

Complexity Theory and Counting Problems

This chapter aims to present some of the fundamental concepts of complexity theory, drawing on Arora and Barak's book *Complexity Theory: A Modern Approach* [1]. Given a task, the central goal of complexity theory is proving the minimum number of resources needed to solve said task.

We begin by studying the simplest type of computational problem: the *decision problem*. This groundwork will motivate the study of counting problems and allow us to derive analogous definitions for the complexity of counting.

3.1 Computation

Computation is the process of transforming an input into an output, by following a finite set of instructions. This concept applies as much to a Babylonian adding and subtracting quantities on an abacus in 2300 B.C.E, as it does to a modern-day computer encrypting plain text to make it indecipherable. A **computational problem** is, then, a set of instances or inputs together with a set of outputs or solutions for every instance.

Computational problems are often phrased as questions or tasks to solve. For example, "Given an integer n, find the prime factorisation of n." or "Given a map of towns in Australia, is there a route that visits every town exactly once and returns to its starting point?" are classic examples of computational problems. We will present computational problems by stating explicitly what the problem's name is, what the instances are, and what the output is. The problems will be presented in the following format:

Name.	Integer Factorisation
Instance.	An integer n.
Output.	The prime factorisation of n.

Throughout this thesis, where context guarantees clarity, we will simply use the word "problem" when referring to computational problems.

The method in which we solve a particular computational problem is called an *algorithm*. More precisely, an algorithm is a finite set of mechanical rules, such that by applying them to an input of a computational problem, we arrive at the correct and desired output. As an example, an algorithm that solves the Integer Factorisation problem tests whether the numbers from 2 up to $\lceil \sqrt{n} \rceil$ divide n to find its

factorisation. This particular type of algorithm would be described as a *brute-force* algorithm, where we simply test all possibilities in the space of solutions, in order to find the correct answer. In the worst case, these types of algorithms take as much time as the size of the solution space. This is less than ideal, especially when we consider larger inputs (such as bigger integers). Indeed, the brute-force algorithm for integer factoring would take roughly 10^{16} years to compute the factorisation of a 60 digit number – this is about 10^6 times the age of the universe! As a result, most brute-force algorithms are said to be *inefficient*. The question is then: "Is it possible to find a better algorithm?". Finding more efficient algorithms than those currently known is the focus of algorithmic design in computer science. However, some problems, including the Integer Factorisation problem, defy attempts to find an efficient algorithm (leaving aside Shor's algorithm for factorisation using quantum computing). For these types of problems, it would be more productive to ask: "Can I prove no efficient algorithm exists?". This is the main goal of complexity theory. Proving that no efficient algorithm exists is a challenging task.

Complexity theory counts the number of steps, or elementary computations, executed to arrive at the output. Each elementary operation is assumed to take constant time, regardless of the size of the input.

A **complexity class** describes a set of problems that can be computed within given resource bounds of time and (memory) space. In this thesis, we will be concerned with worst-time complexity. Given a worst-case instance, how many steps will the algorithm take, and how does the step number scale with the size of the input?

3.1.1 Representation and Turing Machines

In order to study computational problems, the terms "computation" and "computational problem" require precise and rigorous definitions. First, a method is needed to represent all potential instances and outputs mathematically. This is done through *encodings*. An encoding is an injective function that maps a problems' instances and outputs to finite *strings*. A string is a sequence of symbols that belong to a non-empty finite set Σ called an *alphabet*. The set of all finite strings over the alphabet Σ is denoted by Σ^*. We present a few examples of how you could encode various objects:

- A familiar encoding is the alphabet $\{a, b, c, \ldots, z\}$ for the English language.
- The natural numbers can be encoded using the base-2 representation of $\mathbb{N}$ with the binary alphabet $\Sigma = \{0, 1\}$. The number ten, for example, is encoded as 1010 in binary (since $10 = 2^3 + 2^1$).
- A graph $G = (V, E)$ is represented by its $|V| \times |V|$ adjacency matrix. Thus, if we can encode 0,1-matrices, then we can encode graphs. A ternary encoding of 0,1-matrices using the alphabet $\Sigma = \{0, 1, \#\}$ views each row of a matrix as a binary 0,1-string and concatenates all the rows, by putting the # symbol between them. Then, the ternary strings can be encoded as binary strings, by mapping $0 \mapsto 00$, $1 \mapsto 01$, and $\#\mapsto 11$, for example. An example of a graph and its encoding are shown in Figure 3.1.

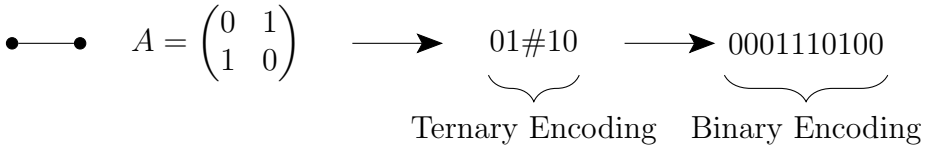


Figure 3.1: A graph, the graph's adjacency matrix A, and an encoding of the graph in the binary alphabet $\{0, 1\}$.

Encodings can capture all sorts of instances and outputs. However, there are certain objects which are not representable as strings, such as the real numbers. All countable objects are encodable.

Through encodings, a computational problem is defined to be a function

$$f : \Sigma^* \to \Sigma^*,$$

where Σ is an alphabet. It is common practice to define computational problems as functions $f : \{0, 1\}^* \to \{0, 1\}^*$, since the choice of encoding scheme makes no difference to the computability or complexity of a problem, and all encodable objects can be represented using the binary alphabet $\{0, 1\}$. Hence, in complexity theory, the specific choice of encoding is usually not mentioned.

The model of computation we adopt in this thesis is the *Turing machine*, developed by Alan Turing in 1936 in his seminal paper *On Computable Numbers, with an Application to the Entscheidungsproblem*. Other models of computation exist, but, under the Church-Turing Thesis, the Turing machine is able to simulate all physically realisable models of computation with little (polynomial time) loss of efficiency. Hence, the choice of model does not affect the computability or efficiency of a function. Two questions arise: "What functions are computable?" and "How efficiently can a (computable) deterministic function be computed?".

A wrinkle on the standard Turing machine makes the model of computation even more flexible and will give rise to a complexity class named after it. A nondeterministic Turing machine is able to branch from its current state in the computation and can compute the same input in more than way. This is opposed to the standard deterministic Turing machine, whose computation always proceeds in one way. The 'N' in the NP-complexity class stands for non-deterministic, and simply means that the computation can be performed on that kind of Turing machine.

An algorithm's *running time* reflects the the number of *elementary operations* performed by a Turing machine on an instance. Typically, the running time scales with the size n of the input. If the time scales as a polynomial in n, then the algorithm is highly efficient. Indeed, we will take a bird's eye view and treat any subcomputation that can be simulated in polynomial time as an elementary operation which will free us from the need to consider low-level details of the Turing machine. In this thesis, we will consider how the running time of an algorithm scales with the number of vertices in the graph or the with the size of a matrix.

We should stress that the computability and intractability of a computational problem is independent of the encoding scheme and computational model.

3.2 Decision Problems

The best understood and simplest type of computational problem is the *decision problem*, in which an algorithm "decides" between two possible outputs. These are commonly phrased as yes or no questions. Formally, a **decision problem** is a function

$$f : \{0,1\}^* \to \{0,1\}.$$

Examples of decision problems are "Is the number n divisible by p?" and "Is it possible to colour the US states using only four colours such that no two states sharing a border have the same colour?".

3.2.1 P and NP

Decision problems that are "efficiently" computable fall under the complexity class **P** (which stands for polynomial time). These are decision problems $f : \{0,1\}^* \to \{0,1\}$ that can be computed by a deterministic Turing machine within a time bounded by some polynomial in the size of the input. Determining if a graph has a perfect matching is notable example of a problem in P. The famous Blossom Algorithm [13] developed by Edmonds finds a maximum-cardinality matching in polynomial time.

The complexity class **NP** (which stands for non-deterministic polynomial time) is the set of decision problems for which we can verify the solutions efficiently, that is, in polynomial time. Formally, they are the set of decision problems $f : \{0,1\}^* \to \{0,1\}$ that can be computed by a non-deterministic Turing machine in polynomial time. As an example, consider the decision problem "Given a graph G, does there exists a vertex cover of size at least k?". Clearly, given a subset of vertices of G, we can easily verify whether it is a vertex cover of size k or not. Many other decision problems are in NP: Garey and Johnson present a catalogue of problems in NP [14].

Of all the many complexity classes that exist, the classes P and NP are among the most important ones. While $\mathrm{P} \subseteq \mathrm{NP}$, whether P equals NP is one of the biggest open problems in computer science, though it is widely believed to be untrue.

3.2.2 Reductions and NP-completeness

This thesis will focus on one of the most powerful tools in complexity theory, which is the ability to interrelate problems through *reductions*. We say that a problem A *reduces* to a problem B if there is a method of A using B. If this is the case, then computing the problem B is *at least as hard* as computing B. For decision problems, the two most commonly used types of reductions are *polynomial-time reductions* (also known as Karp reductions) and *polynomial-time Turing reductions* (also known as Cook reductions).

A polynomial-time reduction from a problem A to a problem B is a polynomial-time algorithm that transforms the instances of A into instances of B, such that the answer to the original instance is "yes" if and only if the answer to the transformed

instance is "yes". If a problem A polynomial-time reduces to a problem B it is denoted as

$$A \leq_p B.$$

Polynomial-time Turing reductions are a more general notion of polynomial-time reductions. A polynomial-time Turing reductions from a problem A to a problem B is an algorithm that computes A in polynomial time using an algorithm for B as a subroutine. If A polynomial-time Turing reduces to B, then we write

$$A \leq_T B.$$

In 1971, Cook proved that every problem in the class NP polynomial-time reduces to a problem called *Boolean satisfiability*, or SAT [7]. In complexity theory terms, every problem in NP *reduces* to SAT. If all problems in NP reduce to another problem, that problem is called **NP-hard**. Note that an NP-hard problem need not be in NP. If in addition to being NP-hard, a problem is in NP, then the problem is said to be **NP-complete**. Problems which are NP-complete are considered to be the "hardest" problems in NP, since a polynomial-time algorithm that solves an NP-complete problem, would imply every other problem in NP can be computed in polynomial time, solving the P vs NP problem. Shortly after Cook proved the NP-completeness of SAT, Karp proved the NP-completeness of 21 problems using polynomial-time reductions [25]. Importantly, if A is a problem known to be NP-complete, and you show for some problem B, that B is in NP and $A \leq_p B$, then B is NP-complete too.

An important property of reductions is that they are transitive, that is

$$\text{if } A \leq_p B \text{ and } B \leq_p C, \text{ then } A \leq_p C.$$

Under the assumption that P $\neq$ NP, Ladner proved in 1975 that there exist computational problems that lie strictly between P and the NP-complete subset of problems. Such problems are called **NP**-intermediate. An example of a decision problem believed to be NP-intermediate is "Is the graph G isomorphic to the graph H?". No one has been able to prove that this problem is in P or is NP-complete.

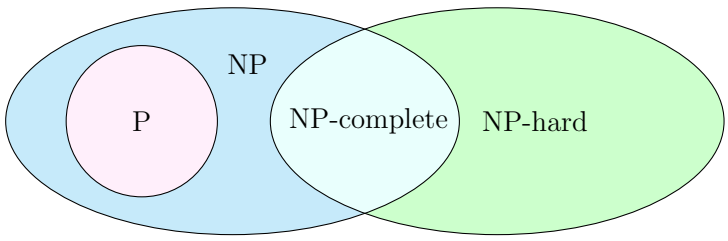


Figure 3.2: Venn Diagram of Complexity Classes assuming P $\neq$ NP.

3.3 Counting Problems

As we have seen, even decision problems lead to challenges (P vs NP). What about counting the number of solutions to a computational problem? These problems, called **counting problems**, are functions

$$f : \{0,1\}^* \to \mathbb{N},$$

that might be *even* harder.

3.3.1 FP and #P

For counting, the complexity classes **FP** and **#P** are the analogues of P and NP for decision problems.

The class FP (which stands for function polynomial) is the set of counting problems $f : \{0,1\}^* \to \mathbb{N}$ that can be computed by a deterministic Turing machine in a time bounded by some polynomial in the size of the input. The number of counting problems in FP are few. Two notable ones are the problem of counting the number of spanning trees in a graph and counting the number of perfect matchings in planar graphs. Proofs of this can be found in Chapter 1 of [23]. An interesting note about these two problems was made by Valiant [35], where these two problems reduce to the problem of computing a determinant. Recall that the determinant of an $n \times n$ matrix A is defined as

$$\det(A) = \sum_{\sigma \in S_n} \operatorname{sign}(\sigma) \prod_{i=0}^{n-1} A_{i,\sigma(i)}$$

where S_n is the symmetric group on n elements. The determinant of any matrix can be computed in polynomial time via the Gaussian elimination.

The class #P consists of all counting problems $f : \{0,1\}^* \to \mathbb{N}$ that be computed in polynomial time by a non-deterministic Turing machine M such that $f(x)$ equals the number of accepting branches in M's computation graph on x. Similar to NP, informally, the problems in #P are those for which we can verify their solutions in polynomial time.

If we could find an algorithm that solves a counting problem, then we would trivially have an algorithm that solves the decision version of this problem. Thus, if someone were to prove that FP = #P, then P = NP. However, the converse is not true. An interesting theorem about counting problems is due to Toda [32]. Informally, Toda proved that if we could use an algorithm for a problem in #P efficiently, then every problem in the *polynomial time hierarchy*, which includes the class NP, could be solved efficiently. Toda's theorem implies counting problems are much harder than decision problems.

3.3.2 #P-completeness and 0,1-PERM

In this thesis, we will use *polynomial-time counting reductions* that makes use of an **oracle**. Similar to the Turing reduction, in this type of reduction, we assume we

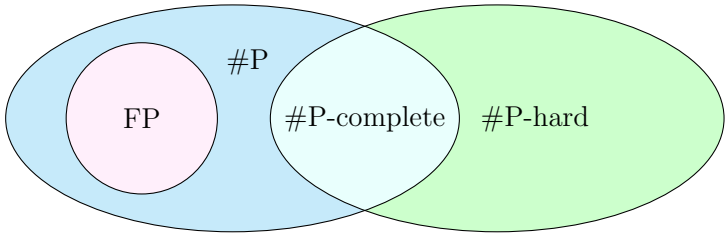


Figure 3.3: Venn Diagram of Complexity Classes assuming FP $\neq$ #P.

have access to an algorithm that computes some problem B in a single computational step. Such algorithm for B, is called an oracle of B. We say that the counting problem A reduces to the counting problem B, if we can solve A in polynomial time using an oracle for B. We will denote this type of reduction by

$$A \leq B.$$

A special type of polynomial-time counting reductions are **parsimonious reductions**. These are counting reductions that preserve the number of solutions from one problem to another. If two problems are parsimoniously interreducible, then they share the same complexity.

As in the decision setting, a problem is said to be **#P-hard** if every problem in #P reduces to it, and **#P-complete** if, in addition, it is in #P. In this thesis, all the problems we consider can be easily verified to be in #P. Hence, we will only focus on proving #P-hardness.

An interesting and open problem regarding counting problems is the following conjecture, for which no counterexamples are known:

Conjecture 3.3.1. NP*-complete decision problems give rise to* #P*-complete counting problems.*

Many examples of problems decision problems which are NP-complete with corresponding counting problem #P-complete are given by Simon [31].

We now introduce the famous problem of computing a quantity called the **permanent**. The permanent of an $n \times n$ matrix A is defined as

$$\text{perm}(A) = \sum_{\sigma \in S_n} \prod_{i=0}^{n-1} A_{i,\sigma(i)},$$

where S_n is the symmetric group of size n. We let 0,1-PERM denote the problem of computing the permanent of a 0,1-matrix:

Name.	0,1-PERM
Instance.	An $n \times n$ 0,1-matrix A.
Output.	The permanent of A, that is perm(A).

In spite of how similar the permanent is to the determinant formula, computing the permanent is very difficult. In 1979, Valiant [35] proved the problem 0,1-PERM is #P-complete from first principles, much like Cook did with SAT.

The 0,1-PERM problem has a combinatorial interpretation as counting perfect matchings in bipartite graphs. Consider a bipartite graph B with vertex partition $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$. We assume X and Y are of the same size, otherwise no perfect matchings exist. Let A be the 0,1-matrix where the ijth entry is equal to 1 if and only if there is an edge connecting x_i and y_j. Then for every permutation $\sigma \in S_n$, the product $\prod_{i=1}^{n-1} A_{i,\sigma(i)}$ equals 1 if and only if the edges which the $A_{i,\sigma(i)}$ entries correspond to form a perfect matching in B.

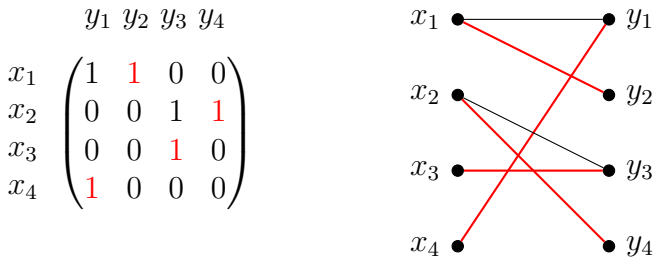


Figure 3.4: A 0,1-matrix and a perfect matching in the bipartite graph corresponding to the matrix.

Interestingly, the decision version of the 0,1-PERM is in P [13]. Jerrum [23] provides a nice proof of the #P-completeness of the 0,1-PERM through a series of reductions from a #P-complete problem which he calls #Exact3Cover to counting perfect matchings in bipartite graphs.

The permanent has another combinatorial interpretation: counting **cycle covers** in directed graphs. A cycle cover of a graph G is a set of cycles that are subgraphs of G and contain all the vertices of G. Now consider the definition of the permanent above: in the sum over permutations, the product of the edge-weights is equal to 1 if and only if the permutation corresponds to a cycle cover, as in Figure 3.5. Otherwise, it equals zero. Hence, we have

$$\text{perm(A)} = \#\text{Cycle covers of the graph with adjacency matrix } A.$$

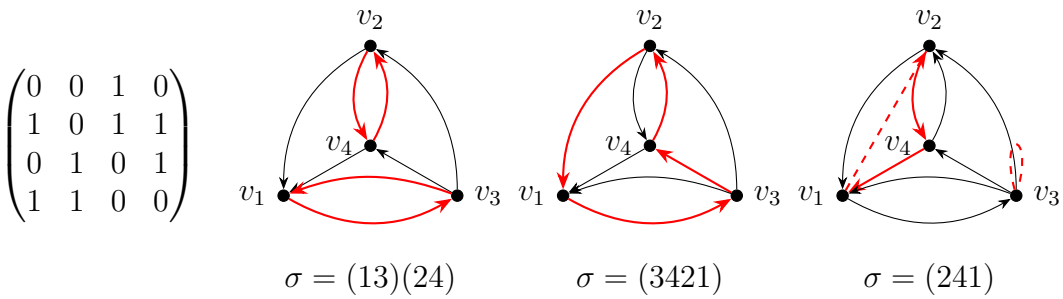


Figure 3.5: A 0,1-matrix and cycle covers highlighted in red its corresponding directed graph along with corresponding permutations $\sigma \in S_4$.

An interesting phenomenon observed about counting problems is the prevalence of so called *dichotomy theorems*. These types of theorems show that for a large

class of problems, the great majority of problems are #P-complete, with only a small subset being in FP. Thus, no problem lies in any of the intermediate classes between FP and #P. We will see that counting homomorphisms to fixed graphs is a dichotomy in Chapter 6.

3.3.3 Polynomial Interpolation via the Vandermonde Matrix

In this section we discuss one of the main techniques used in #P-completeness proofs that will be employed in the many of the reductions presented in this thesis: polynomial interpolation via the Vandermonde matrix. An important property of interpolation via the Vandermonde matrix is that it can be done in polynomial time.

We start by informally describing the procedure: Suppose we wish to find a polynomial $p(x)$ of degree n with unknown coefficients $a_0, a_1, \ldots, a_n$,

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots a_n x^n.$$

Let us assume we are given a set of distinct $n+1$ inputs together with a set of output pairs $(x_0, p(x_0)), \ldots, (x_n, p(x_n))$. We have the following system of equations in matrix form:

$$\begin{pmatrix} p(x_0) \\ p(x_1) \\ \vdots \\ p(x_n) \end{pmatrix} = \begin{pmatrix} 1 & x_0 & \cdots & x_0^n \\ 1 & x_1 & \cdots & x_1^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix}$$

The $(n+1) \times (n+1)$ matrix above is called a Vandermonde matrix. These types of matrices are invertible if and only if all of the x_i entries are distinct. If the Vandermonde matrix is non-zero, we can invert it and recover the a_is. This can be done in polynomial time using Gaussian elimination.

Lemma 3.3.2. *Let $w_1, \ldots, w_r$ be known distinct constants. Suppose that we know values $f_1, \ldots, f_r$ such that*

$$f_s = \sum_{i=1}^{r} c_i w_i^s$$

for $1 \leq s \leq r$. The coefficients $c_1, \ldots, c_r$ can be evaluated in polynomial time.

Proof. We can express the equations in matrix form.

$$\begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_r \end{pmatrix} = \begin{pmatrix} w_1 & w_2 & \cdots & w_r \\ w_1^2 & w_2^2 & \cdots & w_r^2 \\ \vdots & \vdots & \ddots & \vdots \\ w_1^r & w_2^r & \cdots & w_r^r \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_r \end{pmatrix}$$

The $r \times r$ matrix is invertible. If you divide every column by the first entry in the first row of that column, you have a Vandermonde matrix with distinct columns

and this matrix is invertible. Hence, the same is true for its original matrix. We can invert a matrix in polynomial time, to solve for $c_1, \ldots, c_r$. □

How is polynomial interpolation used in reductions involving graphs? The reductions of #P-complete problems involve building new graphs H_is from an instance graph H. The idea is that using an oracle, one is able to compute the outputs of the problem with instance H_i (just as if the oracle was evaluating a polynomial at a point). How the family of graphs H_i are constructed is dependent of the nature of the problem. In many cases, the $H_i s$ should preserve some property, such as the maximum degree of the graph, or remove a property, such as making it triangle free.

Two common operations to create are *stretchings* and *thickenings*. The ***r*-stretch** of a graph G, denoted $S_r G$, is the the graph obtained by replacing every edge $\{v, w\} \in E$ with a copy of the path P_r and making the identifications $v = u_0$ and $w = u_r$. It is also possible to define an r-stretch with respect to a subset of edges. The r-stretch of G with respect to F, is the graph G in which every edge $\{v, w\} \in F$ is replaced with the path P_r using the same identifications as before. This graph is denoted by $S_r^{(F)}(G)$. For $p \geq 1$ the ***p*-thickening** of a graph G is the graph constructed from G by replacing each edge of G by p copies of itself. We denote the p-thickening of a graph G by $T_p G$. The graph $T_p G$ will always be a multigraph for $p > 1$, where each edge has multiplicity p. Similarly to r-stretches, the p-thickening of a graph can be defined with respect to a subset of edges $F \subseteq E$. This is denoted by $T_p^{(F)}(G)$.

Chapter 4

The Complexity of Counting Independent Sets

In this chapter we consider the complexity of #IS, the problem of counting the independent sets of a graph. This problem has important applications to statistical physics (for instance, the hard-core model of a gas) [2] and network design [30]. Similar to the problem of counting perfect matchings in bipartite graphs, the decision problem for #IS can be efficiently computed. All graphs have an independent set, as the empty set is an independent set. On the other hand, the task of finding an algorithm that will determine whether a graph has an independent set of at least size k, where k is an arbitrary integer, is NP-complete [25].

We start by establishing the #P-completeness of #IS for general graphs and follow by considering the complexity of special cases. We will see that it is only in highly restricted instances that counting becomes easy. The main proof of this chapter shows that #IS is #P-complete as soon as graphs have maximum degree greater than two. The original proof can be found in the paper by Greenhill [19]. We provide some extra details.

4.1 Counting Independent Sets and Relationship to Other Graph Properties

We let #IS denote the computational problem of counting the independent sets of an arbitrary graph.

Name.	#IS
Instance.	A graph G.
Output.	The number of independent sets of G that is, $\lvert\mathcal{I}(G)\rvert$.

Provan and Ball [28] proved through reductions that counting independent sets in bipartite graphs is #P-complete. Their result automatically implies #IS is #P-complete, since an algorithm that solved the general problem efficiently would automatically solve the problem on bipartite graphs efficiently.

Counting independent sets is closely linked to two other counting problems. A set is an independent set if and only if it forms a clique in the graph's complement. This one-to-one correspondence between independent sets and cliques implies #IS is parsimoniously interreducible with #CLIQUE, the problem of counting cliques in a graph. Another graph parameter related to independent sets are vertex covers. It follows directly that a set X is an independent set if and only if $V \setminus X$ is a vertex

cover. Thus, the problem of counting vertex covers #VERTEX-COVER is parsimoniously interreducible with #IS. We therefore have the following proposition.

Proposition 4.1.1. *The following problems are parsimoniously interreducible with each other in polynomial time:*

1. #IS - *the problem of counting the independent sets of a graph.*
2. #CLIQUE - *the problem of counting the cliques of a graph.*
3. #VERTEX-COVER - *the problem of counting the vertex covers of a graph.*

Other problems #IS is parsimoniously interreducible with, in polynomial time, include: counting the downsets in partially ordered sets [33], counting solutions to monotone 2-CNF formulas [33], and counting solutions to a specific constraint satisfaction problem [24]. Since these reductions are parsimonious, the #P-completeness and FP results of this chapter will immediately translate to the interreducible problems. Reductions that preserve graph structure, will preserve complexity class.

Even though counting independent sets in general graphs is #P-complete, we can still try to find if there are special classes of graphs for which counting independent sets is easy. Intuitively, it should be easy to determine the number of independent sets for graphs with very few edges, or the opposite, graphs that are almost complete. We determine the "boundary" at which counting independent sets transitions from being easy to hard in the next two sections.

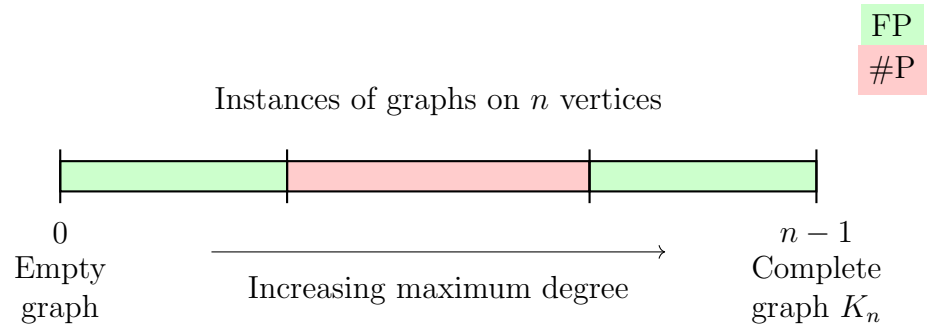


Figure 4.1: Complexity of #IS for regular-degree graphs.

We end this section by stating the simple fact that counting independent sets of a graph is as hard as counting the independent sets of its connected components. Moreover, if G is a graph with connected components $G_1, \ldots, G_\ell$, where $\ell \in \mathbb{N}$, then

$$|\mathcal{I}(G)| = \prod_{i=1}^{\ell} |\mathcal{I}(G_i)|.$$

Any graph will have a polynomial number of components, so computing the number of independent sets amounts to computing it for its connected components. In particular, this tells us that we only need to consider the complexity of #IS for connected graphs. We will make this assumption later.

4.2 Counting Independent Sets in Sparse Graphs

In this section we consider the complexity of counting independent sets in graphs with few edges. These types of graphs are referred to as *sparse* graphs.

Consider graphs of maximum degree 2. The connected components of these graphs are either paths, cycles, or isolated vertices. We prove as a corollary of the next two propositions that finding the number of independent sets of such connected components can be done in polynomial time. The following propositions were stated in paper [19] but we provide short proofs. Let f_r denote the rth Fibonacci number, defined recursively by $f_r = f_{r-1} + f_{r-2}$ for $r \geq 3$, with $f_0 = 0$ and $f_1 = f_2 = 1$. Recall that P_r denotes the path of length r with vertices $\{u_1, \ldots, u_{r+1}\}$ and edges $\{u_i, u_{i+1}\}$ for $i \in \{0, \ldots, r\}$ for $r \geq 0$.

Proposition 4.2.1. *The total number of independent sets of P_r, the path of length r, is f_{r+3}, where f_r denotes the rth Fibonacci number.*

Proof. The proof follows from induction on r. The paths P_0 and P_1 form our base cases and have a total of 3 and 5 distinct independent sets respectively (shown in Figure 4.2).

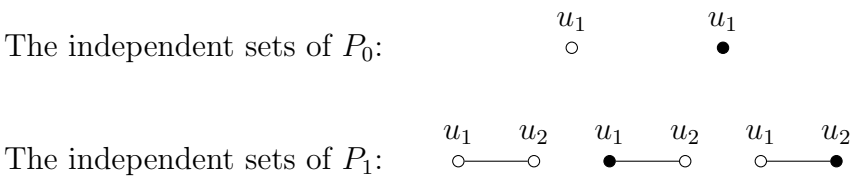


Figure 4.2: The independent sets of P_1 and P_2. The vertices filled in black form the independent set.

Notice that $2 = f_3$ and $3 = f_4$. Thus, $|\mathcal{I}(P_0)| = f_3$ and $|\mathcal{I}(P_1)| = f_4$.

As our inductive hypothesis, assume $|\mathcal{I}(P_k)| = f_{k+3}$ for all $k < r$. We prove the statement holds for r. The independent sets of P_r can be partitioned into two sets: the independent sets which do not contain the last vertex u_{r+1}, which we call I_1, and the independent sets which contain u_{r+1}, which we call I_2. Clearly,

$$|\mathcal{I}(P_r)| = |I_1| + |I_2|.$$

The independent sets of I_1 (those which do not contain u_{r+1}), correspond exactly to the independent sets of P_{r-1}. Hence, we have $|I_1| = |\mathcal{I}(P_{r-1})| = f_{r+2}$ by our induction hypothesis. In the other case, u_{r+1} is contained in the independent sets, and therefore the vertex u_r must not be contained. All other remaining vertices may or may not be included in the independent sets, and therefore correspond

to independent sets of P_{r-2}. Hence, $|I_2| = |\mathcal{I}(P_{r-1})| = f_{r+2}$ by the induction hypothesis. We therefore have

$$|\mathcal{I}(P_r)| = f_{r+2} + f_{r+1} = f_{r+3}.$$

Hence, the number of independent sets of the path of length r is given by the Fibonnacci number f_{r+3}. □

Recall that we let C_r denote the cycle of r edges with vertices $\{u_1, \dots u_r\}$ and edges $\{u_i, u_{i+1}\}$ for $i \in \{1, \dots, r-1\}$ and $\{u_1, u_r\}$.

Proposition 4.2.2. *For $r \geq 3$, the total number of independent sets of the cycle C_r is $f_{r-1} + f_{r+1}$.*

Proof. The cycles C_3 and C_4 form our bases cases and have a total of 4 and 7 and independent sets respectively (shown in Figure 4.3).

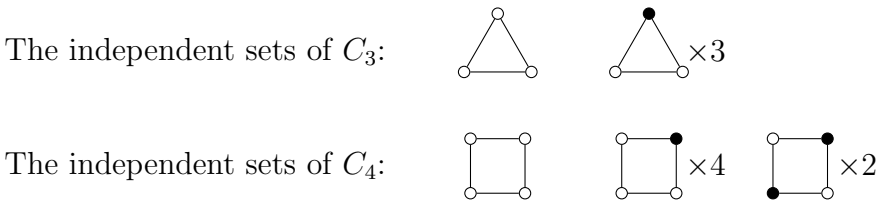
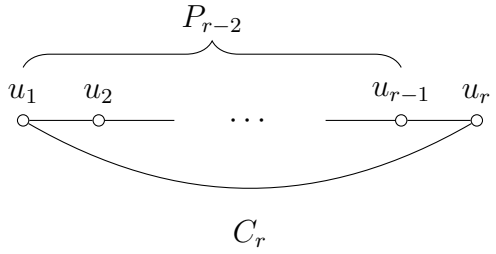


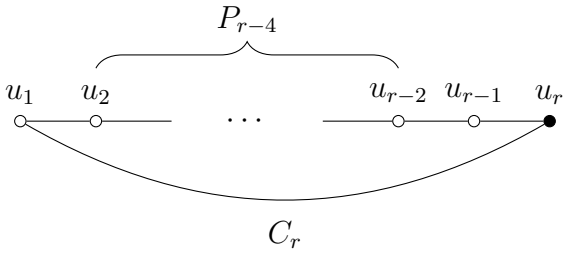
Figure 4.3: The independent sets of C_3 and C_4. The vertices filled in black form the independent set.

Notice that $4 = f_2 + f_4$ and $3 = f_3 + f_5$. Thus, $|\mathcal{I}(C_3)| = f_2 + f_4$ and $|\mathcal{I}(C_4)| = f_3 + f_5$.

We similarly partition the independent sets of C_r by whether they include the vertex u_r or not. The independent sets that do not contain the vertex u_r may contain all other vertices, and therefore correspond to the independent sets of P_{r-2}. The total such number of independent sets is f_{r+1} by Proposition 4.2.1.



The independent sets that do contain the vertex u_r must not include the vertices u_1 and u_{r-1}. All the other vertices may or may not be contained in the independent sets and correspond exactly to independent sets of P_{r-4}. The total such number of independent sets is f_{r+1} by Proposition 4.2.1.



Thus, the total number of independent sets of C_r is $f_{r-1} + f_{r+1}$. □

Corollary 4.2.3. *If G is a graph of maximum degree $\Delta \leq 2$, then computing $|\mathcal{I}(G)|$ can be done in constant time.*

Proof. Let G be a (connected) graph of maximum degree 2. Then G is either a path graph or a cycle. If G is a single vertex, it has a total of 2 independent sets. Otherwise, by Propositions 4.2.1 and 4.2.2, the total number of independent sets of G is given by a finite sum of Fibonacci numbers. The Fibonacci numbers can be computed in polynomial time using Binet's closed form formula:

$$f_n = \frac{1}{\sqrt{5}}(\phi^n - \phi^{-n})$$

where ϕ is the golden ratio, equal to $(1 + \sqrt{5})/2$. □

It turns out that graphs with maximum degree 2 form the boundary beyond which counting independent sets becomes intractable. The author Vadhan proved in [34] that counting the independent sets of bipartite graphs of maximum degree four is #P-complete. This directly implies that counting independent sets of graphs with degree greater than 4 is #P-complete. The paper *The complexity of counting colourings and independent sets in sparse graphs and hypergraphs* by Greenhill [19] closed the gap and proved that it is #P-complete to count the independent sets of graphs of degree greater than two.

Name.	#IS(3)
Instance.	A graph with maximum degree three.
Output.	The number of independent sets in the graph.

Name.	#IS(≥ 4)
Instance.	A graph with maximum degree Δ where $\Delta \geq 4$.
Output.	The number of independent sets in the graph.

The proof reduces the problem of counting independent sets of maximum degree at least four to the problem of counting independent sets of graphs with maximum degree three. It is done by showing you can efficiently solve #IS(≥ 4) by calling an oracle for #IS(3) a polynomial number of times. This is a good example of a proof that employs the commonly used reduction techniques we discussed in Chapter 2: transformations on the vertices or use of gadgets, and polynomial interpolation. We now present the proof of this paper.

Theorem 4.2.4. *The problem* #IS(3) *is* #P*-complete.*

Proof. Let G be an instance of #IS(≥ 4). We construct an instance of #IS(3) which we call $H = (V, E)$ from G. When constructing H, we partition the edges of H into disjoint sets E' and E''. This is done because we will later perform r-stretches with respect to the edges in E''. The construction of H is as follows:

1. Modify all the high-degree vertices. Do the following procedure on all vertices $v \in G$ with $\deg(v) = d > 3$:
   (a) Replace v with a path P_{d-3}. That is, in place of v, add the vertices $v_1, \cdots, v_{d-2}$ along with the edges $\{v_i, v_{i+1}\}$ for $1 \leq i < d-2$. The edges of the path are placed in E''.
   (b) Connect all the neighbours of the original $v \in G$ to vertices on the path in such a way that every v_i for $1 \leq i < d-2$ has degree three. More precisely, connect two distinct neighbours of v to v_1, connect another two distinct neighbours of v to v_{d-2}, and connect all remaining neighbours to distinct v_is. All of the edges introduced in this step are placed in E'.

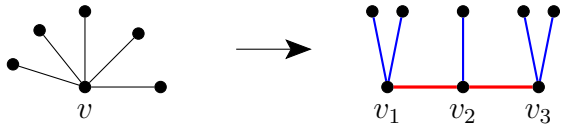


Figure 4.4: Example of how a high-degree vertex in G is transformed in H by steps (a) and (b). The blue edges belong to E' and the red edges E''.

2. After all the high degree vertices have been modified, include all other vertices and edges of G in H. These edges are placed in E'.

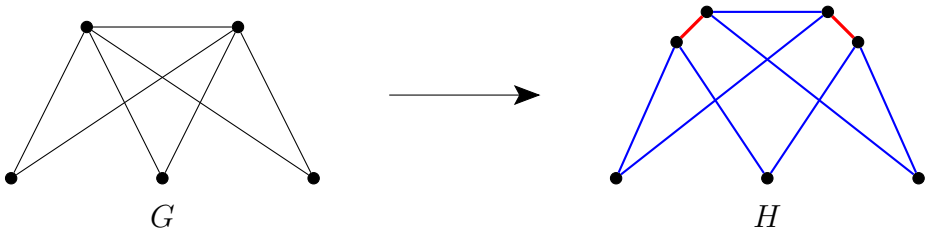


Figure 4.5: Examples of G and H. The blue edges belong to the set E' and the red edges to the set E''.

The construction of H can be done in polynomial time and ensures H is a graph of maximum degree three. We now show how you can compute $|\mathcal{I}(G)|$ by interpolation on H. Let $m = |E''|$. For $0 \leq t \leq m$ and $0 \leq i \leq t$, let $n_{t,i}$ denote the number of subsets X of V such that

(i) if $e \in E'$ then $e \not\subseteq X$,
(ii) $|\{e \in E'' : e \cap X = \emptyset\}| = i$,
(iii) $|\{e \in E'' : e \subseteq X\}| = t - i$.

We remark on what the conditions mean. Condition (i) requires all the vertices joined by edges E' to form an independent set. Condition (ii) tells us the number of edges in E'' that have empty intersection with X and condition (iii) tells us the number of edges in E'' fully contained in X. From conditions (ii) and (iii) we can deduce that the number of edges in E'' where only one of the vertices is in X is $m - (t - i) - i = m - t$. Note that if $t = m$ then all edges of E'' are either in X or not. Moreover, since all the edges in E'' are edges of paths, if $t = m$, either the whole path is included or it is not included at all. From these conditions, it is easy that when $t = m$, subsets that satisfy these conditions are in one-to-one correspondence with independent sets in G. Thus,

$$|\mathcal{I}(G)| = \sum_{i=0}^{m} n_{m,i}.$$

If we can find the value of $n_{m,i}$ for $0 \leq i \leq m$, then we can compute $|\mathcal{I}(G)|$. In order to do so, we perform r-stretches on H with respect to the edges in E'' and use polynomial interpolation. For $1 \leq r \leq m + 1$, perform the r-stretch of H with respect to the edges in E''. That is, replace every edge $\{v, w\} \in E''$ by P_r, and connect all the neighbours of v to u_1 and all the neighbours of w to u_{r+1}. Denote the resulting graph by H_r. The family of graphs $\{H_r : 1 \leq r \leq m + 1\}$ can be constructed in polynomial time and are all graphs of maximum degree three.

We now express $|\mathcal{I}(H_r)|$ in terms of the $n_{t,i}$ variables. Note that H_r has m special paths of length r created by the stretching of the edges in E''. The number $n_{t,i}$ is equal to the number of independent sets S of H_r in which:

1. none of the inner vertices of the paths are included in S,
2. i paths have neither endpoints in S,
3. $t - i$ paths have both their endpoint vertices in S,
4. and $m - t$ paths have exactly one endpoint in the S.

We can partition the independent sets of H_r based on the number of edges of E'' that are fully included, partially included, and not included at all in the independent sets. We first have to compute how many independent sets each path has in each case. Let us compute this for paths of length r:

1. Case 1: Independent sets that do not include the endpoints of P_r. These independent sets can only include vertices in the inner path of length $r - 2$. By Proposition 4.2.1, the total number of such independent sets equals f_{r+1}.
2. Case 2: Independent sets that include the endpoints of P_r. In this case, we can include any of the vertices in the inner path of length $r - 4$. There are exactly f_{r-1} such independent sets.
3. Case 3: Independent sets that include exactly one vertex of P_r. These independent sets correspond to independent sets of a path of length $r - 3$. By Proposition 4.2.1, there are f_r such independent sets.

Thus, the number of independent sets of H_r is

$$\begin{aligned}
|I(H_r)| &= \sum_{t=0}^{m}\sum_{i=0}^{t} n_{t,i} f_{r+1}^{i} f_{r-1}^{t-i} f_r^{m-t} \\
&= \sum_{t=0}^{m}\sum_{i=0}^{t} n_{t,i} f_{r+1}^{i} f_{r+1}^{t-i} f_{r+1}^{m-t} \left(\frac{f_{r-1}}{f_{r+1}}\right)^{t-i} \left(\frac{f_r}{f_{r+1}}\right)^{m-t} \\
&= f_{r+1}^{m} \sum_{t=0}^{m}\sum_{i=0}^{t} n_{t,i} \left(\frac{f_{r-1}}{f_{r+1}}\right)^{t-i} \left(\frac{f_r}{f_{r+1}}\right)^{m-t} \\
&= f_{r+1}^{m} \sum_{t=0}^{m}\sum_{i=0}^{t} n_{t,i} \left(\frac{f_{r+1}-f_r}{f_{r+1}}\right)^{t-i} \left(\frac{f_r}{f_{r+1}}\right)^{m-t} \\
&= f_{r+1}^{m} \sum_{t=0}^{m}\sum_{i=0}^{t} n_{t,i} (1-x_r)^{t-i} x_r^{m-t},
\end{aligned}$$

where $x_r = f_r/f_{r+1}$. The x_r values are distinct for different values of r. Thus the coefficient of $n_{t,i}$ is always distinct. We can compute $|\mathcal{I}(H_r)|$ for every $r > 0$ and perform polynomial interpolation to find the value of $n_{t,i}$ for $0 \leq t \leq m$ and $0 \leq i \leq t$ in polynomial time using Lemma 3.3.2. □

Even though we established that counting independent sets of graphs of maximum degree 2 is in FP, we briefly mention why the ideas of the proof don't let us reduce #IS(3) to #IS(2). In the above proof, we had to turn an instance G of #IS(≥ 4) into and instance H of #IS(3). We were then able to compute the size of $|\mathcal{I}(G)|$ by computing the number of independent sets of the r-stretches of H making oracle calls. The transformation of G into H required us to introduce new vertices into the graph such that the graph stayed connected and now each vertex had degree less than or equal to three. If we were to reduce from #IS(3) to #IS(2), we would have to modify all the vertices of degree three so that they become vertices of degree at most two. However this is not possible. There is no way of turning a degree three vertex into a new set of vertices such that they are all connected and have degree at most two. Clearly, transforming v into anything more complicated would introduce more edges.

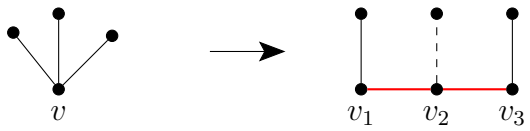


Figure 4.6: A degree three vertex v cannot be transformed into a chain of vertices of degree 2 or less.

Lastly, in addition to #P-completeness result of #IS(3), the paper [19] further proved that counting independent sets in 3-regular graphs is #P-complete.

4.3 Counting Independent Sets in Dense Graphs

In this section, we prove that counting independent sets in almost complete graphs is polynomial-time computable. We make use of the fact that independent sets correspond to cliques in the complement graph.

Proposition 4.3.1. *Let G be a graph on n vertices with minimum degree $\delta \geq n-3$. Computing the number of independent sets of G can be done in polynomial time.*

Proof. If every vertex of the graph G has degree $n-3, n-2$ or $n-1$, then all the vertices of $\overline{G}$ will have either degree 0, 1, or 2. Thus, the complement of G consists of isolated vertices, paths, and cycles. Counting the independent sets of G is equivalent to counting the cliques of $\overline{G}$ and adding 1 (for the empty set). The only possible cliques in $\overline{G}$ are K_1, K_2 or K_3. Detecting these cliques can be done in polynomial time. See Figure 4.3 for an illustration □

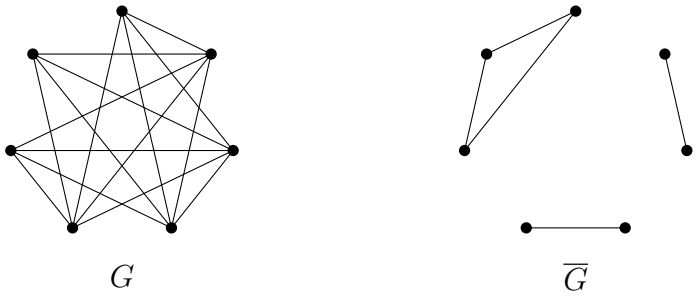


Figure 4.7: The graph G has 7 vertices and minimum degree 4. From the complement of G we can easily see there are exactly 3 cliques.

As soon as a graph has minimum degree $\delta = n-4$, then detecting cliques in the complement graph becomes computationally hard.

4.4 Approximability

We conclude this chapter by stating some results about approximating the number of independent sets. The paper *On the relative complexity of approximate counting problems* [10] the authors proved that approximating #IS is as hard as approximating the number of satisfying assignments to a SAT instance. The problem of approximating #SAT is #P-complete with respect AP-reducibility, where an AP-reduction is an approximation-preserving reductions defined in [10]. This means that #SAT does not have an efficient approximation algorithm unless NP equals the randomised complexity class RP. Thus, it is highly unlikely efficient approximation algorithms for #IS exist.

On the other hand, the approximability complexity of counting independent sets in bipartite graphs, known as #BIS, is unknown. It is conjectured that #BIS lies somewhere between problems that admit a fully polynomial randomised approximation scheme and those interreducible with #SAT. Evidence for this is the high

number of problems interreducible with #BIS, for which neither an FPRAS have been found nor a reduction from #P-complete is proven.

Even though approximating the number of independent sets may be difficult, it might be possible to bound the number of independent sets of certain graphs. We briefly remark on how you could find simple bounds for graphs that are highly connected but sparse. We specifically comment on graphs that have a large number of vertices and are d-regular for small $d \geq 3$. We can find bounds for these types of graphs by exploiting the fact that counting independent sets in paths is easy. In large and connected sparse graphs, from experimental trials, it seems highly likely that on a first try, a long path using a BFS algorithm can be found. After this first path on the graph is found, we can find other paths on the uncovered vertices. This is continued until all the vertices of the graph are covered. Then the number of independent sets of the whole graph is bounded from below by the sum of the number of independent sets of each of the paths. Moreover, the number of independent sets of the whole graph is bounded from above by the product of number of independent sets of the paths (since edges reduce the number of independent sets). If the first path found is especially long and all the other paths are short, then the upper bound will be better.

Figures 4.8 and 4.9 show examples of how this algorithm performed on a random d-regular graphs. The code was generated with the help of ChatGPT, which suggested the use of these standard Python modules: networkx and matplotlib.

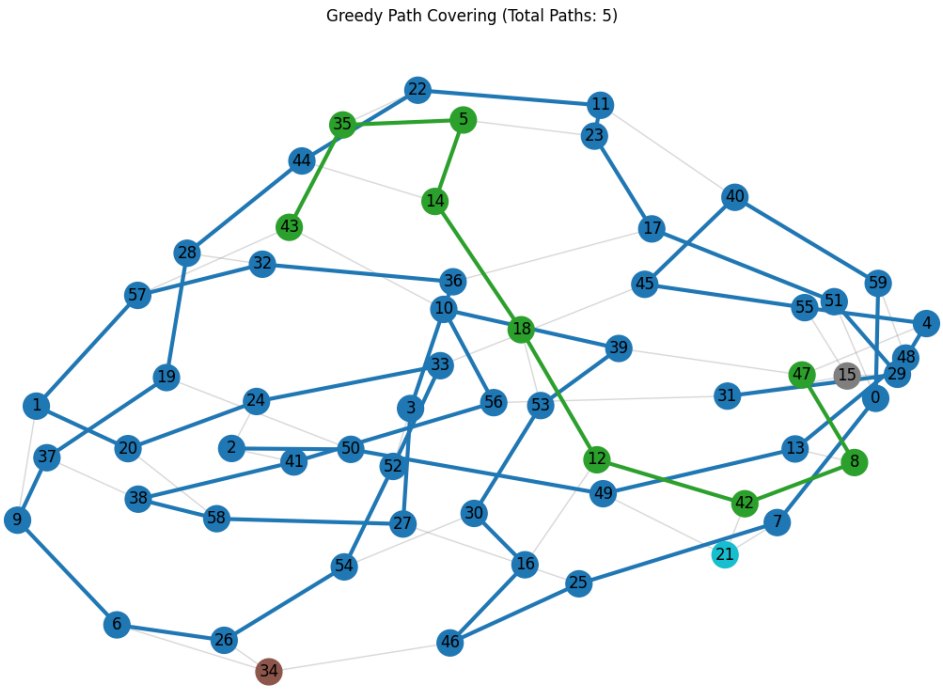


Figure 4.8: A random 3-regular path on 60 vertices with paths covering the graph.

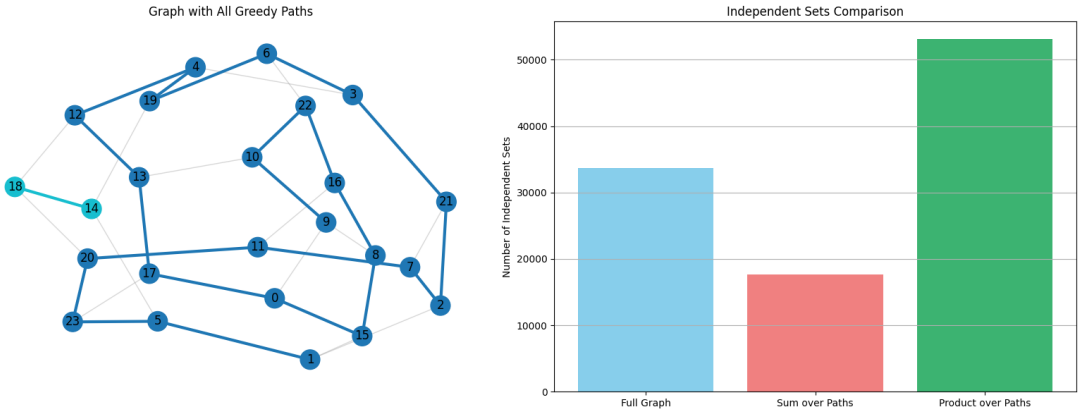


Figure 4.9: Algorithm run on a random 3-regular graph on 24 vertices. Comparison between the number of independent sets of full graph, the sum of the paths, and product of paths.

Chapter 5

Graph Parameters as Homomorphism Counts

This chapter starts by defining homomorphisms between graphs, which will allow us to generalise the concept of k-colourings to express other graph characteristics. In particular, we will be interested in counting the number of homomorphisms to a fixed graph H. This plays a central role in statistical physics, particularly in relationship to the *Ising model*. Furthermore, we discuss two extensions: weighted graph homomorphism counts and quantum graphs, which allow us to express even more graph characteristics.

5.1 Graph Homomorphisms

A **graph homomorphism** from a graph G to a graph H is a mapping f from the vertex set of G to the vertex set of H, denoted $f : G \to H$, such that

$$\text{if } (u, v) \in E(G), \text{ then } (f(u), f(v)) \in E(H).$$

If a graph homomorphism from G to H exists, then G is said to be **homomorphic** to H or H**-colourable**.

A graph homomorphism is a mapping between graphs that preserves the adjacency relation on the vertices.

We let $\Omega_H(G)$ denote the set of all H-colourings of G and $\#\mathrm{Hom}(G, H)$ the size of this set. For ease of reading, we use H to denote an arbitrary target graph, and G to denote an arbitrary source graph, from which graph homomorphisms into H are considered.

The number of homomorphisms from G to H can give us information on the combinatorial properties of G and H. This is because the edges of H determine which vertices can be adjacent in G. For example, if we know H is a graph with no loops, then $\#\mathrm{Hom}(K_n, H)/n!$ is equal to the number of cliques of size n in H. Problems of this kind, where we are counting homomorphisms from a fixed source graph, are related to counting small subgraphs in larger graphs [8]. We will instead focus on counting homomorphisms from arbitrary graphs to small but fixed target graphs, which will allow us to generalise many useful *graph parameters* and express various graph properties in a unified framework. We present three examples of this: how graph homomorphisms generalise graph colourings, how the number of independent sets of a graph is equal to the number of homomorphisms to some fixed graph, and how homomorphisms relate to scheduling problems.

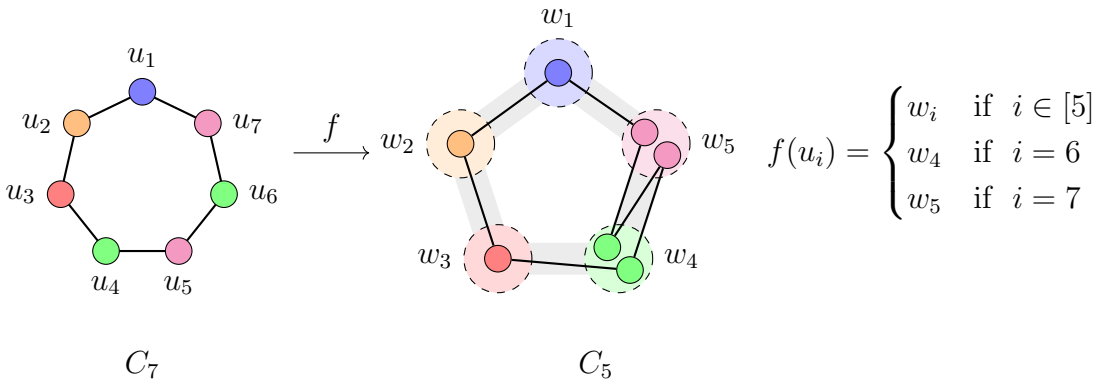


Figure 5.1: Example of a graph homomorphism from C_7 to C_5.

5.1.1 Graph Colourings

Recall that a graph colouring of G is simply an assignment of colours, which we think of as some set $\{1, \ldots, k\}$, to the vertices of G. In this light, the vertices of H can be viewed as the possible colours, each labelled with an element of $\{1, \ldots, k\}$, and the edges of H indicate which vertices (colours) may be adjacent in G. For example, a proper 3-colouring of a graph G is equivalent to a homomorphism of G to K_3, as in Figure 5.2. More generally, if G is K_n-colourable, then G admits a proper n-colouring and $\#\text{Hom}(G, K_n)$ equals the number of proper n-colourings of G. If we were instead interested in counting all the proper and non-proper n-colourings of G, this would be equivalent to counting the number of homomorphisms to the reflexive complete graphs on n vertices (the loops allow same colours to be adjacent in G).

Graph homomorphisms also encompass other types of colourings such as circular and fractional colourings and thus provide a good framework for studying all the possible colourings of a graph. We refer the reader to Hell and Nešetřil's book for further reading on this subject [21].

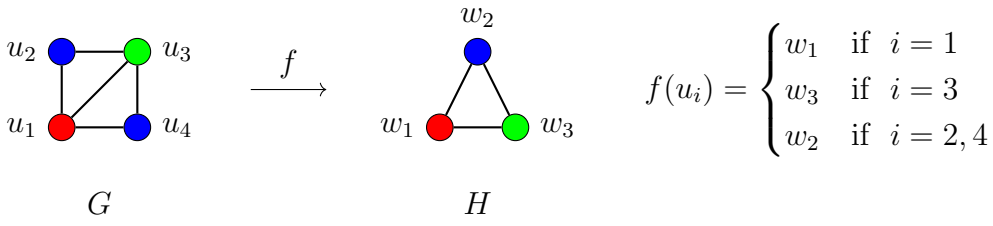


Figure 5.2: Example of a graph homomorphism from the diamond graph to K_3. The explicit homomorphism f tells us exactly how to colour the vertices of G. The total number of homomorphisms from G to H is 6.

5.1.2 Independent Sets

The number of independent sets of G can be expressed as the number of homomorphisms to a fixed graph H. Let H be the graph of Figure 5.3 consisting of two vertices u and v connected by an edge, where only the vertex v has a loop.

If $f : G \to H$ is a homomorphism, then all the vertices mapped to the unlooped vertex u will form an independent set, since all of their neighbours must be mapped to v. Similarly, for every independent set of G, there exists a corresponding homomorphism from G to H which maps the vertices of the independent set to the unlooped vertex u and all other vertices to the looped vertex v. Hence, for this particular graph, $\#\mathrm{Hom}(G, H)$ equals the number of independent sets of G.

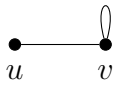


Figure 5.3: Target graph H for which $\#\mathrm{Hom}(G, H)$ equals the number of independent sets of G.

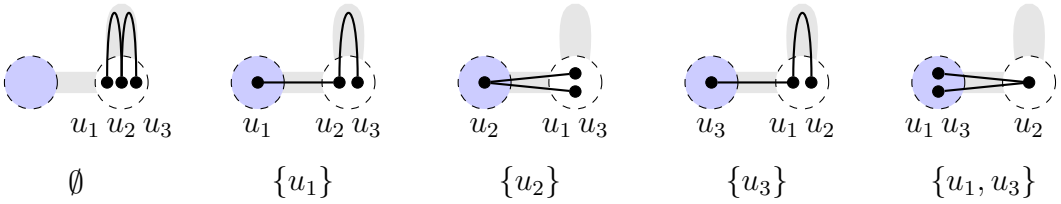


Figure 5.4: All the H-colourings of P_2, where H is the graph for counting independent sets. The vertices in the blue circles form independent sets.

The problem of counting the independent sets of a graph, #IS, is therefore equivalent to the problem of counting graph homomorphisms to the graph H described above. This fact will be used in Chapter 6 when we prove the complexity of counting graph homomorphisms to fixed graphs.

5.1.3 Scheduling Problems

We now explain how scheduling problems can be phrased as homomorphism problems between two graphs. Consider the task of scheduling exams into k periods such that no two exams are scheduled at the same time if there is a student taking both exams. Let G be the graph where each vertex represents an exam and two vertices are connected if there is a student sitting in both exams. Then exams can be scheduled in k periods if and only if G is k-colourable, or equivalently there exists a homomorphism from G to K_k (every vertex in K_k represents a period). Suppose now we want to schedule the exams into k periods such that no student takes two exams in a row. Let H be the complete graph on k vertices minus a path of length $k-1$, as illustrated in Figure 5.5. Then a homomorphism from G to H would correspond to a valid scheduling and $\#\mathrm{Hom}(G, H)$ is the total number of possible schedules. The two highest degree vertices of H represent the first and last periods of the day. The order of the vertices in the $k-1$ path correspond to the order of the periods throughout the day.

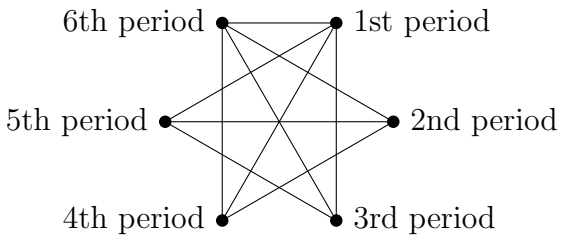


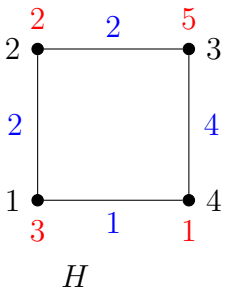
Figure 5.5: The graph H for scheduling exams into 6 periods such that no student takes two exams consecutively.

5.2 Weighted Graph Homomorphisms

In this section we describe a more general notion of homomorphism count: homomorphisms into weighted graphs. We show how counting weighted homomorphisms relates to computing the partition function in statistical physics and give examples of graph parameters expressible in this way.

In this extension of the definition of homomorphism numbers, the source graph G is unweighted and we allow the target graph H to posses vertex weights and/or edge weights. The weights may be elements from any commutative ring but we will only focus on real weights. Let $G = (V, E)$ and $H = (C, E_H)$ where $|V| = N$.

The edge weights of H are stored in its $n \times n$ adjacency matrix, where the ijth entry is equal to the weight of the edge connecting the ith vertex and jth vertex of H. If no edge exists between a pair of vertices, their edge-weight is equal to zero. We call this matrix the *weight matrix* of H and denote it by A. The vertex weights of the graph are stored in an $n \times n$ diagonal matrix D where $(i, i)th$ entry equals λ_i, the the weight of the ith vertex of H.



H

$$A = \begin{pmatrix} 0 & 2 & 0 & 1 \\ 2 & 0 & 2 & 0 \\ 0 & 2 & 0 & 4 \\ 1 & 0 & 4 & 0 \end{pmatrix} \qquad D = \begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 5.6: Example of H with corresponding matrices A and D. The vertices are labelled in black, the vertex weights are given in red and the edge weights are given in blue.

For $X \in \Omega_H(G)$ define the **edge-weighting** of X to be $w_A(X)$ where

$$w_A(X) := \prod_{\{v,w\} \in E} A_{X(v),X(w)}.$$

Since A is a symmetric matrix (because H is not a directed graph), taking the product over edges in G is well-defined. Similarly, we define the **vertex-weighting** of X to be $\widetilde{w}_D(X)$ where

$$\widetilde{w}_D(X) := \prod_{v \in V} \lambda_{X(v)}.$$

If H has both vertex weights and edge weights, we define a quantity associated with the vertex-weighting and the edge-weighting. We call the product of both weightings the **total weight** of X:

$$w_{A,D}(X) = w_A(X)\widetilde{w}_D(X).$$

The **total weight of all H-colourings** of G is the sum of the total weight over all H-colourings of G:

$$Z_{A,D}(G) := \sum_{X \in \Omega_H(G)} w_{A,D}(X).$$

The Z notation comes from the german word "Zustandssumme" meaning "sum over states". It is the notation used for the *partition function* of statistical physics. If all vertex weights of H are equal to one, we simply write $w_A(G)$ and $Z_A(G)$ instead of $w_{A,D}(G)$ and $Z_{A,D}(G)$. Importantly, if every edge has weight 1 also, then

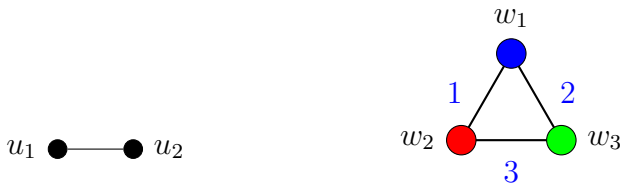
$$Z_A(G) = \sum_{X \in \Omega_H(G)} 1 = |\Omega_H(G)| = \#\mathrm{Hom}(G, H).$$

This shows how $Z_{A,D}$ is a generalisation of the computational problem of counting homomorphisms into unweighted graphs. If we can say something about the complexity of computing $Z_{A,D}$ for different matrices A and D, then we will have results about the complexity of counting homomorphisms into unweighted graphs. In the next chapter, we will prove complexity results about evaluating the functions $Z_{A,D}$ and Z_A for different matrices A and D. We define these computational problems as EVAL(A) and EVAL(A, D).

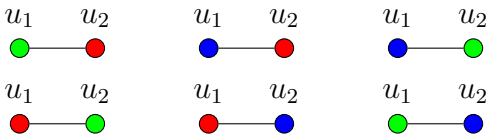
Name.	EVAL(A)
Instance.	A graph G.
Output.	$Z_A(G)$

Name.	EVAL(A,D)
Instance.	A graph G.
Output.	$Z_{A,D}(G)$

We now present three examples of graph properties and how they relate to the Z_A, where A is the adjacency matrix of some graph H we describe. In each example, the graph G is an arbitrary source graph with N vertices. The first example relates Z_A to finding the maximum cut of G, and it be found in [27]. The two other examples were stated in [15] but we give explanations for them. For all three examples, the target weighted graph H is a complete reflexive graph on two nodes. Every mapping from arbitrary G to a complete reflexive graph on two nodes is a valid homomorphism, and hence the total number of homomorphisms is 2^N.



All 6 homomorphisms from G to H:



$$Z_A(G) = 3 + 3 + 2 + 2 + 1 + 1 = 12$$

Figure 5.7: Example of a graph G, a weighted graph H, and evaluating $Z_A(G)$ where A is the adjacency matrix of A.

5.2.1 Maximum Cut

A **cut** of a graph is a partition of the vertices into two disjoint sets. We refer to the edges between the two disjoint sets as **cut-edges**. The size of a cut is equal to its number of cut-edges. A **maximum cut** is a cut of the largest size possible. For example, for any bipartite graph, the size of the maximum cut is equal to the size of the edge set. Let $\text{Maxcut}(G)$ denote the size of the maximum cut of G and let $\text{maxcut}(G)$ denote the *normalised maximum cut*,

$$\text{maxcut}(G) = \frac{\text{Maxcut}(G)}{N^2}.$$

Consider homomorphisms to the edge-weighted graph H shown in Figure 5.8. Every H-colouring of G corresponds to a cut of G, where the vertices mapped to u form one partition and the vertices mapped to v form the other partition. The weight of an H-colouring of G equals 2 raised to the number of cut-edges in the corresponding partition.

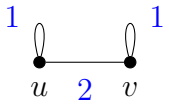


Figure 5.8: Target graph H. The edge weights are given in blue.

At least one of the homomorphisms from G to H will correspond to a maximum cut partition, thus

$$2^{\text{Maxcut}(G)} \leq Z_A(G).$$

On the other hand, the maximum weight of any H-colouring of G is $2^{\text{Maxcut(G)}}$. There are 2^N possible H-colourings, and so

$$Z_A(G) \leq 2^N 2^{\text{Maxcut}(G)}.$$

All together we have

$$Z_A(G)2^{-N} \leq 2^{\text{Maxcut}(G)} \leq Z_A(G).$$

Taking the base-2 logarithm and dividing by N^2, we obtain

$$\frac{\log_2(Z_A(G))}{N^2} - \frac{1}{N} \leq \text{maxcut}(G) \leq \frac{\log_2(Z_A(G))}{N^2}.$$

Therefore, the total weight of homomorphisms to H asymptotically determines the size of the normalised maximum cut as $N \to \infty$. Though maxcut(G) is not exactly expressible as a homomorphism count, homomorphism counts provide these useful bounds.

5.2.2 Induced Subgraphs with an Even Number of Edges

Suppose G is a graph with n induced subgraphs with an even number of edges and m induced subgraphs with an odd number of edges. Clearly, $m + n$ is equal to the total number of induced subgraphs of G. Every homomorphism to the target graph H of Figure 5.9 corresponds to a unique induced subgraph of G. The subgraph is the one induced by the vertices mapped to the vertex v.

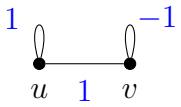


Figure 5.9: Target graph H. The edge weights are given in blue.

Since the total number of homomorphisms to H is 2^N, the total number of induced subgraphs of G is 2^N, thus

$$2^N = m + n.$$

The weight of an H-colouring X of G is equal to 1 if and only if X maps an even number of edges of G to v. Otherwise, the weight of the homomorphism equals -1. Therefore,

$$Z_H(G) = n - m.$$

Then

$$\frac{1}{2}\big(Z_H(G) + 2^N\big) = \frac{1}{2}\big((n - m) + (n + m)\big) = n,$$

which is the number of induced subgraphs of G with an even number of edges. Though the number of induced subgraphs with an even number of edges is not expressible as a single weighted homomorphism count, computing this graph parameter is equivalent in complexity to computing $Z_H(G)$.

5.2.3 Indicator Function for Eulerian Property

A connected graph is **Eulerian** if it has an **Eulerian circuit**. An Eulerian circuit is a sequence of distinct edges joining vertices on a finite graph that visits every

edge exactly once and starts and ends at the same vertex. Euler proved that a connected graph has an Eulerian circuit if and only if every vertex has even degree. We now show how the number of homomorphisms to target graph H of Figure 5.10 is an indicator function for the existence of an Eulerian circuit.

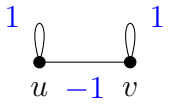


Figure 5.10: Target Graph H. The edge weights are given in blue.

As in the maximum cut example (Section 5.2.1), every homomorphism from G to H corresponds to a cut of G. The weight of the homomorphism will either be 1 or -1 depending on the number of cut-edges produced by the cut. The weight will be 1 if there is an even number of cut-edges, and -1 otherwise.

If G is Eulerian, then any homomorphism into H will have weight 1. This is because we can follow the Eulerian circuit in H, starting at either u or v, and we must cross the -1 edge an even number of times in order to return to the starting vertex. This corresponds to the homomorphism mapping an even number of edges of G to the edge $\{u, v\}$ in H, and thus, the weight of the homomorphism is equal to 1. The total number of homomorphisms from G the graph H is equal to 2^N, and so, for any Eulerian graph G,

$$Z_A(G) = 2^N.$$

Alternatively, fix some H-colouring of G, and let the vertices mapped to the vertex u form the partition A and the vertices mapped to v form the partition B. Suppose there are a edges in the subgraph induced by the vertices in A, b edges in the subgraph induced by the vertices in B, and s edges in the *cut* (edges connecting vertices of A and B). If G is Eulerian, every vertex has even degree, so

$$\sum_{v \in A} \deg(v) = \text{even}.$$

Note that the sum of the degrees in A is equal to two times the number of edges in A plus the number of cut edges

$$\sum_{v \in A} \deg(v) = 2a + s.$$

Hence, in order for $2a + s$ to be even, we must have s is even. In other words, the number of cut edges is even and therefore an even number of edges of G are mapped to the -1 edge in H. If G is Eulerian, then any homomorphism to H will be have weight 1.

Otherwise, suppose G is not Eulerian so it has some number of vertices of odd degree. Fix some H-colouring of G and suppose A has k vertices of odd degree and

B has ℓ vertices of odd degree. By the handshaking Lemma,

$$\sum_{v \in V} \deg(v) = 2|E|,$$

the number of odd degree vertices, $k + \ell$, must be even. Hence, either both k and ℓ or even or they are both odd. If both k and ℓ are even, then the sum of the degrees in A is even. As in the Eulerian case, we have $2a + s$ is even, and thus, there is an even number of edges in the cut. In this case, the homomorphism has weight 1. In the other case, k and ℓ are both odd, and

$$\sum_{v \in A} \deg(v) = \sum_{v \in A \text{ of even degree}} \deg(v) + \sum_{v \in A \text{ of odd degree}} \deg(v) = \text{odd}$$

hence $2a+s$ has to be odd, thus the number of edges in the cut is odd and the weight of the homomorphism is equal to -1. Now, the number of ways to partition $2n$ objects such that both partitions have an odd number of objects equals the number of ways to partition $2n$ objects such that both partitions have an even number of objects. Thus, the number of homomorphisms from G to H of weight 1 equals the number of homomorphisms of weight -1. Hence, when G is not Eulerian, we have

$$Z_A(G) = 0.$$

Therefore,

$$Z_A(G) = \begin{cases} 2^N & \text{if } G \text{ is Eulerian,} \\ 0 & \text{otherwise.} \end{cases}$$

5.2.4 Statistical Physics and Partition Functions

In statistical physics, counting weighted graph homomorphisms corresponds to solving an important quantity called the *partition function* of a system. By system we mean the collective behaviour of many units, and the partition function describes the distribution of (energy) states. In this section we explain what the partition function is in more detail and describe the weighted graph that corresponds to the partition function of a famous model called the *Ising model*. This correspondence is often quoted in papers but I haven't been able to find a source that explains it, so I have provided my own explanation here.

When a system is in thermal equilibrium, meaning there is no net heat flow, the *partition function* encodes its properties. The system's entropy, total energy, free energy, heat capacity can all be expressed in terms of the system's partition function and its derivative. Thus, knowing a system's partition function is incredibly valuable. Computing it, however, is not easy, since the partition function is a sum over all the possible states the system. More specifically, for any given system, the partition function Z is given by

$$Z = \sum_{\sigma} e^{-\beta E(\sigma)}$$

where the sum is over all possible states σ of the system, β is the inverse of the temperature, and $E(\sigma)$ is the energy of the system when it is in state σ.

Ising Model

The Ising model is a model of magnetism created by Lenz [22], in order to understand why magnets suddenly lose their magnetic properties when heated past a critical temperature. It is a simple model that, despite this, exhibits this interesting qualitative behaviour.

In the Ising model, particles are arranged in a lattice Λ and each particle has an intrinsic magnetic field called *spin*. If almost all the spins are aligned in the same direction, then the system is considered magnetic. The spins of each atom are constantly changing direction but not independently of each other, and the goal is to understand how the spins behave as you increase or decrease the temperature. The Ising model makes the simplifying assumption that each spin points either up or down and only neighbouring spins interact. Thus, each lattice site $k \in \Lambda$ has an associated discrete variable σ_k which equals $+1$ or -1 if it is an up spin or down spin respectively. A spin configuration $\sigma = \{\sigma_k\}_{k\in\Lambda}$ is as assignment of spin values to each lattice site.

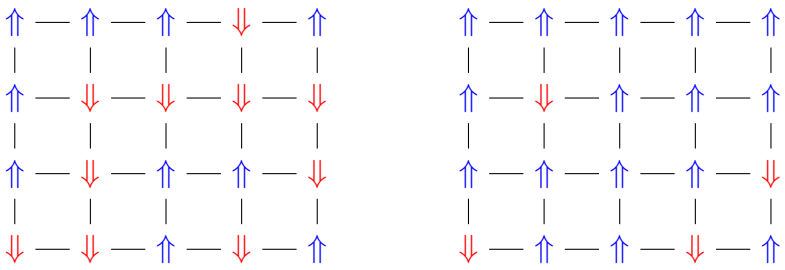


Figure 5.11: Two example configurations of the 2D Ising model. The edges represent neighbouring sites. The configuration on the left is non-magnetic and the configuration on the right is magnetic.

In this model, the energy of a spin configuration σ is

$$E(\sigma) = -J \sum_{\langle i,j \rangle} \sigma_i \sigma_j - h \sum_j \sigma_j$$

where the first sum is over all pairs of adjacent sites, J is real constant called the *interaction strength*, and h is a real constant dependent of the external magnetic field. Note that when two neighbouring spins i and j point in the same direction we have $\sigma_i \sigma_j = +1$. Hence, aligned spins lower the energy of the configuration. Physical systems prefer being in lower energy states and thus the Ising model is a model that favours alignment of spins and therefore magnetism. The partition function of the Ising model equals

$$Z = \sum_\sigma e^{J\beta \sum_{\langle i,j \rangle} \sigma_i \sigma_j + h\beta \sum_j \sigma_j}. \tag{5.2.1}$$

Each spin configuration corresponds to a homomorphism from the lattice sites of Λ to the graph H of Figure 5.12, where the up spins are mapped to the $+$ vertex and the down spins are mapped to the $-$ vertex. The weight of the homomorphism X will be

$$w_{A,D}(X) = (e^{J\beta})^{\alpha_+}(e^{-J\beta})^{\alpha_*}(e^{J\beta})^{\alpha_-}(e^{h\beta})^{n_+}(e^{-h\beta})^{n_-}$$

where α_+ is the number of neighbouring up spins, α_- is the number of neighbouring down spins, α_* is the number of neighbouring and differing spins, n_+ is the number of up spins, and n_- is the number of down spins. Note that

$$(e^{J\beta})^{\alpha_+}(e^{-J\beta})^{\alpha_*}(e^{J\beta})^{\alpha_-} = e^{J\beta(\alpha_+ + \alpha_- + \alpha_*)} = e^{J\beta\sum_{\langle i,j\rangle}\sigma_i\sigma_j}$$

and

$$(e^{h\beta})^{n_+}(e^{-h\beta})^{n_-} = e^{h\beta(n_+ - n_-)} = e^{h\beta\sum_j \sigma_j}$$

Hence, each term in the summand in (5.2.1) is equal to the weight of a homomorphism from the lattice Λ to the graph H of Figure 5.12. Hence, the partition function of the Ising model equals $Z_{A,D}(G)$ where A and D are the weight matrices of H.

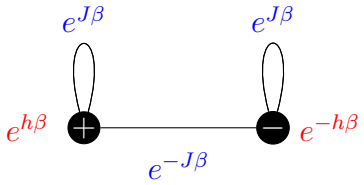
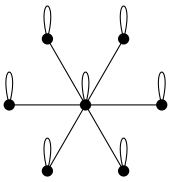


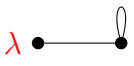
Figure 5.12: The graph H corresponding to the Ising model. The edge weights are given in blue and the vertex weights are given in red.

Other Models

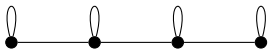
Other examples of the correspondence between weighted graph homomorphisms and statistical physics models are given in Figure 5.13. References for these can be found in [36], [2], and [4].



(a) 6-particle Widom-Rowlinson model.



(b) Hardcore lattice gas model, where λ is a vertex weight > 0.



(c) Beach model.

Figure 5.13: Models of statistical physics and their corresponding graphs. The unweighted edges have weight 1.

5.3 Quantum Graphs

Lastly, we extend the definition of homomorphism count to **quantum graphs** which are linear combinations of graphs over non-zero rational numbers. The restriction to the rational numbers is done because not all real numbers can be computable-meaning their decimal expansion can be determined by an algorithm to any degree of accuracy. Rational numbers are computable however.

A **quantum graph** is defined as a formal linear combination of a finite number of graphs with nonzero rational coefficients. Hence, for a quantum graph $\overline{H}$,

$$\overline{H} = \sum_{H \in \mathcal{C}} \alpha_H H$$

for a finite set of constituent graphs $\mathcal{C}$ and coefficients $\alpha_H \in \mathbb{Q} \setminus \{0\}$.

The definition of homomorphism count is extended linearly to quantum graphs, where for arbitrary graph G and quantum graph $\overline{H} = \sum_{H \in C} \alpha_H H$, we have

$$\#\mathrm{Hom}(G, \overline{H}) = \sum_{H \in C} \alpha_H \cdot \#\mathrm{Hom}(G, H).$$

This concept is useful because some graph parameters are not expressible as a single $Z_{A,D}$ for some fixed graph H with weight matrix A and vertex matrix D. For example, in the paper *The Exponential-Time Complexity of Counting Quantum Graph Homomorphisms* [6], the authors note that the number of surjective homomorphisms onto K_3, denoted $\#\mathrm{VertSurj}(G, K_3)$, is expressible as a linear combination of graph homomorphism counts:

$$\mathrm{VertSurj}(G, K_3) = \#\mathrm{Hom}(G, K_3) - 3\#\mathrm{Hom}(G, K_2) + 3\#\mathrm{Hom}(G, K_1).$$

This is due to the inclusion-exclusion principle. A rich class of graph parameters are expressible by counting the number of homomorphisms to quantum graphs. However, not all graph parameters can be expressed in this framework. For example, the number of perfect matchings has no such representation [27]. Theorem 6.30 in *Large Networks and Graph Limits* [27] gives precise conditions for when this is possible.

Chapter 6

The Complexity of Counting Graph Homomorphisms

In this chapter we will explore when the problem of counting graph homomorphisms to a fixed graph H is #P-complete, where H is a looped-simple graph. Hell and Nešetřil [20] proved a complexity dichotomy theorem for deciding whether a homomorphism from an input graph G to a fixed looped-simple graph H exists. Their result stated that deciding if a homomorphism from G to H exists is NP-complete unless H is bipartite or H has a loop, in which cases the problem is in P. An analogous dichotomy theorem of counting homomorphisms to a fixed looped-simple graphs H was given by Dyer and Greenhill [12]. We present a simplified proof of the counting homomorphisms problem by Chen, Curticapean, and Dell [6]. Missing details of the proof are filled in using results from the original paper by Dyer and Greenhill [12]. The #P-completeness of computing certain partition functions in statistical physics and graph parameters follow as corollaries.

6.1 Preliminaries and Terminology

For a fixed graph H, we define $\#\mathrm{Hom}(\,\cdot\,, H)$ to be the computational problem of counting the number of homomorphisms from an arbitrary graph G to H.

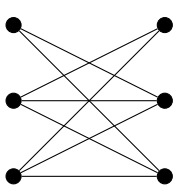
Name.	$\#\mathrm{Hom}(\,\cdot\,, H)$
Instance.	A graph G.
Output.	The number of H-colourings of G that is, $\#\mathrm{Hom}(G, H)$.

The full complexity classification of $\#\mathrm{Hom}(\,\cdot\,, H)$ by Dyer and Greenhill [12]:

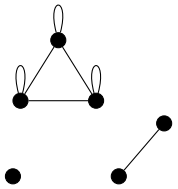
Theorem 6.1.1 (Dyer & Greenhill [12])**.** *Let H be a fixed graph. The problem of counting the number of homomorphisms from a graph G to H, $\#\mathrm{Hom}(G, H)$, is #P-complete if H has a connected component which is not a complete reflexive graph or a complete irreflexive bipartite graph. Otherwise, the problem is in FP.*

We will consider an isolated vertex v with no loop to be a complete bipartite graph with vertex bipartition $v \cup \{\emptyset\}$, as done in the Dyer and Greenhill paper [12]. Thus, isolated vertices satisfy the FP conditions of Theorem 6.1.1. We describe graphs which are complete reflexive graphs or complete irreflexive graphs as **hom-easy**, and all other types of graphs as **hom-hard**.

The paper by Dyer and Greenhill [12] proving Theorem 6.1.1 consists of three parts:



(a) Hom-hard graph.



(b) Hom-easy graph.

Figure 6.1: Examples of graphs for which #Hom($\cdot$, H) are in different complexity classes.

- The first part of the paper proves a set of complexity results concerning the computation of $Z_A(G)$ and $Z_{A,D}(G)$ where A and D are weight matrices of a fixed graph H. This was done using a combination of polynomial interpolation, applying stretchings and thickenings to graphs, and spectral methods.
- The second part proves that if one of the connected components of H is hom-hard, then #Hom($\cdot$, H) is #P-complete.
- The last part proves Theorem 6.1.1 by considering whether H is bipartite or not and whether H has self-loops or not. In each case, the reduction was to the problem of counting the independent sets of a graph or counting the number of 3-colourings of a graph, which are #P-complete problems. Here, we can treat H as connected, thanks to the second part, and the results of the first part are repeatedly used to do the reductions.

Chen, Curticapean, and Dell [6] sketched a new and shortened proof of Theorem 6.1.1, which consisted in a series of reductions from #Hom($\cdot$, H) to the problem of counting independent sets in a graph. As discussed in Chapter 4, the problem of the counting independent sets of a graph, #IS, is #P-complete. The simplifications of the new proof [6] were the following:

- They simplified the reduction from a general graph H to a single connected component of H, and they extended this result, proving a dichotomy theorem for homomorphisms to quantum graphs.
- They removed the need to consider bipartite graphs separately by showing that the complexity of #Hom($\cdot$, H) is equivalent to the complexity of counting homomorphisms to the *double bipartite cover* of H.

We now present some definitions and prove three identities about counting homomorphisms we will use throughout the proof of Theorem 6.1.1.

Let H and F be disjoint graphs. The **tensor product** of H with F, denoted $H \otimes F$, is the graph on vertex set $V(H) \times V(F)$ where (u, v) and (u', v') are adjacent if and only if $(u, u') \in E(H)$ and $(v, v') \in E(F)$. It is a commutative and associative operation for unlabelled graphs. The adjacency matrix of $H \otimes F$ is the Kronecker product of the respective adjacency matrices of H and F (see definition in Chapter 2). The tensor product of an arbitrary graph H and K_2, $H \otimes K_2$, is called the **bipartite double cover** of H. The bipartite double cover of any graph is always bipartite, as in Figure 6.2.

Proposition 6.1.2. *The tensor product $H \otimes K_2$ for an arbitrary graph H is always bipartite. Moreover, if H is connected, then its bipartite double cover is, too.*

Proof. Suppose the vertices of K_2 are v and v'. Let V_1 and V_2 be subsets of the vertex set of $H \otimes K_2$ defined by $V_1 = \{(u, v) \mid u \in G\}$ and $V_2 = \{(u, v') \mid u \in G\}$. The subgraph induced by V_1 has no edges since v is not adjacent to itself in K_2. Similarly, the subgraph induced by V_2 has no edges. Thus, $H \otimes K_2$ is bipartite graph with vertex bipartition V_1 and V_2. □

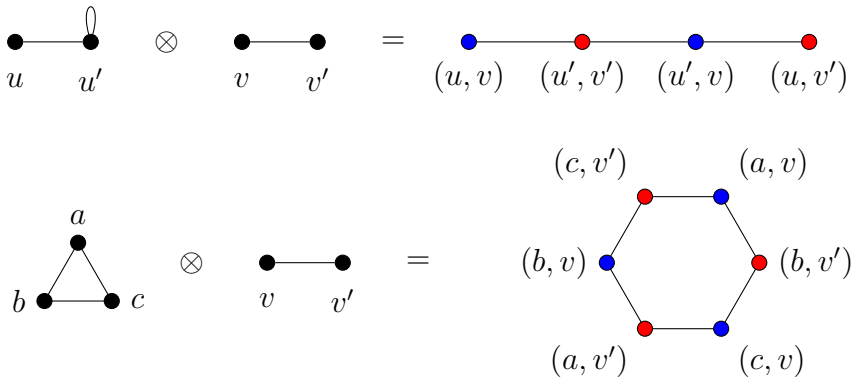


Figure 6.2: Tensor product of two different graphs with K_2. The red and blue vertices of the tensored graph form the two partitions.

For disjoint graphs A and B, we let the disjoint union $A \sqcup B$ be the graph with vertex set $V(A) \sqcup V(B)$ which consists of one copy of A and one of B.

Lemma 6.1.3. *Let G, F, A, B be disjoint graphs. The following identities hold:*

(i) $\#\mathrm{Hom}(G \sqcup F, A) = \#\mathrm{Hom}(G, A) \cdot \#\mathrm{Hom}(F, A)$,
(ii) $\#\mathrm{Hom}(G, A \otimes B) = \#\mathrm{Hom}(G, A) \cdot \#\mathrm{Hom}(G, B)$,
(iii) *Additionally, if G is connected we have* $\#\mathrm{Hom}(G, A \sqcup B) = \#\mathrm{Hom}(G, A) + \#\mathrm{Hom}(G, B)$.

Proof. (i) A homomorphism $h : G \sqcup F \to A$ defines the unique homomorphisms $g : G \to A$ and $f : F \to A$, where g and f are equal to h restricted to G and F respectively. Likewise, homomorphisms $g : G \to A$ and $f : F \to A$ define a unique homomorphism $h : G \sqcup F \to A$. Hence, counting the number of homomorphisms from $G \sqcup F$ to A equals product of the number of homomorphism to A from the disjoint components.

(ii) A homomorphism $f : G \to A \otimes B$ can be described in terms of two unique functions $g : G \to A$ and $h : G \to B$ where $f(u) = (g(u), h(u))$ for $u \in G$. Since f is a homomorphism, for all $(u, v) \in E(G)$ we have

$$((g(u), h(u)), (g(v), h(v)) \in A \otimes B.$$

By definition of the tensor graph, if $((g(u), h(u)), (g(v), h(v))$ is an edge of $A \otimes B$, then $g(u)$ is adjacent to $g(v)$ in A and $h(u)$ is adjacent to $h(v)$ in B. Thus, for $(u, v) \in E(G)$, we have $(g(u), g(v)) \in E(A)$ and $(h(u), h(v)) \in$

$E(B)$. Hence, the maps g and h are homomorphisms. Similarly, if we have homomorphisms $g : G \to A$ and $h : G \to B$, then function $f : G \to A \otimes B$ defined by $f = (g, h)$ is a homomorphism too.

(iii) Since G is connected, all the edges of G must be mapped to a single connected component in order to preserve the vertex adjacency relation. Thus, the total number of homomorphisms to $A \sqcup B$ is equal to the number of homomorphisms to A plus the number of homomorphisms to B.

□

6.2 Complexity of Quantum Graph Homomorphisms

We now present Chen, Curticapean, and Dell's [6] complexity result about quantum graphs, which says counting homomorphisms to a quantum graph is as hard as counting homomorphisms to any of its constituent components. They call it the *complexity monotonicity* property and it will allow us to restrict our attention to graphs that are connected. The proof of the complexity monotonicity property relies on the following lemma:

Lemma 6.2.1. *Let $H_1, \ldots, H_k$ be a set of pairwise non-isomorphic graphs. Then there exist irreflexive graphs $F_1, \ldots, F_k$ such that the $k \times k$ matrix M with $M_{ij} = \#\mathrm{Hom}(F_i, H_j)$ is invertible.*

The proof of this lemma can be found in the appendix of [6]. We omit the proof but comment on an important detail left out by the authors that will be relevant to their complexity monotonicity proof. The graphs $F_1, \ldots, F_k$ of Lemma 6.2.1 can be explicitly constructed in polynomial time given the H_is. The F_is are taken from a set S which contains the H_is, is closed under surjective homomorphisms, such that all graphs in S are non-isomorphic. The "closed under surjective homomorphisms" condition means that if a graph H_i surjectively maps onto a graph H', then $H' \in S$. Though the proof does not state how to find such S in polynomial time, we explain why we are allowed to do this. Note any surjective homomorphism must always map from a graph onto a graph of the same or smaller size. Since each graph H_i is bounded in size, all graphs with the same number or fewer vertices than H_i can be enumerated in constant time (this constant will most likely be very large). Then, from all the subgraphs obtained we can remove extra copies of isomorphic graphs and let S equal the remaining set. Again, since all graphs considered are of bounded size, checking if two graphs are isomorphic can be done in polynomial time, even though checking for graph isomorphisms is not known to be polynomial-time solvable. Given S, we can explicitly find F_is following the rest of the proof of the lemma. Lastly, since the F_is are bounded in size, counting the number of homomorphisms from the H_is to the F_is can be done in polynomial time.

Proposition 6.2.2 (Complexity Monotonicity, Chen, Curticapean, Dell [6])**.** *Let $\overline{H}$ be the quantum graph*

$$\overline{H} = \sum_{j=1}^{k} \alpha_j H_j$$

with pairwise non-isomorphic graphs $H_1, \ldots, H_k$ and coefficients $\alpha_1 \ldots \alpha_k \in \mathbb{Q} \setminus \{0\}$. For every fixed $j \in [k]$, we then have

$$\#\mathrm{Hom}(\,\cdot\,, H_j) \leq \#\mathrm{Hom}(\,\cdot\,, \overline{H}).$$

Proof. Without loss of generality, assume $j = 1$. By Lemma 6.2.1, there exist irreflexive graphs $F_1, \ldots, F_k$ such that the matrix M with $M_{ij} = \#\mathrm{Hom}(F_i, H_j)$ is invertible. By the above discussion, we can explicitly find the F_is and compute the matrix M in polynomial time. On input graph G, we can construct the graphs $G \sqcup F_i$ for all $i \in [k]$ in polynomial time. Then

$$\begin{aligned}
\#\mathrm{Hom}(G \sqcup F_i, \overline{H}) &= \sum_{j=1}^{k} \alpha_j \#\mathrm{Hom}(G \sqcup F_i, H_j) \\
&= \sum_{j=1}^{k} \alpha_j \#\mathrm{Hom}(G, H_j) \#\mathrm{Hom}(F_i, H_j) \\
&= \sum_{j=1}^{k} \alpha_j \#\mathrm{Hom}(G, H_j) M_{ij}
\end{aligned}$$

where the first equality is by definition and the second equality is by Lemma 6.1.3. These equations form a system of linear equations $b = Mx$ where $b_i = \#\mathrm{Hom}(G \sqcup F_i, \overline{H})$ and $x_i = \alpha_i \#\mathrm{Hom}(G, H_i)$. Given an oracle for $\#\mathrm{Hom}(\,\cdot\,, \overline{H})$, we can compute $\#\mathrm{Hom}(G \sqcup F_i, \overline{H})$ for each $i \in [k]$ in polynomial time and by Lemma 6.2.1 we may invert the matrix M. Thus, solve for $\#\mathrm{Hom}(G, H_1)$ since $\alpha_i \neq 0$ in polynomial time. □

The authors of [6] mention that their main contribution in computing the #P-hardness of $\mathrm{Hom}(\,\cdot\,, \overline{H})$ comes from allowing the rational coefficients of $\overline{H}$ to be negative. They claim that #P-hardness of $\mathrm{Hom}(\,\cdot\,, \overline{H})$ is immediate if all the coefficients are positive nonzero rationals, just from the original statement of the theorem by Dyer and Greenhill.

Let us fill in the details. Assume that $\overline{H}$ is a quantum graph

$$\overline{H} = \alpha_1 H_1 + \alpha_2 H_2 + \cdots \alpha_k H_k,$$

where $H_1, \ldots, H_k$ are pairwise non-isomorphic graphs and $\alpha_1, \ldots, \alpha_k \in \mathbb{Q}^+ \setminus \{0\}$. By definition, the number of homomorphisms into the quantum graph $\overline{H}$ is equal to

$$\#\mathrm{Hom}(G, \overline{H}) = \sum_{i=1}^{k} \alpha_i \#\mathrm{Hom}(G, H_i).$$

Since the coefficients are rational numbers, we can assume $\alpha_i = p_i/q_i$ for each $i \in [k]$ where $p_i, q_i \in \mathbb{Z}^+$. Let $\beta = \prod_{i=1}^{k} q_i$ and $\beta_i = \alpha_i \beta$ for each $i \in [k]$. Note that each β_i

is a positive integer, hence multiplying the above equation by β will make all the coefficients positive integers. We have

$$\beta\#\text{Hom}(G, \overline{H}) = \sum_{i=1}^{k} \beta_i \#\text{Hom}(G, H_i).$$

The sum on the right-hand side can be reinterpreted as homomorphism count to a single graph. For each $i \in [k]$, the number of homomorphisms to the graph consisting of β_i disjoint copies of H_i is equal to $\beta_i \text{Hom}(G, H_i)$ by the identity of Lemma 6.1.3. Hence, the sum of $\beta_i \text{Hom}(G, H_i)$ over all $i \in [k]$ is equal to the number of homomorphisms into the graph H' consisting of β_i disjoint copies of H_i for each $i \in [k]$. Dyer and Greenhill showed that if any of the connected component is #P-complete then the whole problem is #P-complete. Thus, if any H_i is #P-complete for some $i \in [k]$, then computing $\sum_{i=1}^{k} \beta_i \text{Hom}(G, H_i)$ is #P-complete. Clearly, if computing this sum is #P-complete, then so is $\text{Hom}(G, \overline{H})$. Otherwise, compute right-hand side sum and divide by β to obtain $\#\text{Hom}(G, \overline{H})$. Although this is not explicitly mentioned by the authors, if all the coefficients are negative rationals, then #P-hardness of $\text{Hom}(\,\cdot\,, \overline{H})$ can be deduced directly from the Dyer and Greenhill theorem also. We cannot do this, however, when we have a mixture of positive and negative coefficients.

An important detail left out by the authors is that the complex monotonicity property of graphs implies that we can assume H is a connected graph when proving the complexity of $\#\text{Hom}(\,\cdot\,, H)$. This is because the number of homomorphisms to a graph equals the number of homomorphism to a quantum graph with coefficients equal to 1 and the constituent graphs correspond to the disjoint connected components of the original graph.

Corollary 6.2.3. *If H is a graph with connected components $H_1, \cdots H_k$, then*

$$\#\text{Hom}(\,\cdot\,, H_j) \leq \#\text{Hom}(\,\cdot\,, H)$$

for every $j \in [k]$.

6.3 Collapsing Vertices

In this section, we cover a missing element needed to complete the proof by Chen, Curticapean, and Dell that is found in the original proof [12]: a graph H could have vertices that share the same neighbourhood, but we will show how one can "collapse" such vertices into a single vertex, thereby creating a new graph H', such that the problems $\#\text{Hom}(\,\cdot\,, H')$ and $\#\text{Hom}(\,\cdot\,, H)$ share the same complexity.

For any unweighted graph H, its adjacency matrix A will have either identical column or row vectors if vertices in H share the same neighbourhood, or a vector of zeroes if a vertex in H is isolated. Dyer and Greenhill state the following:

Lemma 6.3.1 (Dyer & Greenhill [12])**.** *Let A be a symmetric 0-1 matrix which has at least one pair of linearly dependent columns. Then there exists a symmetric 0-1 matrix A' with no two linearly dependent columns, and a positive diagonal matrix*

D, such that the problems EVAL(A', D) *and* EVAL(A) *are equivalent. Moreover, the matrices A' and D can be constructed from A in constant time.*

Proof. Let H be the graph with adjacency matrix A. Then H either has an isolated vertex or a pair of vertices with the same neighbourhood structure. Let H' be the graph constructed from H as follows:

1. Delete all isolated vertices of H.
2. If $\{v_1, \ldots, v_\ell\} \in V(H)$ have the same set of neighbours as each other, then delete $\{v_2, \ldots, v_\ell\}$ from H. Repeat this step until no two vertices have the same set of neighbours as each other.

Let A' be the adjacency matrix of H'. Step 1 removed all the zero rows and columns of A and step 2 removed all the repeated rows and columns of A. Let D be the diagonal matrix with the same number of rows and columns as A', such that D_{ii} equals λ_i, where λ_i is the number of vertices in H which have the same neighbours as the ith vertex of H' (considered as a vertex in H). The idea is that the number of homomorphisms to H' has decreased due to the removal of vertices and edges, but this is countered by adding weights to the vertices. Suppose $\{v_1, \ldots, v_\ell\} \in V(H)$ are vertices with the same set of neighbours in H and v is the "collapsed" vertex in H'. Then for every H'-colouring of G that maps r vertices of G to v, there are ℓ^r distinct H-colourings of G that map the vertices originally mapped to v to one of the v_is in $\{v_1, \ldots, v_\ell\}$ and all other vertices are mapped according to the H'-colouring. By definition, we have

$$\begin{aligned} Z_A(G) &= \sum_{X \in \Omega_H(G)} \prod_{\{v,w\} \in E} A_{X(v),X(w)} = \sum_{X \in \Omega_H(G)} 1 = |\Omega_H(G)| \\ Z_{A',D}(G) &= \sum_{X \in \Omega_{H'}(G)} \prod_{\{v,w\} \in E} A'_{X(v),X(w)} \prod_{v \in V} \lambda_{X(v)} \\ &= \sum_{X \in \Omega_{H'}(G)} \prod_{v \in V} \lambda_{X(v)}. \end{aligned}$$

Hence, it is clear that $Z_A(G)$ and $Z_{A',D}(G)$ are equivalent. How the reductions apply to a graph are illustrated in Figure 6.3. □

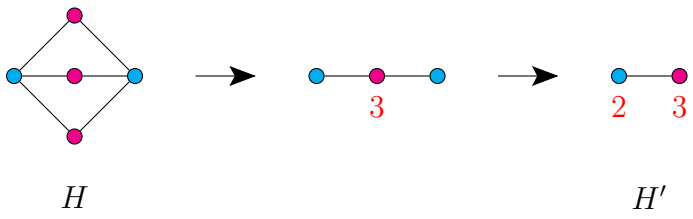


Figure 6.3: The weighted number of homomorphisms to H and H' are equal by Lemma 6.3.1. The vertices with the same colours share the same neighbourhood and the vertex weights are given in red below the vertices.

Theorem 6.3.2 (Dyer & Greenhill [12]). *Let A be a symmetric matrix with no two linearly dependent columns. There is a polynomial-time reduction from* EVAL(A) *to* EVAL(A, D)*, for any diagonal matrix D of positive weights.*

The proof of this theorem is a bit involved because it relies on several other results. We discuss the main ideas without going through the details. Given an instance G of EVAL(A), the proof constructs an instance Γ of EVAL(A, D), where the edges of Γ are partitioned into disjoint sets E' and E''. To construct Γ, all the vertices of G are modified so that they all become the endpoint of exactly two edges in E''. Then, by applying the stretching and thickening operation to the edges in E'', a set $\{S_2 T_p S_r'' \Gamma \mid 1 \leq r \leq (n+1)^k\}$ of new graphs are constructed in polynomial time from Γ (see definition of stretching and thickenings in Section 3.3.3), where p is chosen large enough to satisfy some inequality. By applying other lemmas and computation, you can write the number of weighted homomorphisms to the new and stretched graphs as sum over all functions from Γ to H. Then

$$Z_{A,D}(S_2 T_p S_r'' \Gamma) = \sum_{X : V_\Gamma \mapsto C} w'_A(X) w''_{\sigma_r B}(X).$$

It turns out that the right-hand side equals $Z_A(G)$.

Corollary 6.3.3 (Dyer & Greenhill [12]). *Let H be a graph and let H' be the graph obtained from H by replacing all vertices with the same neighbourhood structure by a single vertex (with that neighbourhood structure). If* $\#\,\mathrm{Hom}(\,\cdot\,, H')$ *is* #P*-complete then so is* $\#\mathrm{Hom}(\,\cdot\,, H)$.

Proof. The graph H' described above is the one used in Lemma 6.3.1. Let A, A' be the adjacency matrices of H and H', respectively. Then EVAL(A', D) and EVAL(A) are equivalent for some diagonal matrix D of positive vertex weights, as in Lemma 6.3.1. Moreover, there is a polynomial-time reduction from EVAL(A') to EVAL(A', D), by Theorem 6.3.2. This completes the proof. □

6.4 The Main Proof

We now provide the proof of Theorem 6.1.1. We start by considering homomorphisms to hom-easy graphs and then to hom-hard graphs. For the rest of the chapter, we assume G is connected by Lemma 6.1.3 and we assume H is connected by Corollary 6.2.3.

6.4.1 Counting to Hom-Easy Graphs

Proposition 6.4.1. *If H is a reflexive complete graph, then* $\#\mathrm{Hom}(\,\cdot\,, H)$ *can be computed in polynomial time.*

Proof. Since H is reflexive and complete, every vertex in H is adjacent to every other vertex in H. So every mapping $f : V(G) \to V(H)$ is a graph homomorphism. Hence, the number of homomorphisms to H is equal to the number of mappings from G to H. Thus, $\#\mathrm{Hom}(G, H) = |V(H)|^{|V(G)|}$. □

Proposition 6.4.2. *If H is an irreflexive complete bipartite graph, then $\#\mathrm{Hom}(\,\cdot\,, H)$ can be computed in polynomial time.*

Proof. Recall that we consider an isolated vertex with no loop to be a complete bipartite graph. If H is an isolated vertex, then H-colourings of G do not allow vertices in G to be adjacent. Thus, if H is an isolated vertex without a loop then $\Omega_H(G)$ is empty unless G is an isolated vertex, in which case $\#\mathrm{Hom}(G, H) = 1$.

Otherwise, suppose H is a complete irreflexive bipartite graph with vertex bipartition $C_1 \cup C_2$, where $|C_i| = k_i$ for $i = 1, 2$. If G is not bipartite then $\Omega_H(G)$ is empty, since any H-colouring of G would give a bipartition of the vertices of G. In the case that G is bipartite, suppose the vertex bipartition of G is $V_1 \cup V_2$, where $|V_i| = n_i$ for $i = 1, 2$. A homomorphism from G to H must map the vertices of V_1 to either L or R, and the vertices of V_2 to the opposite part. There are $k_i^{n_j}$ possible mappings from V_j to C_i for i. Hence, the total number of homomorphisms from G to H that map V_1 to C_1 equals $k_1^{n_1} k_2^{n_2}$, and the number of homomorphisms that map V_1 to C_2 equals $k_1^{n_2} k_2^{n_1}$. Altogether, we have

$$\#\mathrm{Hom}(G, H) = k_1^{n_1} k_2^{n_2} + k_1^{n_2} k_2^{n_1}.$$

Hence, the total number of homomorphisms is polynomial-time computable. □

6.4.2 Counting to Hom-Hard Graphs

The proof by Chen, Curticapean, and Dell [6] consists of a series of steps which in the end show #IS (defined in Chapter 4) reduces to $\#\mathrm{Hom}(\,\cdot\,, H)$ for any hom-hard graph H. The first step proves the complexity of counting homomorphisms to a graph H is equivalent to the complexity of counting homomorphisms to its bipartite double cover $H \otimes K_2$:

$$\#\mathrm{Hom}(\,\cdot\,, H \otimes K_2) \leq \#\mathrm{Hom}(\,\cdot\,, H) \text{ and } \#\mathrm{Hom}(\,\cdot\,, H) \leq \#\mathrm{Hom}(\,\cdot\,, H \otimes K_2).$$

This step allows us to assume the target graph H is always bipartite by Proposition 6.1.2. The second step proves the problem $\#\mathrm{Hom}(\,\cdot\,, B_v)$ reduces to the problem $\#\mathrm{Hom}(\,\cdot\,, B)$ where B is any bipartite graph and B_v is the subgraph induced by vertices at most distance 2 from v in B:

$$\#\mathrm{Hom}(\,\cdot\,, B_v) \leq \#\mathrm{Hom}(\,\cdot\,, B).$$

The result is proved using the property of complexity monotonicity of quantum graphs. Since B_v is always bipartite, this reduction can be iteratively applied by setting B to B_v. This is done until we can no longer achieve a smaller hom-hard graph by applying the reduction. We call this hom-hard base case an *impasse*. The third step of the proof shows that the impasse P is always an "exploded four-vertex path" $P(a_1, a_2, a_3, a_4)$, where a_1, a_2, a_3, a_4 are positive integers. The exploded four-vertex path $P(a_1, a_2, a_3, a_4)$ consists of a graph F where the vertices of F are partitioned into four disjoint sets $V(F) = V_1 \cup V_2 \cup V_3 \cup V_4$ where $|V_i| = a_i$ for $i \in [4]$, and $E(F) = \{\{v_i, v_j\} \mid 1 \leq i < j \leq 4,\ j - i = 1,\ v_i \in V_i,\ v_j \in V_j\}$.
The fourth and last step proves the hardness of the impasse $\#\mathrm{Hom}(\,\cdot\,, P)$. The

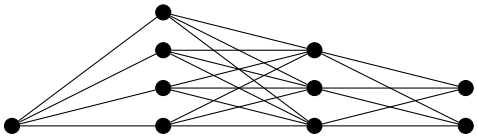


Figure 6.4: Exploded four-vertex path $P(1, 4, 3, 2)$

authors of [6] did not provide details for this reduction other than the fact that counting independent sets reduces to this problem. We use Corollary 6.3.3 of the Dyer and Greenhill paper [12] to prove #IS reduces to #Hom($\cdot$, P) where P is an impasse of the form $P(a_1, a_2, a_3, a_4)$ of arbitrary fixed positive integers a_1, a_2, a_3, a_4. Though these chains of reductions, we obtain that for any hom-hard graph H, the problem #Hom($\cdot$, H) is #P-complete:

$$\#\text{IS} \leq \#\text{Hom}(\,\cdot\,, P) \leq \cdots \leq \#\text{Hom}(\,\cdot\,, H \otimes K_2) \leq \#\text{Hom}(\,\cdot\,, H).$$

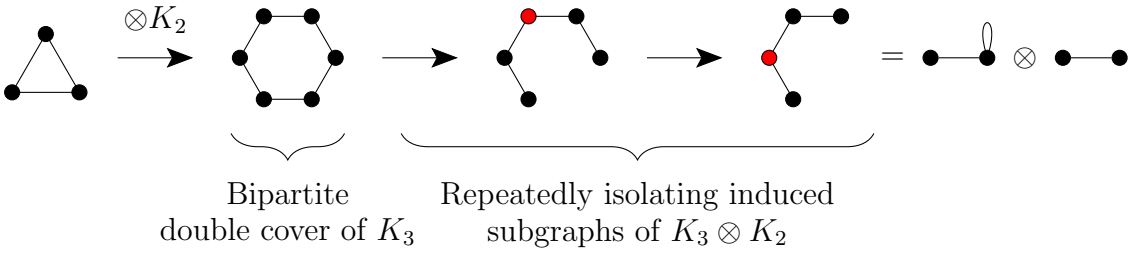


Figure 6.5: Example of how the reductions apply to K_3. The red vertices highlight the vertex v from which we reduce to B_v.

Proposition 6.4.3. *The problems* #Hom($\cdot$, H) *and* #Hom($\cdot$, $H \otimes K_2$) *are equivalent under polynomial-time reductions.*

Proof. By Lemma 6.1.3, we have that for any graphs G and H,

$$\#\text{Hom}(G, H \otimes K_2) = \#\text{Hom}(G, H) \cdot \#\text{Hom}(G, K_2).$$

By Proposition 6.4.2, $\text{Hom}(G, K_2)$ can be calculated in polynomial time. □

We remark that if a graph has a tensor product decomposition of hom-easy graphs, is itself a hom-easy graph.

Recall that for a given bipartite graph B and $v \in V(B)$, we let B_v denote the subgraph induced by vertices of distance at most 2 from v. For the next lemma, we let G^a_L be the graph derived from a bipartite graph G with vertex bipartition $V(G) = L \cup R$ by adding an "apex" vertex a that is adjacent to all of L. We similarly derive G^a_R by adding an apex vertex a adjacent to all of R. These two graphs G^a_L and G^a_R can always be constructed in polynomial time.

Figure 6.6: Example of G_L^a and G_R^a

Lemma 6.4.4. *Let B be a bipartite graph and let G be a connected bipartite graph with bipartition $V(G) = L \cup R$. Then*

$$\#\mathrm{Hom}(G_L^a, B) + \#\mathrm{Hom}(G_R^a, B) = \sum_{v \in V(B)} \#\mathrm{Hom}(G, B_v). \tag{6.4.1}$$

Proof. Homomorphisms from G_L^a to B can be partitioned by what they map the apex vertex a to in B. For every $v \in V(B)$, we denote the number of homomorphisms from G_L^a to B that map a to v by $\#\mathrm{Hom}(G_L^a, B \mid a \mapsto v)$. Then,

$$\#\mathrm{Hom}(G_L^a, B) = \sum_{v \in V(B)} \#\mathrm{Hom}(G_L^a, B \mid a \mapsto v).$$

The same applies to homomorphisms from G_R^a to B. Hence,

$$\#\mathrm{Hom}(G_L^a, B) + \#\mathrm{Hom}(G_R^a, B) = \sum_{v \in V(B)} \Big(\#\mathrm{Hom}(G_L^a, B \mid a \mapsto v) + \#\mathrm{Hom}(G_R^a, B \mid a \mapsto v) \Big). \tag{6.4.2}$$

If the image of a is fixed, then the neighbours of a, so all the vertices in L, have to be mapped to vertices at most distance 1 from the image of a. All remaining vertices of G_L^a are the ones in R, which must be mapped to vertices at most distance 2 from the image of a.

Fix any vertex $v \in V(B)$. We say that a homomorphism h from G_L^a to G_R^a to B is an *extension* of a homomorphism g from G to B_v if h agrees with g on all the vertices of G, and h also maps the additional vertex a to v.

Any homomorphism $h : G_L^a \to B$ that maps a to $v \in B$ is an extension of a homomorphism $g : G \to B$. This can be easily seen by restricting h to G and noting that since homomorphisms must preserve the adjacency relation of the vertices, the image of G must be contained in the subgraph induced by the vertices at most distance two from the image of a, which is precisely B_v.

For every homomorphism $g : G \to B_v$, there exists exactly one homomorphism h from either G_L^a or G_R^a to B that is an extension of g. Let X be the set of the bipartition of B_v not containing v. In other words, let X be the neighbourhood of v in B_v. A homomorphism $g : G \to B_v$ must either map the whole of L to X or the whole of R to X, since G is connected and bipartite. In the first case, g maps L to

X. Then g can be extended to the map $h : G_L^a \to B$, where h acts the same on the vertices of G and maps a to v in B. Since every vertex of G_L^a is at most distance two from a, then the image of h will is the subgraph B_v. All other maps from G_L^a to B that, restricted to G, coincide with g, will not be homomorphisms: if they map v to R then that would force R be mapped to X. In the case that g maps X to R, a similar argument shows there is exactly one extension $h : G_R^a \to B$ of g.

Thus, we have

$$\#\mathrm{Hom}(G_L^a, B \mid a \mapsto v) + \#\mathrm{Hom}(G_R^a, B \mid a \mapsto v) = \#\mathrm{Hom}(G, B_v). \tag{6.4.3}$$

Summing equation 6.4.3 over all vertices in B yields equation 6.4.1. □

Proposition 6.4.5. *For every bipartite graph B and every vertex $v \in V(B)$, the problem* $\#\mathrm{Hom}(\,\cdot\,, B_v)$ polynomial-time reduces to $\#\mathrm{Hom}(\,\cdot\,, B)$.

Proof. Let G be an instance of $\#\mathrm{Hom}(\,\cdot\,, B_v)$. By proposition 6.4.3 we can assume that G bipartite with $V(G) = L \cup R$. Let $\overline{B}$ be the quantum graph

$$\overline{B} = \sum_{v \in V(B)} B_v.$$

By the complexity monotonicity property (Proposition 6.2.2) we have that for any $v \in V(B)$

$$\#\mathrm{Hom}(\,\cdot\,, B_v) \leq \#\mathrm{Hom}(\,\cdot\,, \overline{B}).$$

The number of homomorphisms to $\overline{B}$ is equal to the right-hand side of equation 6.4.1. By Lemma 6.4.4, we have

$$\#\mathrm{Hom}(G_L^a, B) + \#\mathrm{Hom}(G_R^a, B) = \#\mathrm{Hom}(G, \overline{B}).$$

Having an oracle for $\#\mathrm{Hom}(\,\cdot\,, B_v)$, we can compute the left-hand side in polynomial time. And thus, $\#\mathrm{Hom}(\,\cdot\,, \overline{B}) \leq \#\mathrm{Hom}(\,\cdot\,, B)$. By transitivity of reductions, we have $\#\mathrm{Hom}(\,\cdot\,, B_v)$ polynomial-time reduces to $\#\mathrm{Hom}(\,\cdot\,, B)$. □

Lemma 6.4.6. *Let B be a connected bipartite graph, but not a complete one. If for every $v \in V(B)$ the graph B_v is either complete bipartite or equal to B, then B is isomorphic to $P(a_1, a_2, a_3, a_4)$ for positive integers a_1, a_2, a_3, a_4.*

Proof. Let $V(B) = L \cup R$ be a bipartition of B. Given B, we define the set S to be the set of all $v \in V(B)$ such that B_v is equal to B, and T is the set of all $v \in V(B)$ such that B_v is a complete bipartite graph.

Let $A_1 = L \cap T$, $A_2 = R \cap S$, $A_3 = L \cap S$, and $A_4 = R \cap T$. We claim B is the exploded four vertex path $P(|A_1|, |A_2|, |A_3|, |A_4|)$. Since of L and R form a bipartition of G, each A_i is an independent set. There are no edges between the sets A_1 and A_3, since A_1 and A_3 are in L. Similarly, there are no edges between A_2 and A_4.

We prove, by contradiction, that there are no edges between A_1 and A_4. Suppose that $\{u, v\} \in A_1 \times A_4$ is an edge in B. Thus, $u \in N(v)$. By definition of A_4, B_v is a the complete bipartite graph, where $N(v) \subseteq L$ forms one partition, and $N(N(v)) \subseteq R$ forms the other partition. Thus, the vertex u is adjacent to all vertices of $N(N(v))$. By the same argument on B_u, the vertex v is adjacent to all vertices of $N(N(u))$. With these two results, we have $N(v) = N(N(N(v)))$ and $N(u) = N(N(N(u)))$. Thus, $N(u) = L$ and $N(v) = R$. This contradicts the fact that B is not a complete bipartite graph.

Lastly, by definition of S, all the vertices of A_2 are adjacent to the vertices in L, and all the vertices of A_3 are adjacent to the vertices in R. Thus, graphs induced by the vertices A_1 and A_2, A_2 and A_3, A_3 and A_4, are complete bipartite graphs. □

Proposition 6.4.7. *The problem* #Hom($\cdot$, P) *for impasse* P *is #P-hard.*

Proof. By Corollary 6.3.3, we can collapse all the vertices with the same neighbourhood of $P(a_1, a_2, a_3, a_4)$. This results in a path of length three. The path of length three is equal to the graph with two connected vertices where one of the vertices is looped tensored with K_2, as in Figure 6.2. From Section 5.1.2,

$$\#\text{IS} = \#\text{Hom}(\ \cdot\ , \bullet\!\!-\!\!\bullet\!\!\circlearrowleft).$$

Hence, #Hom($\cdot$, P) is #P-hard for any impasse P. □

This concludes Chen, Curticapean, Dell's proof of Theorem 6.1.1. From Theorem 6.1.1 and the discussions in Chapter 5.1, we conclude that counting k-colourings to graphs and computing the partition function of in Widom-Rowlinson and Beach model is #P-complete.

Chapter 7

Conclusion

As we have seen, many counting problems in graph theory can be related to one another through reductions. This allows us to determine the relative computational difficulty of solving these tasks. In particular, we deduced the #P-completeness of counting graph homomorphisms to fixed graphs from the #P-complete problem of counting independent sets in graphs. Due to the generality and flexibility of graph homomorphisms, it followed as corollary that counting k-colourings and solving the partition of some models of statistical physics in #P-complete. The knowledge that a problem is #P-complete can help guide which lines of research are worth pursuing.

The result of counting homomorphisms to a fixed graph H by Dyer and Greenhill [12], was extended in several ways. Bulatov and Grohe [3] proved a dichotomy theorem for when H is a multigraph, or equivalently H has edge weights. Goldberg, Grohe, Jerrum, and Thurley [16], then proved a dichotomy theorem for computing $Z_A(G)$ when A is a real symmetric matrix. Cai, Chen, and Lu [5], further extended this result to when A is a symmetric matrix with complex values.

Similar to the counting independent sets problem we discussed in Chapter 4, the complexity of counting homomorphisms from restricted instances has been considered, for example in [11] and [18]. The approximability of studying graph homomorphisms were studied by [26] and [17], with similar trichotomy theorems found as in the case for independent sets in Section 4.4.

The counting homomorphisms problem was generalised to counting constraint satisfaction problems. Jerrum's survey [24] details much of the work done in this area, which has important applications in artificial intelligence and machine learning [29].

Lastly, homomorphism densities into graphs are useful in studying large networks [27]. Graphons are the generalisation of homomorphism counts into infinite graphs. Many of the basic properties of homomorphism counts carry over. The methods used here are more analytic.

References

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[2] Graham R. Brightwell and Peter Winkler. Graph homomorphisms and phase transitions. *Journal of Combinatorial Theory, Series B*, 77(2):221–262, 1999.

[3] Andrei Bulatov and Martin Grohe. The complexity of partition functions. *Theoretical Computer Science*, 348(2):148–186, 2005. Automata, Languages and Programming: Algorithms and Complexity (ICALP-A 2004).

[4] Robert Burton and Jeffrey E. Steif. Non-uniqueness of measures of maximal entropy for subshifts of finite type. *Ergodic Theory and Dynamical Systems*, 14(2):213–235, 1994.

[5] Jin-Yi Cai, Xi Chen, and Pinyan Lu. Graph homomorphisms with complex values: A dichotomy theorem, 2011.

[6] Hubie Chen, Radu Curticapean, and Holger Dell. The exponential-time complexity of counting (quantum) graph homomorphisms. In Ignasi Sau and Dimitrios M. Thilikos, editors, *Graph-Theoretic Concepts in Computer Science*, pages 364–378, Cham, 2019. Springer International Publishing.

[7] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery.

[8] Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC '17, page 210–223. ACM, June 2017.

[9] Reinhard Diestel. Graph theory, volume 173 of. *Graduate texts in mathematics*, 593, 2012.

[10] Martin Dyer, Leslie Ann Goldberg, Catherine Greenhill, and Mark Jerrum. On the relative complexity of approximate counting problems. In Klaus Jansen and Samir Khuller, editors, *Approximation Algorithms for Combinatorial Optimization*, pages 108–119, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[11] Martin Dyer, Leslie Ann Goldberg, and Mike Paterson. On counting homomorphisms to directed acyclic graphs. *J. ACM*, 54(6):27–es, December 2007.

[12] Martin Dyer and Catherine Greenhill. The complexity of counting graph homomorphisms. *Random Structures & Algorithms*, 17(3-4):260–289, 2000.

[13] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[14] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Mathematical Sciences Series. W. H. Freeman, 1979.

[15] Leslie Ann Goldberg, Martin Grohe, Mark Jerrum, and Marc Thurley. A complexity dichotomy for partition functions with mixed signs, 2009.

[16] Leslie Ann Goldberg, Martin Grohe, Mark Jerrum, and Marc Thurley. A complexity dichotomy for partition functions with mixed signs, 2009.

[17] Leslie Ann Goldberg and Mark Jerrum. The complexity of approximately counting tree homomorphisms. *ACM Transactions on Computation Theory*, 6(2):1–31, May 2014.

[18] Artem Govorov, Jin-Yi Cai, and Martin E. Dyer. A dichotomy for bounded degree graph homomorphisms with nonnegative weights. *CoRR*, abs/2002.02021, 2020.

[19] Catherine Greenhill. The complexity of counting colourings and independent sets in sparse graphs and hypergraphs. *Comput. Complex.*, 9(1):52–72, January 2000.

[20] Pavol Hell and Jaroslav Nešetřil. On the complexity of H-coloring. *Journal of Combinatorial Theory, Series B*, 48(1):92–110, 1990.

[21] Pavol Hell and Jaroslav Nešetřil. 192colouring—variations on a theme. In *Graphs and Homomorphisms*. Oxford University Press, 07 2004.

[22] Ernst Ising. Beitrag zur theorie des ferromagnetismus. *Zeitschrift für Physik*, 31(1):253–258, 1925.

[23] M. Jerrum. *Counting, Sampling and Integrating: Algorithms and Complexity*. Lectures in Mathematics. ETH Zürich. Birkhäuser Basel, 2003.

[24] Mark Jerrum. Counting Constraint Satisfaction Problems. In Andrei Krokhin and Stanislav Zivny, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 205–231. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2017.

[25] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations*, pages 85–103. Springer US, Boston, MA, 1972.

[26] Steven Kelk. *On the relative complexity of approximately counting H-colourings*. Ph.d. thesis, University of Warwick, 2003.

[27] László Lovász. *Large Networks and Graph Limits*, volume 60 of *Colloquium Publications*. American Mathematical Society, 2012.

[28] J. Scott Provan and Michael O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal on Computing*, 12(4):777–788, 1983.

[29] Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1):273–302, 1996.

[30] Pavan Sangha and Michele Zito. Independent sets in line of sight networks. *Discrete Applied Mathematics*, 286:100–115, 2020. Algorithms and Discrete Applied Mathematics (CALDAM 2017).

[31] Janos Simon. On the difference between one and many. In Arto Salomaa and Magnus Steinby, editors, *Automata, Languages and Programming*, pages 480–491, Berlin, Heidelberg, 1977. Springer Berlin Heidelberg.

[32] Seinosuke Toda. Pp is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

[33] Salil P. Vadhan. The complexity of counting. Undergraduate thesis, Harvard University, 1995. Available from `http://people.seas.harvard.edu/~salil/`.

[34] Salil P. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM Journal on Computing*, 31(2):398–427, 2001.

[35] L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

[36] B. Widom and J. S. Rowlinson. New model for the study of liquid–vapor phase transitions. *The Journal of Chemical Physics*, 52(4):1670–1684, 02 1970.

I acknowledge the assistance of ChatGPT 4.0 in writing Python code in Chapter 4, Section 4.4. The AI suggested that I use the dedicated Python module NetworkX for graph-theoretic applications, such as creating expander graphs and the BFS algorithm, which are all standard material. I did not use AI or any large/natural language model for any other part of the thesis.