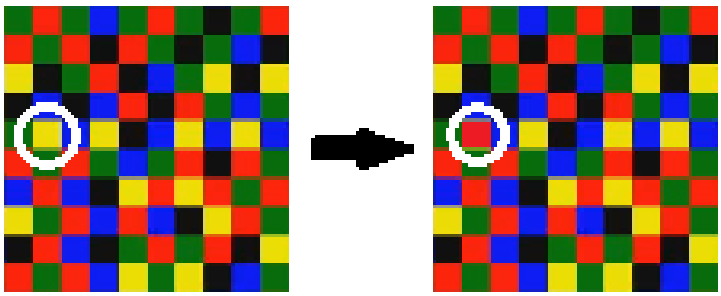
Approximating graph-theoretic counting problems with Markov chain simulation

Wouter Rienks

July 2, 2018

Bachelor thesis Mathematics

Supervisor: dr. Viresh Patel



Korteweg-de Vries Institute for Mathematics
Faculty of Sciences
University of Amsterdam

Abstract

In graph theory, it is hard to find exact answers for certain counting problems efficiently. For example, counting the number of proper k-colourings or independent sets on a graph are hard to do efficiently. In this thesis we aim to find algorithms to provide approximate answers that run in polynomial time. Such algorithms are known as fully polynomial randomised approximation scheme (fpras).

The algorithms we study specify a process to sample approximately uniformly from the set of all objects to count, and use this sampler to estimate the total amount of objects. We discuss some of these sampling processes. One of them is known as the Glauber dynamics which originates from statistical physics. Another process we study is an improvement of the Glauber dynamics, sometimes referred to as the Flip dynamics. And we give a modification of Jerrum [6]'s sampler for matchings to sample independent sets in claw-free graphs.

All of the samplers we study are created by running a Markov chain on the set of all objects to count. We study some of the theory to understand the behaviour of these samplers. A particular concept of interest is the *mixing time*, the time it takes for these Markov chains to converge to their uniform distribution. We provide two ways of bounding the mixing time, known as *coupling* and the *canonical paths* method.

Finally we also implement the Glauber Dynamics using python/C++, and do an empirical analysis of the sampler of Jerrum for some small graphs.

Title: Approximating graph-theoretic counting problems with Markov chain simulation
Authors: Wouter Rienks, wouter.rienks@student.uva.nl, 11007745
Supervisor: dr. Viresh Patel,
Second grader: dr. Arnoud den Boer,
End date: July 2, 2018

Korteweg-de Vries Institute for Mathematics
University of Amsterdam
Science Park 904, 1098 XH Amsterdam
`http://www.kdvi.uva.nl`

Contents

1 Introduction

In graph theory, it is hard to find exact answers for certain counting problems efficiently. Consider for example the problem of counting the number of proper k-colourings in a given graph G on n vertices. It is clear that any general exact formula will probably not be of any use, as there are many possible graphs on n vertices. And if G is sparse (say $\Delta(G) \leq k+2$), it is not to hard to see that the number of k-colourings will grow exponentially in n, so enumerating all colourings with a computer will take a long time even for small n. However, there are algorithms to approximate the answer in polynomial time. These algorithms are so-called *fpras*, which stands for *fully polynomial randomised approximation scheme*.

The methods we will consider have their origin in statistical physics. For example, the Glauber Dynamics (a Markov Chain process to select a graph colouring almost-uniformly, see Section 3.2) is related to the Potts model. The Potts model is used in statistical physics to model interacting spins on a graph $G = (V, E)$. In practice this graph will usually just be a lattice with vertex set $\{-L, \ldots, L\}^D$ for some finite L, D, however we will consider an arbitrary graph. Every vertex is then assigned a spin, which we can model as assigning each vertex one of q colours. A particular arrangement of spins will be referred to as a state (analogous to a single colouring σ). Each state corresponds to an energy level, which is called the Hamiltonian of the state, defined by

$$H(\sigma) = \sum_{(u,v)\in E} J(1 - \delta_{\sigma(u),\sigma(v)}).$$

Here the J is a fixed constant, and $\delta : [q]^2 \to \{0, 1\}$ is the Kronecker delta, which is equal 1 if $\sigma(u) = \sigma(v)$ and 0 otherwise. If $J > 0$ the model is said to be *ferromagnetic* and *antiferromagnetic* if $J < 0$.

The Potts model then defines a probability measure on the space of all possible states. For a fixed constant $\beta = \frac{1}{k_B T}$ (known as the inverse temperature, here k_B is the Boltzmann constant and T is the absolute temperature), the relative weight of a particular state is given by $e^{-\beta H(\sigma)}$. If we thus define the normalizing constant (often called the partition function)

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

then the probability of a given state appearing is given by

$$\mathbb{P}(\sigma) = \frac{e^{\beta H(\sigma)}}{Z(G)}.$$

The Potts model is a generalization of the Ising model, which is the exact same model but taking only the set $\{1, -1\}$ as possible spins instead of an arbitrary number q. By

modifying the Glauber Dynamics with Metropolis-style acceptance probabilities (see for an explanation Norris [7, Section 5.5, p210], we provide an example of how to do this in Section 5.1) to arrive at the desired invariant distribution, one can then construct a process to sample from the Potts model distribution. Such a sampler is called a Gibbs sampler, and has applications in statistical physics.

However, physicists are often interested in evaluating the partition function $Z(G)$ as well. It turns out, that if we set $K = J/k_BT$, the function $Z(G;q,K)$ is (up to some constant) the Tutte polynomial in disguised form. The Tutte polynomial $T(G;x,y)$ is a two-variable polynomial assigned with graphs that can describe a lot of their properties. It is defined recursively by contracting/removing edges from the graph. We will not define it here, one could look at Goodall [3, Section 3.1] for a definition of the Tutte polynomial. If one then parametrizes

$$\begin{aligned} x &= 1 + \frac{qe^{-K}}{1-e^{-K}} \\ y &= e^{K}, \end{aligned}$$

one has that $T(G;x,y) = C \times Z(G;q,K)$ for some constant C. The Tutte polynomial can thus be seen as a continuation of the partition function from some countable set of hyperbola (the image of $\mathbb{N} \times \mathbb{R}$ under the parametrization) to the whole plane.

It is then no surprise that one can retrieve lots of properties of G one could obtain from the Tutte polynomial directly from $Z(G)$ as well. For example, consider letting $\beta \to \infty$, which corresponds to letting the temperature tend to absolute zero, in the antiferromagnetic Potts model ($J < 0$). Then any proper colouring appears with weight $e^{-\beta|E|\times J}$, and any non-proper colouring will appear with weight at most $e^{-\beta(|E|-1)\times J}$. Thus

$$Z(G;q,K) = \#\{\sigma \mid \sigma \text{ is a proper colouring}\}(e^{-\beta J})^{|E|} + \mathcal{O}((e^{-\beta J})^{|E|-1}). \tag{1.1}$$

As the number of edges $|E|$ is easy to count in a graph, by letting $\beta \to \infty$ one can use $Z(G)$ to determine the number of proper colourings on a graph G (which corresponds to evaluating $T(1-q,0)$). To do this, note that from equation 1.1 we obtain

$$\#\{\sigma \mid \sigma \text{ is a proper colouring}\} = \lim_{\beta\to\infty} \frac{Z(G;q,K)}{e^{-\beta J|E|}}$$

as long as $J < 0$.

We will now start off with some prerequisites in Chapter 2. After that, in chapter 3 we will describe the Glauber dynamics Markov Chain mentioned above in detail, and look at some of its properties. In particular we will look at its mixing time, the rate of convergence to its limiting distribution. In chapter 4 we expand the theory by looking at two distinct ways of showing a Markov chain mixes fast. Then in chapter 5 we will use this to construct a new chain for sampling vertex covers, and improve upon the Glauber dynamics. Finally in chapter 6 we will perform an empirical analysis of the Glauber dynamics by implementing the algorithm on a computer.

2 Prerequisites

In this chapter we introduce some notation and state some previously known theorems. Throughout this thesis, we will often consider a discrete time Markov chain on some state space Ω. We will often write P for its transition matrix. Given $x, y \in \Omega$, we will write $P^t(x, y)$ for the probability of moving from x to y in t steps of the Markov chain. We say that x and y are *adjacent* if $P^t(x, y) > 0$.

Given a distrubition π on Ω, we write $P^t(\pi)$ for the distribution obtained after simulating the Markov chain for t steps with initial distribution π. We will often write $P^t(x, \cdot)$ where $x \in \Omega$ for the distribution obtained after simulating the Markov chain for t steps starting at x.

We say that the Markov chain is *irreducible* if for all $x, y \in \Omega$ there exists some $t \in \mathbb{N}$ such that $P^t(x, y) > 0$.

We say that a distribution π is an *invariant distribution* if $P\pi = \pi$. Recall that if an invariant distribution exists and the Markov chain is irreducible, it is in fact unique. Furthermore, we have the following result.

Proposition 2.1 (Detailed Balance)**.** *Let P be the transition matrix of a Markov chain with state space Ω. If π is some distribution on Ω satisfying*

$$\pi(x)P(x, y) = \pi(y)P(y, x) \qquad \forall x, y \in \Omega$$

then π is an invariant distribution of the Markov chain.

Note that if P is symmetric and irreducible, then π is the uniform distribution.

We say that the Markov chain is *aperiodic* if $P^t(x, x) > 0$ for all $x \in \Omega$ and $t \in \mathbb{N}$ sufficiently large. This is equivalent to saying that $\gcd\{t \mid P^t(x, x) > 0\} = 1$. This allows us to state the following general theorem about the limiting behaviour of Markov chains.

Theorem 2.2 (Ergodic Theorem)**.** *Suppose P is a transition matrix for some irreducible and aperiodic Markov chain. Then there exists a* unique *limiting distribution π, and for any other distribution π' we have for any $x \in \Omega$ that*

$$\lim_{t \to \infty} P^t(\pi')(x) = \pi(x).$$

If we are in the situation of the Theorem 2.2, we say that the Markov chain is *ergodic*.

3 From Sampling to Counting

In this chapter we discuss an initial algorithm for counting the total number of k-colourings on a graph for some fixed k. The algorithm will work by sampling colourings randomly. We will describe the process used to sample these colourings, but we will not analyze the performance of the sampler until chapter 3.1. This chapter serves mainly to show, by means of an example, how one could construct an algorithm to estimate the total number of objects given a sampler.

Before we can do this we need some terminology. We thus start by introducing the so-called *mixing time* of Markov chains.

3.1 Mixing Time

In this section we aim to discuss the speed of convergence of a Markov chain to its limiting distribution. The fewer steps a Markov chain will need to get close to its limiting distribution, the faster its mixing time will be. Having a Markov chain converge to its limiting distribution fast is really useful, because it gives us a way to sample from its limiting distribution almost-uniformly, by simulating the chain for not too many steps. This will in turn give us a way to approximate the size of the state space, as we will see by means of an example in the next section.

We start by making more precise what we mean by "almost-uniformly", by defining a metric on the set of all probability distributions on the state space.

Definition 3.1. Given two probability distributions π_1, π_2 on a finite set Ω, we define the *total variation distance* between π_1 and π_2 to be

$$d_{\mathrm{TV}}(\pi_1, \pi_2) = \max_{S \subseteq \Omega} |\pi_1(S) - \pi_2(S)|.$$

It follows immediately from the next remark that this is in fact a metric.

Remark 3.2. For any set $S \subseteq \Omega$, denote temporarily $d(S) := |\pi_1(S) - \pi_2(S)|$ and then define the sets $S^+ = \{s \in S \mid \pi_1(s) > \pi_2(s)\}$ and similarly $S^- = \{s \in S \mid \pi_1(s) < \pi_2(s)\}$. Then clearly

$$d(S) = \max\{d(S^+), d(S^-)\} - \min\{d(S^+), d(S^-)\}.$$

Thus for any maximal S, we have either $S = S^+$ or $S = S^-$. As $d(\Omega^+) \geq d(S^+)$ and $d(\Omega^-) \geq d(S^-)$, one of either Ω^+ or Ω^- must be maximal. But as both π_1 and π_2 sum to 1 over Ω, we must have $d(\Omega) = 0$. Thus $d(\Omega^+) - d(\Omega^-) = 0$, which implies that $d(\Omega^+) = d(\Omega^-) = d_{\mathrm{TV}}(\pi_1, \pi_2)$. Therefore we may finally conclude that we have

$d_{\mathrm{TV}}(\pi_1, \pi_2) = \frac{1}{2}\left(d(\Omega^+) + d(\Omega^-)\right)$, and we obtain

$$d_{\mathrm{TV}}(\pi_1, \pi_2) = \frac{1}{2}\sum_{y\in\Omega} |\pi_1(y) - \pi_2(y)|.$$

Remark 3.3. As Ω is a finite set, the ergodic theorem also holds with respect to this metric (as opposed to converging pointwise for any $x \in \Omega$). This means that for any irreducible and aperiodic matrix P with invariant distribution π we have for any distribution π' that

$$\lim_{t\to\infty} d_{\mathrm{TV}}(P^t(\pi'), \pi) = 0.$$

Now given any initial state x, we can define the distance between the chain starting at x and the limiting distribution π at time t to be

$$d_x(t) := d_{\mathrm{TV}}(P^t(x, \cdot), \pi).$$

The ergodic theorem implies that $d_x(t)$ converges to 0. The question remains however, how fast it converges. The mixing time of a Markov chain aims to describe this.

Definition 3.4. The *mixing time* of an irreducible and aperiodic Markov chain with transition matrix P is defined as a function $\tau : (0, 1) \mapsto \mathbb{N}$ assigning to every $\varepsilon > 0$ a time needed to get within distance ε of π by

$$\tau(\varepsilon) := \max_{x\in\Omega} \min_{t\in\mathbb{N}} \{t \mid d_x(t') \leq \varepsilon \text{ for all } t' \geq t\}.$$

We will say that a family of Markov chains $\mathcal{M}$ is *rapidly mixing* if the mixing time of $M \in \mathcal{M}$ is bounded by some polynomial in the size $\log(|\Omega|)$, where $\Omega = \Omega(M)$ is the state space. For example, given a family of Markov chains on colourings or matchings of graphs, the size of the state space will be exponential in the number of vertices, hence we want our algorithm to be polynomial in the number of vertices.

3.2 Introduction to Glauber Dynamics

In this section we provide a so-called fpras for the number of k-colourings of a graph G, whenever $k > 2\Delta(G)$. This algorithm was originally introduced by Jerrum [5]. We will write $\Omega_k(G)$ for the set of proper k-colourings on G, we thus wish to estimate $|\Omega_k(G)|$. We start off by describing precisely what we mean by a fast approximation algorithm.

Definition 3.5 (FPRAS)**.** Let $k \in \mathbb{N}$. A randomised approximation scheme for k-colourings is a probabilistic algorithm that takes as input a graph G and some $\varepsilon \in (0, 1)$, and outputs a random number Y such that

$$\mathbb{P}\left((1-\varepsilon)|\Omega_K(G)| \leq Y \leq (1+\varepsilon)|\Omega_k(G)|\right) \geq \frac{3}{4}.$$

A randomised approximation scheme is called fully polynomial if it runs in time polynomial in $|G|$ and ε^{-1}. A fully polynomial randomised approximation scheme will be abbreviated as fpras.

Remark 3.6. The constant $\frac{3}{4}$ has no significance beyond lying strictly between 0 and $\frac{1}{2}$. If we repeat the experiment N times and take the median of the results, then the median M is inside the interval $I = ((1-\varepsilon)|\Omega_K(G)|, (1+\varepsilon)|\Omega_k(G)|)$ whenever at least $N/2$ of the results are. Therefore, if the probability of a result ending in I is at least $\frac{3}{4}$, then the probability of less than $N/2$ results ending in I is at most the probability of a r.v. with distribution $\mathrm{Bin}\left(N, \frac{1}{4}\right)$ being at least $\frac{N}{2}$. Thus if we let $Z \sim \mathrm{Bin}(N, \frac{1}{4})$ be a random variable distributed accordingly, then we can bound the probability of M lying outside I using the Hoeffding [4, Theorem 2] inequality by

$$\mathbb{P}(M \notin I) \leq \mathbb{P}\left(Z \geq \frac{N}{2}\right) \leq \mathbb{P}\left(Z - \mathbb{E}[Z] \geq \frac{N}{4}\right) \leq e^{-\frac{N^3}{8}}.$$

From this we see that we can get a success probability p arbitrarily close to one by simulating $\lceil (-8\log(1-p))^{\frac{1}{3}} \rceil$ times and taking the median of the results.

We now wish to find such an fpras. As said before, we will start by constructing a Markov chain on the space of all proper k-colourings to sample k-colourings almost-uniformly.

Definition 3.7 (Glauber Dynamics)**.** Let G be a graph and $k \in \mathbb{N}$. Consider a Markov chain (X_t) with state space $\Omega = \Omega_k(G)$ the set of all proper k-colourings on G, whose transition probabilities from state X_t are described by the following procedure:

1. Pick a vertex $v \in V(G)$ and a colour $c \in \{1, \ldots, k\}$ uniformly at random.
2. Recolour vertex v with colour c. If the resulting colouring X' is a proper colouring, let $X_{t+1} = X'$, otherwise let $X_{t+1} = X_t$.

This process is often referred to as the *Glauber Dynamics* (which originates from statistical physics). It turns out that this chain is almost always ergodic, as shown in the following proposition.

Proposition 3.8. *Let G be a graph and $k \in \mathbb{N}$. The Glauber Dynamics Markov chain is ergodic whenever $k \geq \Delta(G) + 2$.*

Proof. By Theorem 2.2, it suffices to show the chain is aperiodic and irreducible. To see that it is aperiodic, note that given a coloring σ and some $v \in V(G)$, attempting to recolour v with colour $\sigma(v)$ will make the chain stay at colouring σ, therefore $P(\sigma, \sigma) > 0$ for all $\sigma \in \Omega$, hence the chain is aperiodic. To see that the chain is irreducible, note that we can travel from any colouring σ to any colouring τ by enumerating the vertices of G in some way, say $v_1, \ldots, v_n$, and recolouring the v_i from $\sigma(v_i)$ to $\tau(v_i)$ in order. It might be necessary to recolour some v_j with $j > i$ along the way, but we always have a spare colour to do so since $k \geq \Delta(G) + 2$ (note that any v_j with $j < i$ will never need to be recoloured since τ is a proper colouring). □

If $k = \Delta(G) + 1$ it need not be ergodic, consider for example the graph containing two vertices and a single edge, if both are given a different colour then $k = 2, \Delta(G) = 1$ but it's impossible to swap the colours using the Glauber Dynamics Markov Chain.

It turns out that this chain is in fact rapidly mixing provided $k > 2\Delta(G)$.

Theorem 3.9. *Let G be a graph and $k \in \mathbb{N}$. Provided $k > 2\Delta(G)$, the mixing time of the Glauber dynamics with k colours on G is bounded above by*

$$\tau(\varepsilon) \leq \frac{k}{k-2\Delta} n \log(n\varepsilon^{-1}).$$

We will prove this later on, in Corollary 4.6. For now, we will use this bound on the mixing time to construct an fpras for approximating the number of k-colourings of a graph.

Theorem 3.10 (Counting k-colourings rapidly)**.** *Let G be graph and $k \in \mathbb{N}$. If k is such that $k > 2\Delta(G)$, then there exists an fpras for counting the number of k-colourings on G. The time complexity of this algorithm can be bounded above by*

$$\frac{50k}{k-2\Delta} \times \frac{nm^2}{\varepsilon^2} \log\left(\frac{4nm}{\varepsilon}\right),$$

where m is the number of edges in G and the time unit is a single simulation step of the Markov chain (X_t) described above.

Proof. Our proof is entirely thanks to Jerrum [5]. Let $G_0 \subseteq G_1 \subseteq \cdots \subseteq G_m$ be a sequence of subgraphs of G in which $G_m = G$ and each G_i is obtained from G_{i+1} by removing a single edge. Then clearly, since $|\Omega_k(G_0)| = k^n$, we have

$$|\Omega_k(G)| = k^n \prod_{i=1}^{m} \frac{|\Omega_k(G_i)|}{|\Omega_k(G_{i-1})|}.$$

Our goal is to estimate each of the ratios $\rho_i := |\Omega_k(G_i)|/|\Omega_k(G_{i-1})|$ individually and combine those estimates to estimate $|\Omega_k(G)|$.

We first bound the ρ_i away from 0, which will be needed later on. Suppose the graphs G_{i-1} is obtanied from G_i by removing the edge $\{u, v\}$. Clearly $\Omega_k(G_i) \subseteq \Omega_k(G_{i-1})$. Any colouring in $\Omega_k(G_{i-1}) \setminus \Omega_k(G_i)$ assigns the same colour to u and v, and may thus be changed to be a colouring in $\Omega_k(G_i)$ by recolouring vertex u with one of at least $k - \Delta \geq \Delta + 1$ colours. On the other hand, each colouring of $\Omega_k(G_i)$ can be obtained in at most one way as the result of such a pertubation. Therefore

$$|\Omega_k(G_{i-1}) \setminus \Omega_k(G_i)| \leq \frac{1}{\Delta+1} \Omega_k(G_i),$$

and thus (note $k > 1$ hence $\Delta \geq 2$, the case $k = 1$ is trivial anyway)

$$\frac{3}{4} \leq \frac{\Delta+1}{\Delta+2} \leq \rho_i \leq 1. \tag{3.1}$$

Now continue by defining

$$T = \left\lceil \frac{k}{k-2\Delta} n \log\left(\frac{4nm}{\varepsilon}\right) \right\rceil.$$

Let $C_i \in \{0,1\}$ denote the random variable obtained by simulating the Glauber Dynamics Markov Chain with k colours from an arbitrary fixed initial state $c_i \in \Omega_k(G_{i-1})$ for T steps and returning the final colouring. Then C_i has distribution $P_i^T(c_i, \cdot)$, where $P_i : \Omega_k(G_{i-1}) \to \Omega_k(G_{i-1})$ is the transition matrix for the Glauber Dynamics Markov Chain on $\Omega_k(G_{i-1})$. Set Z_i to be the random variable given by

$$Z_i = \begin{cases} 0 \text{ if } C_i \notin \Omega_k(G_i), \\ 1 \text{ if } C_i \in \Omega_k(G_i). \end{cases}$$

Let $\mu_i = \mathbb{E}Z_i$. The idea is that Z_i is almost Bernoulli(ρ_i)-distributed. In fact, we have

$$\begin{aligned} |\mu_i - \rho_i| &= \left| \sum_{y \in \Omega_k(G_i)} P^T(c_i, y) - \sum_{y \in \Omega_k(G_i)} \frac{1}{\Omega_k(G_{i-1})} \right| \\ &= \left| P^T(c_i, \Omega_k(G_i)) - \pi_i(\Omega_k(G_i)) \right| \end{aligned}$$

where π_i is the uniform (limit) distribution of P_i.

By Theorem 3.9 (inserting $\varepsilon = \varepsilon/4m$, and using the global variation bound with $S = \Omega_k(G_i)$) we thus have

$$\rho_i - \frac{\varepsilon}{4m} \leq \mu_i \leq \rho_i + \frac{\varepsilon}{4m}.$$

Using equation 3.1 it follows that

$$\left(1 - \frac{\varepsilon}{3m}\right) \rho_i \leq \mu_i \leq \left(1 + \frac{\varepsilon}{3m}\right) \rho_i. \tag{3.2}$$

Thus the mean of a sufficiently large number of independent copies of Z_i will provide a good estimate of ρ_i. Set

$$s := \lceil 37\varepsilon^{-2} m \rceil,$$

we will see this choice is sufficiently large later on. Let $Z_i^{(1)}, \ldots, Z_i^{(s)}$ be a sequence of i.i.d. copies of Z_i. Let $\overline{Z_i}$ denote their mean. Finally, let $Y := k^n \prod \overline{Z_i}$ be our estimator for $|\Omega_k(G_i)|$, since all $\overline{Z_i}$ are independent we have $\mathbb{E}[Y] = k^n \prod \mu_i$. Since we already know the mean of Y to be close to our actual answer (see 3.2), if Y has low variance as well we would expect the estimator Y to end up near the actual answer with high probability.

For convenience, we will bound the variance of $k^{-n}Y$ instead. We can do so by

$$\begin{aligned} \frac{\operatorname{Var}\left(\prod_{i=1}^m \overline{Z_i}\right)}{\prod_{i=1}^m \mu_i^2} &= \frac{\mathbb{E}\left[\prod_{i=1}^m \overline{Z_i^2}\right] - \prod_{i=1}^m \mu_i^2}{\prod_{i=1}^m \mu_i^2} \\ &= \frac{\prod_{i=1}^m \mathbb{E}\left[\overline{Z_i^2}\right]}{\prod_{i=1}^m \mu_i^2} - 1 && (\overline{Z_i} \text{ are independent}) \\ &= \prod_{i=1}^m \left(1 + \frac{\operatorname{Var} \overline{Z_i}}{\mu_i^2}\right) - 1. && (\operatorname{Var} \overline{Z_i} + \mu_i^2 = \mathbb{E}[Z_i^2]) \end{aligned}$$

Thus to bound the variance of Y we need to bound $\mu_i^{-2} \operatorname{Var} \overline{Z_i^2}$. Since Z_i takes values in $\{0,1\}$, by conditioning on both outcomes we see that for all $i \in \{1,\dots,s\}$ we have

$$\mu_i^{-2} \operatorname{Var} Z_i = \mu_i^{-2}(\mu_i(1-\mu_i)^2 + (1-\mu_i)\mu_i^2) = \mu_i^{-1} - 1.$$

We thus need to bound μ_i away from 0. But we can do this, since by equations 3.1, 3.2 we have $\mu_i \geq \frac{1}{2}$. Hence $\mu_i^{-2} \operatorname{Var} Z_i \leq 1$, and $\mu_i^{-2} \operatorname{Var}(\overline{Z_i^2}) \leq s^{-1}$. Substituting this in our original calculation, we obtain

$$\begin{aligned} \operatorname{Var}(k^{-n}Y) &\leq \left(1 + \frac{1}{s}\right)^m - 1 \\ &\leq \exp\left(\frac{\varepsilon^2}{37}\right) - 1 \leq \frac{\varepsilon^2}{36}. \end{aligned}$$

By Chebyshev's inequality we conclude that (we use that $\operatorname{Var}(k^{-n}Y)^{\frac{1}{2}} \leq \frac{\varepsilon}{6}\prod_{i=1}^m \mu_i$ and $\mathbb{E}[k^{-n}Y] = \prod_{i=1}^m \mu_i$)

$$\mathbb{P}\left(\left|k^{-n}Y - \prod_{i=1}^m \mu_i\right| < \frac{\varepsilon\alpha}{6}\prod_{i=1}^m \mu_i\right) \geq 1 - \frac{1}{\alpha^2}$$

holds for any $\alpha > 0$. By setting $\alpha = \frac{1}{2}$ it follows that

$$\mathbb{P}\left(\left(1 - \frac{\varepsilon}{3}\right)\prod_{i=1}^m \mu_i \leq k^{-n}Y \leq \left(1 + \frac{\varepsilon}{3}\right)\prod_{i=1}^m \mu_i\right) \geq \frac{3}{4}.$$

From equation 3.2, we see using the fact that $(1 - \varepsilon/3m)^m \geq (1 - \varepsilon/2)$ for $m \geq 1$ that

$$\left(1 - \frac{\varepsilon}{2}\right)\prod_{i=1}^m \rho_i \leq \prod_{i=1}^m \mu_i \leq \left(1 + \frac{\varepsilon}{2}\right)\prod_{i=1}^m \rho_i.$$

Combining these two inequalities implies the estimator Y satisfies the required bounds.

Since the computation of the estimator requires ms simulations, each with T steps, we need msT steps in total. The constant factor 50 is chosen to absorb the ceiling functions. □

4 Establishing Rapid Mixing

In the previous chapter we have provided an algorithm that counted k-colourings rapidly. However, we assumed that the Markov chain used to sample k-colourings converged to the invariant distribution rapidly (we will often refer to this as a Markov chain *mixing* rapidly). In this chapter, we dive into the theory to show that certain Markov chains are rapidly mixing. We discuss two methods, known as *coupling* and *the canonical paths method*.

We will always assume our Markov chain to be aperiodic and irreducible so that the ergodic theorem holds, and write π for the unique limiting distribution.

4.1 Coupling

The first method to show that a Markov chain mixes rapidly is often referred to as coupling. A good intuition for a coupling is two people walking around randomly in a city until they meet, and once they meet they will walk together.

Definition 4.1. Suppose we are given a Markov chain M with a transition matrix P and state space Ω. A coupling of M is a Markov chain of the form $(x_t, y_t)_{t\in\mathbb{N}}$ on $\Omega \times \Omega$ such that

1) The marginal chains $(x_t, \cdot)$ and $(\cdot, y_t)$ are copies of M. More precisely, at any time t the identities

$$\mathbb{P}(x_{t+1} = x' \mid x_t = x) = P(x, x'), \tag{4.1}$$
$$\mathbb{P}(y_{t+1} = y' \mid y_t = y) = P(y, y') \tag{4.2}$$

hold.

2) If $x_t = y_t$, then $x_{t+1} = y_{t+1}$.

Remark 4.2. We can look at a coupling as if it were a linear map between jump probabilities. Suppose $(x_t, y_t) = (\xi, \omega)$. Given a jump $\xi \to \xi'$ for x_t, we can look at the probability of each of the individual jumps $\omega \to \omega'$ appearing for y_t. More precisely, using equation (4.2) we can identify a coupling with the coefficients

$$a_{\xi\to\xi'|\omega\to\omega'}(t) = \mathbb{P}(y_{t+1} = \xi' | x_{t+1} = \omega' \text{ and } (x_t, y_t) = (\omega, \xi)),$$

since one can then compute the transition probabilities of the coupling (x_t, y_t) immediately by

$$\mathbb{P}((x_{t+1}, y_{t+1}) = (\omega', \xi') \mid (x_t, y_t) = (\omega, \xi)) = a_{\xi\to\xi'|\omega\to\omega'} P(\omega, \omega'). \tag{4.3}$$

However, not every coupling is very interesting. Consider for example the coupling that selects x_{t+1} and y_{t+1} independently if $x_t \neq y_t$, and selects x_{t+1} by simulating the chain for one step starting at x_t then setting $y_{t+1} = x_{t+1}$ whenever $x_t = y_t$. One can easily verify that this is a coupling, however as long as $x_t \neq y_t$ we are basically just running the chain twice.

However, one could imagine that if coupling occurs rapidly (i.e. $X_T = Y_T$ occurs for small T with high probability) for some particular coupling (even if the starting states are very far apart), then the distance between any two distributions should decline rapidly, since probability mass can move around the state space quickly. Then to establish rapid mixing, all one has to do is find a coupling that 'couples fast'.

The following lemma makes this intuition a bit more precise.

Lemma 4.3 (Coupling Lemma)**.** *Let P be the transition matrix of a Markov chain with state space Ω. Remember that $P^t(x, \cdot)$ is defined as the distribution on Ω obtained by simulating the chain for t steps with initial state x. Let (x_t, y_t) be a coupling of the Markov chain with initial state $(x, y) \in \Omega \times \Omega$. Set*

$$T_{xy} = \min\{t \in \mathbb{N} \mid x_t = y_t\}.$$

Then T_{xy} is a stopping time, and we have the upper bound

$$d_{TV}(P^t(x, \cdot), P^t(y, \cdot)) \leq \mathbb{P}(x_t \neq y_t) = \mathbb{P}(T_{xy} > t).$$

Proof. The proof we give is thanks to Diaconis [2, Chapter 4, Lemma 4].

Let $\Delta = \{(s, s) \mid s \in \Omega\} \subset \Omega \times \Omega$ be the diagonal and $\Delta^c = \Omega^2 \setminus \Delta$, and pick any $A \subseteq \Omega$. Let Q^t be the joint distribution of (x_t, y_t), that is

$$Q^t(\hat{x}, \hat{y}) = \mathbb{P}(x_t = \hat{x}, y_t = \hat{y}) \qquad \forall \hat{x}, \hat{y} \in \Omega.$$

Note that $P^t(x, A) = \mathbb{P}(x_t \in A) = Q^t(A \times \Omega)$ and similarly $P^t(y, A) = Q^t(\Omega \times A)$. The main idea is that therefore $|P^t(x, A) - P^t(y, A)| \leq Q^t(\Delta^c \cap A)$. In fact, we have

$$\begin{aligned}
&|P^t(x, A) - P^t(y, A)| = |Q^t(A \times \Omega) - Q^t(\Omega \times A)| \\
&= |Q^t((A \times \Omega) \cap \Delta) + Q^t((A \times \Omega) \cap (\Delta^c)) - (Q^t((\Omega \times A) \cap \Delta) + Q^t((\Omega \times A) \cap (\Delta^c)))|
\end{aligned}$$

Now since $(A \times \Omega) \cap \Delta = \{(a, a) \mid a \in A\} = (\Omega \times A) \cap \Delta$, only the terms outside the diagonal remain. In other words, one has that

$$|P^t(x, A) - P^t(A, y)| = |Q^t((A \times \Omega) \cap (\Delta^c)) - Q^t((\Omega \times A) \cap (\Delta^c))|$$

The right hand side is obviously at most $Q^t(\Delta^c)$, as it is a difference of two numbers between 0 and $Q^t(\Delta^c)$. However, $Q^t(\Delta^c) = \mathbb{P}(x_t \neq y_t)$, because

$$\mathbb{P}(x_t = y_t) = \sum_{s \in \Omega} \mathbb{P}(x_t = s \wedge y_t = s) = \sum_{s \in \Omega} Q^t(s, s) = Q^t(\Delta).$$

Thus $|P^t(x, A) - P^t(y, A)| \leq \mathbb{P}(x_t \neq y_t)$. Since A was an arbitrary subset of Ω we see that $d_{\mathrm{TV}}(P^t(x, \cdot), P^t(y, \cdot)) \leq \mathbb{P}(x_t \neq y_t)$.

To see that $\mathbb{P}(x_t \neq y_t) = \mathbb{P}(T_{xy} > t)$, remember that if (x_t, y_t) is a coupling and $x_s = y_s$ for some $s < t$, then also $x_t = y_t$. Thus if $x_t \neq y_t$ then also $T_{xy} > t$ and vice versa. □

It is in fact possible to show some Markov chains mix rapidly with purely the coupling lemma, however this becomes quite a lot of work. The problem is that you have to define transition probabilities for each pair of states (x, y), even if they are very far apart. The main issue with the coupling lemma is that the initial states x and y might be very far apart, and it can thus be hard to find a good way to couple them together. The idea is to only consider states that are very close together in some sense that might be useful for our specific chain. We then use this to implicitly define our coupling for states that are farther apart.

Definition 4.4. A neighbour relation on a state space Ω is a graph with vertex set Ω. If x and y are *neighbours*, we write $x \sim y$. (In particular, we can choose this relation to be anything we want, x and y don't have to be adjacent in the sense that $P(x, y) > 0$).

We can then consider paths in our state space obtained by jumping between neighbours. The general idea is that if one can couple two neighbours together quickly, and find a path of neighbours between any two arbitrary states, then by "jumping between neighbours" the two end states will eventually couple together. The precise construction will be shown in the next theorem.

Theorem 4.5 (Path Coupling)**.** *Let M be a Markov Chain with state space Ω and some neighbour relation $\sim$ on Ω. Let $\Phi : \Omega \times \Omega \to \{0, \dots, D\}$ be a metric such that for all $\sigma, \xi \in \Omega$ there exist $\eta_1, \dots, \eta_k \in \Omega$ such that $\eta_1 = \sigma, \eta_k = \xi$, $\eta_1 \sim \eta_2 \sim \cdots \sim \eta_k$ and*

$$\Phi(\sigma, \xi) = \sum_{i=1}^{k-1} \Phi(\eta_i, \eta_{i+1}).$$

Suppose there exists $\beta < 1$ and a coupling (σ_t, ξ_t) of M such that if $\sigma_t \sim \xi_t$ we have

$$\mathbb{E}[\Phi(\sigma_{t+1}, \xi_{t+1})] \leq \beta \Phi(\sigma_t, \xi_t). \tag{4.4}$$

Then

$$\tau(\varepsilon) \leq \frac{\log(D\varepsilon^{-1})}{\log(\beta^{-1})} \leq \frac{\log(D\varepsilon^{-1})}{1 - \beta}.$$

Proof. For any $\sigma, \xi \in \Omega$, let the coefficients $a_{\xi \to \xi' | \sigma \to \sigma'}(t)$ be the coefficients corresponding to the original coupling (as in Remark 4.2).

We start by constructing a new coupling $(\tilde{\sigma}_t, \tilde{\xi}_t)$. Pick any state $(\tilde{\sigma}_t, \tilde{\xi}_t)$. We wish to define a process to select $(\tilde{\sigma}_{t+1}, \tilde{\xi}_{t+1})$. If $\tilde{\sigma}_t = \tilde{\xi}_t$ pick $\tilde{\sigma}_{t+1}$ by simulating the Markov chain for one step starting at $\tilde{\sigma}_t$, and let $\tilde{\xi}_{t+1} = \tilde{\sigma}_{t+1}$.

Otherwise, pick some fixed path $\tilde{\sigma}_t = \eta_1, \dots, \eta_k = \tilde{\xi}_t$. We select new states η_i' according to the probabilities (here P denotes the matrix corresponding to the original Markov chain)

$$\begin{aligned} \mathbb{P}(\eta_1' = x) &= P(\eta_1, x), \\ \mathbb{P}(\eta_i' = y | \eta_{i-1}' = x) &= a_{\eta_i \to y | \eta_{i-1} \to x} \qquad \forall i \in \{2, \dots, k\}. \end{aligned}$$

We then let $(\tilde{\sigma}_{t+1}, \tilde{\xi}_{t+1}) = (\eta'_1, \eta'_k)$. What we're doing here is basically coupling together all of the η_i, η_{i+1} according to the original coupling.

To see that this defines a coupling, we only need to show that equations (4.1), (4.2) hold for $\tilde{\sigma}_t, \tilde{\xi}_t$, since condition (2) holds by definition. To do this, we show by induction on i that

$$\mathbb{P}(\eta'_i = y') = P(\eta_i, y').$$

The case $i = 1$ follows directly from the definition of η'_1. Now suppose it holds for $i-1$. Conditioning on η'_{i-1} we get

$$\begin{aligned}
\mathbb{P}(\eta'_i = y') &= \sum_{x' \in \Omega} \mathbb{P}(\eta'_i = y' \mid \eta'_{i-1} = x') \mathbb{P}(\eta'_{i-1} = x') \\
&= \sum_{x' \in \Omega} a_{\eta_i \to y' | \eta_{i-1} \to x'} \mathbb{P}(\eta'_{i-1} = x') \\
&= \sum_{x' \in \Omega} a_{\eta_i \to y' | \eta_{i-1} \to x'} P(\eta_{i-1}, x') \\
&= P(\eta_i, y'),
\end{aligned}$$

where the third line follows from the induction hypotheses and the last line follows since the original coupling was a coupling. By taking $i = 1$ and $i = k$ respectively, we get that equations (4.1), (4.2) hold for $\tilde{\sigma}_t, \tilde{\xi}_t$; hence this procedure defines a coupling.

Now since $\eta'_1 \sim \cdots \sim \eta'_k$ is a way of traversing Ω from $\tilde{\sigma}_{t+1}$ to $\tilde{\xi}_{t+1}$ (although not necessarily a path), we have by the triangle inequality that

$$\begin{aligned}
\mathbb{E}[\Phi(\tilde{\sigma}_{t+1}, \tilde{\xi}_{t+1})] &\leq \sum_{i=1}^{k-1} \mathbb{E}[\Phi(\eta'_i, \eta'_{i+1})] \\
&\leq \sum_{i=1}^{k-1} \beta \Phi(\eta_i, \eta_{i+1}) \qquad \text{(by 4.4)} \\
&\leq \beta \Phi(\tilde{\sigma}_t, \tilde{\xi}_t),
\end{aligned}$$

where we've used in the second line that we coupled η_i and η_{i+1} together according to the original coupling. If we do this for all possible states $(\tilde{\sigma}_t, \tilde{\xi}_t)$, we end up with a coupling $(\tilde{\sigma}_t, \tilde{\xi}_t)$ such that

$$\mathbb{E}[\Phi(\tilde{\sigma}_t, \tilde{\xi}_t)] \leq \beta^t \Phi(\tilde{\xi}_0, \tilde{\sigma}_0) \leq \beta^t D.$$

Thus for $t > \frac{\log(D\varepsilon^{-1})}{\log(\beta^{-1})}$ we have (since if $\tilde{\sigma}_t \neq \tilde{\xi}_t$ we have $\Phi(\tilde{\sigma}_t, \tilde{\xi}_t) \geq 1$)

$$\mathbb{P}(\tilde{\sigma}_t \neq \tilde{\xi}_t) \leq \mathbb{E}[\Phi(\tilde{\sigma}_t, \tilde{\xi}_t)] \leq \beta^t D = \left(e^{\log(\beta^{-1})}\right)^{-t} D < e^{-\log(D\varepsilon^{-1})} D = \varepsilon.$$

So for any fixed state $x \in \Omega$, if we let $\tilde{\sigma}_0 = x$ and select $\tilde{\xi}_0$ according to the invariant distribution π, then since π is an invariant distribution $\tilde{\xi}_t$ will still have the invariant

distribution. Since we defined a coupling, the marginal chain $\tilde{\sigma}_t$ is a copy of the original chain, therefore $\tilde{\sigma}_t$ has the same distribution as $P^t(x, \cdot)$.

Let $T = T_{\tilde{\sigma}_0, \tilde{\xi}_0}$ be the stopping time given by the smallest t such that $\tilde{\sigma}_t = \tilde{\xi}_t$. Recall that by Lemma 4.3 we had

$$d_{\mathrm{TV}}(\tilde{\sigma}_t, \tilde{\xi}_t) \leq \mathbb{P}(T > t) = \mathbb{P}(\tilde{\sigma}_t \neq \tilde{\xi}_t).$$

Thus for $t > \frac{\log(D\varepsilon^{-1})}{\log(\beta^{-1})}$ we have

$$d_{\mathrm{TV}}(P^t(x, \cdot), \pi) = d_{\mathrm{TV}}(\tilde{\sigma}_t, \tilde{\xi}_t) \leq \mathbb{P}(\tilde{\sigma}_t \neq \tilde{\xi}_t) < \varepsilon,$$

which is precisely what we needed to show by Definition 3.4. □

We can now immediately see why this theory is so powerful. To show that Glauber dynamics mix rapidly whenever $k > 2\Delta(G)$, all we need to do is define a coupling between two neighbouring states such that the expected distance of the states one step later is less than one.

Corollary 4.6. *Let G be a graph and $k \in \mathbb{N}$. Provided $k > 2\Delta(G)$, the mixing time of the Glauber dynamics with k colours on G is bounded above by*

$$\tau(\varepsilon) \leq \frac{k}{k - 2\Delta} n \log(n\varepsilon^{-1}).$$

Proof. We extend the state space Ω to the set of all colourings C^n. Since any proper colouring will remain a proper colouring after any transition, the set of proper colourings is a closed class. Since any improper colouring will eventually become a proper colouring, any improper colouring is a transient state and thus has weight 0 in any invariant distribution. Therefore, an invariant distribution of this extended chain must be assign 0 weight to any non-proper colourings, hence the invariant distribution will remain the same. It thus suffices to show this chain with extended state space mixes rapidly, since we can always interpret the original chain as this chain because starting at a proper colouring we stay at a proper colouring.

We take Φ to be the Hamming distance of two states, i.e. $\Phi(\sigma, \tau)$ is the number of vertices v such that $\sigma(v) \neq \tau(v)$. Since we extended the state space it's clear that all the paths we need for the path coupling theorem exist, simply recolour all different vertices one by one. We will define a neighbour relation on Ω by $\sigma \sim \tau$ if and only if $\Phi(\sigma, \tau) = 1$. We can obviously find a path with length $\Phi(\sigma, \tau)$ from σ to τ by recolouring the vertices in which they differ one by one.

Now given any two adjacent states σ_t, τ_t, note that they must differ in exactly one vertex v. We define our coupling (σ_t, τ_t) as follows. Obtain σ_{t+1} from σ_t by simulating the Glauber Dynamics for one step. If we obtain σ_{t+1} from σ_t by trying to recolour any vertex x not in $N(v)$ (in particular we might have $x = v$) with colour c, let τ_{t+1} also be given by recolouring that x with colour c. Given that σ_{t+1} is obtained from σ_t by trying

to color $w \in N(v)$ with colour c, we obtain τ_{t+1} from τ_t by trying to color vertex x in τ_t with color c', where c' is defined as

$$c' = \begin{cases} c & \text{if } c \notin \{\sigma_t(v), \tau_t(v)\}, \\ \tau_t(v) & \text{if } c = \sigma_t(v), \\ \sigma_t(v) & \text{if } c = \tau_t(v). \end{cases}$$

One can easily verify that both τ and σ still try to recolour each vertex with every colour with probability $\frac{1}{nk}$ (as we run the original chain on σ and only permute some colours sometimes to get the transition probabilities for τ), hence this procedure defines a coupling. Now for any $w \notin N(v)$, for any neighbour x of w we have $\sigma_t(x) = \tau_t(x)$. Hence attempting to color w with color c will succeed for σ_t if and only if it succeeds for τ_t, hence $\sigma_{t+1}(w) = \tau_{t+1}(w)$. Therefore, if $w \neq v$ we have $\Phi(\sigma_{t+1}, \tau_{t+1}) = \Phi(\sigma_t, \tau_t)$.

If $w = v$, then we have $\Phi(\sigma_{t+1}, \tau_{t+1}) = 0$ if there is no $x \in N(v)$ with $\sigma_t(x) = c$, and otherwise $\Phi(\sigma_{t+1}, \tau_{t+1}) = \Phi(\sigma_t, \tau_t)$. There are at most Δ colors c for which such x exists, so since there are k colors we have

$$\mathbb{E}[\Phi(\sigma_{t+1}, \tau_{t+1})| \text{ we attempt to recolour } v\ ] \leq \frac{\Delta}{k}\Phi(\sigma_t, \tau_t) + \frac{k-\Delta}{k} \times 0 = \frac{\Delta}{k}.$$

Finally, if $w \in N(v)$ then if $c \notin \{\sigma_t(v), \tau_t(v)\}$ we see that once again attempting to recolour σ_t will fail if and only if it fails for τ_t, so we still have $\Phi(\sigma_{t+1}, \tau_{t+1}) = \Phi(\sigma_t, \tau_t)$. If $c = \sigma_t(v)$ then by definition $c' = \tau_t(v)$ and we have $\sigma_{t+1} = \sigma_t, \tau_{t+1} = \tau_t$. And if $c = \tau_t(v)$ then we have $\Phi(\sigma_{t+1}, \tau_{t+1}) \leq \Phi(\sigma_t, \tau_t) + 1$ since both attempt to recolour w. Hence

$$\begin{aligned} \mathbb{E}[\Phi(\sigma_{t+1}, \tau_{t+1})| \text{ we recolour a fixed } w \in N(v)\ ] &\leq \frac{k-1}{k}\Phi(\sigma_t, \tau_t) + \frac{1}{k}\left(\Phi(\sigma_t, \tau_t) + 1\right) \\ &= \Phi(\sigma_t, \tau_t) + \frac{1}{k}. \end{aligned}$$

By conditioning on the vertex w we attempt to recolour we see that

$$\begin{aligned} \mathbb{E}[\Phi(\sigma_{t+1}, \tau_{t+1})] &\leq \frac{1}{n}\left(\sum_{w \notin N(v)\cup\{v\}} \Phi(\sigma_t, \tau_t) + \sum_{w \in N(v)} \left[\Phi(\sigma_t, \tau_t) + \frac{1}{k}\right] + \sum_{w=v} \frac{\Delta}{k}\right) \\ &= \frac{n-1}{n} \times \Phi(\sigma_t, \tau_t) + \frac{|N(v)|}{kn} + \frac{\Delta}{kn} \\ &\leq \frac{n-1}{n} \times \Phi(\sigma_t, \tau_t) + \frac{2\Delta}{kn} \qquad (|N(v)| \leq k) \\ &= \left(1 - \frac{k-2\Delta}{kn}\right)\Phi(\sigma_t, \tau_t). \end{aligned}$$

Since the Hamming distance between any two colourings is at most n, by Theorem 4.5 we see that the mixing time of the Glauber dynamics is bounded above by

$$\tau(\varepsilon) \leq \frac{k}{k-2\Delta} n \log(n\varepsilon^{-1}),$$

this completes the proof. □

4.2 Canonical Paths

In this section we discuss a different way of showing a Markov chain mixes rapidly, by constructing so-called canonical paths. The idea is that for any pair $x, y \in \Omega$, we have to route $\pi(x)\pi(y)$ units of flow from x to y, using the transition steps of the matrix as the "pipes" to route the flow through, where the transition probabilities could be seen as the "capacity" of the pipes. If we can route the flow evenly without creating pipes that are particularly congested, we can get a good upper bound on the mixing time.

We will use a lot of ideas from Jerrum [6, Section 5] in this chapter.

Definition 4.7. Let P be the transition matrix of a Markov chain with state space Ω. A a set of *canonical paths* on Ω is a set $\Gamma = \{\gamma_{xy}\}_{x,y\in\Omega}$, where each γ_{xy} is a tuple (path) of the form $(x = z_0, z_1, \ldots, z_l = y)$ of states such that $P(z_i, z_{i+1}) > 0$ for $i \in \{1, \ldots, l-1\}$. The *congestion* of a set of canonical paths is defined as

$$\rho(\Gamma) := \max_{t=(u,v)} \underbrace{\frac{1}{\pi(u)P(u,v)}}_{(\text{capacity of } t)^{-1}} \underbrace{\sum_{x,y:\gamma_{xy} \text{ uses } t} \pi(x)\pi(y)|\gamma_{xy}|}_{\text{total flow through } t},$$

where t runs over all possible transitions in the Markov Chain, and $|\gamma_{xy}|$ denotes the length l of γ_{xy}.

Later on, in Section 5.1 we will give an example of how to find a set of paths with low congestion. For now, we will try to find an upper bound on the mixing time if the congestion of a particular set of canonical paths is low. Thus assume ρ is small for some choice of paths Γ, we aim to show that the mixing time is small as well. For an arbitrary function $f : \Omega \to \mathbb{R}$, we define the *variance and expectation with respect to the probability measure* π as

$$\mathbb{E}_\pi f := \sum_{x\in\Omega} \pi(x)f(x), \qquad \mathrm{Var}_\pi(f) := \sum_{x\in\Omega} \pi(x)(f(x) - \mathbb{E}_\pi f)^2.$$

Recall that given two i.i.d. variables X_1, X_2 the identity $\mathrm{Var}(X_1) = \frac{1}{2}\mathbb{E}[(X_1 - X_2)^2]$ holds. This gives us the identity

$$\mathrm{Var}_\pi f := \frac{1}{2} \sum_{x,y\in\Omega} \pi(x)\pi(y)(f(x) - f(y))^2, \tag{4.5}$$

which shows that $\mathrm{Var}_\pi f$ basically measures the difference between $f(x)$ and $f(y)$ over all possible states, i.e. it is a measure for the global variation of f. We can similarly define

$$\mathcal{E}_P(f, f) := \frac{1}{2} \sum_{x,y\in\Omega} \pi(x)P(x,y)(f(x) - f(y))^2$$

as a measure for the expected square change of f after simulating the Markov Chain for one step (i.e. start at some point x, simulate the chain for one step and end up in

y, then look at $(f(x) - f(y))^2$). Note that $\mathcal{E}_P(f, f)$ depends on P, whereas the variance only depends on π. Recall that the congestion $\rho(\Gamma)$ was a measure for "how easy it is to move along the state space using transitions of P". Now if the congestion is low, then the probability of jumping from any x to y in a small number of steps is high. If the expected square change of f were also low, then one could imagine f cannot vary too much accross the state space. In other words, if $\rho(\Gamma)$ and $\mathcal{E}_P(f, f)$ are both low, one would expect $\operatorname{Var}_\pi f$ to also be low. This is made more precise in the following proposition.

Proposition 4.8. *Let P be the transition matrix of a Markov chain with state space Ω. For any function $f : \Omega \to \mathbb{R}$ and any set of canonical paths Γ, we have that*

$$\mathcal{E}_P(f, f) \geq \frac{1}{\rho(\Gamma)} \operatorname{Var}_\pi f.$$

Proof. The proof is fairly straightforward and can be found in Jerrum [6, Section 5, Theorem 5.2]. □

The general idea is to analyze how f changes as we apply to P to f. We can define an action from P on f in the same way we define an action from P on any probability distribution, by setting

$$[Pf](x) := \sum_{y \in \Omega} P(x, y) f(y).$$

This is basically just matrix-vector multiplication. Note that applying P preserves the average, in other words $\mathbb{E}_\pi P^t f = \mathbb{E}_\pi f$ for all $t \in \mathbb{N}$. If we think of f as the amount of 'flow' contained in a point, applying P basically distributes the flow of f over the state space by simulating the Markov chain for one step. To then show the Markov chain mixes rapidly, we want the 'flow' to distribute evenly (with respect to the invariant measure π) accross the state space in a short amount of time. In other words, we want $\operatorname{Var}_\pi P^t f$ to decay rapidly as t grows.

Since we are analyzing a dynamical system, we would really like to be able to do calculus (so that we could differentiate w.r.t. t). We thus define a continuous-time version of the same Markov chain. Consider the continuous time Markov chain $(\tilde{X}_s)_{|s \in \mathbb{R}}$ on Ω with transition rate matrix $Q = P - I$. This is a continuous Markov chain with exponentially-distributed holding times with mean 1, and jump probabilities equal to the transition probabilities of P.

Let $\tilde{P}^s(x, y) := \mathbb{P}(\tilde{X}_s = y \mid \tilde{X}_0 = x)$ denote the transition probabilities after s time has passed. By definition of Q we have that $\tilde{P}^s = e^{Qs}$. We define the action from $\tilde{P}^s$ on f in the same way we defined it for P. Now using calculus, we can show that the variance of this continuous chain decays rapidly.

Lemma 4.9. *Let P be the transition matrix of a Markov chain with state space Ω. Let $\tilde{P}^s$ be its continuous-time version, as defined above. For any function $f : \Omega \to \mathbb{R}$ and any set of canonical paths Γ, we have that*

$$\mathrm{Var}_\pi(\tilde{P}^s f) \leq e^{-2\frac{s}{\rho(\Gamma)}} \mathrm{Var}_\pi f. \tag{4.6}$$

Proof. The proof we give is inspired by Jerrum [6, Section 5.5]. Note that since both $\mathrm{Var}_\pi(\tilde{P}^s f)$ and $\mathrm{Var}_\pi f$ do not change when we add or subtract a constant to f, we may assume that $\mathbb{E}_\pi f = \mathbb{E}_\pi \tilde{P}^s f = 0$.

Starting with 4.5, we can now differentiate with respect to s and obtain

$$\begin{aligned}
\frac{d}{ds}\mathrm{Var}_\pi(\tilde{P}^s f) &= \frac{d}{ds}\frac{1}{2}\sum_{x,y\in\Omega}\pi(x)\pi(y)\left([\tilde{P}^s f](x) - [\tilde{P}^s f](y)\right)^2 \\
&= \sum_{x,y\in\Omega}\pi(x)\pi(y)\left([\tilde{P}^s f](x) - [\tilde{P}^s f](y)\right)\left(\frac{d}{ds}([\tilde{P}^s f](x) - [\tilde{P}^s f](y))\right) \\
&= \sum_{x,y\in\Omega}\pi(x)\pi(y)\left(2[\tilde{P}^s f](x)\frac{d}{ds}[\tilde{P}^s]f(x) - 2[\tilde{P}^s f](x)\frac{d}{ds}[\tilde{P}^s]f(y)\right).
\end{aligned}$$

The last line follows by multiplying out the product and using the symmetry in x and y. Now, since $\mathbb{E}_\pi \tilde{P}^s f = 0$ we have that

$$\sum_{x,y\in\Omega}\pi(x)\pi(y)[\tilde{P}^s f](x)\frac{d}{ds}[\tilde{P}^s f](y) = \left(\sum_{y\in\Omega}\pi(y)\frac{d}{ds}[\tilde{P}^s f](y)\right)\left(\sum_{x\in\Omega}\pi(x)[\tilde{P}^s f](x)\right) = 0.$$

Thus continuing, it follows that

$$\begin{aligned}
\frac{d}{ds}\mathrm{Var}_\pi(\tilde{P}^s f) &= 2\sum_{x\in\Omega}\pi(x)[\tilde{P}^s f](x)\frac{d}{ds}[\tilde{P}^s f](x) \\
&= 2\sum_{x,y\in\Omega}\pi(x)[\tilde{P}^s f](x)f(y)\frac{d}{ds}\tilde{P}^s(x,y) \\
&= 2\sum_{x,y\in\Omega}\pi(x)[\tilde{P}^s f](x)f(y)\left((P-I)\tilde{P}^s)(x,y)\right) \\
&= 2\sum_{x,y,z\in\Omega}\pi(x)[\tilde{P}^s f](x)f(y)(P(x,z)-I(x,z))\tilde{P}^s(z,y) \\
&= 2\sum_{x,z\in\Omega}\pi(x)(P(x,z)-I(x,z))[\tilde{P}^s f](x)[\tilde{P}^s f](z).
\end{aligned}$$

Setting $s = 0$ and summing over x, y instead, we obtain

$$\begin{aligned}\frac{d}{ds}\operatorname{Var}_\pi(\tilde{P}^s f)\Big|_{s=0} &= 2\sum_{x,y\in\Omega}\pi(x)(P(x,y)-I(x,y))f(x)f(y)\\ &= 2\sum_{x,y\in\Omega}\pi(x)P(x,y)f(x)f(y) - \sum_{x\in\Omega}\pi(x)f(x)^2\\ &= 2\sum_{x,y\in\Omega}\pi(x)P(x,y)f(x)f(y) - \frac{1}{2}\sum_{x,y\in\Omega}\pi(x)P(x,y)\left(f(x)^2+f(y)^2\right)\\ &= -2\mathcal{E}_P(f,f).\end{aligned}$$

By Proposition 4.8, we obtain

$$\frac{d}{ds}\operatorname{Var}_\pi(\tilde{P}^s f)\Big|_{s=0} \le -\frac{2}{\rho(\Gamma)}\operatorname{Var}_\pi f.$$

Now since f is an arbitrary function $\Omega \to \mathbb{R}$, by setting $f = \tilde{P}^r f$ we obtain for any $r > 0$

$$\frac{d}{ds}\operatorname{Var}_\pi(\tilde{P}^s f)\Big|_{s=r} = \frac{d}{ds}\operatorname{Var}_\pi\left(\tilde{P}^s\left(\tilde{P}^r f\right)\right)\Big|_{s=0} \le -\frac{2}{\rho(\Gamma)}\operatorname{Var}_\pi \tilde{P}^r f.$$

Hence for $s \ge 0$, the function $s \mapsto \operatorname{Var}_\pi \tilde{P}^s f$ is bounded above by the solution to the ordinary differential equation $\dot{v} = -(2/\rho)v$ with $v(0) = \operatorname{Var}_\pi f$, therefore we obtain for $s \ge 0$ the inequality

$$\operatorname{Var}_\pi(\tilde{P}^s f) \le e^{-2\frac{s}{\rho}}\operatorname{Var}_\pi f,$$

as desired. □

This bound on the continuous time can be translated to a bound on the discrete-time chain.

Lemma 4.10. *Let P be the transition matrix of a Markov chain with state space Ω. For any function $f : \Omega \to \mathbb{R}$ and any set of canonical paths Γ, we have that*

$$\operatorname{Var}_\pi(Pf) \le \left(1 - \frac{2}{\rho}\right)\operatorname{Var}_\pi f.$$

Proof. Consider once again the continuous time $\tilde{P}^s$ corresponding to P. Conditioning on the number of jumps at time s (which has distribution Poisson(s)) and dropping all terms but the first two, we obtain for any $s > 0$ the inequality

$$\begin{aligned}\operatorname{Var}_\pi \tilde{P}^s f &= \sum_{t=0}^{\infty}\frac{s^t e^{-s}}{t!}\operatorname{Var}_\pi P^t f\\ &\ge e^{-s}\operatorname{Var}_\pi f + se^{-s}\operatorname{Var}_\pi Pf.\end{aligned}$$

Using the previous bound from Lemma 4.9, we find that for any $s > 0$

$$\left(e^{-2\frac{s}{\rho}} - e^{-s}\right)\operatorname{Var}_\pi f \ge se^{-s}\operatorname{Var}_\pi Pf,$$

and thus

$$\operatorname{Var}_\pi Pf \leq \operatorname{Var}_\pi f \cdot \frac{e^{\left(1-\frac{2}{\rho}\right)s} - 1}{s}.$$

Taking the limit $s \to 0$ yields the desired bound. □

We can now easily bound the mixing time using our bound on the variance.

Theorem 4.11 (Canonical Paths)**.** *Let M be a Markov chain with state space Ω. For any set of canonical paths Γ on Ω, the mixing time of M is bounded above by*

$$\tau(\varepsilon) \leq \frac{\rho(\Gamma)}{2}(2\log(\varepsilon^{-1}) + \sup_{x\in\Omega} \log(\pi(x)^{-1})).$$

Proof. Fix some initial starting state x. Let $A \subseteq \Omega$ be some subset of the state space, and let $\mathbf{1}_A$ be the indicator function on the set A. It's clear that $\operatorname{Var}_\pi \mathbf{1}_A \leq 1$, and therefore by using Lemma 4.10 t times we see that

$$\operatorname{Var}_\pi(P^t\mathbf{1}_A) \leq \left(1 - \frac{2}{\rho}\right)^t \leq e^{\frac{-2t}{\rho}}.$$

Set

$$t = \left\lceil \frac{\rho}{2}(2\log\varepsilon^{-1} + \log(\pi(x)^{-1})) \right\rceil,$$

then

$$\operatorname{Var}_\pi P^t\mathbf{1}_A \leq e^{-2\log\varepsilon^{-1} - \log(\pi(x)^{-1})} = \varepsilon^2\pi(x).$$

Also

$$\begin{aligned}\operatorname{Var}_\pi P^t\mathbf{1}_A &\geq \pi(x)\left([P^t\mathbf{1}_A](x) - \mathbb{E}_\pi P^t\mathbf{1}_A\right)^2 \\ &= \pi(x)\left([P^t\mathbf{1}_A](x) - \mathbb{E}_\pi\mathbf{1}_A\right)^2.\end{aligned}$$

Now note that $[P^t\mathbf{1}_A](x) = \sum_{a\in A} P^t(x,a) = P^t(x,A)$. And obviously $\mathbb{E}_\pi\mathbf{1}_A = \pi(A)$. This implies that combining the two bounds we get

$$\varepsilon \geq |[P^t\mathbf{1}_A](x) - \mathbb{E}_\pi\mathbf{1}_A| = |P^t(x,A) - \pi(A)|.$$

Since A was an arbitrary subset of Ω, we conclude that $d_{\mathrm{TV}}(P^t(x,\cdot),\pi) \leq \varepsilon$. Taking the supremum over all x gives the desired bound on the mixing time. □

Thus we have shown that if one can find a set of canonical paths with low congestion, then the mixing time will become low as well. In the next chapter we shall take a look at finding such a set of canonical paths.

5 Constructing Samplers

In this chapter we use the theory from chapter 4 to construct more samplers by bounding the mixing of specific Markov chains. These samplers can then be used to construct a counting algorithm in a similar way as discussed in section 3.2.

5.1 Approximate Uniformly Sampling Independent Sets In Claw-Free Graphs

In this section we aim to construct a process to sample from the set of independent sets of some graph G. We will restrict our procedure to so-called claw-free graphs, since it's much easier to construct a rapidly mixing Markov chain on these graphs. We will adapt the method used in Jerrum [6, Section 5] to sample independent sets instead of matchings. We start off by defining a claw-free graph.

Definition 5.1. The *claw graph* is a graph with 4 vertices: $\{v, w_1, w_2, w_3\}$ and 3 edges: (v, w_i) for $i = 1, 2, 3$. A graph is said to be *claw-free* if there exists no subset $A \subseteq V(G)$ such that the induced subgraph $G[A]$ is isomorphic to the claw graph.

Remark 5.2. Note that in any claw-free graph G, if there exist 3 vertices w_1, w_2, w_3 that are all adjacent to some vertex v, then one of the edges (w_i, w_j) for some $i \neq j$ must be present in G since otherwise $\{w_1, w_2, w_3, v\}$ would induce a subgraph isomorphic to the claw graph.

Recall that a *independent set* is a subset A of $V(G)$ such that the induced subgraph contains no edges. We will define the *weight* of a independent set A to be $\lambda^{|A|}$, where λ is some positive constant. For many problems in statistical physics, one is then interested in sampling from the set of all independent sets with probability proportional to their weight, which gives us the distribution

$$\pi(A) = \frac{\lambda^{|A|}}{\sum_{B \subseteq V(G)} \lambda^{|B|}},$$

where the sum is over all independent sets $B \subseteq V(G)$.

We start by defining a Markov chain on the set of all independent sets.

Definition 5.3 (Independent set chain)**.** Define a Markov chain (X_t) with state space $\Omega = \Omega(G)$ the set of all independent sets on G, whose transition probabilities are described by the following procedure:

1. Select a vertex $v \in G$ uniformly at random.
2. There are four mutually exclusive possibilities:
   a) If $v \in X_t$, let $X' = X_t \setminus \{v\}$.
   b) If $v \notin X_t$ and $w \notin X_t$ for any $w \in N(v)$, let $X' = X_t \cup \{v\}$.
   c) If $v \notin X_t$ and there is exactly one $w \in N(v)$ that is a member of X_t, let $X' = (X_t \setminus \{w\}) \cup \{v\}$.
   d) If $v \notin X_t$ and $|N(v) \cap X_t| > 1$ let $X' = X_t$.
3. With *acceptance probability* $\frac{1}{2}\min\{1, \pi(X_t)/\pi(X')\}$, set $X_{t+1} = X'$, otherwise set $X_{t+1} = X_t$.

The acceptance probability in step (3) is there to ensure we arrive at the correct limiting distribution. It is often referred to as a Metropolis acceptance probability, see Norris [7, Section 5.5, p210] for a more detailed description. The factor $\frac{1}{2}$ is to make the chain 'lazy', to assure the chain is aperiodic.

Proposition 5.4. *Let G be a claw-free graph, and $\lambda > 0$. The independent set chain on G is ergodic, with limiting distribution π given by*

$$\pi(A) = \frac{\lambda^{|A|}}{\sum_{B \subseteq V(G)} \lambda^{|B|}}.$$

Proof. The chain is clearly aperiodic, since $P(A, A) \geq \frac{1}{2}$ for any state A since the acceptance probability in step (3) is at most $\frac{1}{2}$. And the chain is irreducible, since for any two independent sets A, A' we can move from A to A' by first removing all vertices from A using transitions of type (a), and then adding all of the vertices of A' with transitions of type (b).

We now wish to find the limiting distribution. Consider two independent sets A, A' with $P(A, A') > 0$ and assume without loss of generality $\pi(A) \leq \pi(A')$. Then by definition of the acceptance probability we have (note that there is exactly one vertex we can choose to move from A to A' or from A' to A)

$$P(A, A') = \frac{1}{2n}; \qquad P(A', A) = \frac{1}{2n}\frac{\pi(A)}{\pi(A')}.$$

We thus obtain

$$\pi(A)P(A, A') = \pi(A')P(A', A) = \frac{1}{2n}\min\{\pi(A), \pi(A')\}.$$

By Proposition 2.1 it follows that π is the limiting distribution, since the limiting distribution is unique. □

We now aim to bound the mixing time of the independent set chain. We will use the canonical paths method, so we start by defining the set Γ of canonical paths. Given two independent sets I (initial) and F (final), we need to construct a canonical path γ_{IF} from I to F in the adjacency graph on $\Omega(G)$ of the independent set chain. We start by looking at the difference between the two independent sets with a helpful subgraph, think of this as a sort of "symmetric difference".

Definition 5.5. Given a graph G and two independent sets I, F in G, we define the graph $\Delta_G(I, F)$ as the graph with vertex set $V(G)$, and edge set

$$E = \{e \in E(G) \mid e = (v_1, v_2) \text{ for some } v_1 \in I, v_2 \in F\}.$$

As I and F are independent sets it is clear that the edge set of $\Delta_G(I, F)$ is equal to the edge set of the subgraph of G induced by $I \cup F$, but it is helpful to think about it using this definition.

Proposition 5.6. *If G is a claw-free graph, then any vertex has degree at most 2 in $\Delta_G(I, F)$. Thus $\Delta_G(I, F)$ thus partitions into a bunch of even cycles $C_1, \ldots, C_k$, paths $P_1, \ldots, P_l$, and a set of isolated vertices $Q_1, \ldots, Q_r$.*

Proof. We show that any vertex in $v \in \Delta_G(I, F)$ has degree at most 2, the partitioning follows immediately. If $v \notin I$ and $v \notin F$ then clearly $N_{\Delta_G(I,F)}(v) = \emptyset$. If $v \in I$ and $v \in F$, then any neighbour of v can't be in I or F as both are independent sets, hence in this case we also have $N_{\Delta_G(I,F)}(v) = \emptyset$ by definition.

Thus suppose without loss of generality $v \in I, v \notin F$. If v were to have three neighbours in $\Delta_G(I, F)$, then v also has three neighbours in G, say w_1, w_2, w_3. By Remark 5.2, there most exist some edge between the $w_i \in G$, so without loss of generality we may assume $(w_1, w_2) \in E(G)$. Since $v \in I$, we must have $w_i \in F$ by definition of $\Delta_G(I, F)$. But then F is not a independent set since $(w_1, w_2) \in E(G)$. We conclude that v has at most two neighbours in $\Delta_G(I, F)$.

The partition follows immediately, to see that the cycles have even length note that they must alternate between vertices in I and F. □

We now construct our set of canonical paths Γ. Fix some ordering of the vertices of G. Order the components of $\Delta_G(I, F)$ by smallest vertex member. For each component we may identify a "start vertex", in the case of a cycle this will be the lowest vertex that is in I, in the case of a path it will be the lowest endpoint (which can be in I or F). We orient each path away from its start vertex. We orient each cycle by moving from the start vertex to the lowest of the two neighbours.

To get from I to F, we process the components of $\Delta_G(I, F)$ in their respective order. We process each component by the following procedure:

- For any isolated vertex Q_i that is contained in either I or F (but not both), we add or remove it using a transition of type (a) or (b). Any other isolated vertex Q_i (which is in both I and F or neither) we will skip entirely.

- For any path P_i, if the start vertex is in I, we first remove the start vertex using a transition of type (a). We then swap all vertices along the path using a transition of type (c), and in case of a path of odd length we will add the end vertex back using a transition of type (a). If the start vertex is not in I, we will start immediately with the type (c) transitions.

- For any cycle C_i, the start vertex is in I. Thus remove the start vertex using a transition of type (a). Then move along the orientaion of the cycle, swapping vertices using a transition of type (c). Add the final vertex adjacent to the start vertex back using a transition of type (b).

Note that none of our canonical paths ever use a transition of type (d), they will only use transitions of the form (A, A') with $A \neq A'$. Remember that the congestion involves a sum over all (I, F) for which γ_{IF} uses a specific transition $t = (A, A')$. We collect all those (I, F) in the following definition.

Definition 5.7. Let G be a claw-free graph, and $\Gamma = \{\gamma_{IF}\}$ some set of canonical paths from I to F for each $I, F \in \Omega(G)$. Then we define for any transition $t = (A, A')$ in the chain with $A \neq A'$

$$\mathrm{cp}(t) := \{(I, F) \mid t \in \gamma_{IF}\},$$

the set of all pairs (I, F) whose canonical path γ_{IF} uses transition t.

Since the congestion will eventually include a sum over all $(I, F) \in \mathrm{cp}(t)$, in order to bound the congestion it is helpful to bound $|\mathrm{cp}(t)|$. To bound the number of elements in $\mathrm{cp}(t)$, we now construct an (almost) injective encoding $\eta_t : \mathrm{cp}(t) \to \Omega$, which will allow us to bound $|\mathrm{cp}(t)|$ from above by something in size similar to $|\Omega|$. However, as constructing this injection is quite cumbersome, we need a good way of describing the current 'state', given some transition $t = (A, A')$ and $(I, F) \in \mathrm{cp(t)}$. To make describing such a state easier, we introduce some terminology. For now, we will always fix some transition $t = (A, A')$, and some $(I, F) \in \mathrm{cp}(t)$.

Definition 5.8. Let G be a claw-free graph and $\Gamma = \{\gamma_{IF}\}$ be the set of canonical paths for G we just defined. For some fixed transition $t = (A, A')$, and $(I, F) \in \mathrm{cp}(t)$ we define the following terminology:

Any vertex that is changed by t will be referred to as the current vertex. Note that there can be two current vertices, in a transition of type (c). The *current component* is the component in $\Delta_G(I, F)$ containing the current vertex (vertices). A component is *processed* if it comes before the current component in the ordering of the components of $\Delta_G(I, F)$, and *unprocessed* if it comes after the current component. If there are two current vertices, the current component is a cycle or a path. In that case, we will distinguish the first/second current vertex by order of appearance if we follow the cycle/path from the start vertex along its orientation.

A vertex in $\Delta_G(I, F)$ is called *processed/unprocessed* if it is in a processed/unprocessed component. Furthermore, if the current component is a path $P_i = (p_1, \ldots, p_k)$ with start vertex p_1, let j be such that p_j is the (first) current vertex. Any p_l with $l < j$ is also

said to be *processed* and any p_l with $l > j+1$ is said to be *unprocessed*. Similarly, if the current component is a cycle $C_i = (c_0, \ldots, c_k, c_{k+1} = c_0)$ (with start vertex c_0), let j be such that c_j is the (first) current vertex. Any c_l with $0 < l < j$ is also said to be *processed* and any c_l with $l > j+1$ is said to be *unprocessed*. We will refer to c_0 as the *special cycle vertex* (if it exists).

We are now ready to construct a independent set η_t.

Definition 5.9. Let G be a claw-free graph and $\Gamma = \{\gamma_{IF}\}$ be the set of canonical paths for G defined before. For some fixed transition $t = (A, A')$, and $(I, F) \in \mathrm{cp}(t)$, define $\eta_t(I, F) \subseteq \Delta_G(I, F)$ as the set of vertices

$$\eta_t(I, F) := \{w \in I \mid w \text{ is processed}\} \cup \{w \in F \mid w \text{ is unprocessed}\}.$$

In particular, all of the current and special cycle vertices (that exist) are not contained in $\eta_t(I, F)$.

We verify that this is well-defined.

Proposition 5.10. *Let G be a claw-free graph and $\Gamma = \{\gamma_{IF}\}$ be a set of canonical paths for G. For some fixed transition $t = (A, A')$, and $(I, F) \in cp(t)$, the set of vertices $\eta_t(I, F)$ defined above defines a independent set in G for any transition $t = (A, A')$ and $(I, F) \in cp(t)$.*

Proof. Since $\eta_t(I, F) \subseteq I \cup F$, it's clear that any edges present in the subgraph induced by $\eta_t(I, F)$ must be present in $\Delta_G(I, F)$, and thus contain one endpoint in F and one endpoint in I. Therefore any such edges must be included in some component P_i, C_i or Q_i. Since I and F are independent sets and $\eta_t(I, F)$ agrees with either I or F on any component that isn't the current component, the only way for $\eta_t(I, F)$ to include some edge from G is if it contains an edge in the current component. Clearly the current component must be a path or a cycle since a single point doesn't contain any edges.

If the current component is a path, then the current vertex (vertices) of the current component are not in $\eta_t(I, F)$. And if we follow the path along it's orientation, it agrees with I before the current vertex (vertices) and with F after the current vertex (vertices). Since I and F are independent sets, it can't contain any edges.

Finally, if the current component is a cycle, the argument for a path works in the same way, except that the edge involving (c_0, c_{2k}) might be present. But this can't happen since we explicitly let the special cycle vertex $c_0 \notin \eta_t(I, F)$. □

This shows that $\eta_t : \mathrm{cp}(t) \to \Omega$ is well defined. The following proposition shows that knowing $\eta_t(I, F)$ and (A, A'), we can almost recover $I \cup F$ completely, which will help us in making η_t injective.

Proposition 5.11. *Let G be a claw-free graph and $\Gamma = \{\gamma_{IF}\}$ be a set of canonical paths for G. For some fixed transition $t = (A, A')$ and $(I, F) \in cp(t)$, and any vertex $w \in I \cup F$ (that is not a current vertex or the special cycle vertex), one of the following two cases holds.*

I) w is in exactly one of I or F, and in exactly one of A or $\eta_t(I, F)$.

II) w is in both I and F, and in both A and $\eta_t(I, F)$.

Proof. Note that if w is processed, then $w \in A$ if and only if $w \in F$, and $w \in \eta_t(I, F)$ if and only if $w \in I$. And similarly, if w is unprocessed, then $w \in A$ if and only if $w \in I$, and $w \in \eta_t(I, F)$ if and only if $w \in F$. □

We can thus recover all of $I \cup F$, except for the special cycle vertex (if it exists). We can fix this by encoding an extra bit of information.

Definition 5.12. Let G be a claw-free graph and $\Gamma = \{\gamma_{IF}\}$ be a set of canonical paths for G. Fix some transition $t = (A, A')$. Define $\mu_t : \text{cp}(t) \to \{0, \ldots, n\}$ as follows: For any $(I, F) \in \text{cp}(t)$, let $\mu_t(I, F)$ be 0 if the special cycle vertex does not exist, and otherwise be the index of the special cycle vertex in the vertex ordering of G.

Proposition 5.13. *Let G be a claw-free graph and $\Gamma = \{\gamma_{IF}\}$ be a set of canonical paths for G. For every transition t, the function*

$$\eta'_t : cp(t) \to \Omega \times [n] : (I, F) \mapsto (\eta_t(I, F), \mu_t(I, F))$$

is injective.

Proof. Let $t = (A, A')$ be some transition, and $(I, F) \in \text{cp}(t)$. We wish to reconstruct I and F knowing only t and $\text{cp}(t)$, if we can do this then the mapping is clearly injective. By Proposition 5.11, $I \cup F$ can be recovered as a set since $A \cup A' \cup \eta_t(I, F)$ will contain all vertices in $I \cup F$, except the special cycle vertex (if it exists). But we can still find out if it exists and what vertex it is by looking at $\mu_t(I, F)$.

Since $\Delta_G(I, F)$ is simply the subgraph induced by $I \cup F$, we can thus retrieve all of the components Q_i, P_i and C_i. We know the order we were processing the components in since that was only determined by our fixed order of the vertices of G. We can thereby also figure out which vertices have been processed and which haven't, as that was only determined by our vertex ordering and we know the current vertex (the only vertex in $A \cap A'$).

We know if any vertex has been processed. Now any processed vertex is in I if and only if it is in $\eta_t(I, F)$, and in F if and only if it is in A. Any unprocessed vertex is in F if and only if it is in $\eta_t(I, F)$, and in I if and only if it is in A. We can thus tell, for any processed or unprocessed vertex whether it needs to be in I, F, or both.

Any current vertex is in I if and only if it is in A, and in F if and only if it is in A' (it can't be in both, because then we would have skipped it).

Finally, if we found out in which sets all other vertices are, we can then figure out where the special cycle vertex would be. If its neighbours are in I then it is in F and vice versa. It can't be in both, because then it would be an isolated point in $\Delta_G(I, F)$. □

Thus we have found an injective mapping from the set of canonical paths to a set we can bound in size. We can now use this to bound the congestion, as we do in the following lemmas.

Lemma 5.14. *Let G be a claw-free graph, $\lambda \in \mathbb{R}_{>0}$, and $\Gamma = \{\gamma_{xy}\}$ be a set of canonical paths on the collection of independent sets $\Omega(G)$. For all transitions $t = (A, A')$ and all pairs $(I, F) \in cp(t)$, it holds that*

$$\pi(I)\pi(F) \leq 2n\overline{\lambda}^2\pi(A)P(A, A')\pi(\eta_t(I, F)),$$

where $\overline{\lambda} = \max\{1, \lambda\}$.

Proof. Write $C = \eta_t(I, F)$. By Proposition 5.11 and since there are at most 2 current vertices and one special cycle vertex, $|I| + |F| \leq 3 + |A| + |C|$ (and the same holds for A' as they only differ in the current vertex/vertices). Thus

$$\pi(I)\pi(F) \leq \overline{\lambda}^3\pi(A)\pi(C), \qquad \pi(I)\pi(F) \leq \overline{\lambda}^3\pi(A')\pi(C),$$

and

$$\pi(I)\pi(F) \leq \overline{\lambda}^3 \min\{\pi(A), \pi'(A)\}\pi(C) = 2n\overline{\lambda}^3\pi(A)P(A, A')\pi(C),$$

as desired. □

We can now bound the congestion.

Lemma 5.15. *Let G be a claw-free graph and $\lambda \in \mathbb{R}_{>0}$. With the set of canonical paths Γ as defined above, the congestion $\rho(\Gamma)$ is bounded above by*

$$\rho(\Gamma) \leq 2n^2(n + 1)\overline{\lambda}^3.$$

Proof. This follows from a straightforward calculation, we see that

$$\begin{aligned}
\rho &= \max_{t=(A,A')} \left(\frac{1}{\pi(A)P(A, A')} \sum_{(I,F)\in \mathrm{cp}(t)} \pi(I)\pi(F)|\gamma_{IF}| \right) \\
&\leq \max_{t=(A,A')} \left(2n\overline{\lambda}^3 \sum_{(I,F)\in \mathrm{cp}(t)} \pi(\eta_t(I, F))|\gamma_{IF}| \right) && \text{(Lemma 5.14)} \\
&\leq 2n(n + 1)\overline{\lambda}^3\pi(\Omega)n && (\eta'_t \text{ is injective and } |\gamma_{IF}| \leq n) \\
&= 2n^2(n + 1)\overline{\lambda}^3,
\end{aligned}$$

which gives the desired bound. □

Note that our bound is weaker than Jerrum's by an extra factor of $n+1$ and λ, this is because we had to modify η_t a bit to be injective, and our chain can change two vertices at once adding an extra factor λ. We can now bound the mixing time of the independent set chain with the canonical paths theorem.

Theorem 5.16. *Let G be a claw-free graph on n vertices and $\lambda \in \mathbb{R}_{>0}$. The mixing time of the independent set chain described in Definition 5.3 on G is bounded above by*

$$\tau(\varepsilon) \leq n^2(n+1)\overline{\lambda}^2 \left(2\log(\varepsilon^{-1}) + n(|\log(1+\lambda)| + |\log(\lambda)|)\right).$$

Proof. By Theorem 4.11 and Lemma 5.15 together, we obtain the identity

$$\tau(\varepsilon) \leq n^2(n+1)\overline{\lambda}^2 \left(2\log(\varepsilon^{-1}) + \sup_{x\in\Omega} \log(\pi(x)^{-1})\right).$$

Now

$$\log(\pi(x)^{-1}) = \log\left(\frac{\sum_{B\subseteq V(G)} \lambda^{|B|}}{\lambda^{|x|}}\right) \leq \left|\log\left(\sum_{B\subseteq V(G)} \lambda^{|B|}\right)\right| + |\log(\lambda^{-|x|})|.$$

In the worst case, G doesn't have any edges and any $B \subseteq V(G)$ is an independent set. Hence we cannot do any better then crudely bounding the number of independent sets of size i by $\binom{n}{i}$, yielding

$$\sum_{B\subseteq V(G)} \lambda^{|B|} \leq \sum_{i=0}^{n} \binom{n}{i} \lambda^i = (1+\lambda)^n.$$

As the empty set is always an independent set, we also have $1 \leq \sum_{B\subseteq V(G)} \lambda^{|B|}$, thus

$$\left|\log\left(\sum_{B\subseteq V(G)} \lambda^{|B|}\right)\right| \leq n\log(1+\lambda).$$

Combined with the bound $|x| \leq n$ we obtain

$$\log(\pi(x)^{-1}) \leq n|\log(1+\lambda)| + n|\log(\lambda)|,$$

which gives the upper bound. □

5.2 Improving the Glauber Dynamics

In Section 3.2 we described a Markov Chain on the set of all k-colourings $\Omega_k(G)$ of a graph G, which turned out to be rapidly mixing whenever $k > 2\Delta(G)$. In this section we aim to find an improvement, and provide a way to construct a Markov Chain on $\Omega_k(G)$ that mixes rapidly whenever $k > \frac{11}{6}\Delta(G)$. The chain we provide was originally introduced by Vigoda [9] and all described here can be found in the corresponding article.

The key idea is to look at so called 'clusters' of vertices.

Definition 5.17. Let G be a graph, and $\sigma : G \to [k]$ be a (not necessarily proper) colouring of G, and $c \in [k]$. We will say that $(x_0, \dots, x_l)$ is a c_1, c_2-*alternating path* from v to w if $(x_i, x_{i+1}) \in E(G)$ for all i, and $\sigma(x_i) = c_1$ whenever $2 \mid i$ and $\sigma(x_i) = c_2$ otherwise (i.e. σ alternates between c_1 and c_2 along the path).

A *cluster* is a tuple (c_1, c_2, S) where $c_1, c_2 \in [k]$ and $S \subseteq V(G)$ such that

- There exists $v \in S$ with $\sigma(v) = c_1$.
- For any $w \in S$ there exists a c_1, c_2-alternating path from v to w.
- It is maximal, there is no strict superset $S' \subseteq V(G)$ satisfying the above two properties.

We further define the relation $(c_1, c_2, S) = (c_2, c_1, S)$ (so a cluster is actually an equivalence class of 2 tuples). Denote $\mathcal{C}_\sigma$ for the set of all clusters corresponding to a colouring σ.

Remark 5.18. Our new definition for clusters is to describe the conditional probabilities better later on, and to make it more clear which clusters are considered the same. We define, as Vigoda did, for any $v \in V(G)$ and $c \in [k]$ the cluster $S_\sigma(v, c) = (\sigma(v), c, S_\sigma(v, c))$ where

$$S_\sigma(v, c) = \{w \in V(G) \mid \text{There exists a } (\sigma(v), c)\text{-alternating path from } v \text{ to } w.\},$$

with the special case that $S_\sigma(v, c) = \emptyset$ whenever $c = \sigma(v)$.

Remark 5.19. Note that for $x \in S_\sigma(v, c)$ with $\sigma(x) = \sigma(v)$, we have $S_\sigma(x, c) = S_\sigma(v, c)$. Similarly, for $x \in S_\sigma(v, c)$ with $\sigma(x) = c$, we have $S_\sigma(v, c) = S_\sigma(x, \sigma(v))$.

We can now the define the actual Markov Chain.

Definition 5.20 (Flip Dynamics)**.** Define the constants $p_0 = 0$, $p_1 = 1$, $p_2 = \frac{13}{42}$ and for $7 > \alpha > 2$

$$p_\alpha = \max\left\{0, \frac{13}{42} - \frac{1}{7}\left(1 + \frac{1}{2} + \cdots + \frac{1}{\alpha - 2}\right)\right\}.$$

For $\alpha \geq 7$, set $p_\alpha = 0$. Define a Markov Chain on $[k]^{V(G)}$ with transition probabilities given by the procedure:

- Pick a cluster $(c_1, c_2, S) \in \mathcal{C}_\sigma$ with probablity $\frac{p_{|S|}}{nk}$.

- If none of the clusters are picked, do nothing.
- Otherwise, flip the colours c_1, c_2 for all vertices in S.

Remark 5.21. Note that any cluster can be described as some $S_\sigma(v, c)$, thus there are at most nk clusters. Since all the p_i are ≤ 1, we have that

$$\sum_{(c_1,c_2,S)\in\mathcal{C}_\sigma} \frac{p_{|S|}}{nk} \leq 1,$$

thus our transition probabilities are well-defined.

Remark 5.22. It is possible to simulate the above chain by selecting a vertex v and colour c uniformly at random, then flipping the cluster $S = S_\sigma(v, c)$ with probability $p_{|S|}/|S|$ or doing nothing otherwise. To see this, note that a cluster S has $|S|$ different ways of being picked in this way, as for each vertex in the cluster, the colour c required to flip the specific cluster is fixed. The probability of it being flipped by picking each specific vertex is $1/nk$, hence it is flipped with total probability $p_{|S|}/nk$.

Proposition 5.23. *Let G be a graph and $k \in \mathbb{N}$. The flip dynamics on $[k]^{V(G)}$ are ergodic whenever $k \geq \Delta(G) + 2$, with stationary distribution π that is nonzero and uniform on $\Omega_k(G) \subseteq [k]^V(G)$.*

Proof. Note that as long as $k \geq \Delta(G)+2$, for any colouring σ we have $P(\sigma, \sigma) > 0$ since for any v there exists a colour c not adjacent to v in the colouring, so that $S_\sigma(v, c) = \{v\}$ and $p_{|S_\sigma(v,c)|} > 0$. With similar reasoning as in Corollary 3.9 the chain will hit $\Omega_k(G)$ in finite time with probability 1. And once we are in $\Omega_k(G)$ the chain is aperiodic and irreducible, with similar reasoning as in Proposition 3.8, thus ergodic.

To see that the stationary distribution is uniform, note that all transition probabilities are symmetric on $\Omega_k(G)$. □

We aim to bound the mixing time of this chain using the path coupling theorem (Theorem 4.5). We take Φ to be the Hamming distance on Ω. To use the path coupling theorem, we need to define a coupling for any adjacent states σ, τ with $\Phi(\sigma, \tau) = 1$. These states differ in one vertex v, so from now on let $v = v_{\sigma,\tau}$ be the one vertex where $\sigma(v) \neq \tau(v)$.

We proceed by analyzing the clusters that appear in only one of the colourings, which are most interesting for the coupling (since for the rest the identity coupling will usually suffice).

Proposition 5.24. *Let $\sigma, \tau \in \Omega$ with $\Phi(\sigma, \tau) = 1$. For any $c \in [k]$, define*

$$\begin{aligned}
\Gamma_c &= \{w \in N(v) \mid \sigma(w) = \tau(w) = c\},\\
D_{\sigma,c} &= \{S_\sigma(v, c)\} \cup \{S_\sigma(w, \tau(v)) \mid w \in \Gamma_c\},\\
D_{\tau,c} &= \{S_\tau(v, c)\} \cup \{S_\tau(w, \sigma(v)) \mid w \in \Gamma_c\}.
\end{aligned}$$

Then it holds that

$$\mathcal{C}_\sigma \setminus \mathcal{C}_\tau \subseteq \bigcup_{c\in[k]} D_{\sigma,c},$$
$$\mathcal{C}_\tau \setminus \mathcal{C}_\sigma \subseteq \bigcup_{c\in[k]} D_{\tau,c}$$

for all $v \in V(G)$.

Proof. We show the first inclusion, the second is symmetrical in σ and τ.

Suppose we have some cluster $S_\sigma(x,c) \in \mathcal{C}_\sigma \setminus \mathcal{C}_\tau$ for some $x \in V, c \in [k]$. If $\sigma(x) = c$ and $\tau(x) = c$, then $S_\sigma(x,c) = (c,c,\emptyset)$. Since $S_\tau(x,c) \neq S_\sigma(x,c)$, we must have $\tau(x) \neq c$, thus $x = v$. This implies $S_\sigma(x,c) \in D_{\sigma,c}$ which suffices.

Otherwise, since $S_\tau(x,c) \neq S_\sigma(x,c)$, we must have $v \in S_\sigma(x,c)$ or $v \in S_\tau(x,c)$. If $v \in S_\sigma(x,c)$ then there exists $c' \in \{c, \sigma(x)\}$ such that $S_\sigma(x,c) = S_\sigma(v,c')$. Thus $S_\sigma(x,c) \in D_{\sigma,c'}$ which suffices.

Otherwise, since $v \in S_\tau(x,c)$ and τ and σ agree everywhere except at v, there exists an $(\sigma(x),c)$-alternating path (w.r.t. both the colourings σ and τ) from x to some vertex $w \in N(v) \cap S_\sigma(x,c)$. In particular, $\sigma(w) \in \{c, \sigma(x)\}$ and (we may also assume that) $\tau(w) \neq \tau(v)$.

If $\sigma(w) = c$, then also $\tau(w) = c$. Thus $\tau(v) \neq c$, and $\tau(v) = \sigma(x)$ because $v \in S_\tau(x,c)$. Thus $S_\sigma(x,c) = S_\sigma(w,\sigma(x)) = S_\sigma(w,\tau(v))$ as required.

Otherwise, if $\sigma(w) = \sigma(x)$, then also $\tau(w) = \sigma(x)$. Hence $\tau(v) \neq \sigma(x)$, so we also have $\tau(v) \neq \tau(x)$. Thus $\tau(v) = c$ since $v \in S_\tau(x,c)$, and $S_\sigma(x,c) = S_\sigma(w,\tau(v))$ as required. □

Remark 5.25. Suppose a cluster were to appear twice, say in some $D_{\sigma,c}$ and $D_{\sigma,c'}$ with $c \neq c'$. It can't appear as $S_\sigma(v,c)$ and $S_\sigma(v,c')$ as both those clusters have different colours. Similarly, it can't appear in $S_\sigma(w,\tau(v))$ and $S_\sigma(w',\tau(v))$ for $w \in \Gamma_c, w' \in \Gamma_{c'}$ as the first cluster is a $(c,\tau(v))$-cluster and the second a $(c',\tau(v))$-cluster.

Thus any such cluster appears (after possibly exchanging c and c') as $S_\sigma(v,c)$ and $S_\sigma(w,\tau(v))$ for some $w \in \Gamma_{c'}$. We conclude that $c = \tau(v)$ (as $\tau(v) \neq \sigma(v)$), and $\sigma(w) = \sigma(v)$. Thus $S_\sigma(v,\tau(v))$ is the only cluster that can appear in two of the $D_{\sigma,c}$, namely in $D_{\sigma,\tau(v)}$ and $D_{\sigma,\sigma(v)}$.

However, then there exists a $(\sigma(v),\tau(v))$-alternating path from v to w. By removing v from this path, we end up with a $(\sigma(v),\tau(v))$-alternating path between two neighbours of v. Name the other neighbour u, then we must have $\sigma(w) = \sigma(v)$ and $\sigma(u) = \tau(v)$. But then we also have $S_\tau(u,\sigma(v)) = S_\tau(v,\sigma(v))$ as the same path between u and w exists in τ.

Thus if this special case, of this special cluster appearing in two distinct $D_{\sigma,c}$, happens, it also happens for τ (and vice versa). Since this originates from Vigoda's case (*), we will refer to the clusters $S_\sigma(v,\tau(v)) = S_\sigma(w,\sigma(v))$ and $S_\tau(v,\sigma(v)) = S_\tau(u,\tau(v))$ as the σ^* and τ^*-clusters respectively.

We can now define our coupling.

Definition 5.26 (Flip Coupling)**.** To define a coupling all we need to do is define the probabilities of picking clusters S, T to flip σ and τ with. For brevity, write $\mathbb{P}(s,t) = \mathbb{P}(S = s, T = t)$. If $\Phi(\sigma,\tau) \neq 1$, we will simply select S and T independently using the original transition probabilities for the chain (with the extra requirement that if $\sigma = \tau$, we set $S = T$). Remember that to apply the path coupling theorem later on, we are only interested in 'adjacent' states σ, τ anyway (which we will define to be the states with $\Phi(\sigma,\tau) = 1$).

Set

$$\mathcal{E} = \mathcal{C}_\sigma \setminus \left(\bigcup_{c\in[k]} D_{\sigma,c} \cup D_{\tau,c} \right) \subseteq \mathcal{C}_\sigma \cap \mathcal{C}_\tau,$$

and let

$$\mathbb{P}(s,s) = \frac{p_{|s|}}{nk}$$

for any $s \in \mathcal{E}$, to couple all clusters that are in both σ and τ together. For the other clusters, we will only couple together clusters in $D_{\sigma,c}$ and $D_{\tau,c}$ for each colour c. By Remark 5.25 we may later analyze each of these couplings independently, as long as we take care of the double cluster that might appear. For each $c \in [k]$, let $\delta_c = |\Gamma_c|$ and split two cases to define some transitions corresponding to the sets $D_{\sigma,c}$ and $D_{\tau,c}$ (we will refer to these as *c-flips* for each c).

a) ($\delta_c = 0$) In this case $D_{\sigma,c} = \{S_\sigma(v,c)\}$ and $D_{\tau,c} = \{S_\tau(v,c)\}$. Then for this specific transition we set

$$\mathbb{P}(S_\sigma(v,c), S_\tau(v,c)) = \frac{1}{nk}.$$

b) ($\delta_c > 0$) Let $w_1, \ldots, w_{\delta_c} \in \Gamma_c$ denote all the neighbours of v with colour c in both σ and τ. Define

$$\begin{aligned} a_i &= a_i(c) = |S_\tau(w_i, \sigma(v))|, \\ b_j &= b_j(c) = |S_\sigma(w_j, \tau(v))|, \\ A &= A(c) = |S_\sigma(v,c)|, \\ B &= B(c) = |S_\tau(v,c)|. \end{aligned}$$

Since $S_\tau(w_i, \sigma(v))$ might be equal to $S_\tau(w_j, \sigma(v))$ for some $j < i$, if such j exists we redefine $a_i = 0$, and similar for the b_j. Furthermore, if we are in the special case $c = \tau(v)$ and $S_\sigma(v,c)$ is the σ^* cluster, we redefine $A(c) = 0$. Similarly, if $S_\tau(w_j, \sigma(v))$ is the τ^*-cluster, we will redefine $a_j(c) = 0$ (This way we only consider the flip of these $*$-clusters when $c = \sigma(v)$).

Let $i_{\max}$ be the least i for which a_i is maximal, and similarly $j_{\max}$.

The idea is to couple the big clusters together as much as possible, thus define

$$\mathbb{P}(S_\sigma(v,c),\ S_\tau(w_{i_{\max}}, \sigma(v))) = \frac{p_A}{nk}, \qquad (I)$$

$$\mathbb{P}(S_\sigma(w_{j_{\max}}, \tau(v)),\ S_\tau(v,c)) = \frac{p_B}{nk}. \qquad (II)$$

Then couple the remaining clusters together by defining

$$q_l = \begin{cases} p_{a_l} - p_A & \text{if } l = i_{\max} \\ p_{a_l} & \text{otherwise} \end{cases} \qquad q'_l = \begin{cases} p_{b_l} - p_B & \text{if } l = j_{\max} \\ p_{b_l} & \text{otherwise} \end{cases},$$

and setting

$$\mathbb{P}(S_\tau(w_l, \sigma(v)), S_\sigma(w_l, \tau(v))) = \frac{\min\{q_l, q'_l\}}{nk}, \tag{IIIa}$$

$$\mathbb{P}(S_\tau(w_l, \sigma(v)), \emptyset) = \frac{q_l - \min\{q_l, q'_l\}}{nk}, \tag{IIIb}$$

$$\mathbb{P}(\emptyset, S_\sigma(w_l, \tau(v))) = \frac{q'_l - \min\{q_l, q'_l\}}{nk}. \tag{IIIc}$$

Let all other probabilities by 0, and pick $\mathbb{P}(\emptyset, \emptyset)$ to be the exact number for the probabilities to sum to 1.

Remark 5.27. Note that for $c \neq \sigma(v)$ we have

$$S_\sigma(v, c) = \left\{ \bigcup_{i=1}^{\delta_c} S_\tau(w_i, \sigma(v)) \right\} \cup \{v\},$$

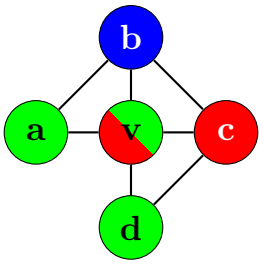
and for $c \neq \tau(v)$ we have

$$S_\tau(v, c) = \left\{ \bigcup_{i=1}^{\delta_c} S_\sigma(w_i, \tau(v)) \right\} \cup \{v\}$$

as sets of vertices (not cluster-triples). And if $c = \sigma(v)$ or $c = \tau(v)$ then $S_\sigma(v, c)$ resp. $S_\tau(v, c)$ consist of one element. Thus $A \leq 1 + \sum a_i$ and $B \leq 1 + \sum b_j$. Since the p_i are decreasing, this shows the q_l and q'_l are at least nonnegative.

As the definition isn't very concise, here is an example to get a better feel for what is going on.

Example 5.28. Consider the graph



denote σ for the colouring that colours the middle node green, and τ for the colouring that colours it red (σ and τ colour a, b, c, d according to the shown colouring). We will use the labels B = Blue, G = Green, R = Red for the encodings. We explicitly write down all non-empty clusters (thus excluding, for example, the cluster $S_\sigma(a, G, G)$).

Table 5.1: σ-clusters

Name	Encodings	Vertex Set	Colours	Size	Flip Probability
σ_1	$S_\sigma(a,B), S_\sigma(b,G), S_\sigma(v,B)$	{a, b, v}	Green, Blue	3	1/90
σ_2	$S_\sigma(a,R)$	{a}	Green, Red	1	1/15
σ_3	$S_\sigma(b,R), S_\sigma(c,B)$	{b, c}	Blue, Red	2	13/630
σ_4	$S_\sigma(d,B)$	{d}	Green, Blue	1	1/15
σ_5	$S_\sigma(v,R), S_\sigma(d,R), S_\sigma(c,G)$	{v, c, d}	Green, Red	3	1/90

Table 5.2: τ-clusters

Name	Encodings	Vertex Set	Colours	Size	Flip Probability
τ_1	$S_\tau(a,B), S_\tau(b,G)$	{a, b}	Green, Blue	2	13/630
τ_2	$S_\tau(a,R), S_\tau(v,G),$ $S_\tau(d,R), S_\tau(c,G)$	{a, v, d, c}	Green, Red	4	2/315
τ_3	$S_\tau(d,B)$	{d}	Green, Blue	1	1/15
τ_4	$S_\tau(v,B), S_\tau(c,B), S_\tau(b,R)$	{v, b, c}	Blue, Red	3	1/90

We find that $\sigma_4 = \tau_3$, yet all other clusters are distinct. Now we wish to calculate the exact transition probabilities for the coupling. We start of by finding $\Gamma_R = \{c\}$, $\Gamma_B = \{b\}$ and $\Gamma_G = \{a, d\}$. Then we find all the sets $D_{\sigma,C}$ and $D_{\tau,C}$ for $C \in \{R, B, G\}$.

$$D_{\sigma,R} = \{\sigma_5\} \qquad D_{\tau,R} = \{\tau_2\}$$
$$D_{\sigma,B} = \{\sigma_1, \sigma_3\} \qquad D_{\tau,B} = \{\tau_4, \tau_1\}$$
$$D_{\sigma,G} = \{\sigma_2, \sigma_5\} \qquad D_{\tau,G} = \{\tau_2\}$$

We thus see that the special case (*) does occur, with σ_5 as σ^* cluster, and τ_2 as τ^* cluster. Furthermore, we see that $\mathcal{E} = \{\sigma_4\} = \{\tau_3\}$.

We will thus start of by coupling σ_4 and τ_3 together by setting $\mathbb{P}(\sigma_4, \tau_3) = \frac{1}{15}$. We then continue to find the C-flips for each $C \in \{R, G, B\}$.

- **(R-flips)**. We have $\delta_R = 1$ and $w_1 = c$. Furthermore, we have the special case $c = \tau(v)$. Thus we explicitly set $A = 0$ as $S_\sigma(v, R)$ is the σ^* cluster. As $S_\tau(c, \sigma(v))$ is the τ^* cluster, we thus also set $a_1 = 0$. Furthermore we have $B = |\emptyset| = 0$, and $b_1 = |\emptyset| = 0$, and $i_{\max} = j_{\max} = 1$. As $p_0 = 0$, this case doesn't yield any positive probabilities.

- **(B-flips)**. We have $\delta_B = 1$ and $w_1 = b$. In this case, $A = |\sigma_1| = 3$, $B = |\tau_4| = 3$, $a_1 = |\tau_1| = 2$ and $b_1 = |\sigma_3| = 2$. Once again $i_{\max} = j_{\max} = 1$ as $\delta_B = 1$. Thus we obtain

$$\mathbb{P}(\sigma_1, \tau_1) = \frac{1}{90}, \qquad (I)$$
$$\mathbb{P}(\sigma_3, \tau_4) = \frac{1}{90}, \qquad (II)$$
$$\mathbb{P}(\sigma_3, \tau_1) = \frac{1}{105}, \qquad (IIIa)$$

since $q_1 = q_1' = \frac{1}{7}$ cases (IIIb) and (IIIc) give probability zero.

- **(G-flips)**. We have $\delta_G = 2$ and $w_1 = a, w_2 = d$. In this case, $A = |\emptyset| = 0$, $B = |\tau_2| = 4$, $a_1 = |\emptyset| = 0$, $a_2 = \emptyset = 0$, $b_1 = |\sigma_2| = 1, b_2 = |\sigma_5| = 3$. We have $i_{\max} = 1$ and $j_{\max} = 2$. As $A = 0$ case (I) yields a zero probability. We thus find

$$\mathbb{P}(\sigma_5, \tau_2) = \frac{2}{315}, \qquad (II)$$

Note now that $q_1 = 0, q_1' = p_1, q_2 = 0, q_2' = p_3 - p_4$. Hence only (IIIb) gives non-zero probability, which yields

$$\mathbb{P}(\sigma_2, \emptyset) = \frac{1}{15}, \qquad (IIIb)$$
$$\mathbb{P}(\sigma_5, \emptyset) = \frac{1}{210}. \qquad (IIIb)$$

since $q_1 = q_1' = \frac{1}{7}$ cases (IIIb) and (IIIc) give probability zero.

We have thus found all the coupling probabilities. To make clear what is going on, the following table shows the probability of flipping each pair of clusters we just calculated. The column/row named 'Total' shows for each cluster the total probability it gets flipped

Table 5.3: Coupling Probabilities

Clusters	τ_1	τ_2	τ_3	τ_4	$\emptyset$	Total
σ_1	$\frac{1}{90}$	0	0	0	0	$\frac{1}{90}$
σ_2	0	0	0	0	$\frac{1}{15}$	$\frac{1}{15}$
σ_3	$\frac{1}{105}$	0	0	$\frac{1}{90}$	0	$\frac{13}{630}$
σ_4	0	0	$\frac{1}{15}$	0	0	$\frac{1}{15}$
σ_5	0	$\frac{2}{315}$	0	0	$\frac{1}{210}$	$\frac{1}{90}$
$\emptyset$	0	0	0	0	0	
Total	$\frac{13}{630}$	$\frac{2}{315}$	$\frac{1}{15}$	$\frac{1}{90}$		

by the coupling, by which we verify that at least in this particular case, the coupling is well defined.

Lemma 5.29. *This procedure defines a coupling.*

Proof. One can easily verify that equations (4.1), (4.2) hold for all clusters that actually change a colouring. Since this becomes tedious quickly we only provide a sketch. The probability that $S = s$ for some s not in any of the $D_{\sigma,c}$ is obviously $\frac{p_{|s|}}{nk}$.. The probability of S becoming a cluster of the form $S_\sigma(v, c)$ with $\delta_c = 0$ is $1/nk$ by (a), and since $p_1 = 1$ this is exactly $\frac{p_{|S_\sigma(v,c)|}}{nk}$ as required. Finally per choice of the A, a_i, B, b_i the clusters belonging to case (b) also appear with their corresponding probabilities, since they all belong to exactly one colour c, except in the special case of a σ^*-cluster, but then we redefined the A, a_i for it to still work out exactly. □

Now to apply the path coupling theorem, we wish to bound

$$\mathbb{E}[\Delta\Phi(\sigma,\tau)] := \mathbb{E}[\Phi(\sigma',\tau')] - \Phi(\sigma,\tau).$$

away from 1, where the σ' and τ' are the colourings obtained by applying flips S and T to σ and τ respectively. Since our coupling is defined by a bunch of disjoint events, we can analyze each of these events individually. It's clear that a choice of the form (s,s) for $s \in \mathcal{E}$ leaves $\Phi(\sigma,\tau)$ intact. We thus only have to consider the c-flips for each colour c. For $c \in [k]$, let $\mathbf{1}_c$ be the r.v. that is 1 if and only if the coupling selects a c-flip, and 0 otherwise. Set $\Delta_c\Phi(\sigma,\tau) = \mathbf{1}_c \cdot \Delta\Phi(\sigma,\tau)$. Then clearly

$$\mathbb{E}[\Delta\Phi(\sigma,\tau)] = \sum_{c\in[k]} \mathbb{E}[\Delta_c\Phi].$$

Note that we lost the factor nk that Vigoda had since we're using probabilities instead of weight, to make it easier to understand the actual conditioning. The following lemma contains the heart of the calculation. It is quite cumbersome to proof, and we will only sketch some parts of its proof.

Lemma 5.30. *Let $\sigma,\tau \in \Omega$ with $\Phi(\sigma,\tau) = 1$. Then for each $c \in [k]$ we have*

a) If $\delta_c = 0$, then $\mathbb{E}[\Delta_c\Phi(\sigma,\tau)] = -\frac{1}{nk}$.

b) If $\delta_c > 0$, then $\mathbb{E}[\Delta_c\Phi(\sigma,\tau)] \leq \frac{1}{nk}\left(\frac{11}{6}\delta_c - 1\right)$.

Proof. We proof (a) and sketch (b).
a) In this case, remember that $D_{\sigma,c} = \{S_\sigma(v,c)\}$ and $D_{\tau,c} = \{S_\tau(v,c)\}$. There is only one c-flip, the move $(S_\sigma(v,c), S_\tau(v,c))$ which is selected with probability $1/nk$. As $\delta_c = 0$, both $S_\sigma(v,c)$ and $S_\tau(v,c)$ only contain the point v. Before the flip σ and τ disagree on v, and after the flip they agree on v (and remain the same everywhere else. Hence

$$\mathbb{E}[\Delta_c\Phi(\sigma,\tau)] = \frac{1}{nk}\left((\Phi(\sigma,\tau) - 1) - \Phi(\sigma,\tau)\right) = -\frac{1}{nk}.$$

b) In the case $\delta_c > 0$, consider the effect of each of the possible c-flips seperately. Let us for now ignore the special case of a σ^*,τ^* cluster appearing. For a move of type (I), the colourings still remain the same on the vertex set of the cluster $S_\tau(w_{i_{\max}}, \sigma(v)) \subseteq S_\sigma(v,c)$. As they also agree on v now, their Hamming distance has increased by at most $A - a_{\max} - 1$. For a move of type (II) we conclude by similar reasoning that the Hamming distance increases by at most $B - b_{\max} - 1$.

For a move of type (IIIa), the Hamming distance increases by at most $a_l + b_l - 1$ (as they both flip w_l we only have to take it once into account). And finally moves (IIIb) and (IIIc) increase the hamming distance by a_l and b_l respectively. We denote the effect of the flips (IIIa,b,c) with the function $f(w_l)$, defined as

$$f(w_l) = \frac{1}{nk}(a_l q_l + b_l q_l' - \min\{q_l, q_l'\}).$$

Then we thus have, by considering each of the events described above with their appropriate probability,

$$\mathbb{E}[\Delta_c\Phi(\sigma,\tau)] \leq \frac{1}{nk}\left((A-a_{\max}-1)p_A + (B-b_{\max}-1)p_B\right) + \sum_{l=1}^{\delta_c} f(w_l). \tag{5.1}$$

What remains is to carefully verify that for all of the variables a_i, b_i that may appear, the parameter choices satisfy

$$\frac{1}{nk}\left((A-a_{\max}-1)p_A + (B-b_{\max}-1)p_B\right) + \sum_{l=1}^{\delta_c} f(w_l) \leq \frac{1}{nk}\left(\frac{11}{6}\delta_c - 1\right).$$

The entire calculation takes Vigoda about 4 pages. We will instead consider only the case $\delta_c = 1$ to give some idea of what remains to be done. In this case, the cluster $S_\sigma(v,c)$ consists (as long as it's not empty) of the vertex set $\{v\} \cup \{S_\tau(w_1, \sigma(v))\}$. Hence $|A| \leq a_1 + 1$ and similarly $|B| \leq b_1 + 1$. Thus $q_1 = p_{a_1} - p_A$ and $q_1' = p_{b_1} - p_B$. We may assume $q_1 \geq q_1'$ without loss of generality. This gives us

$$\begin{aligned}
\mathbb{E}[\Delta_c\Phi(\sigma,\tau) &\leq \frac{1}{nk}\left((A-a_1-1)p_A + (B-b_1-1)p_B\right) + f(w_1) \\
&\leq \frac{1}{nk}\left((a_1+1-a_1-1)p_A + (b_1+1-b_1-1)p_B\right) + f(w_1) \\
&= \frac{1}{nk}(a_1q_1 + b_1q_1' - q_1') \\
&= \frac{1}{nk}(a_1(p_{a_1} - p_A) + (b_1-1)(p_{b_1} - p_B)) \\
&\leq \frac{1}{nk}(a_1(p_{a_1} - p_{a_1+1}) + (b_1-1)(p_{b_1} - p_{b_1+1})).
\end{aligned}$$

One easily verifies that for each of the $a, b \in \{1, \ldots, 7\}$ (as otherwise all p_a, p_b are 0 anyway) we have $(b-1)(p_b - p_{b+1}) \leq \frac{1}{7}$ and $a(p_a - p_{a+1}) \leq \frac{29}{42}$. From this we conclude

$$\mathbb{E}[\Delta_c\Phi(\sigma,\tau)] \leq \frac{1}{nk} \times \frac{5}{6} = \frac{1}{nk}\left(\frac{11}{6}\delta_c - 1\right),$$

as desired.

Finally, consider the special case of a σ^*, τ^* cluster appearing. Remember that the only cluster probabilities we redefined were those for $c = \tau(v)$. But for this particular case, remember that we explicitly defined $|A| = 0$ and $a_j = 0$, where j is such that $S_\tau(w_j, \sigma(v))$ is the τ^*-cluster. As $c = \tau(v)$, we also have $|B| = 0$ and $b_i = 0$ for all i, as the cluster $S_\tau(v, \tau(v)) = \emptyset$, and for $u \in \Gamma_{\tau(v)}$ also $S_\sigma(u, \tau(v)) = \emptyset$. Hence the only terms in equation 5.1 that remain are

$$\frac{1}{nk}\left(\sum_{1 \leq l \leq \delta_c, l \neq j} a_l p_{a_l}\right) \leq \frac{1}{nk}(\delta_c - 1)p_1 \leq \frac{1}{nk}\left(\frac{11}{6}\delta_c - 1\right),$$

which shows the required bound holds. □

Using this lemma we can now bound the mixing time of the flip dynamics.

Theorem 5.31. *Let G be a graph and $k \in \mathbb{N}$. Provided $k > \frac{11}{6}\Delta(G)$, the mixing time of the flip dynamics is bounded above by*

$$\tau(\varepsilon) \leq \frac{nk \log(n\varepsilon^{-1})}{k - \frac{11}{6}\Delta}.$$

Proof. Call two colourings σ, τ adjacent ($\sigma \sim \tau$) if $\Phi(\sigma, \tau) = 1$, and consider two adjacent states σ, τ. Let $v = v_{\sigma,\tau}$ the vertex in which σ and τ disagree, and $\delta = \delta(v)$ the degree of v in G. To apply the previous lemma, we need to find the number of colours c such that $\delta_c = 0$. As $\delta = \sum_{c:\delta_c>0} \delta_c$, we have the identity

$$\sum_{c:\delta_c=0} 1 = k - \sum_{c:\delta_c>0} 1 = k - \delta + \sum_{c:\delta_c>0} (\delta_c - 1). \tag{5.2}$$

This implies that

$$\begin{aligned}
\mathbb{E}[\Delta\Phi(\sigma,\tau)] &= \sum_{c:\delta_c=0} \mathbb{E}[\Delta_c\Phi(\sigma,\tau)] + \sum_{c:\delta_c>0} \mathbb{E}[\Delta_c\Phi(\sigma,\tau)] \\
&\leq \left(k - \delta + \sum_{c:\delta_c>0} (\delta_c - 1)\right)\left(-\frac{1}{nk}\right) + \sum_{c:\delta_c>0} \mathbb{E}[\Delta_c\Phi(\sigma,\tau)] \\
&\leq \left(k - \delta + \sum_{c:\delta_c>0} (\delta_c - 1)\right)\left(-\frac{1}{nk}\right) + \sum_{c:\delta_c>0} \frac{1}{nk}\left(\frac{11}{6}\delta_c - 1\right) \\
&= \frac{\delta - k}{nk} + \sum_{c:\delta_c>0} \frac{5}{6} \times \frac{\delta_c}{nk} \\
&\leq \frac{\frac{11}{6}\Delta - k}{nk},
\end{aligned}$$

where the second line follows from equation 5.2 and Lemma 5.30(a), the third line follows from Lemma 5.30(b), and in the last line we used $\delta \leq \Delta$ and $\sum_{c:\delta_c>0} \delta_c = \delta$. Hence if $\sigma_t \sim \tau_t$, we have that ($\Phi(\sigma_t, \tau_t) = 1$)

$$\begin{aligned}
\mathbb{E}[\Phi(\sigma_{t+1}, \tau_{t+1})] &= \Phi(\sigma_t, \tau_t) + \mathbb{E}[\Delta\Phi(\sigma, \tau)] \\
&\leq 1 - \frac{k - \frac{11}{6}\Delta}{nk} = \left(1 - \frac{k - \frac{11}{6}\Delta}{nk}\right)\Phi(\sigma_t, \tau_t).
\end{aligned}$$

Thus, applying Theorem 4.5 with $\beta = 1 - \frac{k - \frac{11}{6}\Delta}{nk}$, we see that the mixing time of the flip dynamics is bounded above by

$$\tau(\varepsilon) \leq \frac{\log(n\varepsilon^{-1})}{1 - \left(1 - \frac{k - \frac{11}{6}\Delta}{nk}\right)} = \frac{nk \log(n\varepsilon^{-1})}{k - \frac{11}{6}\Delta}.$$

□

6 Empirical Analysis

In this chapter we try and analyze the approximation scheme for counting colourings empirically, to see how good it actually performs in practice. An implementation of the algorithm described in Theorem 3.10 is given (in C++) in the appendix, see file 9.3.1.

Unfortunately, we can't provide a good analysis of the mixing speed of the chain, as that would require us to perform calculations that scale exponentially in either n or ε. Instead, we will verify that thee output of the algorithm agrees with the exact number of colourings, or at least doesn't vary too much between simulations.

6.1 Single algorithm simulations on cyclic graphs

We start off by running the algorithm on a cyclic graph, where we can find the exact answer using the chromatic polynomial. Recall that the chromatic polynomial of a cycle on n vertices is given by

$$\chi_{C_n}(k) = (k-1)^n + (k-1)(-1)^n.$$

The following table shows some results of running the algorithm once, with $k = 6$, $\varepsilon = 0.1$ and minimum succes probability $\frac{3}{4}$, for various n.

Table 6.1: Approximating colouring count, $k = 6$, $\varepsilon = 0.1$

n	6	7	8	9	10
Exact Answer	15630	78120	390630	1953120	9803980
Approximation	15738	78160	386467	1963260	9765630

We see that in this case, the algorithm performs about ten times better than expected. The error margin is usually about 0.01 instead of the 0.1 it should be in three out of four times.

We can't let k be much smaller, but as the running time is decreasing in k we can actually try this for very high k. This isn't a very interesting case, as for big k the approximation k^n gets really good anyway (as for k much bigger than n, most colourings will be proper anyway). Nevertheless, we will still try it (fix $n = 10$ this time).

Once again we see that the relative error bound remains at most 0.01 instead of the 0.1. (The reason there are so many trailing zeroes is because we're using the terminal to transfer information from C++ to python, and as C++ outputs floats with 6 digits in scientific notation we end up with only six digits of the actual answer, which is good enough for our tests here).

Table 6.2: Approximating colouring count, $n = 10$, $\varepsilon = 0.1$

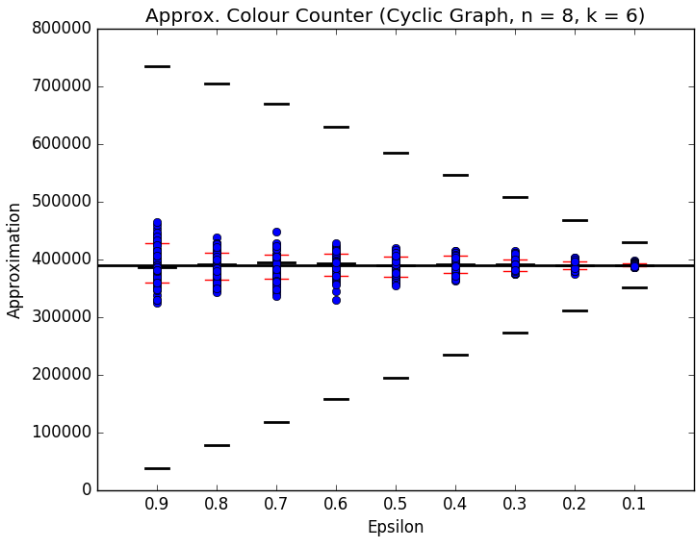
k	6	20	40
Exact Answer	1953120	6131066257820	8140406085191640
Approximation	1963260	6058320000000	8104130000000000

It is most interesting to vary epsilon (to see if the algorithm keeps performing about ten times better), but as the running time is square in ε we can't do many of those simulations. Furthermore, running the algorithm once doesn't really give a good idea about the general distribution of outputs the algorithm can give. We will therefore simulate the algorithm repeatedly in the next sections.

6.2 Repeated algorithm simulations on a cyclic graph

The previous section seems to suggest the algorithm performs a lot better than expected, i.e. the bounds in Theorem 3.10 can be improved. To get a good idea of what the output distribution of the algorithm looks like and varies as ε varies, we will now vary ε and perform 100 simulations for each ε.

The following picture shows the output of these simulations.

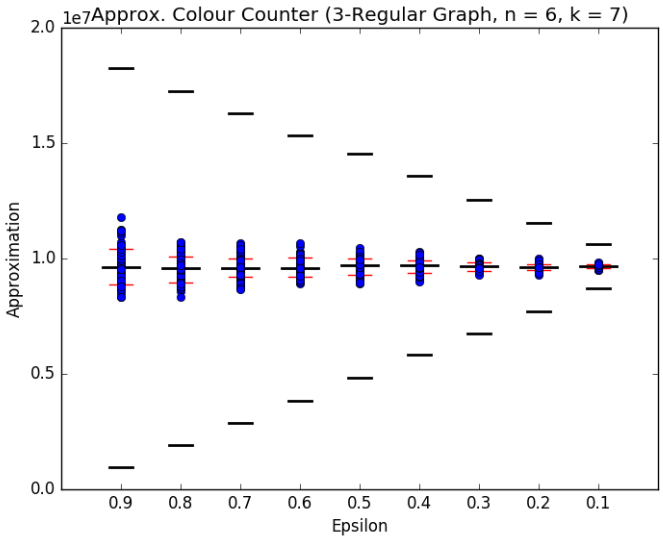


The long middle line is the actual answer (390630). The blue dots are the individual simulation results. The outer black lines are $(1 + \varepsilon)$ and $(1 - \varepsilon)$ times the median, and the short inner black lines are the median. The red lines indicate the middle 75% of the data, which is the only part we should be interested in as we have a succes probability of $\frac{3}{4}$.

6.3 Repeated algorithm simulations on a regular graph

We can do the same thing for a regular graph, except that we don't know the exact answer. We can thus only see how much it varies between different simulations.

The following picture shows the output of these simulations on a 3-regular graph with 6 vertices, when $k = 7$.



Note that there is no long middle line, as we don't know the actual answer. This picture seems to suggest that our approximation is very good, as the algorithm output doesn't vary too much. However, even if the algorithm were to output the same number each time, it doesn't need to be the exact answer. The problem is that our estimator is biased (all we know is that $\mathbb{E}[Y]$ is within $((1-\varepsilon)|\Omega_k(G), (1+\varepsilon)|\Omega_k(G)|)$.

However, the median of the results doesn't seem to vary as ϵ decreases. And if we let $\varepsilon \to 0$ the algorithm does start providing better approximations. In fact, we know for sure that $\lim_{\varepsilon \to 0} \mathbb{E}[Y(\varepsilon)] = \Omega_k(G)$. So this at least suggests that the algorithm does in fact perform a bit better then the bounds provided in Theorem 3.10.

7 Conclusion

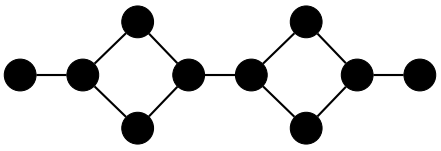
In section 3.2 have described the Glauber dynamics on a graph G with k colours. After developing some theory on mixing time of Markov chains, we then showed in Corollary 4.6 it mixes rapidly whenever $k > 2\Delta(G)$. Now it turns out that one can show the Glauber Dynamics also mix rapidly whenever $k > \frac{11}{6}\Delta(G)$, by using the flip dynamics we described in section 5.2, see Vigoda [9, Section 6]. However, in that case one finds that the mixing time (for some fixed ε) is bounded above by

$$\tau(\varepsilon) = O(n^2 \log(n)),$$

as opposed to the $\tau(\varepsilon) = O(n \log(n))$ we found for $k > 2\Delta(G)$. Recall that the Glauber Dynamics were ergodic whenever $k \geq \Delta(G) + 2$. An interesting open question that thus remains, is what happens when $\Delta(G) + 2 \leq k \leq \frac{11}{6}\Delta(G)$. The bound by Vigoda seems to suggest that the mixing time remains polynomial in n for $k < 2\Delta(G)$, however it might still become exponential as k approaches $\Delta(G)$. In particular, the running time might be of the form $n^{f(c)}$ whenever $k \leq c\Delta$, where $f(c)$ is finite for any $c > 1$ but $\lim_{c \to 1} f(c) = \infty$. However, there is too little known to state anything specific.

We also constructed a Gibbs sampler for independent sets in claw-free graphs in section 5.1. The construction of the chain itself easily generalizes to general graphs. One would like to know under what conditions the algorithm remains rapidly mixing.

Another thing one could look at is the behavior of the independent set chain as λ varies. Letting $\lambda \to \infty$ one can use the independent distribution $\pi(\lambda, A)$ to count the number of maximal independent sets, in a similar way as we did this for vertex colourings in the introduction (note that one can estimate π in a similar way as we did for the Glauber dynamics in section 3.2). This would suggest one could create an fpras for counting maximal independent sets in claw free graphs. However, unfortunately the mixing time also depends on λ, and to approximate this limit λ would have to be exponentially big in n as there are graphs in which the maximal matchings make an insignificant contribution to the limiting distribution unless λ is exponentially big in n. Consider for example graphs of the form



but with k rhombi instead of 2, these have 1 maximal independent set but 2^k independent sets with 1 vertex less. The question of whether there is an fpras for counting maximal independent sets in a claw free graphs thus remains open.

Bibliography

[1] C. Merino D. J. A. Welsh. The Potts model and the Tutte polynomial. *Journal Of Mathematical Physics*, 41(3), 2000.

[2] P. Diaconis. *Group representations in probability and statistics.* Institute of Mathematical Statistics, 1988.

[3] A. Goodall. The Tutte polynomial and related polynomials, 2010, 2012, 2014.

[4] W. Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 58(301), 1963.

[5] M. Jerrum. A very simple algorithm for estimating the number of k-colorings of a low-degree graph. *Random Structures and Algorithms*, 7(2), 1995.

[6] M. Jerrum. *Counting, Sampling and Integrating: Algorithms and Complexity.* ETH Zürich, 2003.

[7] J. R. Norris. *Markov Chains.* Cambride University Press, 1997.

[8] M. Dyer R. Bubley. Path Coupling: a Technique for Proving Rapid Mixing in Markov Chains. *Proceedings 38th Annual Symposium on Foundations of Computer Science*, 1997.

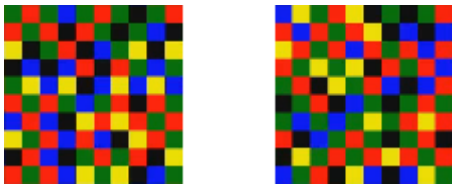
[9] E. Vigoda. Improved Bounds For Sampling Colourings. *Journal Of Mathematical Physics*, 41(3), 2000.

8 Populaire samenvatting

Het natuurkundige model voor (anti)ferromagnetisme bestaat uit een raster met deeltjes, bijvoorbeeld een vierkant van 10 bij 10 (met 100 deeltjes totaal). Op een vast tijdstip kan elk deeltje zich in een vast aantal toestanden bevinden. De toestand waarin een deeltje zich bevind wordt de *spin* van het deeltje genoemd. In het meest eenvoudige model bestaan er enkel de spins "+" en "-". en bestaat één configuratie van het model dus simpelweg uit een vierkant van plusjes en minnetjes. Er bestaan echter ook ingewikkeldere modellen, met bijvoorbeeld vier, vijf of in het algemene geval k mogelijke spins. In plaats van deze spins aan te duiden met plus of min, kunnen we ze ook aanduiden met een kleur, bijvoorbeeld rood, groen of blauw.

In het algemene model voor magnetisme hebben deeltjes de neiging om dezelfde spin te hebben als hun buren. Echter, in sommige metalen (bijvoorbeeld chroom) blijkt dat deeltjes de neiging hebben om een andere spin te hebben dan hun buren. Dus als twee deeltjes naast elkaar zitten, en de temperatuur is erg laag, dan is het onwaarschijnlijk dat ze allebei "blauw" zijn.

In dit model zijn er echter heel veel mogelijke configuraties, hieronder staan er twee weergegeven in het geval dat we met 5 kleuren werken.



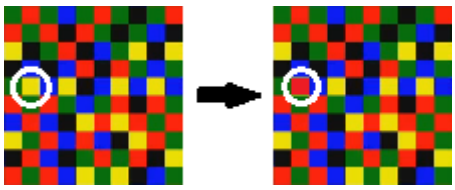
Een interessante vraag voor wiskundigen is hoeveel mogelijke configuraties er zijn. Een formule voor het aantal configuraties blijkt lastig te vinden, en het zijn er veel te veel om allemaal af te gaan met een computer. Dus het vinden van een exacte oplossing is lastig, en in plaats daarvan zijn we geïnteresseerd in het vinden van een manier om de oplossing (snel) te kunnen benaderen met een computer.

Het idee is om met behulp van een algoritme eerst een (bijna) willekeurige configuratie te genereren. Als men dan eenmaal een manier heeft gevonden om willekeurige configuraties te trekken, kan men dat gebruiken om het totale aantal configuraties te schatten.

Helaas kunnen we niet gewoon alle punten een willekeurige kleur geven en dan hopen dat het een geldige configuratie is, omdat dit bijna nooit een geldige configuratie oplevert. In meer wiskundige termen, als we op deze manier een configuratie genereren (op een $n \times n$ vierkant), dan zal de kans dat het een geldige configuratie van de orde $\frac{1}{k^n}$ zijn. Dit betekent dat als we op deze manier een geldige configuratie wilden genereren, we

iets van k^n pogingen nodig hebben. Als n heel groot wordt zal dit al gauw veel te lang duren.

In plaats daarvan kunnen we een random walk op de collectie van alle configuraties maken, door telkens een configuratie in één deeltje te veranderen. In het plaatje hieronderen zien we één stap van deze random walk, door het omcirkelde deeltje van geel naar rood te veranderen.



Als we dat een aantal keer doen (vaak genoeg dat we elke deeltje minstens k keer raken, dus zeg kn^2 keer) dan lijkt onze configuratie misschien wel op een willekeurige geldige configuratie.

Om in te zien dat dit nieuwe algoritme veel sneller is dan het willekeurige algoritme, is hier een tabel met de waarden van $5n^2$ en 5^n voor verschillende waarden van n.

Table 8.1: Snelheid van algoritmen

n	1	2	5	10	100
$5n^2$	5	20	125	500	50.000
5^n	5	25	3125	9765625	een getal met 69 cijfers

Men ziet dat 5^n al gauw vele malen groter wordt dan $5n^2$. Dit komt omdat de n in de exponent staat in plaats van op de plek van het grondtal (met de 2 als exponent). Indien de n in de exponent staat wordt gezegd dat een algoritme in *exponentiële tijd* kan worden uitgevoerd. Als de n in het grondtal staat (dus bij de $5n^2$) dan zegt men dat het algoritme in *polynomiale tijd* kan worden uitgevoerd. De algoritmes in polynomiale tijd zijn (voor grote n) altijd veel sneller, en men is dus altijd op zoek naar dit soort algoritmen.

In deze scriptie wordt uitgerekend hoe vaak je de boven uitgelegde stap (van het veranderen van de kleur van één deeltje) moet herhalen om op een willekeurige configuratie uit te komen. Ook wordt hetzelfde proces geanalyseerd op veel ingewikkeldere structuren dan een rooster (zogeheten *grafen*), en worden enkele soortgelijke processen geanalyseerd. Daarnaast wordt uitgelegd hoe je een algoritme om willekeurige configuraties te kiezen kan gebruiken om het totaal aantal configuraties te tellen. Tot slot wordt theorie beschreven om goed over dit soort processen te kunnen nadenken, en te kunnen bewijzen dat de processen echt werken, iets wat wiskundigen heel belangrijk vinden.

9 Appendix

9.1 Build and Usage Instructions

9.1.1 Building and Installing

The python program "approximate_counting.py" requires an executable called "graph_colouring_approximator" to be present in the same directory. This executable is build from the corresponding "graph_colouring_approximator.cpp". To build this file, a C++ compiler has to be installed on your computer, and some build system (I suggest cmake: `https://cmake.org/download/` along with either the MSVC Build tools (Windows) or the GNU compilers (Unix)).

Create a new file called "CMakeLists.txt" in the same directory as the ".cpp" file, containing the following:

```
cmake_minimum_required (VERSION 2.6)
project (graph_colouring_approximator)
add_executable(graph_colouring_approximator
    graph_colouring_approximator.cpp)
```

Open a shell in the directory containing "graph_colouring_approximator.cpp" and "Cmakelists.txt". Then generate the build files

```
> cmake .
```

and build the executable

```
> cmake --build .
```

I recommend building a release version as well, as it appears to be around 100 times faster. For windows, this can be done using:

```
> cmake --build . --config Release
```

For Unix systems one should generate new build files first, then rebuild the application.

```
> cmake -DCMAKE_BUILD_TYPE=Release .
> cmake --build .
```

Then copy the generated executable into the folder with the python program.

Both python programs require networkx 1.11. This can be installed using pip by typing

```
> pip install networkx==1.11 (--upgrade)
```

9.1.2 Using the programs

The python program "approximate_counting.py" can be used for 2 things. After invoking the program with

```
> python approximate_counting.py
```

one can select one of two functions:

1. Approximately counting colourings on a cyclic graph (i.e: running Jerrums algorithm once). Simply invoke the program, press 1, input the length of the cycle, number of colours k, error bound and succes probability, and it will calculate an approximation for the total number of k-colourings on the cycle. It will also output the exact result to allow you to compare the two.

2. Approximately counting colourings on cyclic/regular graphs for varying epsilon (100 times each), then generate a plot containing all the samples. This allows one to monitor the behavior of the algorithm for varying epsilons. Simply select the graph type you want, input the various constants, then input a list of epsilons seperated by spaces, and it will generate a plot.

The python program "gif_generation.py" can be used to generate a video simulating the glauber dynamics on a 100 by 50 grid. Simply invoke the program

```
> python gif_generation.py
```

and input the number of frames to generate, the number of simulation steps per frame, the number of miliseconds each frame will show and finally the filename.

I would suggest not invoking the C++ program directly, as it requires quite a lot of arguments to be inserted in a very specific order. The main reason this program is here is to speed up the calculations performed by "approximate_counting.py", as python turned out to add way too much overhead to do these calculations efficienty.

9.2 Python Code

9.2.1 approximate_counting.py

```
"""
Python program for approximately counting colourings.
"""
import statistics
import networkx as nx
import math
import matplotlib.pyplot as pyplot
import subprocess
import numpy as np


if nx.__version__[0] != "1":
    print("This script is written for an older version of networkx, please roll back to version
        1.11")
    exit()

# Constant determining the number of trials used for generating boxplots when
# trying to find constant efficiency.
BOXPLOT_TRIALS = 100


def exact_count_cycle_colourings(n, k):
    """
    Exactly counts the number of colourings on a cyclic graph for comparing.
    :param n: (int) Length of cycle
    :param k: (int) Number of colours
    :return: (int) Number of colourings
    """
    return (k - 1) ** n + ((-1) ** n) * (k - 1)


def ap_co_custom_constants(graph, k, time, s, trials):
    """
    C++ wrapper for approximately counting colourings.
    Calls the corresponding C++ program to do the actual counting.
    :param graph: (nx.Graph) graph to count colourings on
    :param k: (int) number of colours
    :param time: (int) number of steps to simulate Glauber Dynamics for sampling
    :param s: (int) number of z_i's to approximate mu_i
    :param trials: (int) number of (simultaneous) approximations to compute
    :return: (list of float) list of approximations
    """

    # extract edges and vertices from graph
    n = graph.number_of_edges()
    edge_list = []
    for x, y in graph.edges():
        edge_list.extend([str(x), str(y)])

    # call C++ program
    arg_list = ["graph_colouring_approximator.exe", str(n), str(k), str(time), str(s), str(
        trials)]
    arg_list += edge_list
    pipe = subprocess.PIPE
    process = subprocess.Popen(arg_list, stdin=pipe, stdout=pipe, stderr=pipe)

    # retrieve output
    output, err = process.communicate()
    output = output.decode("utf-8")
    return list(map(lambda l: int(float(l)), output.split()))


def approximate_colouring_count(graph, k, epsilon, p_success):
    """
    Approximates the total number O of proper colourings for a graph G.
    Returns a number that lies in ((1 - e)O, (1 + e)O)
    :param graph: (nx.Graph) Graph to count colourings on
    :param k: (int) Number of colours to use for colouring
    :param epsilon: (float) error bound e
    :param p_success: (float) success probability p
    :return: (int) Approximation given by the procedure described in section 3.1
    """
    # For a good explanation of this implementation, look over section 3 in the report

    # Calculate number of approximations for the median
    trials = int(math.ceil((8 * math.log(1 / (1 - p_success))) ** 1 / 3))

    # Calculate all required constants
```

```
    m = len(graph.edges())
    n = len(graph.nodes())
    d = max(graph.degree(graph.nodes()).values())
    time = math.ceil((k / (k - 2 * d)) * n * math.log(4 * n * m / epsilon))  # big T in the
        report
    s = math.ceil(37 * m / (epsilon ** 2))

    print("Simulating trials of {0} samplings with simulation time {1}.".format(s, time))

    return statistics.median(ap_co_custom_constants(graph, k, time, s, trials))


def generate_plots_varying_epsilon(graph, k, epsilons, title="", exact=-1):
    """
    Generates a series of boxplots measuring precision of algorithm while varying time constant
        .
    To make things easier, we will fix the s constant to 500.
    :param graph: (nx.Graph) Graph to approximate on.
    :param k: (int) Number of colours to use
    :param epsilons: (list of float) epsilons to try, should be between 0 and 1
    :param title: (str) Title to display above the graph
    :param exact: (int) Exact answer to draw on the graph. If this is set to -1, this line isn'
        t drawn.
    """

    m = len(graph.edges())
    n = len(graph.nodes())
    d = max(graph.degree(graph.nodes()).values())

    # List of lists of sample results for each k
    results = []

    fig, ax = pyplot.subplots()

    for i, epsilon in enumerate(epsilons):
        assert 0 < epsilon < 1
        print("Generating data for epsilon = {0}".format(epsilon))

        # Calculate T and s
        time = math.ceil((k / (k - 2 * d)) * n * math.log(4 * n * m / epsilon))
        s = math.ceil(37 * m / (epsilon ** 2))
        print("s = {0}, T = {1}".format(s, time))

        # Generate actual approximations
        result = ap_co_custom_constants(graph, k, time, s, trials=BOXPLOT_TRIALS)
        results.append(result)

        # Plot lines mean and (1 +- epsilon)*median
        median = statistics.median(result)
        pyplot.plot([i + 0.7, i + 1.3], [median, median], 'k-', lw=2)
        pyplot.plot([i + 0.8, i + 1.2], [(1 - epsilon) * median, (1 - epsilon) * median], 'k-',
            lw=2)
        pyplot.plot([i + 0.8, i + 1.2], [(1 + epsilon) * median, (1 + epsilon) * median], 'k-',
            lw=2)

        # Plot lines indicating middle 75% of data
        lower_percentile = np.percentile(result, 12)
        upper_percentile = np.percentile(result, 87)
        pyplot.plot([i + 0.8, i + 1.2], [lower_percentile, lower_percentile], 'k-', color='r')
        pyplot.plot([i + 0.8, i + 1.2], [upper_percentile, upper_percentile], 'k-', color='r')

    if exact != -1:
        pyplot.plot([0, len(epsilons) + 1], [exact, exact], 'k-', lw=2)

    # Plot all data points
    pyplot.plot(range(1, len(epsilons) + 1), results, "bo")

    # Label axes, title
    pyplot.xlabel("Epsilon")
    ax.set_xlim(0, len(epsilons) + 1)
    pyplot.ylabel("Approximation")
    pyplot.title(title)
    pyplot.xticks(range(1, len(input_epsilons) + 1), input_epsilons)

    pyplot.show()


if __name__ == "__main__":
    print("Select action by pressing number key:")
    print("[1] Approximately count colourings on cyclic Graph")
    print("[2] Measure precision by varying epsilon")
    option = input()

    # Approximately Count Colourings UI
```

```
if option == "1":
    print("Length of cycle:")
    input_n = int(input())
    print("Number of colours:")
    input_k = int(input())
    assert input_k >= 3
    print("Error tolerance (epsilon):")
    input_epsilon = float(input())
    assert 0 < input_epsilon < 1
    print("Minimum success probability:")
    input_prob = float(input())
    assert 0 <= input_prob < 1
    test_graph = nx.cycle_graph(input_n)
    approx_answer = approximate_colouring_count(test_graph, input_k, input_epsilon,
        input_prob)
    print("Approximate answer: {0}".format(approx_answer))
    print("Exact answer: {0}".format(exact_count_cycle_colourings(input_n, input_k)))

# Boxplots varying time UI
elif option == "2":
    print("[1] Cyclic Graph")
    print("[2] Random Regular Graph")
    option = input()

    if option == "1":
        print("Length of cycle:")
        input_n = int(input())
        print("Number of colours:")
        input_k = int(input())
        assert input_k >= 3
        test_graph = nx.cycle_graph(input_n)
        print("List of epsilons to try:")
        input_epsilons = list(map(float, input().split()))
        exact_answer = exact_count_cycle_colourings(input_n, input_k)
        title_str = "Approx. Colour Counter (Cyclic Graph, n = {0}, k = {1})"
        title_input = title_str.format(input_n, input_k)
        generate_plots_varying_epsilon(test_graph, input_k, input_epsilons, title_input,
            exact_answer)

    elif option == "2":
        print("Number of vertices:")
        input_n = int(input())
        print("Degree:")
        input_d = int(input())
        print("Number of colours:")
        input_k = int(input())
        assert input_k >= input_d + 1
        assert (input_n * input_d) % 2 == 0
        assert input_d <= input_n - 1
        test_graph = nx.random_regular_graph(input_d, input_n)
        print("List of epsilons to try:")
        title_str = "Approx. Colour Counter ({2}-Regular Graph, n = {0}, k = {1})"
        title_input = title_str.format(input_n, input_k, input_d)
        input_epsilons = list(map(float, input().split()))
        generate_plots_varying_epsilon(test_graph, input_k, input_epsilons, title_input)
```

9.2.2 gif_generation.py

```
import random
import itertools as it
import networkx as nx
import math
import time
import numpy as np
import matplotlib.pyplot as pyplot
import matplotlib.colors as colors
import matplotlib.animation as animation

if nx.__version__[0] != "1":
    print("This script is written for an older version of networkx, please roll back to version
        1.11 ")
    exit()


class DrawableGridColouredGraph(object):
    def __init__(self, width, height, colours):
        """
        Class used to generate Glauber Dynamics Anmiations.
        :param width: (int) width of grid
        :param height: (int) height of grid
        :param colours: (list of str) List of colours to use to draw the grid
        """

        self.height = height
        self.width = width

        # generate colour_grid, used to pass colours on to matplotlib
        self.colour_grid = np.zeros(height * width).reshape(width, height)
        self.colours = colours
        self.k = len(colours)

    def generate_initial_colouring(self):
        # Reset all colours
        for x, y in it.product(range(self.width), range(self.height)):
            self.set_colour((x, y), (x + y) % 2)

    def set_colour(self, v, k):
        """
        Tries to recolour node v with colour k
        :param v: Vertex to recolour
        :param k: Colour to use
        :return: (boolean) returns true if v has been recoloured and false otherwise
        """
        # Check if allowed to recolour
        x, y = v
        for dx in (-1, 1):
            if 0 <= x + dx < self.width:
                if self.colour_grid[x + dx][y] == k:
                    return False
        for dy in (-1, 1):
            if 0 <= y + dy < self.height:
                if self.colour_grid[x][y + dy] == k:
                    return False

        # Recolour
        self.colour_grid[x][y] = k
        return True

    def simulate_glauber_dynamics(self, t, k=1):
        """
        Simulate the Glauber dynamics for n steps with a graph G.
        :param t: (float) Minimum number of steps to simulate (in case of non-int)
        :param k: (int) Number of colours to use to recolour vertices. Works as if inputting
            range(k).
        :return: None
        """

        # First retrieve list of nodes and setup a list of colours to use
        new_colours = list(range(k))

        # Make sure we run for at least t steps
        t = int(math.ceil(t))
        for _ in range(t):
            # Pick a random vertex and colour
            x = random.randint(0, self.width - 1)
            y = random.randint(0, self.height - 1)
            colour = new_colours[random.randint(0, k - 1)]

            # Attempt to recolour
```

```
            self.set_colour((x, y), colour)

    def generate_animation(self, frames, steps_per_frame, interval, filename="animation.mp4"):
        """
        Generates the actual Glauber Dynamics animation and saves it as a file
        :param steps_per_frame: (int) Number of Glauber Dynamic steps to simulate per frame
        :param interval: (float) Time interval of a frame
        :param frames: (int) Number of frames to simulate for.
        :param filename: (str) Filename to save animation to
        """
        # generate random colouring to start on
        self.generate_initial_colouring()
        # We need one of each colour present or matplotlib will optimize our colours away
        for i in range(len(self.colours)):
            self.set_colour((0, i), i)

        # setup matplotlib context
        fig, ax = pyplot.subplots()
        colour_map = colors.ListedColormap(self.colours)
        ax = ax.imshow(self.colour_grid, cmap=colour_map, interpolation='none',
                       extent=[0, self.height, 0, self.width],
                       animated=True)

        pyplot.xticks([])
        pyplot.yticks([])

        old_time = time.time()

        def _update(f):
            nonlocal old_time
            if time.time() - old_time > 2:
                print("{0} / {1} frames generated".format(f, frames))
                old_time = time.time()
            # simulate glauber for a bit and update matplotlib plot
            ax.set_data(self.colour_grid)
            self.simulate_glauber_dynamics(steps_per_frame, self.k)
            return ax,

        # generate animation and save to file
        ani = animation.FuncAnimation(fig, _update, interval=interval, frames=frames)
        ani.save(filename, dpi=400)


if __name__ == "__main__":
    print("Frames:")
    input_frames = int(input())
    print("Steps per frame:")
    input_steps = int(input())
    print("Timestep:")
    input_interval = float(input())
    print("Filename:")
    input_filename = input()
    my_graph = DrawableGridColouredGraph(100, 50, ["red", "green", "blue"])
    my_graph.generate_animation(input_frames, input_steps, input_interval, input_filename)
```

9.3 C++ Code

9.3.1 graph_colouring_approximator.cpp

```
/**
        graph_colouring_approximator.cpp
        Purpose: Calculates approximations for number of colourings on a graph.
        @author: Wouter Rienks
        @version: 1.2 14/05/2018

        Invoke like
                > Graph Colouring Approximator <n> <k> <t> <s> <trials> e1 e2 e3 ...
        Where
                (int) n = number of vertices
                (int) k = number of colours
                (int) t = number of steps to simulate glauber dynamics
                (int) s = number of samples to generate per approximator
                (int) trials = number of approximations to generate

                (2x int) ei = some edge

        For example:
                > Graph Colouring Approximator 5 5 100 200 10 0 1 1 2 2 3 3 4 4 0
        Will generate a cyclic graph with 5 vertices, simulate using 5 colours, using
        100 steps to sample a colouoring, 200 colourings to sample a ratio, and write
        10 (independent) approximations to cout. The part
        0 1 1 2 2 3 3 4 4 0
        specifies the list of edges.
*/


#include <list>
#include <vector>
#include <random>
#include <iostream>
#include <algorithm>
#include <cstdio>

using namespace std;


struct Edge {
        // Struct used to represent an edge
        Edge(int v1, int v2) : v1(v1), v2(v2) {};

        // Endpoints
        int v1;
        int v2;
};


class ColouredGraph {
        // These are private as changing them at will might cause exceptions
        // vertex count
        int n;

        // number of colours
        int k;

        // edge count
        int m = 0;

        // Matrix of edges present
        vector<vector<bool>> edges;

        // List of colours
        vector<int> colours;

        mt19937 rng;
        uniform_int_distribution<mt19937::result_type> n_dist;
        uniform_int_distribution<mt19937::result_type> k_dist;

public:
        /**
                Class used to represent a graph to simulate Glauber Dynamics on. Much
                faster implementation than the old python one.

                @param n number of vertices
                @param k number of colours
                @param edge_list list of edges present in the graph
        */
```

```
ColouredGraph(int n, int k, vector<Edge> edge_list) : n(n), k(k), n_dist(0, n - 1), k_dist
    (0, k - 1)
    {
            // Setup random number generator for glauber dynamics
            rng.seed(random_device()());

            // Read edges from list into n x n boolean matrix
            edges = vector<vector<bool>>(n, vector<bool>(n, false));
            colours = vector<int>(n, 0);
            for (Edge e : edge_list)
                    add_edge(e);

            // Generate initial colouring (greedy)
            for (int i = 0; i < n; ++i)
            {
                    int c = 0;
                    while (!update_colour(i, c))
                            c += 1;
                    if (c >= k)
                            throw exception("Not enough colours!");
            }
    }

    /**
            Try and recolour a vertex.
            @param v the vertex index to recolour
            @param c the colour to recolour with

            @return true if v has been recoloured to c, false if it couldn't do so
            because a neighbour had colour c already.
    */
    bool update_colour(int v, int c) {
            for (int i = 0; i < n; ++i)
            {
                    if (edges[i][v] && colours[i] == c)
                            return false;
            }
            colours[v] = c;
            return true;
    }

    /**
            Add edge to the graph.
            @param e edge to add.
    */
    void add_edge(Edge e) {
            if (!edges[e.v1][e.v2])
                    m += 1;
            edges[e.v1][e.v2] = true;
            edges[e.v2][e.v1] = true;
    }

    /**
            Remove edge from the graph.
            @param e edge to remove.
    */
    void remove_edge(Edge e) {
            if (edges[e.v1][e.v2])
                    m -= 1;
            edges[e.v1][e.v2] = false;
            edges[e.v2][e.v1] = false;
    }

    /**
            Simulate (simple) Glauber Dynamics, keeping a proper colouring.
            @param t number of steps to simulate for.
    */
    void simulate_glauber_dynamics(int t) {
            for (int i = 0; i < t; ++i) {
                    int v = n_dist(rng);
                    int c = k_dist(rng);
                    update_colour(v, c);
            }
    }

    /**
            Calculate maximum degree (non-cached).
            @return maximum degree of a vertex in the graph.
    */
    int max_degree() {
            int best = 0;
            for (int i = 0; i < n; ++i) {
                    int i_sum = 0;
                    for (bool neighbour : edges[i])
```

```
                        if (neighbour) i_sum += 1;
                best = max(1, i_sum);
        }
        return best;
    }

    /**
        Retrieve the number of edges in the graph (cached)
        @return number of edges in the graph.
    */
    int number_of_edges() {
        return m;
    }

    /**
        Find the "first" edge in the graph
        (wrt to lexicographic ordering on vertices)
        @return edge in the graph
    */
    Edge find_next_edge() {
        for (int x = 0; x < n; ++x) {
            for (int y = 0; y < x; ++y) {
                if (edges[x][y])
                    return Edge(x, y);
            }
        }
        throw exception("No edges!");
    }

    /**
        Lookup colour of a vertex.
        @param v vertex to lookup.
        @return colour of v.
    */
    int get_colour(int v) {
        return colours[v];
    }

    /**
        Retrieve the number of vertices in the graph (cached)
        @return number of edges in the graph.
    */
    int number_of_vertices() {
        return n;
    }

    /**
        Retrieve the maximum number of colours
        @return maximum number of colours used for colourng
    */
    int max_number_colours() {
        return k;
    }
};


/**
    Approximate the total number of colourings on a graph using Jerrum's algorithm.
    @param graph graph to approximate on.
    @param k number of colours to use
    @param time number of steps to simulate glauber dynamics for sampling colouring
    @param s number of samples used to estimate Omega_k(G_i) / Omega_k(G_(i + 1))
    @param trials total number of approximations to make (i.e. repeat the algorithm this
        many times)
    @return vector of approximations (of length <trials>)
*/
vector<float> approximate_colouring_count(ColouredGraph graph, int time, int s, int trials) {
    int d = graph.max_degree();
    int n = graph.number_of_vertices();
    int k = graph.max_number_colours();
    vector<float> results = vector<float>(trials, (float) pow(k, n));

    // Simulate all trials simultanuously
    while (graph.number_of_edges() > 0) {
        // Remove edge from graph
        Edge next_edge = graph.find_next_edge();
        graph.remove_edge(next_edge);

        for (int trial = 0; trial < trials; ++trial) {
            // Generate approximator for z_i
            float z_i_approximator = 0;
            for (int ss = 0; ss < s; ++ss) {
                graph.simulate_glauber_dynamics(time);
```

```
                                        // if colouring remains proper when adding edge back, z_i = 1,
                                            otherwise z_i = 0.
                                        int c1 = graph.get_colour(next_edge.v1);
                                        int c2 = graph.get_colour(next_edge.v2);
                                        if (c1 != c2)
                                                z_i_approximator += 1;
                                }
                                z_i_approximator /= s;

                                // Multiply trial-th result by z_i
                                results[trial] *= z_i_approximator;
                        }
                }
                return results;
}


/**
        Entry point.
*/
int main(int argc, char* argv[])
{
        int n = atoi(argv[1]);
        int k = atoi(argv[2]);
        int t = atoi(argv[3]);
        int s = atoi(argv[4]);
        int trials = atoi(argv[5]);

        int edges = (argc - 6) / 2;

        // Retrieve edges from argv and store them in vector of edges
        vector<Edge> edge_list(edges, Edge(0, 0));
        for (int i = 0; i < edges; ++i) {
                edge_list[i] = Edge(atoi(argv[2 * i + 6]), atoi(argv[2 * i + 7]));
        }

        // Create graph with edges
        ColouredGraph graph(n, k, edge_list);

        // Approximate colouring count
        vector<float> results = approximate_colouring_count(graph, t, s, trials);

        // Write approximations back to console
        for (float result : results)
                cout << result << endl;

        return 0;
}
```

9.4 Logbook

This section contains a 'logbook' of the ways I deviated from the original texts in certain proofs, at the request of my supervisor. For all proofs not mentioned here, I have mentioned the source either in the proof or at the start of the chapter/section.

- Most of the stuff in Chapter 3 comes from the article of Jerrum [5]. The discussion in Remark 3.2 is something I recalled from a lecture. The discussion in Remark 3.6 is something I figured out after a talk with my supervisor, the idea of using the Hoeffding inequality was suggested by Guus Regts.
- For Theorem 4.5, I had a look at the article by R. Bubley [8] but got very confused by their notation. I'm pretty sure the verification of the new coupling 'being a coupling' is something I did myself, but I haven't read the article careful enough to point out if and where I differ from it. The proof of Theorem 4.6 is something I came up with myself.
- For section 4.2 I looked at the Jerrum [6]. I came up with Lemma 4.9 by filling in the details of Jerrum's sketch in section 5.5. The translation to the discrete time case in Lemma 4.10 is something I came up with myself.
- Section 5.1 also originates from Jerrum [6]. However, instead of doing it for matchings I translated everything to independent sets in claw-free graphs (and added some small details here and there).
- Section 5.2 stems from Vigoda [9]. I mostly tried to make things more explict (and clearer) by translating from weights to actual probabilities, and adding details. Except for the example I didn't add anything original here.