Markov Chain Monte Carlo Algorithms for the Uniform Sampling of Combinatorial Objects

Dissertation

zur Erlangung des
Doktorgrades der Naturwissenschaften (Dr. rer. nat.)

der

Naturwissenschaftlichen Fakultät III

der Martin-Luther-Universität
Halle-Wittenberg,

vorgelegt

von Herrn Steffen Rechner
geb. am 02.12.1986 in Bad Frankenhausen

Gutachter:
Prof. Dr. Matthias Müller-Hannemann
Prof. Dr. Ulrik Brandes

Tag der Verteidigung: 04.06.2018

Acknowledgements

I wish to thank Matthias Müller-Hannemann for his always prompt feedback and valuable suggestions. I very much appreciate our regular discussions and your helpful advice.

No less thanks to Annabell Berger for her many years of support, for our joint work, and for her numerous ideas, which enhanced my thesis in many ways. Thank you for introducing me to this challenging research topic.

I thank Louise Molitor, Hjalmar Boulouednine, Benjamin Schmidt, and Linda Strowick for writing their final theses under my supervision. Furthermore, I thank Hjalmar Boulouednine for his reliable work as student assistant.

I thank my friends and colleagues for their support and for the plentiful distractions during the last years. Many thanks to Sascha Heße for his very thorough proofreading.

Last but not least, I wish to thank Jördis-Ann Schüler, who reacted to my regular extra hours with much understanding and support.

Abstract

The random sampling of combinatorial objects is an integral part of scientific computing. We discuss a family of sampling methods known as Markov chain Monte Carlo (MCMC) algorithms. Such an algorithm can be seen as a random walk on the state graph of an associated Markov chain. In our discussion, we lay a particular focus on the efficiency of such methods, which primarily depends on the length of the random walk. As the *total mixing time* of the associated Markov chain provides an upper bound on the necessary number of steps, it is of large interest in many applications. However, deriving a sharp upper bound on this quantity is often a difficult challenge.

To support the analysis of MCMC sampling algorithms, we developed the software tool *marathon*, designed to determine properties of Markov chains that are usually hard to find analytically. As one of several applications, our software allows us to systematically study the total mixing time of Markov chains on a large set of instances. We apply our software to assess the efficiency of several MCMC algorithms from three sampling applications. In doing so, we derive several non-trivial insights into the nature of these methods.

First, we assess the efficiency of three well-known MCMC algorithms for the uniform sampling of bipartite graphs with fixed degrees. In a set of experiments, we determine the total mixing time of the *classical switch* chain, the *edge switch* chain, and the recently suggested *curveball* chain. By processing a large number of ecological real-world instances, we show that the edge switch chain is superior to the other chains in the majority of cases. Finally, we show how the sampling problem can be accelerated by a clever preprocessing of the vertex degrees.

Motivated by the work with incomplete data, we next address the uniform sampling of bipartite graphs whose degrees lie in prescribed intervals. After introducing two new Markov chains, we prove the correctness of the associated sampling algorithms. By experimentally assessing the efficiency of these methods, we find that the first chain is superior to the second when the degrees of the bipartite graphs are near-regular, while the second chain works best for the degrees of scale-free networks. As an interesting side-result, we present an efficient algorithm for the construction of a bipartite graph whose degrees lie in prescribed intervals.

Finally, we address the uniform sampling of perfect matchings in bipartite graphs. In a set of experiments with two special classes of bipartite graphs, we identify initial states that require either a polynomial or an exponential number of steps. Our findings highlight the influence of the initial state on the efficiency of MCMC methods.

Contents

Chapter 1

Introduction

A fundamental task in scientific computing is the evaluation of the expected value

$$E_\pi[f(X)] := \sum_{x\in\Omega} \pi(x) f(x) \tag{1.1}$$

of a function $f\colon \Omega \to \mathbb{R}$ on a large but finite set Ω of combinatorial objects with respect to a probability distribution function $\pi\colon \Omega \to [0,1]$. In many applications, Ω is a set of high-dimensional combinatorial objects like graphs or matrices with well-defined properties. Due to the complex nature of these objects, and as Ω is too large to be enumerated, the exact evaluation of the expected value is infeasible in nearly all practical applications. However, by randomly drawing independent samples $x_1, x_2, \ldots, x_N$ according to the specified probability distribution function π, we may approximate the expected value by the sample mean

$$\mu_\pi[f(X)] := N^{-1} \sum_{i=1}^{N} f(x_i). \tag{1.2}$$

By the law of large numbers, the sample mean $\mu_\pi[f(X)]$ approaches its associated expected value when $N \to \infty$. The approximation of a certain value of interest with the help of randomly sampled objects established a large family of algorithms known as *Monte Carlo* methods (see Refs. [1, 2, 3] for a general introduction on the topic). Today, such methods are an essential part of statistics and applied sciences, including the following applications.

- In null-network analysis, the structure of an observed network is typically compared with the structure that would be expected "from chance" if edges were distributed randomly. In this case, Ω is defined as a set of so-called *null-networks* that share specific properties with the network $y \in \Omega$ of interest. If $g\colon \Omega \to \mathbb{R}$ is an arbitrary function quantifying the structure of a network, we may define

$$f(x) = \begin{cases} 1, & \text{if } g(x) \geq g(y) \\ 0, & \text{else.} \end{cases}$$

Then, the expected value $E_\pi[f(X)]$ describes the probability of observing a random network that is at least "as structured" as the observed one. If this probability is small, we will conclude that the structure of the observed network is unlikely to be created by chance. In the last decades, null-network analysis evolved into an integral part of ecology and other life-sciences [4, 5, 6].

- By setting $f(x) = \pi(x)^{-1}$, the expected value $E_\pi[f(X)]$ is equal to the total number of objects in Ω. This quantity is often known as *normalizing constant* or *partition function* and is of high interest in statistical physics, including spinning models and monomer-dimer systems [7]. In computer science, it is equivalent to the number of solutions of the associated decision problem. While the exact calculation of $|\Omega|$ is often #P-complete [8], the number of solutions of a *self-reducible* problem can be approximated in polynomial time using randomly sampled objects [9].

- In combinatorial optimization, famous meta-heuristics like *simulated annealing* [10, 11] are widely used to find near-optimal solutions of typically hard optimization problems. Based on the Metropolis algorithm that will be specified soon, this method defines Ω as the set of valid solutions of an optimization problem. Starting with an arbitrary initial solution, the method iteratively randomizes the current solution x to create a solution $y \in \Omega$. Based on the objective values of x and y, and on the current *temperature* of the system, the method either moves to y, or stays at x. In doing so, the method produces a random solution according to a target distribution π that favors solutions with a near-optimal objective value.

- Many real-life systems are too complex to be analysed analytically. Examples of such systems come from economics, physics, engineering, chemistry, and many more. A widely adapted approach to study the properties of such systems is stochastic simulation. After building a simplified model of the complex system, its properties can be analysed by computer simulation. For this purpose, models typically include some randomness. By the simulation of such a model on a computer, we produce random samples from a highly complex sample space Ω of model configurations [2, 12].

- In the running time analysis of deterministic algorithms, one typically constructs a worst-case instance for which the algorithm behaves poorly. To counter this so-called adversary argument, it is sometimes better to consider a randomized algorithm [13, 14]. As such algorithms make use of randomly sampled objects, their running time is a random variable. While an adversary may show that a deterministic algorithm behaves poorly for a single worst-case instance, it is often hard to show that a randomized algorithm behaves poorly on average.

These and other applications highlight the importance of Monte Carlo sampling.

However, the construction of random objects according to a specified target probability distribution π is often a challenging algorithmic problem.

1.1 Markov Chain Monte Carlo Sampling

In 1953, Metropolis et al. [15] suggested a general method designed to construct random samples according to an arbitrary probability distribution that is specified by the weight function $w\colon \Omega \to \mathbb{R}^+$. This method became later known as the Metropolis algorithm (see Alg. 1.1).

Algorithm 1.1: Metropolis Algorithm

Input: initial state $s \in \Omega$, weight function $w\colon \Omega \to \mathbb{R}^+$, integer $t \in \mathbb{Z}^+$.
Output: state $x \in \Omega$ with probability $\pi(x) \propto w(x)$.

```
1 x ← s
2 for i = 1, 2, ..., t do
3 |   randomly modify x to create y ∈ Ω            // candidate selection
4 |   select a real number u ∈ [0, 1) uniformly at random
5 |   if u < w(y)/w(x) then                        // Metropolis rule
6 |   |   x ← y
7 |   end
8 end
9 return x
```

As will be specified in Chapter 2, the Metropolis algorithm simulates a random walk on the state graph of a well-defined Markov chain, whose state space is the set Ω from which we want to draw samples, and whose stationary distribution is proportional to the weight function $w\colon \Omega \to \mathbb{R}^+$.

Starting at an arbitrary initial state $s \in \Omega$, the algorithm iteratively transforms the current state x into a state $y \in \Omega$ by applying some random modification on x. Depending on the weights $w(x)$ and $w(y)$, the algorithm will move to y or will stay at the current state x. After a certain number of steps specified by the parameter t, the algorithm returns the final state as a random sample. By design, the random sample produced by Alg. 1.1 will depend on the initial state s. One of the major challenges of the MCMC method is to choose the number t of steps large enough such that the final state is "sufficiently random" for practical purpose.

Scope In this thesis, we discuss MCMC sampling algorithms that can be seen as special cases of Alg. 1.1. In our discussion, we lay a particular focus on the efficiency of these methods. Essentially, the running time of Alg. 1.1 depends on two aspects.

- Primarily, the number t of steps must be chosen large enough such that the final state is sufficiently random. The number of steps required to sample

from a probability distribution which is "close" (in some well-defined sense) to the target distribution π is known as the *total mixing time* of the associated Markov chain, and is of high interest for the running time of such sampling algorithms. However, it is hard to establish a general upper bound on the total mixing time of non-trivial Markov chains.

- In addition to the number of steps, the random modification in Line 3 of Alg. 1.1 requires special attention as it determines the running time spent in each step. Intuitively, if a Markov chain applies very small modifications on the current state in each step, it will need a large number of steps to produce independent samples. In contrast, the more complex the random modification is in each step, the less steps will be needed to produce samples. Both aspects have to be balanced carefully to gain an efficient sampling method.

As the total mixing time of non-trivial Markov chains is typically hard to assess analytically, it is in practical applications often completely unclear how to choose the number t of steps. For that reason, the parameter t is usually chosen empirically, e.g. as a linear function on the length of the input. As we will show in Chapters 4 and 6, this will in many cases be insufficient and thus leads to a sample that is not as random as desired. As a consequence, the sampling application might behave unexpectedly or might even be incorrect. Thus, our key motivation is to assess the difficult question of how to derive a more reliable bound in practical applications.

In some cases, techniques like the *canonical path method* [16] can be applied to gain an upper bound on the total mixing time of a Markov chain. When applied successfully, a thereby established upper bound is often a high-degree polynomial on the size of the input [17, 18, 19]. Unfortunately, this is far too large to be of practical use. However, as such bounding techniques are fairly general, and worst-case instances in terms of total mixing time are unknown, most scientist would assume that there must be a considerable gap between the total mixing time and the established upper bound. Thus, we try to assess whether the total mixing time is really as large as proposed by the upper bounds, or if they are too pessimistic.

1.2 Contribution and Overview

To support the analysis of MCMC based sampling methods, we developed a software tool called *marathon* [20]. This tool serves several purposes.

- First of all, our software allows to experimentally derive properties of interest that are usually hard to find in theory. In particular, we can exactly calculate the total mixing time of a Markov chain on certain input instances, the value of the second-largest eigenvalue of its transition matrix, or the associated average loop probability. Thereby, we can precisely calculate the necessary number of steps for an MCMC sampling algorithm on a certain input instance.

- In addition, our software allows to systematically test hypotheses on properties of Markov chains by running experiments on a large number of instances. In doing so, we may quickly reject an incorrect conjecture on a certain property of interest, thereby actively assisting the analysis of Markov chains.

- Furthermore, we can precisely assess the effect of parameters on the efficiency of the sampling methods. For example, we can study the influence of the initial state s, or the effect of the number t of steps. Thereby, we can determine whether or not an intended improvement on an existing sampling algorithm pays off.

- Last but not least, we can assess and compare the efficiency of MCMC methods by running the algorithms and determining the associated running time. In doing so, we can fairly compare the efficiency of competing algorithms on real-world instances.

In summary, *marathon* is designed to accompany theoretical studies with practical experiments. For this purpose, the software needs to be versatile enough to embrace several sampling methods of various applications. In addition, it has to be easily expandable to allow the integration of additional Markov chains. Last but not least, the implemented algorithms and data structures need to be efficient enough to support the analysis of large state graphs. Based on our software, we experimentally assessed the efficiency of several MCMC algorithms from three sampling problems.

Structure In the remainder of this chapter, we describe the structure of this thesis and briefly summarize our main results.

1. The first chapter contains this introduction.

2. In the next chapter, we summarize the mathematical foundation on which our methods are built on. This includes well-known theorems from Markov chain theory as well as the concept of total mixing time. Central to our discussion is the so-called *state graph* of a Markov chain, from which we calculate various properties of the associated sampling method. In addition to a review of the literature, we introduce the concept of empirical mixing time, and show how it can efficiently be approximated to derive a lower bound on the total mixing time of a Markov chain.

3. In Chapter 3, we present our software tool *marathon* that has been designed to support the analysis of MCMC algorithms. Our software has two main applications. First, it contains optimized sampling routines that can be used to produce random samples in practical applications. Second, *marathon* can be used to construct and analyse the state graph of a Markov chain. Alongside with a short discussion of the software design, we describe essential algorithms for the construction and analysis of a Markov chain's state graph. In particular,

we show how to calculate the total mixing time of the associated Markov chain, and how to derive lower or upper bounds on the total mixing time.

4. The following chapters are dedicated to three sampling problems. Chapter 4 discusses the uniform sampling of bipartite graphs with prescribed degrees. This is a classical problem that has been in the focus of research for many years. From the large set of sampling algorithms, we address the *classical switch* chain [17, 21], the *edge switch* chain [22], and the recently introduced *curveball* algorithm [23]. We assess the efficiency of each sampling algorithm in a set of experiments.

   (a) By experimentally calculating the total mixing time of each Markov chain on a large set of input instances, we determine how the total mixing time depends on the size of the input instance. In doing so, we show that the total mixing time of both switch chain variants is often a super-linear function on the size of the input. In some cases, we find that even a quadratic number of steps does not suffice to produce a random sample.

   (b) In an additional experiment, we assess the quality of several methods designed to derive lower and upper bounds on the total mixing time of a Markov chain. In particular, we focus on the famous *canonical path method* [16] that is often applied to derive general upper bounds on the total mixing time of a certain Markov chain. By applying the method to concrete state graphs, we show how sharp such bounds may be if full information about state graphs were available. In doing so, we find that the canonical path method is unlikely to produce sharp bounds, as its quality decreases fast. Our findings indicate that existing upper bounds on the total mixing time of the classical switch chain seem to be too pessimistic.

   (c) To compare the efficiency of the three sampling algorithms on real-world instances, we experimentally determine the number of steps each algorithm is required to produce "similarly random" samples. By measuring the running time of each sampling algorithm on a large set of ecological instances, we assess the efficiency of the sampling algorithms from a practical point of view. Our experiments show that the edge switch chain is superior to the other chains in the majority of cases. In contrast, the classical switch chain is least efficient.

   We close this chapter by showing how the sampling problem can be accelerated by a clever preprocessing of the input instance. Our algorithm decomposes an input instance into a set of primitive instances that can be processed independently from each other. In an experimental evaluation, we show that our method significantly accelerates the sampling process.

5. A variation of the classical sampling problem is considered in Chapter 5, where we address the uniform sampling of bipartite graphs whose degrees lie within

prescribed bounds. Motivated by the work with incomplete data, we introduce two new Markov chains. While the first chain affects a constant number of edges per step, the second Markov chain is designed to affect a large number of edges. In doing so, we intend to reduce the total mixing time of the second chain at the cost of a larger running time spent in each step. In a series of theorems, we show that the stationary distribution of both Markov chains is uniform, thereby proving the correctness of the sampling algorithms. Afterwards, we assess the efficiency of both Markov chains in a set of experiments.

(a) By experimenting with the degrees of scale-free and regular graphs, we show that the first chain is superior to the second if the degrees of a bipartite graph are near-regular. In contrast, the second chain works best when the degrees are distributed scale-free.

(b) In a sampling application, we demonstrate how our methods can be applied to study the expected properties of partially observed networks. By assuming that several edges of an imaginary "true" network remained unobserved, we model the true network to be an unknown element of a set of graphs whose degrees may exceed the observed ones by a certain constant. We apply our sampling algorithms to approximate the expected properties of the assumed true network. Our experiments show that the presence of unobserved edges may largely influence the structure of the network.

We conclude this chapter by addressing the problem of constructing a bipartite graph whose degrees lie in prescribed intervals. This is a sub-problem of the associated sampling problem, when no initial state is known. After presenting a realization algorithm, we give a proof of its correctness and show that its running time is superior to that of existing algorithms.

6. In Chapter 6, we address the uniform sampling of perfect matchings in bipartite graphs. As it is difficult to directly sample from the set of perfect matchings, a common approach is to extend the sample space by enclosing near-perfect matchings. In this chapter, we experimentally analyse the total mixing time of two famous MCMC algorithms. While the total mixing time of Broder's chain [24] is known be an exponential function on the size of the underlying bipartite graph, the Markov chain presented by Jerrum et al. [19] is proven to possess a polynomial total mixing time. In a set of experiments with two special classes of bipartite graphs, we investigate the influence of the initial state on the efficiency of the sampling methods.

   (a) By processing several bipartite graphs from each graph class, we experimentally determine how the growing rate of the mixing time depends on the initial state. In doing so, we identify states that lead to an exponential mixing time, and states whose mixing time can be described by a polynomial function.

(b) In a final experiment, we evaluate a strategy to derive a lower bound on the total mixing time of a Markov chain when the construction of state graphs is infeasible. Our method is based on the eigenvalues of the weighted adjacency matrix of a state graph's induced subgraph. However, our experiments show that the lower bound gained by this method is likely to be too small to be of use in practical sampling applications.

7. We close our discussion with a summary of our essential results and briefly point out open problems for future work.

This thesis builds on and extends the following publications, which are joint work with Annabell Berger, Linda Strowick, and Matthias Müller-Hannemann.

A. Berger and S. Rechner. *Broder's Chain Is Not Rapidly Mixing.* arXiv preprint (2014). arXiv: `1404.4249v1 [cs.DM]` [25]

S. Rechner and A. Berger. *Marathon: An open source software library for the analysis of Markov-Chain Monte Carlo algorithms.* PLOS ONE **11** (2016). DOI: `10.1371/journal.pone.0147935` [20]

S. Rechner, L. Strowick, and M. Müller-Hannemann. *Uniform sampling of bipartite graphs with degrees in prescribed intervals.* Journal of Complex Networks (2017). DOI: `10.1093/comnet/cnx059` [26]

S. Rechner. *An Optimal Realization Algorithm for Bipartite Graphs with Degrees in Prescribed Intervals.* arXiv preprint (2017). arXiv: `1708.05520v1 [cs.DS]` [27]

Chapter 2

Preliminaries

In this chapter, we summarize the mathematical background on which the remaining chapters are built on. We assume the reader to have some basic understanding of graph and probability theory. While Diestel [28] gives a comprehensive introduction into graph theory, the classic work of Feller [29] is a good starting point to learn about probability theory.

2.1 Ergodic Markov Chains

An important tool for understanding MCMC based sampling processes are Markov chains. The following definitions and theorems are classical results of Markov chain theory and can be found in standard textbooks like the one by Levin, Peres, and Wilmer [30].

Definition 2.1 (time-homogeneous Markov chain)**.** *Let $X = (X_0, X_1, \ldots)$ be a sequence of random variables from a finite state space $\Omega = \{x_1, x_2, \ldots, x_{|\Omega|}\}$. The random variables X are called* Markov chain *if and only if the so-called* Markov property *holds.*

$$\forall t \in \mathbb{N}\colon \Pr[X_{t+1} = x | X_0 = x_{i_0}, X_1 = x_{i_1}, \ldots, X_t = x_{i_t}] = \Pr[X_{t+1} = x | X_t = x_{i_t}].$$

A Markov chain is called time-homogeneous *if and only if*

$$\forall t \in \mathbb{N}\colon \Pr[X_{t+1} = x | X_t = y] = \Pr[X_t = x | X_{t-1} = y].$$

As the probability of the next state depends solely on the present state and not on any previous state, the *transition probability* $p\colon \Omega \times \Omega \to [0, 1]$ of passing from state x to state y in a time-homogeneous Markov chain can be summarized by a *transition matrix* $P = (p_{ij})$, defined by

$$p_{ij} := p(x_i, x_j) = \Pr[X_{t+1} = x_j | X_t = x_i],$$

for each $i, j \in \{1, 2, \ldots, |\Omega|\}$. As for each state $x \in \Omega$, the sum $\sum_{y \in \Omega} p(x, y)$ equals one, the transition matrix P is a *stochastic* matrix. Thus, the elements of each row

sum to one. The matrix can be interpreted as the weighted adjacency matrix of a directed graph $\Gamma = (\Omega, \Psi)$ whose vertices correspond to the state space Ω. Such a graph is called *state graph*.

Definition 2.2 (state graph)**.** *Let $X = (X_0, X_1, \ldots)$ be a time-homogeneous Markov chain with state space Ω and transition probability $p\colon \Omega \times \Omega \to [0, 1]$. The directed graph $\Gamma = (\Omega, \Psi)$ with $\Psi := \{(x, y) \in \Omega \times \Omega\colon p(x, y) > 0\}$ is called* state graph *of the associated Markov chain.*

In this context, Alg. 1.1 can be seen as a random walk on the state graph of a well-defined Markov chain. Starting at an arbitrary vertex $s \in \Omega$, the algorithm selects a random vertex $y \in \Omega$ that is adjacent to the current vertex and moves to y with probability $\min(1, w(y)/w(x))$. Thus, the associated transition probability $p(x, y)$ can be described by

$$p(x, y) = \kappa(x, y) \cdot \min\left(1, \frac{w(y)}{w(x)}\right),$$

where $\kappa\colon \Omega \times \Omega \to [0, 1]$ is the *proposal probability* of transforming state x into y in Line 3 of Alg. 1.1.

Stationary Distribution We denote by $p_s^{(t)}(x)$ the probability of being at state $x \in \Omega$ after t steps of a random walk starting at state $s \in \Omega$, i.e.

$$p_s^{(t)}(x) := \Pr[X_t = x | X_0 = s].$$

The probability distribution function $p_s^{(t)}$ is known as the *t-step probability distribution* of the associated Markov chain. By the law of total probability, $p_s^{(t+1)}$ is derived from $p_s^{(t)}$ by

$$p_s^{(t+1)}(x) = \sum_{y \in \Omega} p_s^{(t)}(y) \cdot p(y, x). \tag{2.1}$$

Under certain conditions, the t-step distribution of a Markov chain converges to the associated *stationary distribution* when $t \to \infty$.

Definition 2.3 (stationary distribution)**.** *Let $p\colon \Omega \times \Omega \to [0, 1]$ denote the transition probability of a time-homogeneous Markov chain. A probability distribution function π is called* stationary distribution *of the associated Markov chain if and only if*

$$\forall x \in \Omega\colon \pi(x) = \sum_{y \in \Omega} \pi(y) \cdot p(y, x).$$

In other words, once a Markov chain has reached a stationary distribution, the distribution will not change by the further evolution of the chain. It is well-known under which conditions such a stationary distribution exists.

Definition 2.4 (irreducibility, aperiodicity, ergodicity)**.** *Let $\Gamma = (\Omega, \Psi)$ the state graph of a Markov chain $X = (X_0, X_1, \ldots)$ with state space Ω. The Markov chain is called* irreducible *if Γ is strongly connected. It is called* aperiodic *if Γ is not bipartite. It is called* ergodic *if it is both irreducible and aperiodic.*

If a Markov chain is not aperiodic, it can easily be transformed into an aperiodic chain by adding an artificial *loop transition*. In each step, a so-called *lazy* Markov chain stays at its current state with probability $0 < \alpha < 1$. As the associated state graph contains a loop arc, it cannot be bipartite. Thus, the lazy version of any Markov chain is aperiodic.

The following classical result is of central importance for the correctness of MCMC algorithms.

Theorem 2.1 (fundamental theorem)**.** *Let $X = (X_0, X_1, \ldots)$ be an ergodic Markov chain. Let $s \in \Omega$ be an arbitrary initial state and let $p_s^{(t)}$ be the associated t-step probability distribution. Then, there is a unique stationary distribution π such that*

$$\forall x \in \Omega\colon \lim_{t\to\infty} p_s^t(x) = \pi(x).$$

The fundamental theorem is the theoretical foundation of the MCMC approach. As the t-step distribution of an ergodic Markov chain approaches the associated stationary distribution independently from the initial state s, an MCMC algorithm will produce a random sample distributed by π after an infinite number of steps, regardless of at which state the algorithm started.

We can easily identify the stationary distribution π of an ergodic Markov chain by showing that the Markov chain is *reversible* with respect to this distribution.

Definition 2.5 (reversibility)**.** *A time-homogeneous Markov chain with transition probability $p\colon \Omega \times \Omega \to [0, 1]$ is called* reversible *with respect to a probability distribution function π if and only*

$$\forall x, y \in \Omega\colon \pi(x) \cdot p(x, y) = \pi(y) \cdot p(y, x). \tag{2.2}$$

Eq. 2.2 is known as *detailed balance condition*. We can prove that a probability function π is the stationary distribution of a Markov chain by showing that Eq. 2.2 holds.

Theorem 2.2. *Let $X = (X_0, X_1, \ldots)$ be an ergodic Markov chain with transition probability $p\colon \Omega \times \Omega \to [0, 1]$ and let π be a probability distribution function on Ω. When X is reversible with respect to π, then π is its stationary distribution.*

Proof. By Eq. 2.1, one step of a Markov chain transforms the probability distribution function π into

$$\bar{\pi}(x) = \sum_{y\in\Omega} \pi(y)p(y, x) = \sum_{y\in\Omega} \pi(x)p(x, y) = \pi(x) \sum_{y\in\Omega} p(x, y) = \pi(x).$$

Thus, π is stationary. □

Nicholas Metropolis [15] proved the following theorem which states that Alg. 1.1 produces random samples proportionally to the weight function w, whenever the proposal probabilities κ are symmetric.

Theorem 2.3. *If $\kappa\colon \Omega \times \Omega \to [0,1]$ is a symmetric proposal probability function, then the stationary distribution of the Markov chain defined by Alg. 1.1 is given by $\pi(x) = w(x)/Z$, where $Z := \sum_{y\in\Omega} w(y)$.*

Proof. We show that the detailed balance condition holds. Without loss of generality, we may assume $w(y) < w(x)$. Then, the following equations hold for each $x, y \in \Omega$.

$$\begin{aligned}
\pi(x) \cdot p(x,y) &= \frac{w(x)}{Z} \cdot \kappa(x,y) \cdot \min\left(1, \frac{w(y)}{w(x)}\right) \\
&= \frac{w(y)}{Z} \cdot \kappa(x,y) \\
&= \pi(y) \cdot \kappa(y,x) \\
&= \pi(y) \cdot \kappa(y,x) \cdot \min\left(1, \frac{w(x)}{w(y)}\right) \\
&= \pi(y) \cdot p(y,x).
\end{aligned}$$

□

In 1970, Hastings generalized Alg. 1.1 to allow non-symmetric proposal probabilities [31]. This extension is known as the *Metropolis-Hastings* algorithm.

Uniform Sampling A special but practically important case is the *uniform* sampling of combinatorial objects. In this case, the target probability distribution π is the uniform distribution on Ω, i.e. $\pi(x) = |\Omega|^{-1}$ for each $x \in \Omega$. In such cases, Alg. 1.1 can be simplified as $w(x)$ is constant and thus, the Metropolis rule in Line 5 never rejects the transition of moving to state y. The transition probability $p(x,y)$ is in such cases identical with the proposal probability $\kappa(x,y)$. Thus, the stationary distribution of a Markov chain will automatically be uniform if the transition probability function is symmetric.

Corollary 2.4. *Let $X = (X_0, X_1, \ldots)$ be an ergodic Markov chain with state space Ω and symmetric transition probability function $p\colon \Omega\times\Omega \to [0,1]$. Then, the stationary distribution of X is the uniform distribution on Ω.*

Proof. As $p(x,y) = p(y,x)$ holds for each $x, y \in \Omega$, X is reversible with respect to the uniform distribution $\pi(x) = |\Omega|^{-1}$ as the detailed balance condition holds.

$$\forall x, y \in \Omega\colon \pi(x) \cdot p(x,y) = \pi(x) \cdot p(y,x) = |\Omega|^{-1} \cdot p(y,x) = \pi(y) \cdot p(y,x).$$

By Theorem 2.2, the stationary distribution π is the uniform distribution on Ω. □

In other words, if the target distribution π of an MCMC sampling algorithm is supposed to be the uniform distribution on Ω, it suffices to show that the transition probability function the underlying Markov chain is symmetric.

Summary The methods presented in this section provide a methodical foundation for the construction of MCMC sampling algorithms. To create a sampling algorithm, we need to define the transition rules of an ergodic Markov chain whose state space is the set Ω from which we want to draw samples. By showing that the associated transition probability function $\kappa\colon \Omega \times \Omega \to [0,1]$ is symmetric, Alg. 1.1 will generate random samples from Ω according to any target distribution $\pi(x) \propto w(x)$.

2.2 Total Mixing Time

By Theorem 2.1, an infinite number of steps will lead to a truly random sample. But what happens if we stop the random walk after a finite number of steps? In this case, the random sample will be distributed according to a probability function that is "close" to the target distribution π. We can measure the distance between two probability distributions by the *total variation distance*.

Definition 2.6 (total variation distance)**.** *Let μ and η be probability distribution functions on a common set Ω. The total variation distance $||\mu - \eta||$ is defined by*

$$||\mu - \eta|| = \frac{1}{2} \sum_{x \in \Omega} |\mu(x) - \eta(x)|. \tag{2.3}$$

Following from Theorem 2.1, the total variation distance $||p_s^{(t)} - \pi||$ approaches zero when $t \to \infty$. The number of steps required to reach a probability distribution with a total variation distance of ε or less is known as the *mixing time* of a Markov chain, and is of high interest for the running time of the associated sampling algorithm.

Definition 2.7 (mixing time, total mixing time)**.** *Let $s \in \Omega$ be an arbitrary initial state. The* mixing time $\tau_s\colon [0,1] \to \mathbb{N}$ *of a Markov chain is defined by*

$$\tau_s(\varepsilon) := \min\{t \in \mathbb{N}\colon ||p_s^{(t)} - \pi|| \leq \varepsilon\}. \tag{2.4}$$

We define the total mixing time $\tau_{\max}\colon [0,1] \to \mathbb{N}$ *by*

$$\tau_{\max}(\varepsilon) := \max_{s \in \Omega} \{\tau_s(\varepsilon)\}. \tag{2.5}$$

The total mixing time provides a formal measure for the efficiency of a Markov chain. A Markov chain is called *rapidly mixing*, if $\tau_{\max}(\varepsilon)$ can be bounded from above by a polynomial which depends on the input size and the parameter ε^{-1}. The total mixing time can be bounded by several techniques, from which we present some important ones. An overview about several bounding techniques is given by Sinclair [16].

Spectral Bounds Let P the transition matrix of an ergodic Markov chain. It is easy to show that P has the eigenvalue 1 corresponding to the eigenvector $\mathbf{v} = (v_1, v_2, \ldots, v_{|\Omega|})$ defined by $v_i := \pi(x_i)$. (By Def. 2.3, $\mathbf{v}P = 1 \cdot \mathbf{v}$ holds as π is a stationary distribution.) In addition, as the Markov chain is aperiodic (and the state graph is therefore not bipartite), the eigenvalue -1 cannot occur. Furthermore, we assume for now that P is a symmetric matrix. In this case, its eigenvalues are real and can be ordered non-increasingly. As P is stochastic, the Perron-Frobenius theorem [32, 33] shows that its eigenvalues lie in the range

$$1 = \lambda_1 > \lambda_2 \geq \ldots \geq \lambda_{|\Omega|} > -1.$$

Let $\lambda_{\max} := \max\{\lambda_2, |\lambda_{|\Omega|}|\}$. The quantity $(1 - \lambda_{\max})$ is known as the *spectral gap* of the transition matrix P. It can be used to bound the total mixing time of the associated Markov chain.

Theorem 2.5 (spectral bounds [16])**.** *Let P be a symmetric transition matrix of an ergodic Markov chain with stationary distribution π. Let $\lambda_{\max} := \max\{\lambda_2, |\lambda_{|\Omega|}|\}$. Then, the total mixing time $\tau_{\max}(\varepsilon)$ can be bounded by*

$$\tau_{\max}(\varepsilon) \leq \frac{1}{1 - \lambda_{\max}} \cdot \ln(\varepsilon \cdot \pi_{\min})^{-1} \tag{2.6}$$

$$\tau_{\max}(\varepsilon) \geq \frac{1}{2} \cdot \frac{\lambda_{\max}}{1 - \lambda_{\max}} \cdot \ln(2\varepsilon)^{-1}, \tag{2.7}$$

where $\pi_{min} := \min_{x \in \Omega}\{\pi(x)\}$.

If the stationary distribution π of an ergodic Markov chain is not uniform, the associated transition matrix P will not be symmetric. However, it can be transformed into a symmetric matrix P' by calculating

$$P' := D^{-1/2} \cdot P \cdot D^{1/2}, \tag{2.8}$$

where

$$D^{1/2} := \operatorname{diag}\left(\sqrt{\pi(x_1)}, \sqrt{\pi(x_2)}, \ldots, \sqrt{\pi(x_{|\Omega|})}\right), \text{ and}$$
$$D^{-1/2} := \operatorname{diag}\left(\sqrt{\pi(x_1)^{-1}}, \sqrt{\pi(x_2)^{-1}}, \ldots, \sqrt{\pi(x_{|\Omega|})^{-1}}\right)$$

are $|\Omega| \times |\Omega|$ matrices with positive values on their main diagonal. It is straightforward to show that P' has the same eigenvalues as P.

Congestion Bound The *canonical path method* [16] is an elegant technique to gain upper bounds on the total mixing time of a Markov chain. Let

$$\mathcal{P} := \{p_{xy} \colon x \neq y \in \Omega\}$$

be a family of simple paths in Γ, where each p_{xy} is a simple path from x to y. Let

$$\mathcal{P}_{uv} := \{p_{xy} \in \mathcal{P} \colon (u, v) \in p_{xy}\}$$

be the subset of paths that contain the arc $(u, v) \in \Psi$. Then, the *maximum load congestion* $\rho(\mathcal{P})$ is defined as

$$\rho(\mathcal{P}) := \max_{(u,v)\in\Psi} \frac{\sum_{p_{xy}\in\mathcal{P}_{uv}} \pi(x)\pi(y)|p_{xy}|}{\pi(u)p(u,v)}, \tag{2.9}$$

where $|p_{xy}|$ denotes the number of arcs in p_{xy}. For any path system $\mathcal{P}$, the total mixing time of a reversible Markov chain can be bounded from above by the *congestion bound*.

Theorem 2.6 (congestion bound [16])**.** *Let $\rho(\mathcal{P})$ be the maximal load congestion of an arbitrary family of paths $\mathcal{P}$ in the state graph of an ergodic Markov chain. Then, the total mixing time $\tau_{\max}(\varepsilon)$ is bounded from above by*

$$\tau_{\max}(\varepsilon) \quad \leq \quad \rho(\mathcal{P}) \cdot \ln(\varepsilon \cdot \pi_{\min})^{-1}. \tag{2.10}$$

For various applications, the congestion bound has successfully been applied to gain an upper bound on the total mixing time [17, 18, 19]. To show that a Markov chain is rapidly mixing, it suffices to define a set of paths such that the maximal load congestion $\rho(\mathcal{P})$ can be bounded from above by a polynomial function. This, however, is often a demanding challenge.

2.3 Empirical Mixing Time

The mathematical concepts presented in the previous section can be used to assess the efficiency of an MCMC algorithm. In particular, we can calculate the total mixing time $\tau_{\max}(\varepsilon)$ to determine the number of steps after which a Markov chain is "close" to its stationary distribution. However, the calculation of the total mixing is infeasible in many practical applications as it requires the construction of the associated state graph. For that reason, we introduce a generalized concept of mixing time which can be approximated through simulation. This technique is based on the evaluation of an arbitrary auxiliary function

$$f \colon \Omega \to \mathbb{R}.$$

Let η be a probability distribution on $\mathbb{R}$ such that $\eta(y)$ describes the probability of observing $y = f(x)$ when $x \in \Omega$ is distributed according to π, i.e.

$$\eta(y) := \Pr[f(X) = y | X \sim \pi].$$

Analogously, the probability distribution function $q_s^{(t)}$ describes the probability that $f(x)$ equals y when $x \in \Omega$ is distributed according to $p_s^{(t)}$, i.e.

$$q_s^{(t)}(y) := \Pr[f(X) = y | X \sim p_s^{(t)}].$$

Definition 2.8 (empirical mixing time)**.** *Let $f\colon \Omega \to \mathbb{R}$ be an arbitrary auxiliary function and let $s \in \Omega$ be an initial state. The* empirical mixing time $\bar{\tau}_s\colon [0,1] \to \mathbb{N}$ *(with respect to f) is defined by*

$$\bar{\tau}_s(\varepsilon) := \min\{t \in \mathbb{N}\colon ||q_s^{(t)} - \eta|| \leq \varepsilon\}.$$

It is easy to show that the empirical mixing time $\bar{\tau}_s(\varepsilon)$ is a lower bound for the total mixing time $\tau_{\max}(\varepsilon)$.

Theorem 2.7. *Let $X = (X_0, X_1, ...)$ be an ergodic Markov chain with state space Ω and total mixing time $\tau_{\max}(\varepsilon)$. Let $s \in \Omega$ be an arbitrary initial state and $f\colon \Omega \to \mathbb{R}$ be an auxiliary function. Then, $\bar{\tau}_s(\varepsilon)$ is a lower bound on $\tau_{\max}(\varepsilon)$.*

Proof. We prove the theorem by showing that $||q_s^{(t)} - \eta|| \leq ||p_s^{(t)} - \pi||$ holds for all $t \in \mathbb{N}$. The application of the function f partitions the domain Ω into disjoint subsets $\Omega_1, \Omega_2, \ldots, \Omega_k$ that share a common function value, i.e.

$$\forall i \in \{1, \ldots, k\}\colon \forall x_1, x_2 \in \Omega_i\colon f(x_1) = f(x_2) =: y_i.$$

By the law of total probability, η and $q_s^{(t)}$ are calculated by

$$\eta(y_i) = \sum_{x \in \Omega_i} \pi(x) \quad \text{and} \quad q_s^{(t)}(y_i) = \sum_{x \in \Omega_i} p_s^{(t)}(x).$$

Thus, we can express the variation distance $||q_s^{(t)} - \eta||$ by

$$\begin{aligned} ||q_s^{(t)} - \eta|| &= \frac{1}{2} \sum_{y \in \mathbb{R}} \left| q_s^{(t)}(y) - \eta(y) \right| \\ &= \frac{1}{2} \sum_{i=1}^{k} \left| \sum_{x \in \Omega_i} \left(p_s^{(t)}(x) - \pi(x) \right) \right|. \end{aligned}$$

By the triangle inequality, the variation distance can be bounded from above by

$$\begin{aligned} ||q_s^{(t)} - \eta|| &\leq \frac{1}{2} \sum_{i=1}^{k} \sum_{x \in \Omega_i} \left| p_s^{(t)}(x) - \pi(x) \right| \\ &= \frac{1}{2} \sum_{x \in \Omega} \left| p_s^{(t)}(x) - \pi(x) \right| \\ &= ||p_s^{(t)} - \pi||. \end{aligned}$$

Thus, $\bar{\tau}_s(\varepsilon) \leq \tau_s(\varepsilon) \leq \tau_{\max}(\varepsilon)$, which completes the proof. □

The quality of the lower bound depends on the auxiliary function $f\colon \Omega \to \mathbb{R}$ and the initial state s. In Section 4.3, we assess the quality of several functions on the empirical mixing time. Here, we briefly discuss two extreme cases.

- If $f\colon \Omega \to \mathbb{R}$ is an injective function, then $\bar{\tau}_s(\varepsilon)$ is equal to $\tau_s(\varepsilon)$ as each Ω_i contains a single element and thus, $||p_s^{(t)} - \pi|| = ||q_s^{(t)} - \eta||$ for all $t \in \mathbb{N}$.
- If there is a constant $c \in \mathbb{R}$ such that $f(x) = c$ for all $x \in \Omega$, then $\bar{\tau}_s(\varepsilon) = 0$ as

$$q_s^{(t)}(y) = \eta(y) = \begin{cases} 1, & \text{if } y = c, \\ 0, & \text{else} \end{cases}$$

holds for all $s \in \Omega$ and $t \in \mathbb{N}$.

Approximation The empirical mixing time $\bar{\tau}_s(\varepsilon)$ can be approximated by the simulation of the associated Markov chain. For this purpose, we need to approximate the t-step-distributions $q_s^{(t)}$ for increasing values of t, and the limiting distribution η.

1. To approximate the t-step-distribution $q_s^{(t)}$, we may simulate t steps of the associated Markov chain (starting at initial state $s \in \Omega$) to create a random sample $x \in \Omega$ with probability $p_s^{(t)}(x)$. By evaluating the auxiliary function f on x, the real number $y = f(x)$ is consequently distributed according to $q_s^{(t)}(y)$.

   By repeating this process N times (always starting at state $s \in \Omega$), we create a sequence of independent and identically distributed random samples $y_1, y_2, \ldots, y_N$, from which we construct a sampling histogram as an approximation for the t-step-distribution $q_s^{(t)}$.

2. To approximate the limiting distribution η, we may use one of several strategies.

   (a) If $|\Omega|$ is not too large, we may *enumerate* the whole state space and precisely calculate the limiting distribution η by evaluating $y = f(x)$ for each $x \in \Omega$. This is feasible only if Ω is comparably small.

   (b) If the enumeration of the state space Ω is infeasible, we may instead use an *exact* sampling algorithm to produce N independent random samples from the target distribution π and to construct a sampling histogram that approximates η. Although the asymptotic running time of such algorithms is typically unpleasantly large, they can be surprisingly efficient for small and medium-sized real-world instances [34].

   (c) If exact sampling accoring to the target distribution π is infeasible, it can sometimes be more efficient to create a random sample x according to a different distribution $\tilde{\pi}$, and to subsequently calculate the probability $\pi(x)$ of interest. In particular, it has been show that methods based on *importance sampling* are very efficient for this purpose. [35].

   (d) If none of the previous techniques is feasible, we can approximate the limiting distribution η using an MCMC sampling algorithm to iteratively produce random samples according to a t-step-distribution $p_s^{(t)}$, where t

is a large number and s is the final state from the previous run. To determine the number t of steps for each MCMC run, we can use an experimental procedure based on trial and error. In this process, we iteratively construct sampling histograms for several large values of t until the variation distance of two succeeding sampling histograms becomes neglectable.

In summary, we can approximate the empirical mixing time by the simulation of the associated Markov chain. This does not require the construction of a state graph and can therefore be executed for instances in which the exact calculation of the mixing time is infeasible.

2.4 Convergence of Sample Means

By the approximation of a Markov chain's empirical mixing time, we can efficiently calculate a lower bound on the Markov chain's total mixing time. However, constructing the sampling distributions $q_s^{(t)}$ and η may still be infeasible for large input instances as we need to draw a large number of samples to construct an accurate approximation of the probability distributions. For that reason, a rather simplified variant of the empirical mixing time is often used to experimentally assess the efficiency of MCMC methods (see Refs. [36, 37, 38, 39] for examples).

By simulating t steps of an ergodic Markov chain with state space Ω and initial state $s \in \Omega$, we produce the sample $x \in \Omega$ with probability $p_s^{(t)}(x)$. Repeating this experiment N times – always starting at state s – we produce a sequence of independent and identically distributed random samples $x_1, x_2, \ldots, x_N$ according to the t-step probability $p_s^{(t)}$. By evaluating an arbitrary auxiliary function $f\colon \Omega \to \mathbb{R}$ on each random sample and calculating the sample mean

$$\mu_{p_s^{(t)}}[f(X)] := N^{-1} \sum_{i=1}^{N} f\left(x_i\right),$$

we approximate the expected value

$$E_{p_s^{(t)}}[f(X)] = \sum_{x \in \Omega} p_s^{(t)}(x) f(x)$$

of the function f with respect to the t-step distribution $p_s^{(t)}$. By Theorem 2.1 and the law of large numbers, the sample mean $\mu_s^{(t)}[f(X)]$ will approach the expected value

$$E_\pi[f(X)] = \sum_{x \in \Omega} \pi(x) f(x)$$

as $t, N \to \infty$. By letting t grow and observing the sample means $\mu_{p_s^{(t)}}[f(X)]$, we can determine the number of steps after which the sample means stabilize. This number can be used as an indicator for the efficiency of a Markov chain.

In contrast to the empirical mixing time, this technique does not require sampling histograms for increasing values of t. As the approximation of an expected value requires less samples than the construction of the associated sampling histogram, we can apply this technique to instances for which the calculation of the empirical mixing time would be too time-consuming.

Unfortunately, there seems to be no direct way to use the convergence of the sample means to derive a lower bound on the total mixing time of the associated Markov chain. Thus, this method is not as well-founded as the empirical mixing time discussed previously.

2.5 Summary

In this chapter, we introduced the mathematical concepts for our following discussion. After reviewing fundamental definitions and theorems on Markov chains and mixing time, we introduced the concept of empirical mixing time and showed that it is a lower bound on the total mixing time of a Markov chain. As the empirical mixing time can be approximated by the simulation of the associated Markov chain, it can efficiently be evaluated even if the construction of a state graph is infeasible.

In cases where even the approximation of the empirical mixing time is too costly, a simplified variant based on the approximation of expected values can be used to assess the efficiency of MCMC algorithms. As the approximation of an expected value requires less samples than the construction of a sampling histogram, it can be carried out more efficiently in practical applications.

Chapter 3

The *marathon* Software

This thesis involves a large number of practical experiments that require efficient implementations of highly non-trivial algorithms. As these algorithms proved to be very useful for both the analysis of Markov chains and for practical sampling applications, we compiled a software library called *marathon* that contains implementations of the methods that are discussed in this thesis. The source code is written in C++ and is freely available at `https://github.com/srechner/marathon`. In this chapter, we introduce the main features of the library. It provides two main applications.

1. First, *marathon* is a collection of efficient sampling routines that can be used to create random samples for several applications. In particular, each Markov chain discussed in this thesis is implemented within *marathon* and can be simulated to produce random samples. This is useful for scientists who want to integrate random sampling functionality into their applications. In addition, such methods can be used to study the convergence behavior of the associated Markov chains (see Section 2.3).

2. Second, *marathon* is designed to study structural properties of Markov chains, respectively their corresponding state graphs. These methods include the computation of the total mixing time, its lower and upper bounds, and many more. By experimentally analysing a Markov chain's state graph, one can learn about properties which are typically hard to find in theory. This makes *marathon* useful for theoretical scientists who want to study the properties of Markov chains.

3.1 Main Features

Our software library is designed to support the simulation and analysis of MCMC algorithms that follow the generic principle outlined in Alg. 1.1. In this thesis, we cover three sampling problems. Each problem asks for a uniformly drawn sample from its associated sample space Ω. The sample space Ω depends on the type of problem and is defined as

(a) the set of bipartite graphs with fixed degrees (discussed in Chp. 4),

(b) the set of bipartite graphs with bounded degrees (discussed in Chp. 5),

(c) the set of perfect matchings in a bipartite graph (discussed in Chp. 6).

For each type of sampling problem, *marathon* provides the following set of features.

- A family of `RandomGenerator` classes implements several sampling algorithms designed to construct random samples from their sample space Ω. Most of these algorithms are based on the MCMC approach and thus, require assumptions on the necessary number of steps. In addition, a set of *exact* sampling algorithms can be used to create unbiased samples. These methods are proven to create uniformly distributed samples and do not rely on a given number of steps. However, the running time of these methods is typically larger than that of the MCMC algorithms.

- As one of its central applications, *marathon* can be used to construct and analyse the state graphs of each Markov chain discussed in this thesis. In particular, it can calculate the total mixing time, or its lower and upper bounds. Of high importance for the construction and analysis of state graphs are the classes `MarkovChain` and `StateGraph` that will be specified soon.

- Our software provides `Enumerator` classes designed to enumerate the sample space Ω. As this can be carried out via backtracking, the enumeration of states is more efficient than the construction of state graphs. Enumerating the set of states can be used to exactly calculate expected values without the need of random sampling.

- A set of `Counter` classes is designed to calculate the exact size of the state space Ω. By exploiting symmetries and through the use of memoization techniques, the size $|\Omega|$ of the sample space can in practice be determined more efficiently than by the enumeration. Such methods are the basis of the *exact* sampling methods that are designed to produce unbiased samples.

3.2 Software Design

The *marathon* software has been designed with several goals in mind.

- To analyse Markov chains for very diverse sampling problems, the software has to be oblivious to specific applications. For that reason, we require an abstract Markov chain representation that is general enough to include a large range of sampling problems.

- The software is designed to be easily expandable by adding additional Markov chains and analysis methods. To simplify the programming effort and to hide complexity from the developer, a hierarchy of classes and interfaces is used to iteratively derive specialized sub-classes from abstract base classes.

- Our software must be efficient enough to process large state graphs with several million states. For this reason, we chose the C++ programming language and integrated efficient third-party libraries to accelerate the computation.

In the following discussion, we present *marathon*'s class structure. To represent a diverse set of MCMC algorithms, we use a system of abstract classes that define which methods must be implemented by its sub-classes.

States In this thesis, we focus on three sampling problems. Each problem uses its own class to represent the states of its associated Markov chains. While the states of applications (a) and (b) are represented by binary matrices, the states of Markov chains from application (c) are bipartite matchings. To unify the use of different types, an abstract base class `State` models the general representation of a state object. If a Markov chain is designed to be solely used for simulation purpose, we do not specify any requirement to a state sub-class at all. For the construction of state graphs, however, a sub-class must implement the following abstract methods.

- As states will be stored in hash maps, the method `size_t hashValue()` must be implemented to calculate a hash value of the corresponding state object. The better the quality of the hash function, the more efficient the construction of the state graphs will be.

- To construct a total order on Ω, each `State` sub-class must implement the method

  ```
  int compareTo(const State*)
  ```

  to support the pairwise comparison of two state objects $x, y \in \Omega$. The method must be implemented to return 0 if and only if $x = y$. If x and y are not identical, the method must decide which state is lesser with respect to an arbitrary total order $<$ on Ω. For this purpose, it must return a negative value if $x < y$, and a positive value if $y < x$.

- To create a copy of a state, the method

  ```
  State* copy()
  ```

  must be implemented to return a new state object that is identical to the original one.

Derived from the base class `State`, the classes `BinaryMatrix` and `Bipartite-Matching` are specialized (see Fig. 3.1).

Markov Chains As the focus of this thesis is on the analysis of MCMC algorithms, the heart of *marathon* is a hierarchy of classes representing Markov chains (see Fig. 3.2). The abstract base class `MarkovChain` defines which methods a Markov chain sub-class must implement and provides basic functionality that is common to

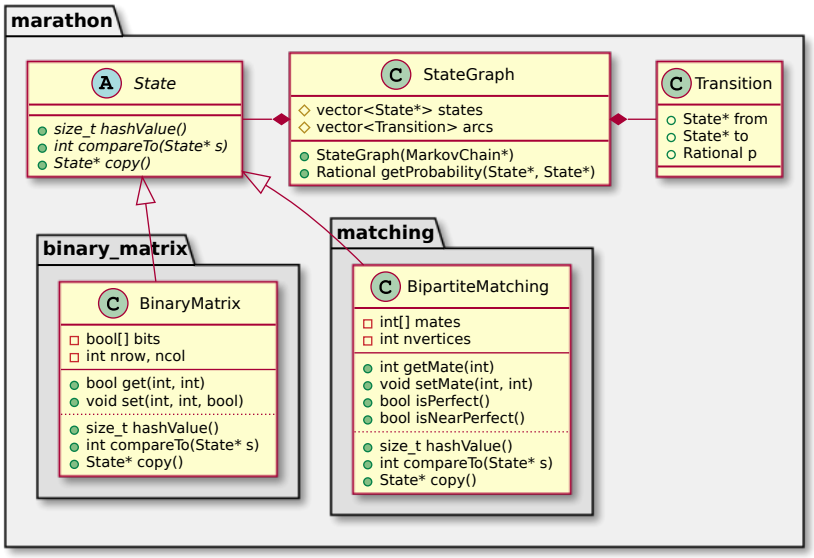

Figure 3.1: `State` and `StateGraph` classes.

each chain. Each `MarkovChain` object holds a single state which is modified by the simulation of the chain. Derived from this base class, a set of several Markov chains can be analysed or simluated to produce random samples. New Markov chains can easily be implemented by adding a new class to the hierarchy. Depending on the application of the chain, a new sub-class must implement one or more of the abstract methods defined by `MarkovChain`.

3.2.1 Random Sampling

As its most basic application, a Markov chain can be used for the construction of random samples according to the associated target distribution. For this purpose, a `MarkovChain` sub-class needs to implement the method

```
void step(),
```

which applies a single step of the associated Markov chain to its current state. After this method has been implemented, the predefined method

```
State* randomize(int steps)
```

can be used to simulate a random walk with a certain number of steps. The application of this methods requires an initial state that can be specified when creating a `MarkovChain` object. If no state is known in advance, an initial state must be constructed in the initialisation of a `MarkovChain` object. For this purpose, the method

```
State* initialState(string inst, int strategy)
```

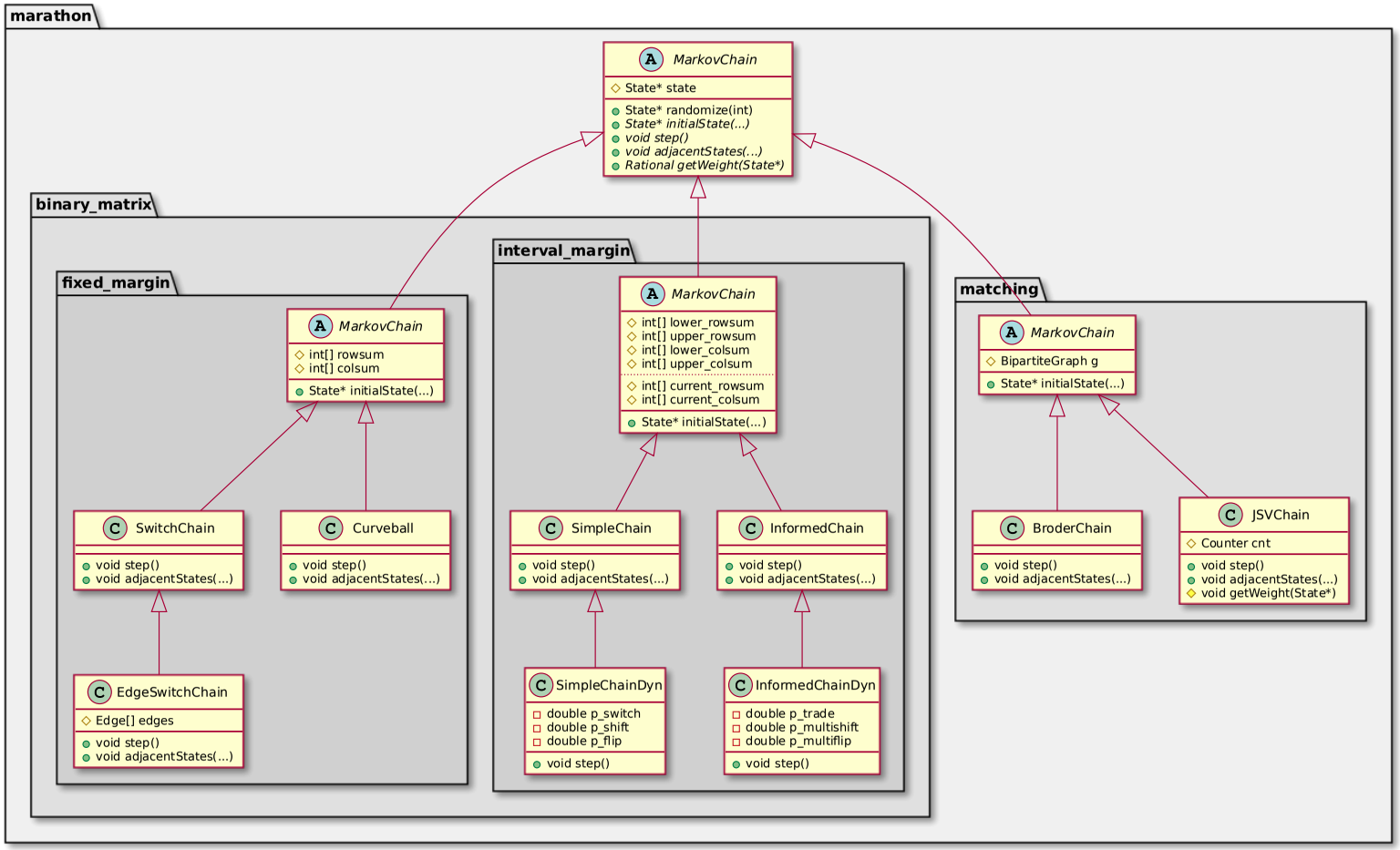

Figure 3.2: The `MarkovChain` class hierarchy.

may be implemented to produce an initial state from a problem-specific instance description. If more than one strategy for the construction of an initial state is implemented, the parameter `strategy` can be used to determine which algorithm is used.

3.2.2 State Graphs

Several interesting properties of a Markov chain require the construction of the Markov chain's state graph. For that reason, we use the class `StateGraph` to represent the state graph of a Markov chain as a weighted directed graph (see Fig. 3.1). While Ω is used as the vertex set of this graph, the set Ψ of directed arcs is implemented as a vector of three-tuples (x, y, p), where x and y are states from Ω and p is the associated transition probability $p(x, y)$. The data type `Rational` is used to store the transition probability as a rational number with arbitrary precision.

State Graph Construction As will be further specified in Section 3.3, the construction of state graphs is based on iterated calls of the abstract method

```
void adjacentStates(State* x, function<void(State*, Rational)> f),
```

that must be implemented to enumerate the set

$$N(x) := \{(y, p(x, y)) \colon y \in \Omega \wedge p(x, y) > 0\}$$

of states $y \in \Omega$ that are adjacent to the state $x \in \Omega$, alongside with the associated transition probability $p(x, y)$ of transforming x into y. Enumerating this set can typically be achieved by simulating each possible random choice according to a Markov chain's set of transition rules.

To support non-uniform stationary distributions, each state $x \in \Omega$ may possess a weight $w(x)$ that is proportional to the target probability $\pi(x)$. Thus, the abstract method

```
Rational getWeight(State* x)
```

of the `MarkovChain` base class can be overridden to return the weight $w(x)$ of the state $x \in \Omega$. Unless otherwise specified, the default implementation of the `MarkovChain` base class returns a unit weight of $w(x) = 1$ for each state $x \in \Omega$.

State Graph Analysis To analyse properties of a state graph, a family of classes can be used to compute properties like the total mixing time or its upper bounds (see Fig. 3.3). The precision of the calculation can be controlled by the template parameter `T`. Most methods provide single (`T=float`), double (`T=double`), and unlimited precision (`T=Rational`). The latter one eliminates problems arising from floating point accuracy at the cost of a larger running time. The necessary precision primarily depends on the size $|\Omega|$ of a state graph and on the parameter ε. While the

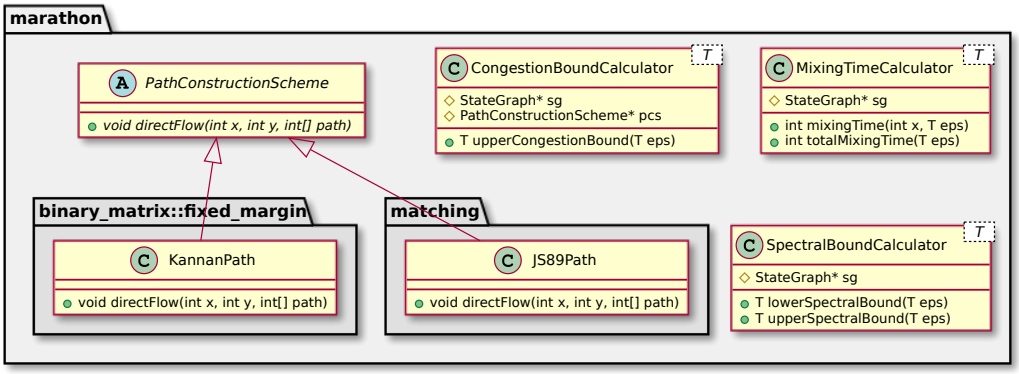

Figure 3.3: Classes for the analysis of state graphs.

usage of the `float` type may easily cause significant inaccuracy, `double` precision proved to be sufficient for most of the experiments discussed in this thesis.

Beneath others, it is possible to compute the following quantities from a state graph.

- The mixing time $\tau_s(\varepsilon)$ as defined by Eq. 2.4.
- The total mixing time $\tau_{\max}(\varepsilon)$ as defined by Eq. 2.5.
- The lower and upper spectral bound via Ineq. 2.6 and 2.7.
- The upper congestion bound via Ineq. 2.10.

The computation of the congestion bound via Ineq. 2.10 requires a canonical path construction scheme that defines how to construct a directed path between each pair of states of a state graph. Such a construction scheme is in *marathon* represented by a class that extends the abstract base class `PathConstructionScheme`. For this purpose, the method

```
void directFlow(int i, int j, int[] path)
```

can be implemented to construct a directed path p from x_i to x_j. The indices of the vertices in p are stored in the reference parameter `path`. In Sections 4.3.3 and 6.2.3, we analyse the quality of two construction schemes known from literature.

Our representation of the state graphs is versatile enough to allow the integration of arbitrary graph algorithms. As state graphs can be seen as weighted directed graphs, all kinds of graph algorithms can be integrated into *marathon*. As an example, we added functions for computing the diameter of a state graph, the average path length, as well as several clustering coefficients and centrality measures.

3.3 Implementation Details

As *marathon* is designed to process large state graphs with several million states, our methods need to be implemented efficiently. Thus, a major challenge in the development of *marathon* was the design of efficient algorithms for the construction and analysis of state graphs. In this section, we give a detailed description of fundamental algorithms that are essential to our experiments.

State Graph Construction The first step in many experiments is the construction of the state graph of an associated Markov chain. Alg. 3.1 constructs a sparse representation of the state graph $\Gamma = (\Omega, \Psi)$.

Algorithm 3.1: State Graph Construction

Input: instance description $\mathcal{I}$.
Output: state graph $\Gamma = (\Omega, \Psi)$ of an associated Markov chain.

```
1  s ← initialState(I)
2  Ω ← {s}                                              // set of nodes
3  Ψ ← ∅                                                // set of weighted arcs
4  S ← ∅                                                // set of marked nodes
5  while Ω \ S ≠ ∅ do                                   // while there is an unmarked node
6  |  select an arbitrary state x ∈ Ω \ S               // select unmarked node
7  |  N(x) ← adjacentStates(x)                          // create adjacent states
8  |  for (y, p_xy) ∈ N(x) do
9  |  |  Ω ← Ω ∪ {y}
10 |  |  Ψ ← Ψ ∪ {(x, y, p_x,y)}
11 |  end
12 |  S ← S ∪ {x}                                       // mark node x
13 end
14 return (Ω, Ψ)
```

The algorithm is based on iterated calls of the `adjacentStates` method, which enumerates a state's neighborhood. Starting with a trivial graph consisting of one node, the algorithm iteratively expands the existing graph by adding new nodes and arcs. In each step, the algorithm selects an unmarked node $x \in \Omega$ and enumerates its neighborhood $N(x)$. For each element (y, p_{xy}) of $N(x)$, the state y is added to the vertex set Ω and a weighted arc going from x to y is added to Ψ. After x has been marked, the algorithm continues the expansion until every node has been marked. The algorithm relies on the connectedness of the state graph, which is given whenever the associated Markov chain is irreducible.

Apart from the insert operation in Line 9, all set operations of Alg. 3.1 can be implemented with constant time. In contrast, to efficiently support the insertion in Line 9, the algorithm needs to determine whether the state y is already included in Ω. For this purpose, we use a hash set representation of Ω. Assuming that each

neighborhood set $N(x)$ can be constructed in $\Theta(|N(x)|)$ time, and further assuming that each of the remaining operations has a running time of $\Theta(1)$, the construction of a state graph requires $\Theta(|\Omega| + |\Psi|)$ time. Assuming that each state has constant size, we can store an adjacency list representation of the final state graph $\Gamma = (\Omega, \Psi)$ within $\Theta(|\Omega| + |\Psi|)$ space.

Mixing Time In *marathon*, the mixing time $\tau_s(\varepsilon)$ of a Markov chain is calculated by the iterated application of Eq. 2.1 (see Alg. 3.2).

Algorithm 3.2: Computation of Mixing Time

Input: state graph $\Gamma = (\Omega, \Psi)$, initial state $s = x_k \in \Omega$, real $\varepsilon \in [0, 1]$.
Output: mixing time $\tau_s(\varepsilon)$.

```
1  t ← 0                                              // will be returned
2  v ← [0]^|Ω|                                        // v = (v_1, v_2, ..., v_|Ω|)
3  v_k ← 1                                            // invariant: ∀i: v_i = p_s^(t)(x_i)
4  while ||v − π|| > ε do                             // while ||p_s^(t) − π|| > ε
5  |  w ← [0]^|Ω|                                     // w = (w_1, w_2, ..., w_|Ω|)
6  |  for i = 1, 2, ..., |Ω| do                       // calculate w_i = p_s^(t+1)(x_i)
7  |  |  w_i ← 0                                     // evaluate Eq. 2.1
8  |  |  for (x_j, x_i) ∈ Ψ do
9  |  |  |  w_i ← w_i + v_j · p(x_j, x_i)
10 |  |  end
11 |  end
12 |  v ← w
13 |  t ← t + 1                                      // invariant restored
14 end
15 return t
```

Starting from a one-point-distribution $p_s^{(0)}$ defined by

$$p_s^{(0)}(x) = \begin{cases} 1, & \text{if } x = s \\ 0, & \text{else,} \end{cases}$$

the algorithm iteratively evaluates Eq. 2.1 to construct the probability distribution $p_s^{(t)}$ for increasing values of t. In each step, the algorithm calculates the total variation distance to the stationary distribution. The process stops if the distance is smaller or equal than ε.

By passing through the sorted arc set Ψ, the running time of the for loop in Line 8 of Alg. 3.2 is proportional to the number of states x_j that can be transformed into x_i. As this number can be as large as $|\Omega|$, Alg. 3.2 calculates the mixing time $\tau_s(\varepsilon)$ in $\mathcal{O}(\tau_s(\varepsilon) \cdot |\Omega|^2)$ time and $\Theta(|\Omega|)$ space.

Total Mixing Time In *marathon*, the total mixing time $\tau_{\max}(\varepsilon)$ can be computed by two different strategies. Most simple, Alg. 3.3 evaluates the mixing time $\tau_s(\varepsilon)$ for each initial state $s \in \Omega$ and returns the maximal one. This strategy has a running time of $\mathcal{O}(\tau_{\max}(\varepsilon) \cdot |\Omega|^3)$ and requires $\Theta(|\Omega|)$ space.

Algorithm 3.3: Computation of Total Mixing Time (1)

Input: state graph $\Gamma = (\Omega, \Psi)$, real $\varepsilon \in [0, 1]$.
Output: total mixing time $\tau_{\max}(\varepsilon)$.

```
1 t_max ← 0                                   // will be returned
2 for k = 1, 2, ..., |Ω| do                   // for each x_k ∈ Ω
3 |   t ← τ_{x_k}(ε)                          // calculate mixing time
4 |   t_max ← max(t_max, t)
5 end
6 return t_max
```

The second strategy is superior to the first one in terms of efficiency. By interpreting $p_s^{(t)}$ and $p_s^{(t+1)}$ as row vectors of length $|\Omega|$, we may understand Eq. 2.1 as the application of the vector-matrix multiplication

$$p_s^{(t+1)} := p_s^{(t)} \cdot P,$$

where $P = (p_{ij})$ is the transition matrix of the Markov chain. Iterating this argument, the t-step distribution of a Markov chain can be calculated by

$$p_s^{(t)} := p_s^{(0)} \cdot P^t.$$

Thus, the i-th row of the matrix P^t represents the t-step distribution $p_{x_i}^{(t)}$. The total mixing time $\tau_{\max}(\varepsilon)$ can therefore be computed from P by iterated matrix multiplication. As a classical result following from the coupling lemma [40], it is well-known that $||p_s^{(t)} - \pi||$ decreases monotonically with t. Thus, we can find $\tau_{\max}(\varepsilon)$ with a two-step procedure (see Alg. 3.4).

1. In the first step, the algorithm determines the smallest integer d such that

   $$2^{d-1} < \tau_{\max}(\varepsilon) \leq 2^d$$

   holds. Starting with $d = 0$, the algorithm iteratively increments d until

   $$\max_{s \in \Omega} ||p_s^{(2^d)} - \pi|| \leq \varepsilon$$

   becomes true. This requires a sequence of $d = \lceil \log_2(\tau_{\max}(\varepsilon)) \rceil$ matrix squaring operations to compute the matrices $P^2, P^4, P^8, \ldots, P^{2^d}$.

Algorithm 3.4: Computation of Total Mixing Time (2)

Input: transition matrix $P = (p_{ij})$, real $\varepsilon \in [0, 1]$.
Output: total mixing time $\tau_{\max}(\varepsilon)$.

```
/* step one */
d ← 0
M ← P                                          // invariant: M = P^{2^d}
Δ ← max_{1≤i≤|Ω|} ||M_{i,·} − π||              // M_{i,·} denotes the i-th row of M
while Δ > ε do                                 // while max_{s∈Ω} ||p_s^{(2^d)} − π|| > ε
    d ← d + 1
    M ← M · M                                  // M = P^1, P^2, P^4, P^8, ...
    Δ ← max_{1≤i≤|Ω|} ||M_{i,·} − π||
end

/* step two */
(l, r) ← (2^{d−1}, 2^d)                        // invariant: l < τ_max(ε) ≤ r
M ← P^l                                        // invariant: M = P^l
while l < r − 1 do
    m ← ⌊(l + r)/2⌋                            // half search space
    A ← M · P^{m−l}                            // A = P^l P^{(m−l)} = P^m
    Δ ← max_{1≤i≤|Ω|} ||A_{i,·} − π||          // A_{i,·} denotes i-th row of A
    if Δ > ε then
        l ← m                                  // update lower bound
        M ← A
    else
        r ← m                                  // update upper bound
    end
end
return r
```

2. In the second step, we apply binary search to find $\tau_{\max}(\varepsilon)$. Two variables l and r represent lower and upper bounds on the total mixing time. Iteratively, we calculate $P^m = P^l \cdot P^{m-l}$ with $m = \lfloor (l+r)/2 \rfloor$, and evaluate the maximal total variation distance $\max_{x \in \Omega} ||p_x^{(m)} - \pi||$. Maintaining the invariant

$$\max_{x \in \Omega} ||p_x^{(l)} - \pi|| > \varepsilon \geq \max_{x \in \Omega} ||p_x^{(r)} - \pi||,$$

we either update the lower bound l or the upper bound r. We stop when $l \geq r - 1$. The value of r is the total mixing time $\tau_{\max}(\varepsilon)$. As the search space is halved in each step, the number of iterations in the second phase is

$$\log_2(2^d - 2^{d-1}) = \log_2(2^{d-1}) = d - 1 = \lceil \log_2(\tau_{\max}(\varepsilon)) \rceil - 1.$$

The running time of this strategy is $\Theta(\log(\tau_{\max}(\varepsilon)) \cdot |\Omega|^\omega)$, where $2 \leq \omega \leq 3$ is the exponent of the best-known matrix multiplication algorithm. In practice, we use highly optimized BLAS routines for matrix multiplication to vastly accelerate the computation of the total mixing time. Thus, this strategy is clearly superior to the first one in terms of efficiency. However, as a major drawback, it requires $\Theta(|\Omega|^2)$ space to store densely filled matrices. Hence, the second strategy is only applicable for small graphs ($|\Omega| \leq 10^5$, depending on the available main memory).

Spectral Bound The difference between the largest and the second largest eigenvalue $(1 - \lambda_{\max})$ of a Markov chain's transition matrix is known as the *spectral gap* and can be used to bound the total mixing time via Ineq. 2.6 and 2.7. To calculate $\lambda_{\max}$, we use the linear algebra library *Armadillo* [41]. This library offers procedures to compute the k largest real eigenvalues of a sparsely packed symmetric matrix and is therefore well-applicable for our purpose. In case of non-symmetric transition rules, we first transform the transition matrix P into a symmetrical matrix via Eq. 2.8. Having calculated the two eigenvalues with the largest magnitude, we use the smaller one to compute the upper and lower spectral bound via Ineq. 2.6 and 2.7. As this method requires only a sparse representation of the transition matrix, it is applicable to large state graphs with several million states.

3.4 Summary

In this chapter, we presented the *marathon* software, a tool designed to support the analysis of MCMC based sampling algorithms. Our software has two main applications. First, it contains sampling algorithms that can be used to create random samples for several applications. Second, *marathon* is designed to study properties of the associated sampling methods. By the construction and analysis of a Markov chain's state graph, *marathon* can be used to calculate the total mixing time or its lower and upper bounds. Our software is designed to be expandable and to be versatile enough to study MCMC algorithms of various applications. After giving a description of *marathon*'s class structure, we showed how the software constructs state graphs and calculates the associated total mixing time.

Chapter 4

Bipartite Graphs with Fixed Degrees

Bipartite graphs are widely used to describe the interactions or relations between two different groups of objects. Such graphs can be used to model the relation between films and actors, authors and publications, products and customers, and many more [42, 43, 44, 45]. In ecology, bipartite graphs model the interaction between species like plants and pollinators, or the presence or absence of species in geographical regions [46, 47]. The latter type of data is commonly represented as a *presence-absence table*, in which rows represent species and columns represent sites. If a species is present at a certain site, the associated table entry is set to one, otherwise it is zero. Prominent presence-absence tables like the famous "Darwin's finches" data set (see Table 4.1) have been objects of intense study in the last decades [48, 49, 34].

To study structural properties of networks, Connor and Simberloff [50] suggested statistical hypothesis testing. In such tests, the structure of an observed network is compared with that of randomly sampled null-networks. To quantify how structured an observed network is, a scientist may ask for the probability of randomly selecting a network that is at least "as structured" as the observed one. If this probability is small, the scientist will conclude that the structure of the observed network is unlikely to be created by pure chance.

A much-discussed question is the choice of an appropriate null model that describes how properties of an observed network are reflected by the null-networks. To preserve the characteristics of the observed network, Connor and Simberloff suggested the uniform distribution over the set of bipartite graphs whose degrees match those of the observed one. Different null-models with various advantages and disadvantages have been discussed during the last decades [4, 5]. However, the basic null-model suggested by Connor and Simberloff is still widely applied. This kind of network analysis requires a method for the uniform sampling from a set of graphs with predefined degrees. In this chapter, we focus on this classical sampling problem.

Table 4.1: "Darwin's finches": Presence of 13 finch species on 17 islands of the Galápagos archipelago (data from Ref. [49]).

	Island																	
Species	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	**r**
A	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	14
B	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	0	0	13
C	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	0	14
D	0	0	1	1	1	0	0	1	0	1	0	1	1	0	1	1	1	10
E	1	1	1	0	1	1	1	1	1	1	0	1	0	1	1	0	0	12
F	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	2
G	0	0	1	1	1	1	1	1	1	0	0	1	0	1	1	0	0	10
H	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
I	0	0	1	1	1	1	1	1	1	1	0	1	0	0	1	0	0	10
J	0	0	1	1	1	1	1	1	1	1	0	1	0	1	1	0	0	11
K	0	0	1	1	1	0	1	1	0	1	0	0	0	0	0	0	0	6
L	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2
M	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	17
c	4	4	11	10	10	8	9	10	8	9	3	10	4	7	9	3	3	122

Parts of this chapter have previously been published in the following papers. However, most of the material has been rearranged for the purpose of this thesis.

> S. Rechner and A. Berger. *Marathon: An open source software library for the analysis of Markov-Chain Monte Carlo algorithms.* PLOS ONE **11** (2016). DOI: 10.1371/journal.pone.0147935 [20]

> S. Rechner, L. Strowick, and M. Müller-Hannemann. *Uniform sampling of bipartite graphs with degrees in prescribed intervals.* Journal of Complex Networks (2017). DOI: 10.1093/comnet/cnx059 [26]

4.1 Definitions and Notation

While we consider the term *network* to refer to an "informal concept describing an object composed of elements and interactions" [51], we use the term *graph* to denote an abstract mathematical representation of such objects. In such a representation, we neglect the original interpretation of the data and focus on its underlying structure.

Definition 4.1 (bipartite graph)**.** *We define the term* bipartite graph *to describe a three-tuple $G = (U, V, E)$, where $U := \{u_1, u_2, \ldots, u_{|U|}\}$ and $V := \{v_1, v_2, \ldots, v_{|V|}\}$ are disjoint vertex sets, and $E := \{e_1, e_2, \ldots, e_{|E|}\} \subseteq U \times V$ is a set of edges.*

When $G = (U, V, E)$ is a bipartite graph, we denote by $\delta_G \colon U \cup V \to \mathbb{N}_0$ the degree *of a vertex in G.*

Problem Definition Let m and n be positive integers and let

$$\mathbf{r} = (r_1, r_2, \ldots, r_m) \qquad \text{and} \qquad \mathbf{c} = (c_1, c_2, \ldots, c_n)$$

be integer vectors of length m and n. We define $\Omega(\mathbf{r}, \mathbf{c})$ as the set of bipartite graphs whose degrees match the integers in $\mathbf{r}$ and $\mathbf{c}$, i.e.

$$\begin{aligned}\Omega(\mathbf{r}, \mathbf{c}) := \{G = (U, V, E) \colon\ & |U| = m \ \wedge\ |V| = n \\ & \wedge\ \forall i \in \{1, 2, \ldots, m\} \colon \delta_G(u_i) = r_i \\ & \wedge\ \forall j \in \{1, 2, \ldots, n\} \colon \delta_G(v_j) = c_j\}.\end{aligned}$$

The problem discussed in this chapter is to uniformly sample from the set $\Omega(\mathbf{r}, \mathbf{c})$.

Binary Matrices A bipartite graph $G = (U, V, E)$ can uniquely be represented by an $m \times n$ binary matrix $M = (m_{ij})$. This matrix is called *bi-adjacency matrix* of G and is defined by

$$m_{ij} = \begin{cases} 1, & \text{if } \{u_i, v_j\} \in E \\ 0, & \text{otherwise.} \end{cases}$$

Vice versa, each binary matrix can be interpreted as the bi-adjacency matrix of a bipartite graph. Thus, the problem discussed in this chapter can equivalently be reformulated as the uniform sampling of a binary matrix $M \in \{0, 1\}^{m \times n}$ with row sums $\mathbf{r}$ and column sums $\mathbf{c}$.

Bipartite Graph Realization In parts of this chapter, we will utilize concepts from the *graph realization* problem. This is a classical combinatorial problem that asks whether there is a graph whose degrees match a given vector of integers. In our context, we focus on the bipartite version of the graph realization problem: Given a pair of integer vectors $(\mathbf{r}, \mathbf{c})$, is there a bipartite graph $G = (U, V, E)$ such that $\delta_G(u_i) = r_i$ for each $i \in \{1, 2, \ldots, m\}$ and $\delta_G(v_j) = c_j$ for each $j \in \{1, 2, \ldots, n\}$? In other words, the problem asks whether or not $\Omega(\mathbf{r}, \mathbf{c})$ is empty.

Definition 4.2 (bi-graphical, realization)**.** *A pair $(\mathbf{r}, \mathbf{c})$ of integer vectors is called* bi-graphical *if and only if $\Omega(\mathbf{r}, \mathbf{c}) \neq \emptyset$. If $(\mathbf{r}, \mathbf{c})$ is bi-graphical, any graph $G \in \Omega(\mathbf{r}, \mathbf{c})$ is called* realization *of the associated vector pair.*

In the following, we will summarize under which conditions a pair of integer vectors is bi-graphical. These theorems are classical results from combinatorial graph theory.

Definition 4.3. *An integer vector* $\mathbf{r} = (r_1, r_2, \ldots, r_m)$ *is called* non-increasing *if and only if* $r_i \geq r_{i+1}$ *for* $1 \leq i < m$. *If* $\mathbf{r}$ *and* $\bar{\mathbf{r}}$ *are integer vectors of length* m, *we write* $\mathbf{r} \leq \bar{\mathbf{r}}$ *if and only if* $r_i \leq \bar{r}_i$ *holds for all* i *in range* $1 \leq i \leq m$.

Definition 4.4 (conjugate sequence)**.** *Let* $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ *be an integer vector of length* n. *We define the infinite sequence of integers* $\mathbf{c}' = (c'_1, c'_2, \ldots)$ *to be called* conjugate sequence *of* $\mathbf{c}$ *if and only if*

$$\forall j \in \mathbb{N}\colon c'_j := |\{i \in \{1, 2, \ldots, n\}\colon c_i \geq j\}|.$$

When $\mathbf{c}' = (c'_1, c'_2, \ldots)$ is the conjugate sequence of a non-increasing integer vector $\mathbf{c} \in \mathbb{N}^n$, than $\mathbf{c}'' = (c''_1, c''_2, \ldots)$ is called *double conjugate sequence* of $\mathbf{c}$. It is straight-forward to show that $c''_i = c_i$ holds for each i in the range $1 \leq i \leq n$.

For convenience, we will abbreviate the sum $\sum_{i=1}^{k} r_i$ by $\Sigma^k_{\mathbf{r}}$ and define $\Sigma_{\mathbf{r}}$ to be the sequence of *partial sums* of $\mathbf{r}$, i.e. $\Sigma_{\mathbf{r}} = (\Sigma^1_{\mathbf{r}}, \Sigma^2_{\mathbf{r}}, \ldots)$.

Definition 4.5 (domination)**.** *Let* $\mathbf{r} = (r_1, r_2, \ldots, r_m)$ *and* $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ *be integer vectors and let* $\mathbf{c}' = (c'_1, c'_2, \ldots)$ *be the conjugate sequence of* $\mathbf{c}$. *We say that* $\mathbf{c}'$ dominates $\mathbf{r}$ *and write* $\mathbf{r} \trianglelefteq \mathbf{c}'$ *if and only if* $\Sigma^k_{\mathbf{r}} \leq \Sigma^k_{\mathbf{c}'}$ *holds for each* $1 \leq k \leq m$.

Finally, the following famous theorem shows under which conditions a pair of integer vectors is bi-graphical.

Theorem 4.1 (Gale [52], Ryser [53])**.** *Let* $\mathbf{r} = (r_1, r_2, \ldots, r_m)$ *be a non-increasing integer vector and let* $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ *be an integer vector. The vector pair* $(\mathbf{r}, \mathbf{c})$ *is bi-graphical if and only if* $\Sigma^m_{\mathbf{r}} = \Sigma^m_{\mathbf{c}'}$ *and* $\mathbf{r} \trianglelefteq \mathbf{c}'$.

To find a realization of a bi-graphical vector pair, Ryser's [53] famous algorithm can be used to construct a bipartite graph $G = (U, V, E) \in \Omega(\mathbf{r}, \mathbf{c})$ in $\mathcal{O}(|U| + |V| + |E|)$ time (see Alg. 4.1).

4.2 Markov Chains

Due to its high importance for network analysis, the uniform sampling of bipartite graphs with fixed degrees has received much attention within the last decades. Hence, a large number of sampling algorithms has been proposed, from which we will briefly review the most important ideas. Such methods can roughly be divided into two groups.

The first group of algorithms [54, 37, 49, 55, 35] use approaches like *rejection sampling* or *importance sampling* to create random samples that are proven to be distributed according a certain target distribution, without having to rely on a necessary number of steps. In our discussion, we will call such algorithms *exact* sampling methods. Although they do not necessarily produce uniformly distributed samples, they can often be used for the unbiased estimation of expected values via Eq. 1.2. Harrison and Miller [35] showed that sequential importance sampling can be very

Algorithm 4.1: Ryser's algorithm for constructing a bi-graphical realization

Input: bi-graphical pair $(\mathbf{r}, \mathbf{c})$ of integer vectors.
Output: bipartite graph $G = (U, V, E) \in \Omega(\mathbf{r}, \mathbf{c})$.

$U \leftarrow \{u_i \colon 1 \leq i \leq m\}$
$V \leftarrow \{v_j \colon 1 \leq j \leq n\}$
$E \leftarrow \emptyset$
for $i = 1, \ldots, m$ **do**
  for $k = 1, \ldots, r_i$ **do**
    $j \leftarrow \arg\max\{c_j \colon j \in \{1, \ldots, n\} \wedge \{u_i, v_j\} \notin E\}$
    $E \leftarrow E \cup \{\{u_i, v_j\}\}$
    $c_j \leftarrow c_j - 1$
  end
end
return $G = (U, V, E)$

efficient in practice, although there are counter examples in which the approach behaves poorly [56]. Exploiting the connection between counting and sampling, Miller and Harrison [34] showed that the exact sampling of bipartite graphs is feasible for many small-sized presence-absence tables.

In this thesis, we will focus on a second group of algorithms based on the MCMC approach. In contrast to the *exact* sampling algorithms, MCMC methods typically produce *near-uniform* samples, i.e. they create samples according to a probability distribution that is arbitrary close to the uniform distribution. Currently, two MCMC-based methods are proven to produce near-uniformly distributed samples from $\Omega(\mathbf{r}, \mathbf{c})$ in polynomial running time.

- Bezáková et al. [57] presented a simulated annealing algorithm whose worst case running time is bounded from above by $\mathcal{O}((m+n)^{11} \ln^5(m+n))$.

- Tutte [58] showed how to transform the sampling problem into an associated perfect matching sampling problem. As we will specify in Chapter 6, the uniform sampling of perfect matchings can be processed in polynomial running time [19]. Thus, a bipartite graph can be sampled in polynomial time, too.

As the above-mentioned approaches are fairly complicated, they are primarily of theoretical interest. In contrast, a family of simple and intuitive Markov chains is widely used in practice.

4.2.1 Classical Switch Chain

The *classical switch chain* [17] has been used for decades to construct random bipartite graphs. It is based on its name-giving *classical switch* operation.

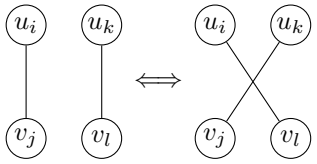

Figure 4.1: Illustration of a switch operation.

Definition 4.6 (classical switch)**.** *Let G be a bipartite graph with bi-adjacency matrix $M = (m_{ij})$.*

1. *Select four integers $1 \leq i < k \leq m$ and $1 \leq j < l \leq n$ uniformly at random.*
2. *Consider the 2×2 sub-matrix*
$$A := \begin{bmatrix} m_{ij} & m_{il} \\ m_{kj} & m_{kl} \end{bmatrix}.$$
3. *If A is either*
$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad \textit{or} \qquad \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$
switch the matrix entries $m_{ij} \leftrightarrow m_{kj}$ and $m_{il} \leftrightarrow m_{kl}$, otherwise do nothing.
4. *Return G.*

Fig. 4.1 illustrates the switch operation.

Definition 4.7 (classical switch chain)**.** *Let $G \in \Omega(\mathbf{r}, \mathbf{c})$.*

1. *Apply a classical switch operation to transform G into G'*
2. *If the maximal number of steps has been reached, return G'.*
3. *Set $G \leftarrow G'$ and go to Step 1.*

The classical switch chain produces a sequence of random variables $X = (X_0, X_1, \ldots)$ from $\Omega(\mathbf{r}, \mathbf{c})$. As each state is constructed from the previous one, the Markov property holds and thus, the classical switch chain is indeed a Markov chain.

Irreducibility As a switch operation does not change the degree of any vertex, it never produces an element $G \notin \Omega(\mathbf{r}, \mathbf{c})$. It is folklore that all bipartite graphs with fixed degree can be generated by iterated application of switches [59]. Thus, the classical switch chain is irreducible.

Aperiodicity Excluding one single exception, the classical switch chain is aperiodic whenever $m > 1$ and $n > 1$. The exception occurs if and only if $\mathbf{r} = \mathbf{c} = (1, 1)$. In this case, there is just one way to choose the sub-matrix A. As A will always have one of the requested forms, the random walk will oscillate between the two realizations. In all other cases however, a classical switch may choose a row or column with two entries being zero, or two entries being one. In such cases, the Markov chain will stay at the current state. As thus the associated state graph contains at least one loop, it is not bipartite. Consequently, the classical switch chain is aperiodic for all non-trivial instances.

Symmetry When two different states $x \in \Omega(\mathbf{r}, \mathbf{c})$ and $y \in \Omega(\mathbf{r}, \mathbf{c})$ can be transformed into each other by a classical switch operation, the associated transition probability $p(x, y)$ is

$$p(x, y) = \binom{m}{2}^{-1} \binom{n}{2}^{-1} = p(y, x),$$

as the random integers in Step 1 are chosen uniformly. Consequently, Corollary 2.4 shows that the classical switch chain's stationary distribution is the uniform distribution on $\Omega(\mathbf{r}, \mathbf{c})$.

Discussion By representing the bipartite graph G by its bi-adjacency matrix, a classical switch operation can be implemented to have a constant running time. However, the associated sampling algorithm has a memory requirement of $\Theta(mn)$.

Kannan et al. [17] successfully applied the canonical path method to show that the classical switch chain is rapidly mixing whenever the bipartite graph is *regular*. In doing so, they bounded the total mixing time of the classical switch chain from above by

$$\tau_{\max}(\varepsilon) \leq 8(nk)^{12} \cdot \ln\left(|\Omega| \cdot \varepsilon^{-1}\right), \tag{4.1}$$

where k is the degree of each node in a regular bipartite graph with $2n$ vertices. Miklós et al. [60] extended the class of rapidly mixing instances by *half-regular* bipartite graphs, in which only one of both vertex sets needs to be regular. Recently, Erdős et al. [61, 62] showed that the classical switch chain is rapidly mixing even when the bipartite graph is *almost* regular. However, it is still unknown whether this is true in general.

The classical switch chain has been customized for the uniform sampling of non-bipartite graphs, multi-graphs, and directed graphs with fixed degrees [21]. It is unclear whether these variants are rapidly mixing in general, although special cases are known, in which the total mixing time can be bounded from above by high-degree polynomials [63, 64].

4.2.2 Edge Switch Chain

There are several variants of the classical switch chain that aim at improving its efficiency by avoiding the random vertex selection in Step 1. For example, the so-called *edge switch chain* replaces Step 1 of the classical switch operation by a more informed edge selection [22, 65]. The chain is based on an arbitrary order of the edge set $E = \{e_1, e_2, \ldots, e_{|E|}\}$.

Definition 4.8 (edge switch)**.** *Let $G = (U, V, E)$ be a bipartite graph.*

1. *Select two integers $1 \leq a < b \leq |E|$ uniformly at random.*
2. *Consider the edges $e_a = \{u_i, v_j\} \in E$ and $e_b = \{u_k, v_l\} \in E$.*
3. *(a) If $e'_a := \{u_i, v_l\} \notin E$ and $e'_b := \{u_k, v_j\} \notin E$, create the edge set*

$$E' := (E \setminus \{e_a, e_b\}) \cup \left\{e'_a, e'_b\right\}.$$

   (b) Otherwise set $E' := E$.

4. *Return $G' := (U, V, E')$*

Definition 4.9 (edge switch chain)**.** *Let $G = (U, V, E) \in \Omega(\mathbf{r}, \mathbf{c})$.*

1. *Apply an edge switch operation to transform G into G'.*
2. *If the maximal number of steps has been reached, return G'.*
3. *Set $G \leftarrow G'$ and go to Step 1.*

Irreducibility The state graph of the edge switch chain is structurally identical to that of the classical switch chain. Following from the definition of the Markov chains, two states $x \neq y \in \Omega(\mathbf{r}, \mathbf{c})$ can be transformed into each other by a classical switch whenever they can be transformed into each other by an edge switch. More precisely, if

$$A = \begin{bmatrix} m_{ij} & m_{il} \\ m_{kj} & m_{kl} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

is a sub-matrix considered in Step 2 of a classical switch operation, then the two edges $e_a = \{u_i, v_j\} \in E$ and $e_b = \{u_k, v_l\} \in E$ can also be selected by an edge switch. Replacing the edges e_a and e_b by e'_a and e'_b is therefore equivalent to the classical switch. The case $A = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$ is symmetric. As the classical switch chain is irreducible, the edge switch chain is irreducible, too.

Aperiodicity The edge switch chain is aperiodic whenever G contains a vertex of degree two or more, as in this case, an edge switch operation may select $e_a \in E$ an $e_b \in E$ to have a common vertex. As the condition in Step 3 (a) will fail in such cases, the state graph contains a loop and thus, cannot be bipartite.

In the special cases where each node has a degree of one, however, the vertex pairs e'_a and e'_b cannot be included in E, and thus, the condition in Step 3 (a) will never fail. The state graph of the edge switch chain will then be bipartite. However, if every node has a degree of one, the vertex sets U and V must be of equal size. The sampling problem is then equivalent to the uniform sampling of a perfect matching in the complete bipartite graph with $m + n = 2m = 2n$ nodes. As each perfect matching corresponds to an $n \times n$ permutation matrix, we can easily address this problem by using a classical permutation algorithm like the Fisher-Yates shuffle [66].

Thus, the edge switch chain is aperiodic for all non-trivial cases.

Symmetry Whenever two different states $x \in \Omega(\mathbf{r}, \mathbf{c})$ and $y \in \Omega(\mathbf{r}, \mathbf{c})$ can be transformed into each other via a single edge switch operation, the associated transition probability is

$$p(x, y) = \binom{|E|}{2}^{-1} = p(y, x).$$

As the number of edges is constant, the transition probability function is symmetric. By Corollary 2.4, the stationary distribution of the edge switch chain is the uniform distribution on $\Omega(\mathbf{r}, \mathbf{c})$.

Discussion To efficiently select two edges at random in Step 2, the edge switch chain requires an additional data structure that keeps track of the edge set E. This data structure needs to be updated after each randomization step. As the number of edges is constant, we represent E by an array of integer pairs. To switch two edges, it suffices to modify the array entries at the positions a and b. An edge switch replaces $e_a = \{u_i, v_j\}$ by $e'_a = \{u_i, v_l\}$, and $e_b = \{u_k, v_l\}$ by $e'_b = \{u_k, v_j\}$, thus swapping the second component of each pair.

To still support the query of the existence of the edges e'_a and e'_b in Step 3 (a) in constant time, the edge switch chain additionally requires a binary matrix representation of the bipartite graph. Thus, an edge switch operation can be implemented with constant running time at the cost of an increased memory complexity of $\Theta(mn) + \Theta(|E|)$. This is in practical applications often about twice as large as the memory requirement of the classical switch chain.

4.2.3 Curveball

Recently, improvements to the classical switch chain like the *curveball* algorithm presented by Strona et al. [23] or the algorithm presented by Verhelst [67] received

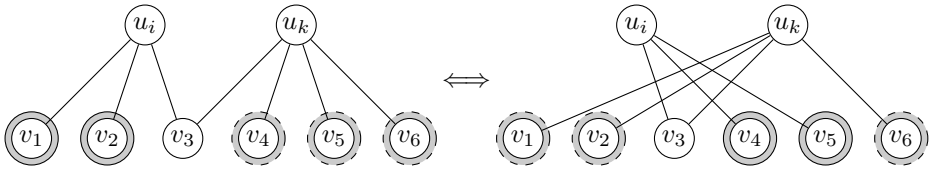

Figure 4.2: Visualization of a trade operation. Transforming the left graph into the right: $V_i = \{v_1, v_2\}$ (gray, solid border), $V_k = \{v_4, v_5, v_6\}$ (gray, dashed border). The set $S = \{v_4, v_5\}$ of size $|V_i|$ was chosen uniformly at random from $V_i \cup V_k$.

much attention. Such algorithms try to improve the efficiency of the sampling process by affecting multiple edges in each step. The key idea of the curveball algorithm is to replace the switch operation by the more complex *trade* operation.

Definition 4.10 (trade)**.** *Let $G = (U, V, E)$ be a bipartite graph.*

1. *Select two integers $1 \leq i < k \leq m$ uniformly at random.*

2. *Determine two disjoint vertex sets $V_i \subseteq V$ and $V_k \subseteq V$ defined by*

$$V_i := \{v \in V \colon \{u_i, v\} \in E \wedge \{u_k, v\} \notin E\},$$
$$V_k := \{v \in V \colon \{u_k, v\} \in E \wedge \{u_i, v\} \notin E\}.$$

3. *Select a subset $S \subseteq V_i \cup V_k$ of size $|S| = |V_i|$ uniformly at random.*

4. *Create a new edge set E' by defining*

$$E_i := \{\{u_i, v\} \colon v \in S\},$$
$$E_k := \{\{u_k, v\} \colon v \in (V_i \cup V_k) \setminus S\},$$
$$E_0 := \{\{u_i, v\}, \{u_k, v\} \colon v \in V_i \cup V_k\}$$
$$E' := (E \setminus E_0) \cup E_i \cup E_k.$$

5. *Return $G' := (U, V, E')$.*

Fig. 4.2 illustrates a trade operation. By design, a trade operation does not change the degree of any node in G as the set S is chosen to possess the same size as V_i.

Definition 4.11 (curveball)**.** *Let $G \in \Omega(\mathbf{r}, \mathbf{c})$.*

1. *Apply a trade operation to transform G into G'.*

2. *If the maximal number of steps has been reached, return G'.*

3. *Set $G \leftarrow G'$ and go to Step 1.*

Irreducibility Carstens [68] showed that the curveball chain is irreducible. Whenever the set S in Step 3 is chosen such that $|S \setminus V_i| = 1$, the trade operation is equivalent to a classical switch. Thus, each classical switch operation that can be applied to a bipartite graph $G \in \Omega(\mathbf{r}, \mathbf{c})$ can be simulated by an equivalent trade operation. Hence, the state graph of the classical switch chain is a subgraph of the corresponding state graph of the curveball chain. Consequently, as the state graph of the switch chain is strongly connected, so is the state graph of the curveball chain.

Aperiodicity A trade may return $G' = G$ when S is chosen in Step 3 to be identical with V_i. Thus, each state has a positive loop probability of

$$p(x, x) = \sum_{1 \leq i < k \leq m} \binom{m}{2}^{-1} \binom{|V_i \cup V_k|}{|V_i|}^{-1}.$$

Consequently, the curveball chain is aperiodic.

Symmetry When $x \in \Omega(\mathbf{r}, \mathbf{c})$ and $y \in \Omega(\mathbf{r}, \mathbf{c})$ are two different bipartite graphs that can be transformed into each other via a single trade operation, then the bi-adjacency matrices of x and y differ only at rows i and k. The probability of transforming x into y is thus equal to

$$p(x, y) = \binom{m}{2}^{-1} \binom{|V_i \cup V_k|}{|V_i|}^{-1}.$$

Carstens [68] showed that $p(x, y) = p(y, x)$ holds for every pair of states. Thus, the transition probability function is symmetric. By Corollary 2.4, the stationary distribution of the curveball chain is the uniform distribution on $\Omega(\mathbf{r}, \mathbf{c})$.

Discussion In contrast to the classical switch and to the edge switch operation, the trade operation is designed to possess a running time of $\mathcal{O}(|V|)$. To support the efficient construction of the edge and vertex sets during a trade operation, we use the bi-adjacency matrix representation of G. Thus, the curveball algorithm has a memory complexity of $\Theta(mn)$. Recently, Carstens and Kleer [65] proved that the curveball Markov chain is rapidly mixing, whenever the classical switch chain is.

4.3 Experiments on Mixing Time

In this section, we experimentally study the efficiency of the introduced algorithms for the uniform sampling of bipartite graphs with fixed vertex degrees. The efficiency of these sampling algorithms has often been assessed experimentally (see e.g. Refs. [36, 37, 38, 39] for variants of the switch chain and Refs. [23, 69] for the curveball algorithm). In most cases, such experiments are based on the observation of samples means as described in Section 2.4. However, none of the existing experimental surveys has yet systematically studied properties like the total mixing time of the

associated Markov chains on a large set of instances. In addition, the running time spent in each step is often ignored in experimental studies. Thus, our experiments complement existing results by a more solid methodological foundation.

In this set of experiments we will address the following main questions.

1. First, we want to study how the total mixing time of the Markov chains depends on the number $m + n$ of nodes in the bipartite graph.
2. Furthermore, how do properties of state graphs relate with the total mixing time of each Markov chain? Can we identify a certain property of the state graph that induces a large total mixing time?
3. Finally, we compare the efficiency of the three sampling algorithms to examine which chain is suited best for the randomization of bipartite graphs.

Data Sets We assessed the efficiency of each sampling method on the following data sets.

- To systematically study the properties of state graphs, we compiled a list of bi-graphical vector pairs describing the degrees of small bipartite graphs with up to 6+6 vertices. In doing so, we created a list of 1326 small instances.
- To examine the relation between the size $m + n$ of an input instance and the associated total mixing time, we created a couple of *scalable* instance classes. Each class of vector pairs has a parameter $n \in \mathbb{N}$ that defines the length of the vector $\mathbf{c}$.

Type A:	$\mathbf{r} = (n-1, n-2, 1, 1, 1)$,	$\mathbf{c} = (2, 2, \ldots, 2)$
Type B:	$\mathbf{r} = (n-1, n-2, 2, 1)$,	$\mathbf{c} = (2, 2, \ldots, 2)$
Type C:	$\mathbf{r} = (n-1, n-2, 3)$,	$\mathbf{c} = (2, 2, \ldots, 2)$
Type D:	$\mathbf{r} = (n-1, n-1, 1, 1)$,	$\mathbf{c} = (2, 2, \ldots, 2)$
Type E:	$\mathbf{r} = (n-2, n-2, 2, 2)$,	$\mathbf{c} = (2, 2, \ldots, 2)$
Type F:[1]	$\mathbf{r} = (\lceil n/2 \rceil, \lfloor n/2 \rfloor)$,	$\mathbf{c} = (\underbrace{1, 1, \ldots, 1}_{n \text{ times}})$

  As the members of each instance class are half-regular, the classical switch chain is known to be rapidly mixing [60]. As shown by Carstens and Kleer [65], the curveball chain is rapidly mixing, too. Thus, the total mixing time of the associated Markov chains can be bounded from above a polynomial function on $m + n$ and ε^{-1}.

  Instance class F is designed such that the state graph of the curveball Markov chain is the complete directed graph on $|\Omega(\mathbf{r}, \mathbf{c})|$ vertices. In each trade operation, the set $V_0 \cup V_1$ is equal to the vertex set V of the bipartite graph

[1] We thank Annabell Berger for bringing this class to our attention.

$G = (U, V, E)$. As a trade operation selects a subset $S \subseteq V$ uniformly at random, each state $x \in \Omega(\mathbf{r}, \mathbf{c})$ is equally likely the result of an operation. Consequently, the total mixing time of the curveball chain is exactly one for all members of class F.

- Bipartite networks like "Darwin's finches" (see Table 4.1) have been objects of study in ecological null model analysis for many years. We compiled a list of bi-graphical vector pairs that correspond to the degrees of the 62 binary bipartite networks in Bascompte's *web-of-life* [70] data set (see Table B.2 in Appendix B). The number $m + n$ of vertices of these networks range from 6 to 1500.

4.3.1 Structural Properties of State Graphs

We started our experimental analysis by relating the total mixing time of each Markov chain with structural properties that can be calculated from a state graph. In this set of experiment, we focused on the following basic properties.

- The size $m + n$ of the input instance.
- The number $|\Omega(\mathbf{r}, \mathbf{c})|$ of states of the state graph.
- The total mixing time $\tau_{\max}(\varepsilon)$ for $\varepsilon = 0.01$.
- The average loop probability

$$\ell := \sum_{x \in \Omega(\mathbf{r},\mathbf{c})} \pi(x) \cdot p(x, x) = |\Omega(\mathbf{r}, \mathbf{c})|^{-1} \sum_{x \in \Omega(\mathbf{r},\mathbf{c})} p(x, x).$$

Experiment 4.1 To gain an impression concerning the dimension of the total mixing time and its relation to other properties, we processed each of the small instances. Even though the vector pairs are small, the resulting state graphs can be large. For example, the largest state graph processed in this experiment corresponds to the integer vectors $\mathbf{r} = \mathbf{c} = (3, 3, 3, 3, 3, 3)$ and has $|\Omega(\mathbf{r}, \mathbf{c})| = 297,200$ states. Fig. 4.3 shows how the properties of the state graphs relate. We will discuss some essential observations.

- We observe that the total mixing time of both switch chains is significantly larger than that of the curveball chain. However, as a single step of the curveball is more expensive than that of the other chains, our observations do not yet imply the efficiency of the curveball in practice. We will assess the practical efficiency of the sampling algorithms in a different set of experiments.
- By comparing the size $|\Omega(\mathbf{r}, \mathbf{c})|$ of the state graphs with the total mixing time, we find that certain instances with comparably small state graphs have a high total mixing time. This observation holds for each type of Markov chain.

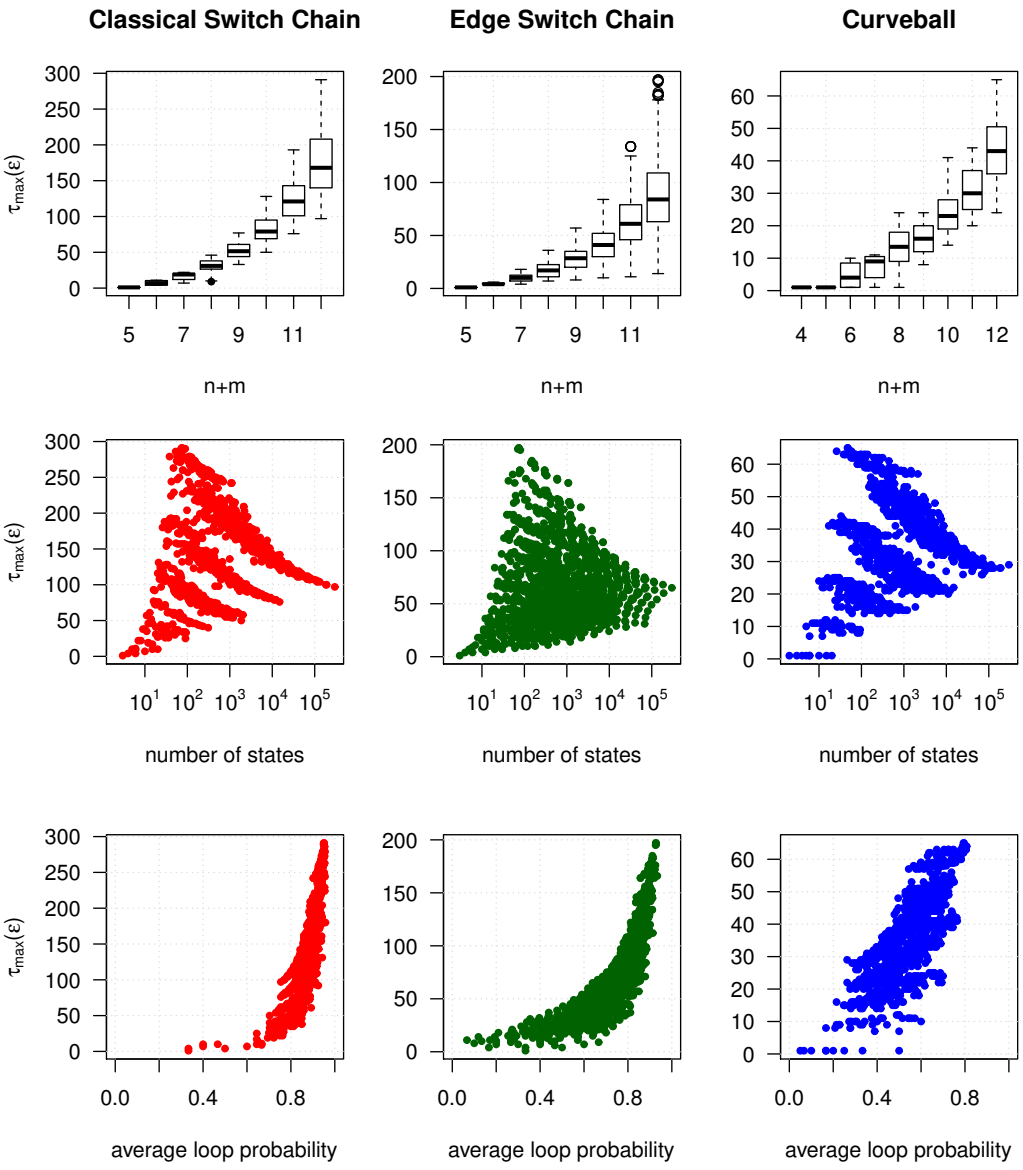

Figure 4.3: Number $m+n$ of nodes, total mixing time $\tau_{\max}(\varepsilon)$, average loop probability ℓ, and number $|\Omega(\mathbf{r},\mathbf{c})|$ of states for the set of small instances ($\varepsilon = 0.01$).

- By comparing the total mixing time of each Markov chain with the associated average loop probability, we found that there is a strong correlation between both quantities. In general, the larger the average loop probability, the larger is the total mixing time. In addition, we found that the classical switch chain has a higher loop probability than the other chains. May the small loop probability be the reason for the efficiency of the curveball chain? We will study the influence of the loops in a subsequent set of experiments.

Experiment 4.2 Next, we further investigated how the total mixing time depends on the instance size $m + n$. For this purpose, we considered the *scalable* instance classes. By letting the parameter n grow, we produced several members of each instance class, for which we constructed the associated state graphs and calculated the total mixing time. Figs. 4.4 and 4.5 show the results of this experiment. We summarize some important observations.

- We observe that the total mixing time of the curveball chain grows considerably slower than that of the other Markov chains. This observations holds for each instance class. We will further assess the growth rate of the total mixing time in a forthcoming experiment.

- In addition, we observe that the total mixing time of the edge switch chain is smaller than that of the classical switch chain in most cases. Exceptions occur for the instances of type C and F.

  In case of class C, the edge switch chain's total mixing time is higher than that of the classical switch chain. This may be explained by the observation that its average loop probability exceeds that of the classical switch chain for this class of instances.

  For class F, the total mixing time of the classical and the edge switch chain align, as their associated state graphs are identical.

- By considering the average loop probability, we find that the classical and edge switch chain's loop probability increases with growing instance size, while it decreases for the curveball chain. We will further investigate the connection between the average loop probability and the total mixing time in a following set of experiments.

Experiment 4.3 As the members of each instance class are half-regular, and as ε is constant, the total mixing time can be bounded from above by a polynomial function on the input size $m + n$. Assuming that this upper bound is sharp, we model the total mixing time to be a function of the form

$$\begin{aligned} & f(m+n) = c \cdot (m+n)^k \\ \Leftrightarrow \quad & \ln f(m+n) = \ln c + k \ln(m+n). \end{aligned}$$

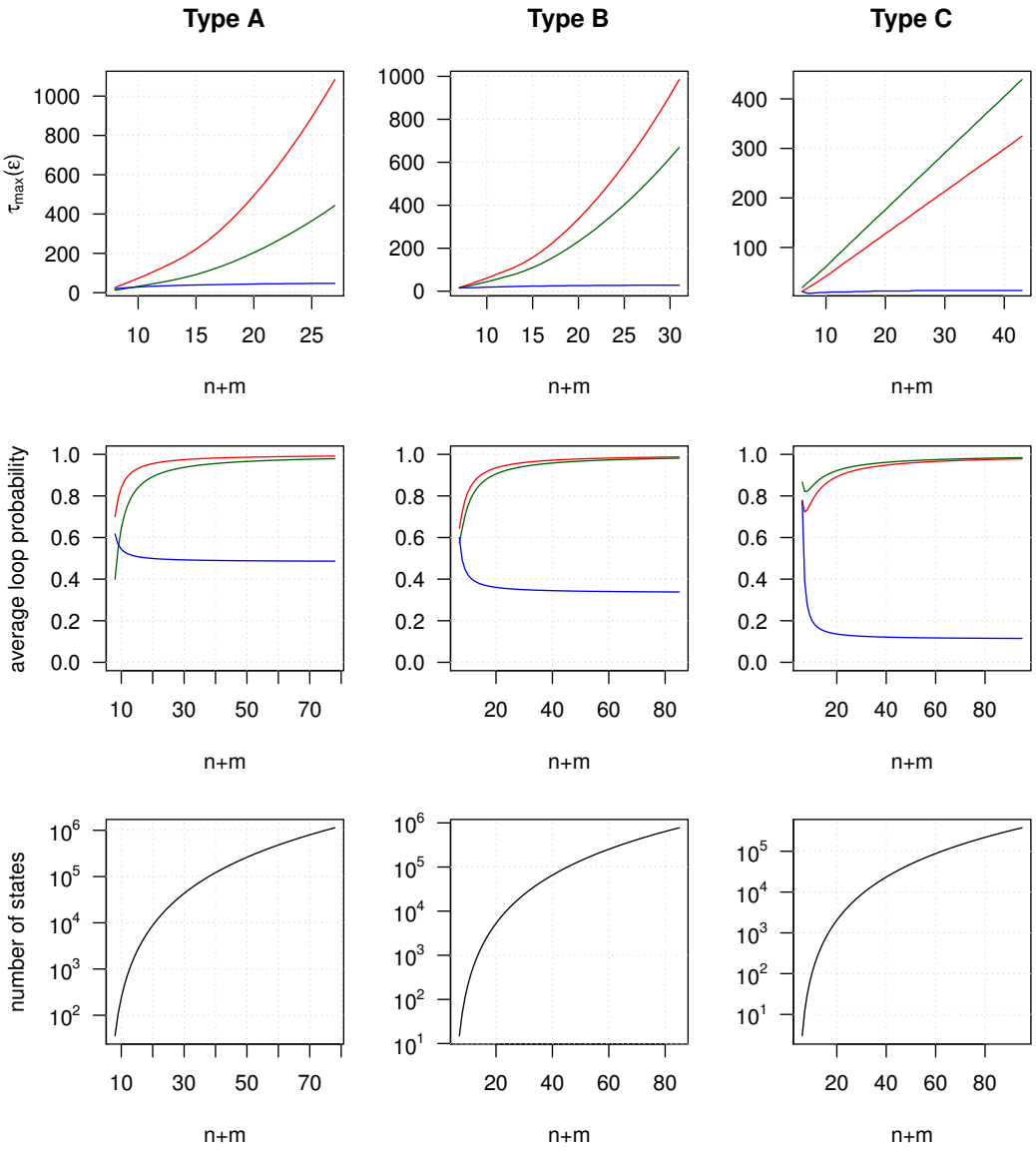

Figure 4.4: Total mixing time, average loop probability, and number of states for members of instance classes A, B, C. Red: classical switch chain. Green: edge switch chain. Blue: curveball chain. ($\varepsilon = 0.01$).

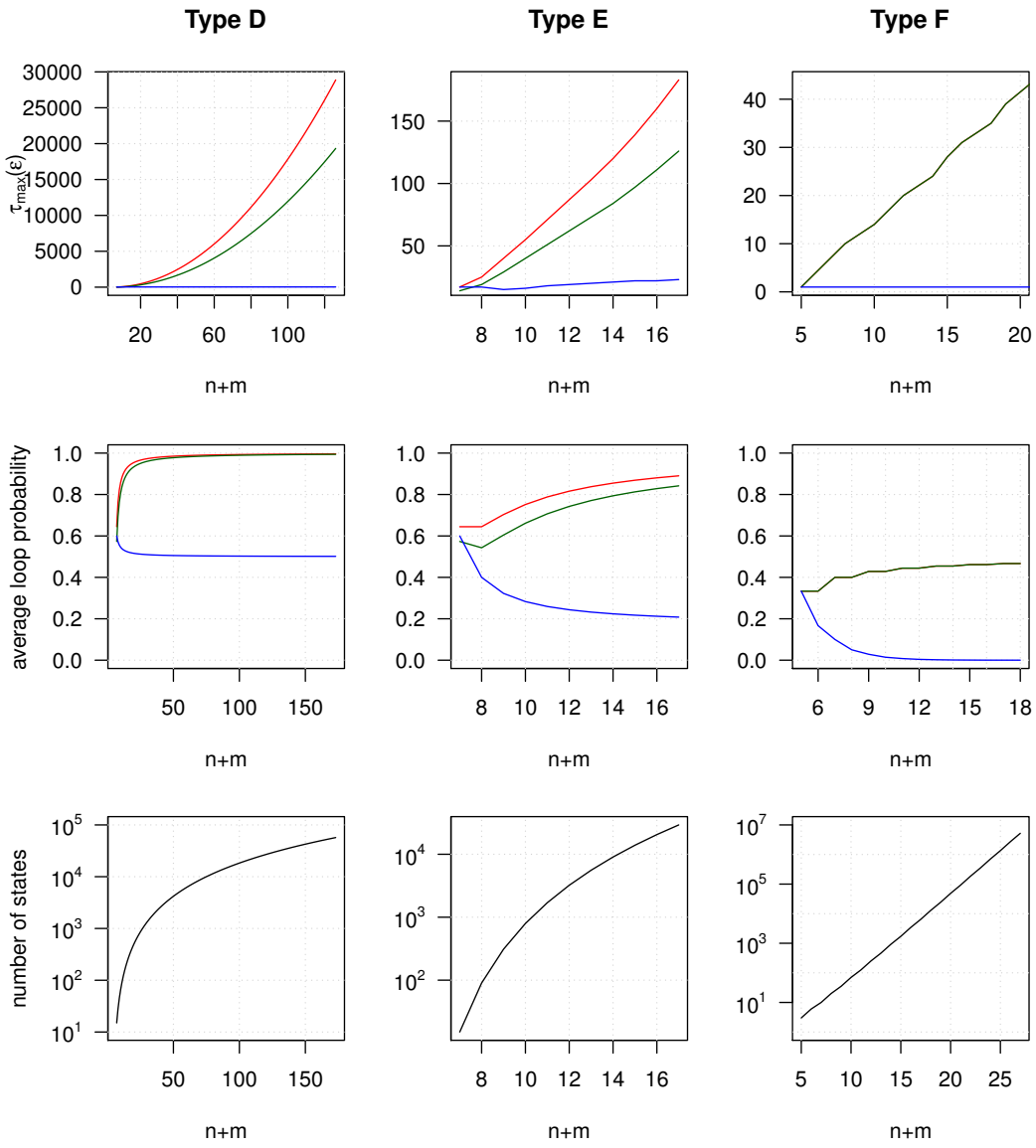

Figure 4.5: Total mixing time, average loop probability, and number of states for members of instance classes D, E, F. Red: classical switch chain. Green: edge switch chain. Blue: curveball chain. ($\varepsilon = 0.01$). In case of class F, the total mixing time of the classical and edge switch chain align.

In this experiment, we experimentally determined the degree k that describes the asymptotic growth of the total mixing time for each of our instance classes. Having calculated the total mixing time of several members of a certain instance class, we gained a set of data points

$$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N),$$

where x_i denotes the number $m + n$ of nodes of the i-th processed instances, and y_i is the associated total mixing time. Based on these data points, we applied the following two-step procedure to estimate the degree k of the polynomial that fits best to our observations.

1. First, we evaluated whether our assumption is valid, i.e. whether the polynomial model fits well to our data. For this purpose, we transformed each data point by taking the natural logarithm of the number $m+n$ of nodes, and of the associated total mixing time. If the transformed data looks clearly non-linear, we must reject our assumption of polynomial mixing time.

2. When our assumption of polynomial mixing time is justified, we applied a conjugate-gradient method to estimate the model parameters c and k that minimize the sum of squared errors

$$\min \sum_{i=1}^{N} \left(y_i - c \cdot (x_i)^k \right)^2.$$

   For this purpose, we used the general-purpose optimization method `optim` from the R software framework.

Fig. 4.6 shows the transformed data points. We discuss some essential observations.

- Ignoring the first few data points of each class, we observe that the polynomial model fits well to the classical and the edge switch chain, as the transformed data relates in a linear manner. The deviation of the first few data points may be explained by the lack of terms of minor degree in our very simple model.

- As the polynomial model fits well to our observations in case of the classical and edge switch chain, we estimated the degree k of the polynomials describing the associated total mixing time (see Table 4.2, Figs. D.1 and D.2 in Appendix D show the estimated polynomials).

- We observe that the total mixing time grows super-linearly in most cases. Consequently, a linear number of switches must ultimately fail and will result in a non-random sample. In many cases, even a quadratic number of switches would not be sufficient. As the number $|E|$ of edges is a linear function on the number $m+n$ of vertices for our instances, our observations imply that even a number of $\mathcal{O}(|E|)$ steps of both switch chain variants is insufficient to produce random samples.

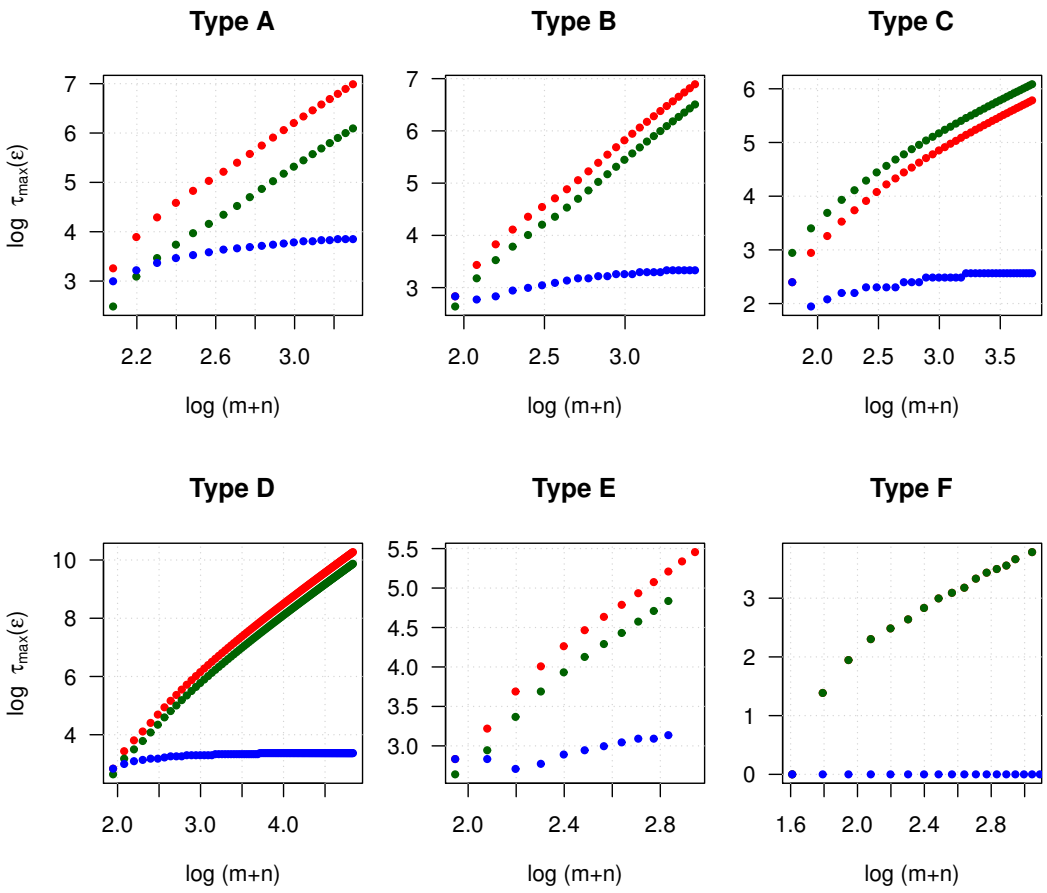

Figure 4.6: Number of nodes and associated total mixing time of the classical switch chain (red), edge switch chain (green), and the curveball chain (blue). ($\varepsilon = 0.01$.)

Table 4.2: Estimated degrees k of polynomials.

Type	Classical Switch	Edge Switch
A	2.68	2.65
B	2.48	2.44
C	1.28	1.25
D	2.12	2.11
E	2.28	2.23
F	1.59	1.59

- In contrast to the switch chain variants, we observe that the polynomial model poorly describes the total mixing time of curveball chain, as the transformed data points grow sub-linearly. Consequently, the total mixing time of the curveball must be a sub-linear function on the instance size $m + n$ for all instance classes considered here. A more refined experiment shows that the total mixing time of the curveball chain is almost constant for these instances. This observation is plausible as the instances classes considered here possess a vertex set U of constant size. We will assess the efficiency of the sampling methods on more realistic instances in a further set of experiments.

Summary In this set of experiments, we investigated the efficiency of three sampling algorithms on a set of instances by calculating the total mixing time of the associated Markov chains. In doing so, we found that the total mixing time of the curveball chain is significantly smaller than that of the other chains. Our findings qualitatively agree with existing experimental results [23, 69].

By relating the total mixing time with properties of the associated state graphs, we found that the total mixing time in our experiments does not correlate with the number of states. In the contrary, we found that the largest total mixing time occurs when the number of states is small.

We assessed the growing behavior of the Markov chain's total mixing time on special classes of scalable instances by experimentally determining the degree k of the polynomial functions that describe how the total mixing time relates to the input size. In doing so, we found that the classical and the edge switch chain require a super-linear number of steps, while the total mixing time of the curveball chain is almost constant. We conclude that the curveball chain is superior to the other chains in terms of iterations.

4.3.2 Influence of Loops

The previous experiments showed that the total mixing time of the curveball chain is smaller than that of the other chains. This may have several reasons. For example,

its efficiency may be caused by its small loop probability. As an alternative explanation, the complex structure of the state graphs may accelerate the convergence of the curveball chain. In this set of experiments, we addressed the reason for the efficiency of the curveball chain by assessing the influence of the loops on the total mixing time.

Methodology To assess the influence of the loop probability, we artificially manipulated the state graphs $\Gamma = (\Omega, \Psi)$ of the three Markov chains. For this purpose, we applied a two-step procedure to modify the weights $p\colon \Omega \times \Omega \to [0, 1]$ of the transition arcs so that the average loop probability becomes equal to a predefined constant ℓ.

1. In the first step, we chose a real number $\alpha \in (0, 1)$ and reduced the weight of all loop arcs in Ψ by that quantity. In doing so, we created the modified edge weights $p'\colon \Omega \times \Omega \to [0, 1]$ with

$$p'(x, y) := \begin{cases} p(x, y), & \text{if } x \neq y, \\ p(x, y) - \alpha, & \text{else.} \end{cases}$$

2. By reducing the loop probability, the transition matrix of the associated Markov chain is not longer stochastic, as each row sums to

$$\sum_{y \in \Omega} p'(x, y) = 1 - \alpha.$$

   To fix this problem, we applied a second step to rescale the weight of all transitions $(x, y) \in \Psi$ to $q\colon \Omega \times \Omega \to [0, 1]$ such that

$$\forall (x, y) \in \Psi\colon q(x, y) := \frac{p'(x, y)}{1 - \alpha}.$$

   It is easy to see that the transition probability $q\colon \Omega \times \Omega \to [0, 1]$ is symmetric whenever p is. Thus, the edge weights q of the modified state graph define an ergodic Markov chain whose stationary distribution is uniform.

By reducing the weight of the loops, we can inflate and deflate the average loop probability of a state graph to values within a certain range. For this purpose we have to determine the amount α by which the weight of each loop arc must be reduced such that the average loop probability of the state graphs becomes equal to a constant ℓ. We can calculate α by using the following equation.

$$\begin{aligned} \ell &= \frac{1}{|\Omega|} \sum_{x \in \Omega} q(x, x) \\ \Leftrightarrow \ell|\Omega| &= \sum_{x \in \Omega} q(x, x) = \sum_{x \in \Omega} \frac{p'(x, x)}{1 - \alpha} = \sum_{x \in \Omega} \frac{p(x, x) - \alpha}{1 - \alpha} = \frac{S - |\Omega|\alpha}{1 - \alpha}, \end{aligned}$$

where $S := \sum_{x \in \Omega} p(x, x)$ is the trace of the original transition matrix P. Solving this equation for α gives the amount by which the weight of each loop must be reduced such that the average transition probability of the modified state graph becomes equal to ℓ:

$$\alpha = \frac{S - |\Omega|\ell}{|\Omega| - |\Omega|\ell}.$$

As we must ensure that each arc weight is non-negative, the variable α must not be chosen larger than $p_{\min} := \min\{p(x, x) \colon x \in \Omega\}$. It is easy to show that the smallest possible value $\ell_{\min}$, to which the average loop probability ℓ may be scaled to, is

$$\ell_{\min} := \frac{S - |\Omega|p_{\min}}{|\Omega| - |\Omega|p_{\min}}.$$

By scaling the average loop probability to a constant of $\ell \geq \ell_{\min}$ and calculating the total mixing time of the modified state graphs, we may gain insights into the influence of the loops on the total mixing time.

Experiment 4.4 We started this set of experiments by evaluating the influence of the loops on the total mixing time of each Markov chain. For this purpose, we scaled the average loop probability of each state graph to a set of values $\ell \in [\ell_{\min}, 1)$ and calculated the total mixing time of the modified state graph. Fig. 4.7 shows how the total mixing time depends on the average loop probability of four exemplary small instances.

- As the state graphs of the classical switch chain and the edge switch chain are structurally identical, the corresponding curves align perfectly.
- Interestingly, the total mixing time behaves non-monotonically for a few instances in which the number of states is small (see Fig. 4.7, bottom right). Although this may seem counter-intuitive at first sight, there is a plausible explanation, as the corresponding state graph is "almost" bipartite when the loop probability is small. In such a case, Alg. 1.1 oscillates between two disjoint sets of states until a rarely occurring loop is used. For most instances, however, we observed that the total mixing time grows monotonically.
- By considering the logarithmic scale of the y-axis, we observe that the total mixing time of the Markov chains appears to be an exponential function on the average loop probability for most instances.
- In addition, we observe that if scaled to the same average loop probability, the total mixing times of the switch chain variants and the curveball chain are very close.

We conclude that the average loop probability has a large influence on the total mixing time of a Markov chain.

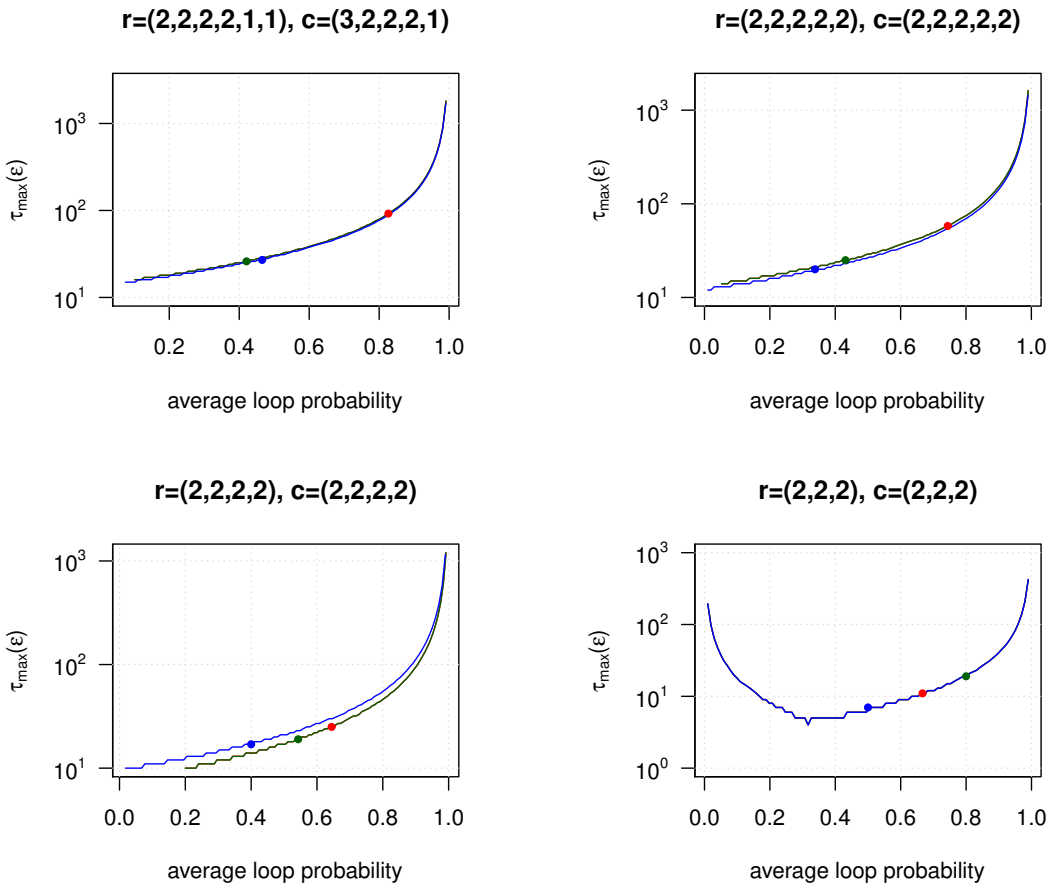

Figure 4.7: Average loop probability and total mixing time of the classical switch chain (red), edge switch chain (green), and curveball chain (blue) on four exemplary instances. The three points indicate the original average loop probability of each Markov chain. ($\varepsilon = 0.01$.)

Experiment 4.5 To assess whether the small loop probability of the curveball chain is the primary reason for its efficiency, we applied a refined experiment. In this experiment, we modified the state graphs of the classical switch and the curveball Markov chain such that their average loop probability becomes $\ell = 1/2$. (The choice of this constant is purely arbitrary and does not influence the qualitative result of this experiment.) As before, we calculated the total mixing time of the modified state graphs. Here, we denote by $\tau_{\max}^{\text{switch}}(\varepsilon)$ and $\tau_{\max}^{\text{curve}}(\varepsilon)$ the total mixing time of the switch chain and the curveball Markov chain, respectively. Then, the quotient

$$q(\varepsilon) := \frac{\tau_{\max}^{\text{switch}}(\varepsilon)}{\tau_{\max}^{\text{curve}}(\varepsilon)}$$

describes the speedup of the curveball in comparison to the switch chain. If this quotient is larger than one, the curveball mixes faster. We calculated the ratio $q(\varepsilon)$ with $\varepsilon = 0.01$ for several members of our scalable instance classes. Fig. 4.8 shows the result of this experiment.

- We observe that the modified switch chain is typically superior to the modified curveball when n is small, as the ratio q is smaller than one. In contrast, when the size of the instances grows larger, the curveball chain becomes more and more efficient.
- In case of the instance classes C and E, the total mixing time of the curveball chain is only fairly smaller than that of the switch chain and we cannot observe a clear trend.

As the average loop probability of the modified state graphs is identical, we can exclude the influence of the loops on the total mixing time. Consequently, we conclude that the complex structure of the curveball's state graph has a positive effect on the efficiency of this Markov chain.

Summary In this set of experiments, we addressed the influence of the loops on the total mixing time of the Markov chains. In a first experiment, we artificially scaled the average loop probability of each Markov chain to a set of values between zero and one, and calculated the total mixing time of the modified chains. In doing so, we found that the average loop probability has a huge influence on the total mixing time.

In a second experiment, we analysed whether the small loop probability is the primary reason for the efficiency of the curveball chain. For this purpose, we scaled the average loop probability of the switch and curveball chain to a common constant and calculated the total mixing time of the modified chains. In doing so, we found that due to its more complex structure, the curveball Markov chain is superior to the switch chain when the size of the instances is sufficiently large. We conclude that its small loop probability is an important but not the sole reason for the efficiency of the curveball chain.

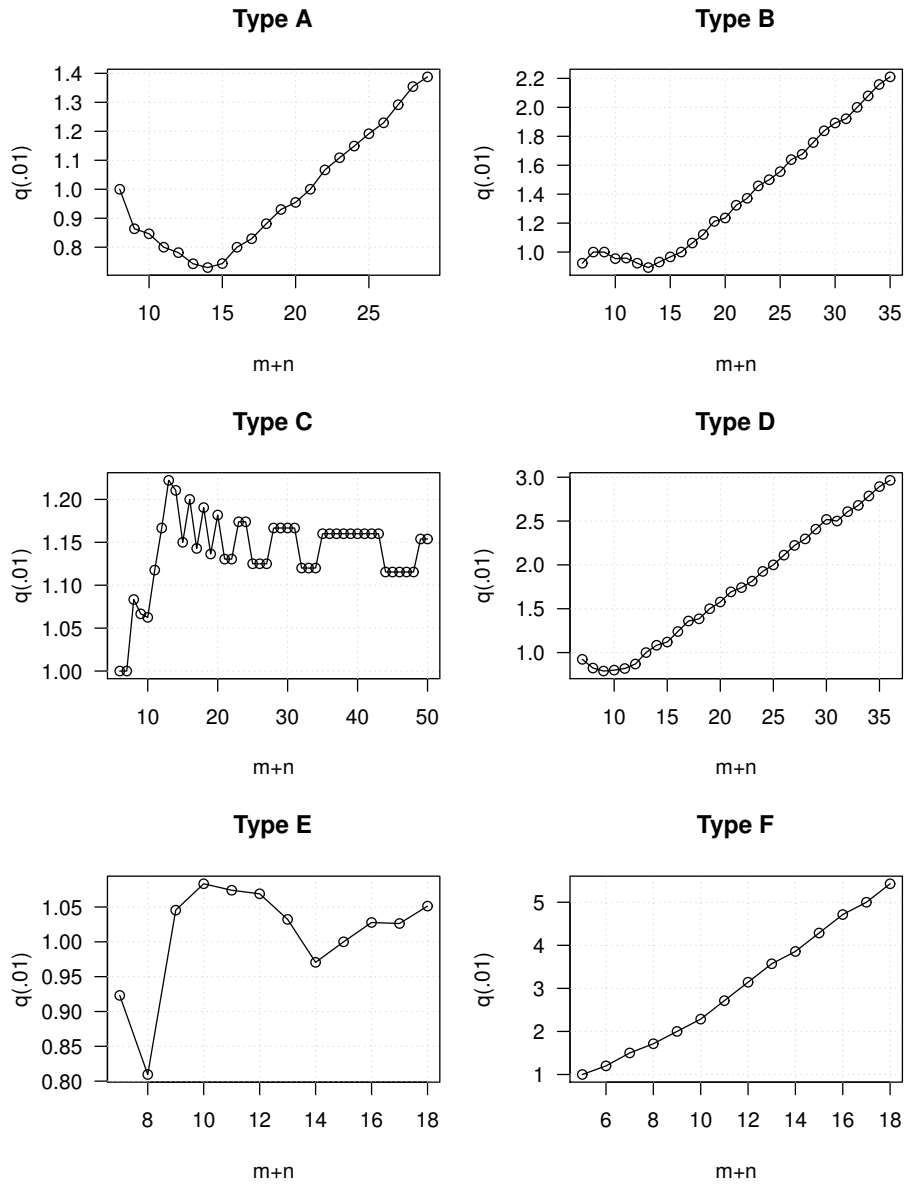

Figure 4.8: Quotients of the total mixing time of the modified curveball and modified classical switch chain. ($\varepsilon = 0.01$.)

4.3.3 Quality of Bounding Techniques

In some cases, techniques like the canonical path method can successfully be applied to gain a general upper bound on the total mixing time of a particular Markov chain. When applied successfully, such upper bounds most often are high-degree polynomials on the size of the input (see Ineq. 4.1). Such upper bounds are far too large to be applicable in practice, where more than a linear number of steps is infeasible when the input size is sufficiently large.

However, it might be the case that an upper bound on the total mixing time is just too pessimistic. Since the bounding techniques in Markov chain analysis are often fairly general and worst-case instances in terms of total mixing time are not known, it is not clear, whether the upper bounds gained by such methods are tight for some worst-case instance. For practical applicability, however, it is of eminent importance to find as sharp bounds as possible. In general, there is very little knowledge about the gap between the so established upper bounds and tight bounds for actual worst-case instances. Probably most researches will suspect that a considerable real gap exists, but for specific Markov chains it is unknown how many orders of magnitude this gap may be large. For that reason, we believe that the true total mixing time might be much smaller than proven by theoretical methods. Therefore, we address the following questions in a set of experiments.

1. Is the total mixing time as large as the bounding methods propose, or is it possible that the bounding methods are just not precise enough to tightly bound the total mixing time? The latter case would support the thesis that the total mixing time actually is far smaller than known in theory.

2. Which bounding method has the best potential and could lead to better results when further information about the structure of state graphs is given?

Experiment 4.6 In a first experiment, we processed the set of small instances and computed the following quantities from the associated state graphs

- the total mixing time $\tau_{\max}(\varepsilon)$ for $\varepsilon = 0.01$,
- its lower and upper spectral bound for $\varepsilon = 0.01$ via Ineq. 2.6 and 2.7, and
- the canonical path congestion bound for $\varepsilon = 0.01$ via Ineq. 2.10.

  To compute the congestion bound via Ineq. 2.10, we applied the path construction scheme presented by Kannan et al. [17] to concrete state graphs. As this path construction scheme was used to gain theoretical bounds without full knowledge of a Markov chain's structure, applying it to concrete state graphs shows how sharp such bounds could be if full structural information were available. Consequently, the general upper bounds can not be better than the bounds gained by applying the scheme on an actual state graph.

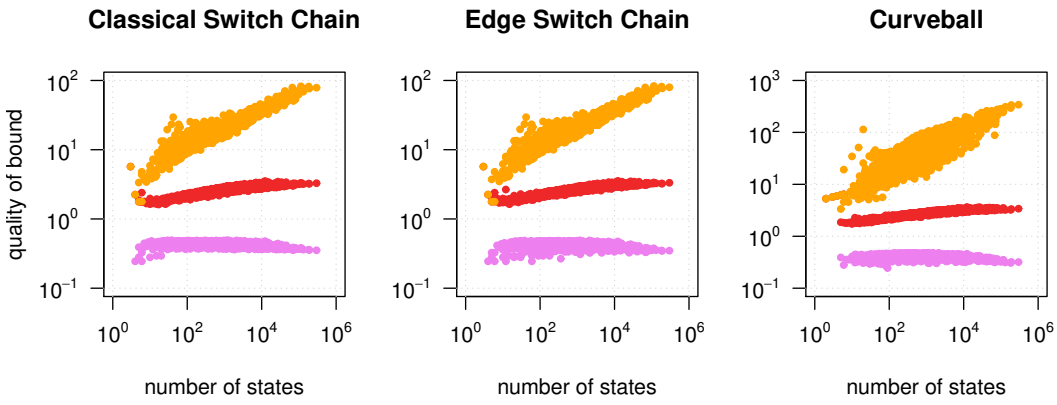

Figure 4.9: Quotient of congestion bound (orange), upper spectral bound (red), and lower spectral bound (violet) with total mixing time for the set of small instances.

We use the quotient of a lower or upper bound and the total mixing time to assess the quality of the bounding method. Fig. 4.9 shows the results of this experiment.

- We immediately observe that the lower and upper spectral bound are quite close to the total mixing time for all instances as the quotient is near one in all cases.

- In contrast, the congestion bound is far larger than the total mixing time, and its quality decreases with the growing size of a state graph. In case of the curveball chain, the congestion bound is up to several hundred times larger than the associated total mixing time for this set of instances.

Experiment 4.7 In a second experiment, we applied the previous experimental setup to members of the *scalable* instance classes. While Fig. 4.10 exemplarily shows the results of this experiment for instance class C, Figs. D.3 and D.4 in Appendix D show the results for all instance classes.

- We observe that the gap between the congestion bound and the total mixing time grows with the input size $m+n$ for each Markov chain. The same is true for the upper spectral bound, but in a much slower way.

- Comparing the lower spectral bound with the total mixing time of both switch chain variants, we observe that these quantities have an almost linear relationship for some of our instance classes (see Fig. D.5 and D.6 in Appendix D). Thus, the lower spectral bound can be used as a very good approximation for the total mixing time. The reason for this close relationship is an interesting topic for future studies. In contrast, the relationship between the total mixing time and its lower spectral bound is more complicated in case of the curveball.

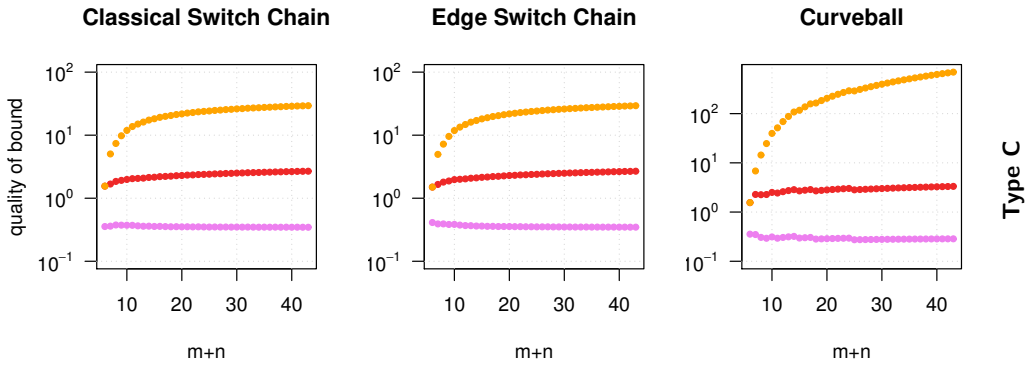

Figure 4.10: Quotient of congestion bound (orange), upper spectral bound (red), and lower spectral bound (violet) with total mixing time for the *scalable* instance class C.

Summary Our observations indicate that the theoretical bounds gained by the canonical path method are likely too pessimistic. Moreover, although we know the exact structure of a state graph in our experiments (which can never be the case in a normal practical scenario) the congestion bound is far larger than the total mixing time. Thus, the canonical path method will not lead to tight bounds, even when further information about a state graphs structure is included. This gives hope to the hypothesis that the true total mixing time is smaller than existing theory is able to prove.

4.3.4 Empirical Mixing Time

In the next set of experiments, we assessed the efficiency of the sampling algorithms on larger instances, for which the construction of state graphs in infeasible. Without a state graph, we can neither calculate the exact mixing time nor its lower or upper spectral bounds. Instead, we approximate the empirical mixing time. In doing so, we can process much larger instances for which the construction of the state graph is infeasible.

Methodology To evaluate the empirical mixing time, we need to approximate the t-step-distributions $q_s^{(t)}$ for increasing values of t, and the limiting distribution η with respect to an auxiliary function $f\colon \Omega(\mathbf{r},\mathbf{c}) \to \mathbb{R}$. To approximate the t-step distributions, we simulated t steps of each Markov chain to produce $N = 10^6$ random samples from $\Omega(\mathbf{r},\mathbf{c})$ according to $p_s^{(t)}$. By evaluating the auxiliary function on each random sample, we gain an approximation of $q_s^{(t)}$. To approximate the limiting distribution η, we evaluated the auxiliary function on 10^6 random samples generated by an exact sampling algorithm similar to that suggested by Miller and Harrison [34].

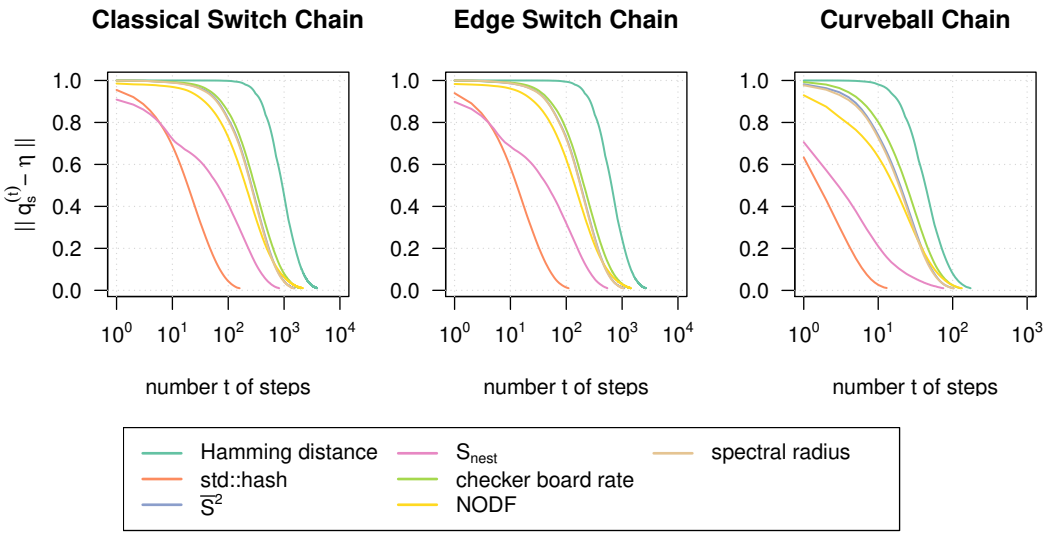

Figure 4.11: Total variation distance to limiting distributions of several auxiliary functions for the "Darwin's finches" instance. The definitions of the metrics can be found in Appendix C.

Initial State The empirical mixing time requires a fixed initial state $s \in \Omega(\mathbf{r}, \mathbf{c})$. As we will show later in Chapter 6, the choice of the initial state may have a large influence on the efficiency of the sampling methods. In this chapter, however, we ignore the question of what makes a good initial state and use a deterministically constructed bipartite graph as a common initial state for each Markov chain. For this purpose, we use Ryser's algorithm [53] to construct a realization of the bi-graphical vector pairs $(\mathbf{r}, \mathbf{c})$.

Experiment 4.8 The empirical mixing time $\bar{\tau}_s(\varepsilon)$ is defined with respect to an auxiliary function $f\colon \Omega(\mathbf{r}, \mathbf{c}) \to \mathbb{R}$ that is evaluated on a large set of random samples. In a first experiment, we tested several auxiliary functions, whose definitions can be found in Appendix C, on a large set of instances (see Fig. 4.11 for the example of "Darwin's finches").

- We observe that the Hamming distance metric is the slowest to approach zero. This metric is defined as the function $f(x) := h(s, x)$, where s is the initial state of the Markov chain and $h\colon \{0,1\}^{m \times n} \times \{0,1\}^{m \times n} \to \mathbb{N}$ is the *Hamming distance* between matrices.

As the Hamming distance to the initial state gives the sharpest approximation of the total mixing time, we will further use this metric as the auxiliary function in all subsequent experiments.

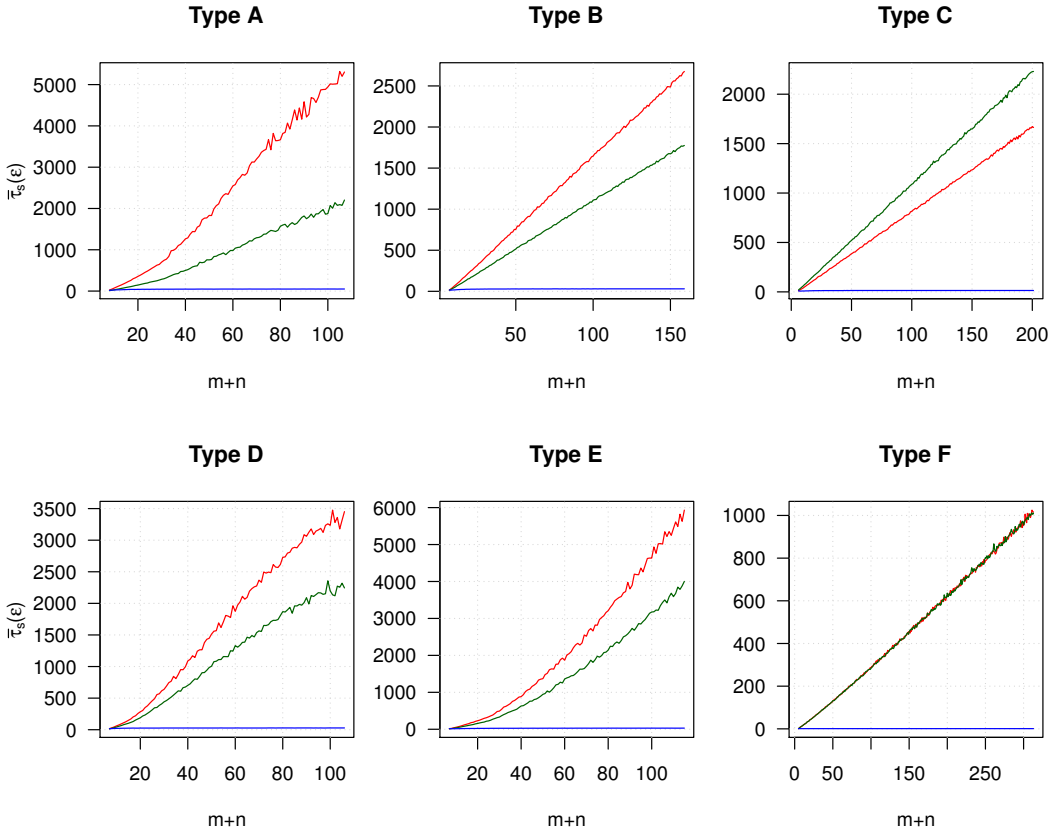

Figure 4.12: Empirical mixing time $\bar{\tau}_s(\varepsilon)$ of the scalable instance classes with respect to the Hamming distance metric. Red: classical switch chain. Green: edge switch chain. Blue: curveball chain. ($\varepsilon = 0.01$, $N = 10^6$ samples).

Experiment 4.9 Next, we approximated the empirical mixing time $\bar{\tau}_s(\varepsilon)$ for members of the scalable instance classes. As before, we used $N = 10^6$ random samples to approximate the probability distributions $q_s^{(t)}$ and η. In doing so, we could easily process instances for which the construction of a state graph is infeasible. Fig. 4.12 shows the result of this experiment.

- The experiment qualitatively confirms our previous observations made while assessing the total mixing time of the scalable instance classes. In particular, the classical switch chain is least efficient for the classes A,B,D, and E, while the curveball requires just a few steps to produce random samples.

- Similar as before, the empirical mixing time of the classical switch chain is smaller than that of the edge switch chain in case of class C. In addition, the empirical mixing times of both chains align for instances of class F.

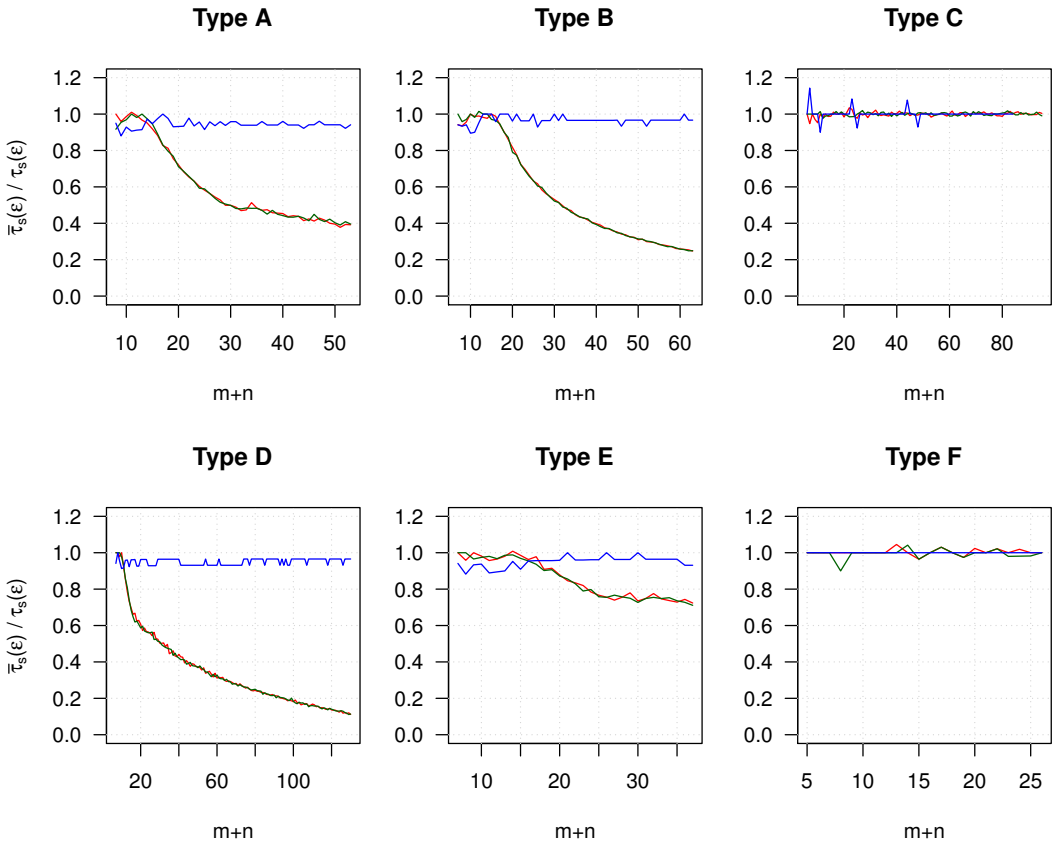

Figure 4.13: Quality of empirical mixing time $\bar{\tau}_s(\varepsilon)$ for the scalable instance classes with respect to the Hamming distance metric. Red: classical switch chain. Green: edge switch chain. Blue: curveball chain. ($\varepsilon = 0.01$, $N = 10^6$ samples).

From our observations, we conclude that the use of $\bar{\tau}_s(\varepsilon)$ instead of $\tau_{\max}(\varepsilon)$ does not change the qualitative results of our experiments.

Experiment 4.10 To quantify the quality of the lower bound $\bar{\tau}_s(\varepsilon)$, we calculated the ratio $r_s(\varepsilon) := \bar{\tau}_s(\varepsilon)/\tau_s(\varepsilon)$ for several members of each instance class. Fig. 4.13 shows how the quality of the lower bound depends on the size $m+n$ of the instances.

- As $\bar{\tau}_s(\varepsilon)$ is a lower bound of $\tau_s(\varepsilon)$, the ratio $r_s(\varepsilon)$ can in theory not be larger than one. However, we observe that it is sometimes slightly larger than one. This is caused by the fact that we do not calculate the empirical mixing time $\bar{\tau}_s(\varepsilon)$ precisely but rather use an approximation. We could improve the quality of our approximation by increasing the number N of samples used to create the sampling histograms.

- Considering the switch chain variants, we observe that the quality of the lower bound depends on the type of the instances. For instance classes A, B, D, and E, the ratio $r_s(\varepsilon)$ rapidly decreases, while it stays constant for instance classes C and F.

- Considering the curveball chain, we observe that the empirical mixing time is very tight in all cases. This is most likely caused by the small scale of its total mixing time induced by the constant size of vertex set U. To test this hypothesis, we repeated this experiment with inversed roles of $\mathbf{r}$ and $\mathbf{c}$ (see Fig. D.7 in Appendix D). In doing so, we found that the quality of $\bar{\tau}_s(\varepsilon)$ decreases even faster in case of the curveball. Thus, we conclude that the constant quality of the ratio $r_s(\varepsilon)$ is a result of the way we defined our instance classes.

We will briefly discuss the implications of our observations. As we observed that the quality of the empirical mixing time $\bar{\tau}_s(\varepsilon)$ depends on the type of Markov chain, we have to assume that the random samples produced after $\bar{\tau}_s(\varepsilon)$ steps of each chain will not be "similarly random".

More precisely, our observations suggest that the distance $||p_s^{(\bar{\tau}_s(\varepsilon))} - \pi||$ between the stationary distribution π and the $\bar{\tau}_s(\varepsilon)$-step probability distribution of the curveball chain is approximately ε, as the ratio $r_s(\varepsilon)$ is near one. In contrast, the total variation distance $||p_s^{(\bar{\tau}_s(\varepsilon))} - \pi||$ exceeds ε for the classical and edge switch chain. Consequently, the $\bar{\tau}_s(\varepsilon)$-step distribution $p_s^{(\bar{\tau}_s(\varepsilon))}$ of the curveball chain may be very different from that of the classical and edge switch chain.

However, as the distance $||q_s^{(\bar{\tau}_s(\varepsilon))} - \eta||$ is less or equal to ε for *each* type of Markov chain, we may still assess the efficiency of the sampling algorithms by comparing their associated empirical mixing time $\bar{\tau}_s(\varepsilon)$. In doing so, we assess the number of steps that are required to produce a random sample from $\mathbb{R}$ according to a distribution $q_s^{(\bar{\tau}_s(\varepsilon))}$ that is close up to ε to the limiting distribution η.

Experiment 4.11 To evaluate the efficiency of the sampling algorithms on real-world instances, we approximated the empirical mixing time of each chain on the set of ecological instances. Similar as before, we approximated the t-step probability distributions $q_s^{(t)}$ for increasing values of t by producing $N = 10^6$ random samples from each Markov chain. However, as the exact sampling methods are infeasible for many of the ecological instances, we approximated the limiting distribution η by an experimental approach similar to that sketched in Sec. 2.3. Fig. 4.14 shows the result of this experiment.

- We observe that the classical switch chain is least efficient for most of the ecological instances as its empirical mixing time is multiple orders of magnitude larger than that of the other chains. In contrast, we observe that the curveball chain is superior to the other chains in most cases.

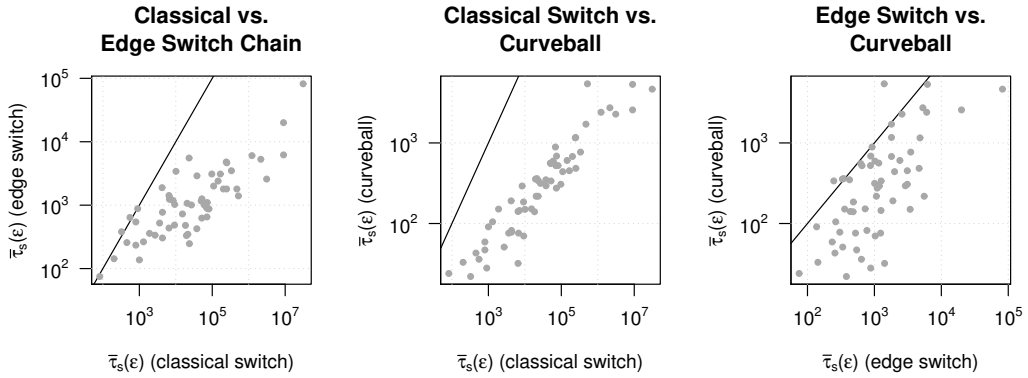

Figure 4.14: Empirical mixing time $\bar{\tau}_s(\varepsilon)$ for the ecological instances with respect to the Hamming distance metric. The black lines separate the areas where one chain is superior to the other. ($\varepsilon = 0.01$, $N = 10^6$.)

- There are a few exceptions to our observations. In some cases, the empirical mixing time of the edge switch chain is similar to that of the classical switch chain. In others, the edge switch chain is similarly efficient as the curveball chain.

Summary In this set of experiments, we approximated the empirical mixing time of each Markov chain. We started our experiments by evaluating several auxiliary functions and found that the Hamming distance metric gives the best approximation to the total mixing time. Subsequently, we calculated the empirical mixing time for members of our scalable instances and found the qualitative results from previous experiments confirmed. By assessing the quality of $\bar{\tau}_s(\varepsilon)$ as a lower bound on the mixing time $\tau_s(\varepsilon)$, we found the quality of the lower bounds depends on the type of Markov chain and instance class. Finally, we processed the real-life instances from the *web-of-life* data set and observed that curveball is most efficient in terms of $\bar{\tau}_s(\varepsilon)$.

4.3.5 Running Time

We close this section by addressing the efficiency of the sampling methods in practice. The previous experiments did not consider the running time of each step. However, this aspect is obviously of crucial importance for the total running time of a sampling algorithm.

Experiment 4.12 By simulating $\tau_s(\varepsilon)$ steps of a Markov chain, we produce a random sample according to a probability distribution $p_s^{(\tau_s(\varepsilon))}(x)$ that is "close" to the uniform distribution up to ε. As each Markov chain needs a different number of steps to reach this distribution, we can fairly compare the efficiency of the sampling

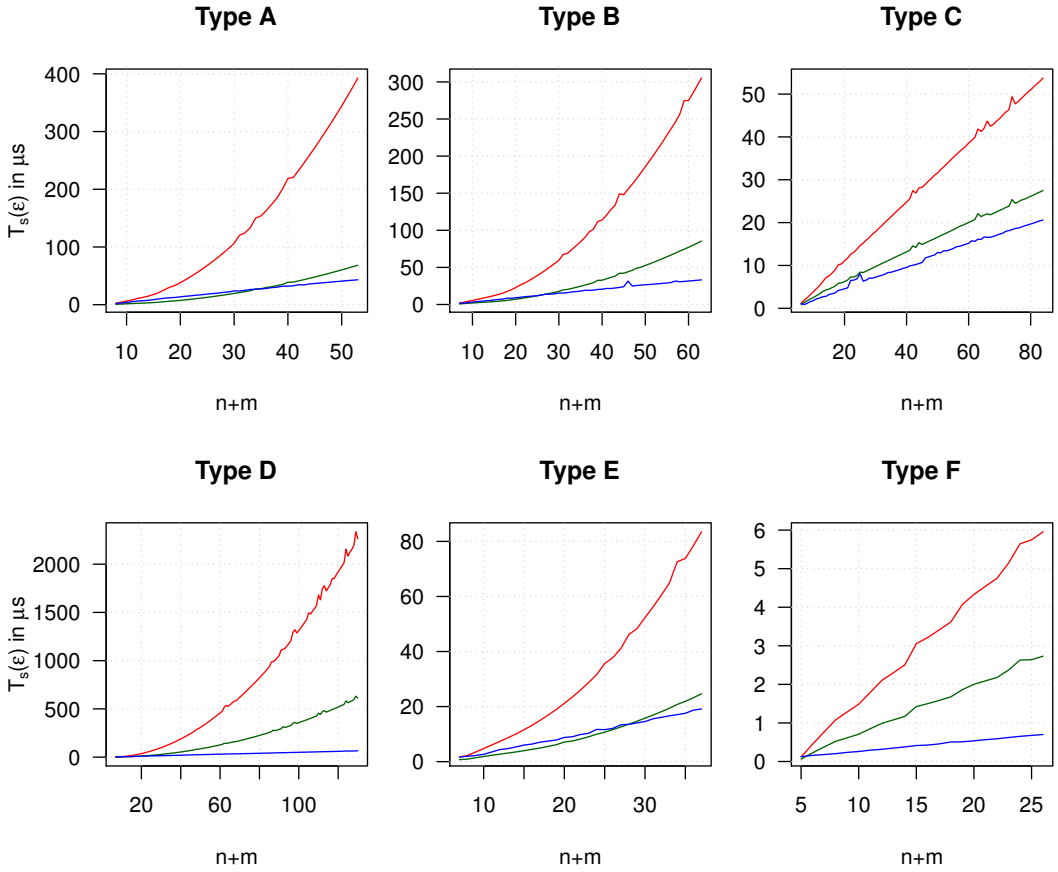

Figure 4.15: Running time $T_s(\varepsilon)$ for $\tau_s(\varepsilon)$ steps of each Markov chain. Red: classical switch chain. Green: edge switch chain. Blue: curveball chain. ($\varepsilon = 0.01$, average running time of 10^6 runs.)

algorithms by measuring the running time $T_s(\varepsilon)$ required for $\tau_s(\varepsilon)$ steps of the associated chain. Fig. 4.15 shows the running time $T_s(\varepsilon)$ of each Markov chain for our scalable instance classes. We discuss some important observations.

- We observe that the classical switch chain is least efficient in terms of running time, similarly as it was in terms of iterations. Even in case of class C, where we found the total mixing time of the classical switch chain to be smaller than that of the edge switch chain, we observe that the classical switch is least efficient. The reason for this observation lies in the definition of the classical switch operation. As each classical switch needs to draw four random numbers (instead of two in case of an edge switch), the running time of a classical switch exceeds that of an edge switch.

- Considering the instance classes A,B,D, and E, the edge switch chain is superior

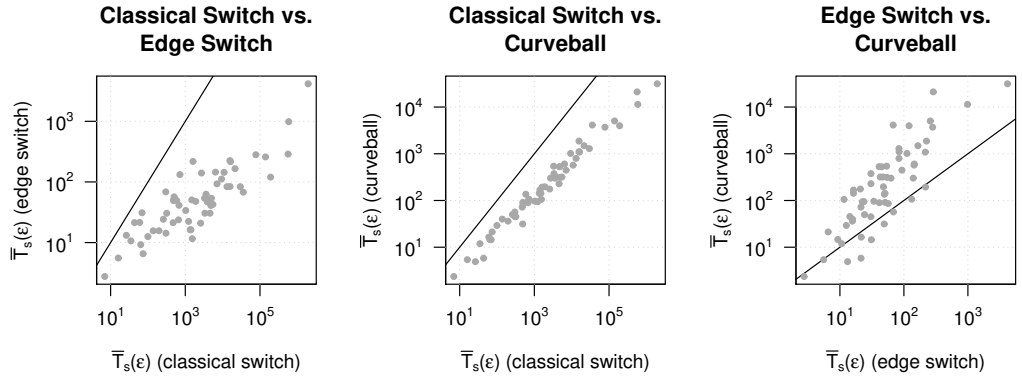

Figure 4.16: Running time $\overline{T}_s(\varepsilon)$ in μs for $\bar{\tau}_s(\varepsilon)$ steps of each Markov chain for the web-of-life instances. The black line separates the areas where one chain is superior to the other. ($\varepsilon = 0.01$, $N = 10^6$, average running time of 10^6 runs.)

to the other chains when the number of nodes is small. However, when n grows, the curveball Markov chains becomes more efficient. In case of classes C and F, we find the curveball chain superior for all instances.

Our experiment suggests that the curveball chain is superior to the other chains if the size of an instance is sufficiently large. However, we should take this with a grain of salt as our instance classes are constructed favorably for the curveball chain. Being specific, the number $|V|$ is designed to grow linearly, while the number $|U|$ is constant. As the ratio $|V|/|U|$ is rarely that extreme in real-world applications, our observations may not hold in practice. Thus, we will next discuss the efficiency of the sampling algorithms for more realistic instances.

Experiment 4.13 In a final experiment, we assessed efficiency of the Markov chains for the instances of the *web-of-life* data set. As the construction of state graphs is infeasible for most ecological instances, we measure the running time $\overline{T}_s(\varepsilon)$ required for $\bar{\tau}_s(\varepsilon)$ steps of each Markov chain. Fig. 4.16 shows the result of this experiment.

- We observe that the classical switch chain is least efficient in terms of running time for all of the ecological instances. This coincides with our previous observations on the scalable instances.
- In contrast to the scalable instances, we observe that the edge switch chain is now superior to the curveball chain for the majority of real-world instances. By quantifying the speedup of the edge switch chain, we observe that it outperforms the curveball chain by a factor of up to 73.

As we observed in the previous experiment that the quality of $\bar{\tau}_s(\varepsilon)$ differs between the Markov chains, our observations do not prove the efficiency of the Markov chains

for the original sampling problems. However, they show that the edge switch chain is suited best for the approximation of expected values via Eq. 1.2.

Summary In this set of experiments, we assessed the efficiency of the Markov chains from a practical point of view. For this purpose, we measured the running time of each chain that is required to produce random samples according to probability distributions that are similarly close to a common limiting distribution. Our observations indicate that the classical switch chain is clearly least efficient, while the curveball chain is superior to the other chains when the vertex set V is large. By considering ecological real-world instances from the *web-of-life* data set, we found that the edge switch chain outperforms the curveball chain in the majority of cases. Our experiments suggest that the edge switch chain is suited best for the uniform sampling of bipartite graphs with fixed vertex degrees, with the curveball chain being superior to the other chains if one vertex set is far larger than the other.

4.4 Preprocessing

Naively applying an MCMC algorithm to produce random samples may result in a poor running time as the mixing time of the underlying Markov chain can be unpleasantly large. However, we can sometimes accelerate the sampling process by a clever preprocessing of the problem instance. For example, considering the "Darwin's Finches" instance shown in Table 4.1, we observe that each entry of the last row must be set to one as the associated row sum is 17 and is thereby equal to the number of columns. Consequently, we can speed-up the sampling process by fixing the entries of the last row to one and omitting them from sampling. In doing so, we replace the original problem of size 13×17 by a sub-problem of size 12×17 that can be solved more efficiently. In the following section, we will generalize this idea and show how to simplify a sampling problem by replacing it by an equivalent set of sub-problems.

In practical applications, obvious preprocessing techniques like the removal of trivial rows or columns are commonly used to accelerate the sampling process [49, 34]. Our method contains these obvious techniques as special cases and augments them by exploiting a more systematic approach.

4.4.1 Methodology

The algorithm described in this section iteratively divides a bi-graphical pair of integer vectors into so-called *partitions*. For the following definition, let $\mathbf{vw}$ describe the *concatenation* of two vectors $\mathbf{v}$ and $\mathbf{w}$.

Definition 4.12 (partition)**.** *Let* $\mathbf{r} = (r_1, r_2, \ldots, r_m)$ *and* $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ *be integer vectors. We say that* $((\bar{\mathbf{r}}, \bar{\mathbf{c}}), (\hat{\mathbf{r}}, \hat{\mathbf{c}}))$ *is a* partition *of* $(\mathbf{r}, \mathbf{c})$ *if and only if*

a) $\mathbf{c} = \bar{\mathbf{c}}\hat{\mathbf{c}}$ *and* $r_i = \bar{r}_i + \hat{r}_i$ *for each* $i \in \{1, 2, \ldots, m\}$*, or*

b) $\mathbf{r} = \bar{\mathbf{r}}\hat{\mathbf{r}}$ *and* $c_i = \bar{c}_i + \hat{c}_i$ *for each* $i \in \{1, 2, \ldots, n\}$.

Example The following sets of integer vectors (and many more) are partitions of the vector pair $\mathbf{r} = (4, 4, 2, 1)$ and $\mathbf{c} = (3, 3, 3, 1, 1)$.

$$\begin{array}{llll} \bar{\mathbf{r}} = (4,4), & \hat{\mathbf{r}} = (2,1), & \bar{\mathbf{c}} = (3,2,1,1,1), & \hat{\mathbf{c}} = (0,1,2,0,0), \\ \bar{\mathbf{r}} = (4,4), & \hat{\mathbf{r}} = (2,1), & \bar{\mathbf{c}} = (2,2,2,1,1), & \hat{\mathbf{c}} = (1,1,1,0,0), \\ \bar{\mathbf{r}} = (2,1,2,0), & \hat{\mathbf{r}} = (2,3,0,1), & \bar{\mathbf{c}} = (3,3,3,1), & \hat{\mathbf{c}} = (1). \end{array}$$

The technique described in this section is based on the partitioning of a bi-graphical vector pair $(\mathbf{r}, \mathbf{c})$ into a pair $((\bar{\mathbf{r}}, \bar{\mathbf{c}}), (\hat{\mathbf{r}}, \hat{\mathbf{c}}))$ of vector pairs such that $\bar{\mathbf{r}}$, $\hat{\mathbf{r}}$, $\bar{\mathbf{c}}$, and $\hat{\mathbf{c}}$ are non-increasing, and $(\bar{\mathbf{r}}, \bar{\mathbf{c}})$ and $(\hat{\mathbf{r}}, \hat{\mathbf{c}})$ are again bi-graphical. The following lemma shows under which conditions we may find such a partition.

Lemma 4.2. *Let* $\mathbf{r} = (r_1, r_2, \ldots, r_m)$ *be a non-increasing integer vector and let* $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ *be an integer vector such that* $(\mathbf{r}, \mathbf{c})$ *is bi-graphical. Let* $k < m$ *be an arbitrary positive integer such that* $\Sigma^k_{\mathbf{r}} = \Sigma^k_{\mathbf{c}'}$ *holds. Let*

$$\begin{array}{ll} \bar{\mathbf{r}} = (\bar{r}_1, \bar{r}_2, \ldots, \bar{r}_k), & \hat{\mathbf{r}} = (\hat{r}_1, \hat{r}_2, \ldots, \hat{r}_{m-k}) \\ \bar{\mathbf{x}} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k), & \hat{\mathbf{x}} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_{m-k}) \end{array}$$

be integer vectors defined by

$$\begin{array}{llll} \bar{r}_i := r_i & \text{and} & \bar{x}_i := c'_i & \text{for } 1 \le i \le k, \\ \hat{r}_i := r_{i+k} & \text{and} & \hat{x}_i := c'_{i+k} & \text{for } 1 \le i \le m-k. \end{array}$$

Finally, let $\bar{\mathbf{c}} = (\bar{c}_1, \bar{c}_2, \ldots, \bar{c}_n)$ *and* $\hat{\mathbf{c}} = (\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_n)$ *be integer vectors defined by*

$$\bar{c}_i := \bar{x}'_i \quad \text{and} \quad \hat{c}_i := \hat{x}'_i \quad \text{for } 1 \le i \le n.$$

Then, $(\bar{\mathbf{r}}, \bar{\mathbf{c}})$ *and* $(\hat{\mathbf{r}}, \hat{\mathbf{c}})$ *are bi-graphical, and* $((\bar{\mathbf{r}}, \bar{\mathbf{c}}), (\hat{\mathbf{r}}, \hat{\mathbf{c}}))$ *is a partition of* $(\mathbf{r}, \mathbf{c})$.

Example To apply Lemma 4.2 to the integer vectors $\mathbf{r} = (4, 4, 2, 1)$ and $\mathbf{c} = (3, 3, 3, 1, 1)$, we set-up the associated conjugate sequences and partial sums.

$$\begin{array}{rr} \mathbf{r} = (4,4,2,1) & \mathbf{c} = (3,3,3,1,1) \\ \Sigma_{\mathbf{r}} = (4,8,10,11) & \Sigma_{\mathbf{c}} = (3,6,9,10,11) \\ \mathbf{r}' = (4,3,2,2,0,\ldots) & \mathbf{c}' = (5,3,3,0,\ldots) \\ \Sigma_{\mathbf{r}'} = (4,7,9,11,11,\ldots) & \Sigma_{\mathbf{c}'} = (5,8,11,11,\ldots). \end{array}$$

We observe that $\Sigma^k_{\mathbf{r}} = \Sigma^k_{\mathbf{c}'}$ holds for $k = 2$. Thus, we define

$$\begin{array}{ll} \bar{\mathbf{r}} = (4,4) & \hat{\mathbf{r}} = (2,1) \\ \bar{\mathbf{x}} = (5,3) & \hat{\mathbf{x}} = (3,0) \\ \bar{\mathbf{c}} = (2,2,2,1,1) & \hat{\mathbf{c}} = (1,1,1,0,0). \end{array}$$

We can easily verify that $((\bar{\mathbf{r}}, \bar{\mathbf{c}}), (\hat{\mathbf{r}}, \hat{\mathbf{c}}))$ is a partition of $(\mathbf{r}, \mathbf{c})$. In addition, Theorem 4.1 shows that $(\bar{\mathbf{r}}, \bar{\mathbf{c}})$ and $(\hat{\mathbf{r}}, \hat{\mathbf{c}})$ are bi-graphical.

$$\begin{array}{llll} \bar{\mathbf{r}} = (4,4) & \bar{\mathbf{c}} = (2,2,2,1,1) & \hat{\mathbf{r}} = (2,1) & \hat{\mathbf{c}} = (1,1,1,0,0) \\ \Sigma_{\bar{\mathbf{r}}} = (4,8) & \bar{\mathbf{c}}' = (5,3,0,\ldots) & \Sigma_{\hat{\mathbf{r}}} = (2,3) & \hat{\mathbf{c}}' = (3,0,\ldots) \\ & \Sigma_{\bar{\mathbf{c}}'} = (5,8,8,\ldots) & & \Sigma_{\bar{\mathbf{c}}'} = (3,3,\ldots). \end{array}$$

Proof of Lemma 4.2. We start by showing that $((\bar{\mathbf{r}}, \bar{\mathbf{c}}), (\hat{\mathbf{r}}, \hat{\mathbf{c}}))$ is a partition of $(\mathbf{r}, \mathbf{c})$. As $\mathbf{r} = \bar{\mathbf{r}}\hat{\mathbf{r}}$ holds by construction, it suffices to show $c_i = \bar{c}_i + \hat{c}_i$ for each $i \in \{1, 2, \ldots, n\}$. As

$$\bar{c}_i = \bar{x}'_i = |\{j \in \{1, \ldots, k\} \colon \bar{x}_j \geq i\}| = \left|\{j \in \{1, \ldots, k\} \colon c'_j \geq i\}\right|$$

and

$$\begin{aligned} \hat{c}_i = \hat{x}'_i &= |\{j \in \{1, \ldots, m-k\} \colon \hat{x}_j \geq i\}| \\ &= \left|\{j \in \{1, \ldots, m-k\} \colon c'_{j+k} \geq i\}\right| \\ &= \left|\{j \in \{k+1, \ldots, m\} \colon c'_j \geq i\}\right|, \end{aligned}$$

the sum $\bar{c}_i + \hat{c}_i$ can be written as

$$\begin{aligned} \bar{c}_i + \hat{c}_i &= \left|\{j \in \{1, \ldots, k\} \colon c'_j \geq i\}\right| + \left|\{j \in \{k+1, \ldots, m\} \colon c'_j \geq i\}\right| \\ &= \left|\{j \in \{1, \ldots, m\} \colon c'_j \geq i\}\right| = c''_i = c_i. \end{aligned}$$

Thus, $((\bar{\mathbf{r}}, \bar{\mathbf{c}}), (\hat{\mathbf{r}}, \hat{\mathbf{c}}))$ is a partition of $(\mathbf{r}, \mathbf{c})$. It remains to show that $(\bar{\mathbf{r}}, \bar{\mathbf{c}})$ and $(\hat{\mathbf{r}}, \hat{\mathbf{c}})$ are bi-graphical. We start by observing that $\bar{\mathbf{r}}$ and $\hat{\mathbf{r}}$ are non-increasing by construction. We first show that $(\bar{\mathbf{r}}, \bar{\mathbf{c}})$ is bi-graphical. From $\Sigma^k_{\mathbf{r}} = \Sigma^k_{\mathbf{c}'}$ and $\bar{x}_i = \bar{x}''_i$ we conclude

$$\sum_{i=1}^{k} \bar{r}_i = \sum_{i=1}^{k} r_i = \sum_{i=1}^{k} c'_i = \sum_{i=1}^{k} \bar{x}_i = \sum_{i=1}^{k} \bar{x}''_i = \sum_{i=1}^{k} \bar{c}'_i$$

and thus $\Sigma^k_{\bar{\mathbf{r}}} = \Sigma^k_{\bar{\mathbf{c}}'}$. In addition, as $(\mathbf{r}, \mathbf{c})$ is bi-graphical, $\Sigma^j_{\mathbf{r}} \leq \Sigma^j_{\mathbf{c}'}$ holds for each j in range $1 \leq j \leq m$. Hence,

$$\sum_{i=1}^{j} \bar{r}_i = \sum_{i=1}^{j} r_i \leq \sum_{i=1}^{j} c'_i = \sum_{i=1}^{j} \bar{x}_i = \sum_{i=1}^{j} \bar{x}''_i = \sum_{i=1}^{j} \bar{c}'_i$$

holds for each j in range $1 \leq j \leq k$. Consequently, $\bar{\mathbf{r}} \trianglelefteq \bar{\mathbf{c}}'$ holds, too. By Theorem 4.1, $(\bar{\mathbf{r}}, \bar{\mathbf{c}})$ is bi-graphical. Similarly, we show that $(\hat{\mathbf{r}}, \hat{\mathbf{c}})$ is bi-graphical. As $\Sigma^m_{\mathbf{r}} = \Sigma^m_{\mathbf{c}'}$ and $\Sigma^k_{\mathbf{r}} = \Sigma^k_{\mathbf{c}'}$, we conclude

$$\begin{aligned} \sum_{i=1}^{m-k} \hat{r}_i = \sum_{i=1}^{m-k} r_{i+k} &= \sum_{i=1}^{m} r_i - \sum_{i=1}^{k} r_i = \sum_{i=1}^{m} c'_i - \sum_{i=1}^{k} c'_i \\ &= \sum_{i=1}^{m-k} c'_{i+k} = \sum_{i=1}^{m-k} \hat{x}_i = \sum_{i=1}^{m-k} \hat{x}''_i = \sum_{i=1}^{m-k} \hat{c}'_i. \end{aligned}$$

In addition, as $\Sigma_{\mathbf{r}}^{j} \leq \Sigma_{\mathbf{c}'}^{i}$ for $1 \leq j \leq m$, it follows that

$$\begin{aligned}\sum_{i=1}^{j} \hat{r}_i &= \sum_{i=1}^{j} r_{i+k} = \sum_{i=k+1}^{k+j} r_i = \sum_{i=1}^{k+j} r_i - \sum_{i=1}^{k} r_i = \sum_{i=1}^{k+j} r_i - \sum_{i=1}^{k} c'_i \\ &\leq \sum_{i=1}^{k+j} c'_i - \sum_{i=1}^{k} c'_i = \sum_{i=k+1}^{k+j} c'_i = \sum_{i=1}^{j} c'_{i+k} = \sum_{i=1}^{j} \hat{x}_i = \sum_{i=1}^{j} \hat{x}''_i = \sum_{i=1}^{j} \hat{c}'_i.\end{aligned}$$

holds for each $1 \leq j \leq m-k$. Thus, $(\hat{\mathbf{r}}, \hat{\mathbf{c}})$ is bi-graphical by Theorem 4.1. □

From Lemma 4.2, we derive a simple algorithm (see Alg. 4.2) that can be used for the partitioning of a vector pair $(\mathbf{r}, \mathbf{c})$.

Algorithm 4.2: PARTITION

Input: bi-graphical vector pair $(\mathbf{r}, \mathbf{c})$ with $\mathbf{r} \in \mathbb{N}^m$ and $\mathbf{c} \in \mathbb{N}^n$ such that $\mathbf{r}$ is non-inreasing, integer k with $\Sigma_{\mathbf{r}}^{k} = \Sigma_{\mathbf{c}'}^{k}$

Output: non-increasing integer vectors $\bar{\mathbf{r}}$, $\hat{\mathbf{r}}$, $\bar{\mathbf{c}}$, and $\hat{\mathbf{c}}$ such that $((\bar{\mathbf{r}}, \bar{\mathbf{c}}), (\hat{\mathbf{r}}, \hat{\mathbf{c}}))$ is a partition of $(\mathbf{r}, \mathbf{c})$, and $(\bar{\mathbf{r}}, \bar{\mathbf{c}})$ and $(\hat{\mathbf{r}}, \hat{\mathbf{c}})$ are bi-graphical

1 $(\bar{\mathbf{r}}, \hat{\mathbf{r}}) \leftarrow ((r_1, r_2, \ldots, r_k), (r_{k+1}, r_{k+2}, \ldots, r_m))$
2 $(\bar{\mathbf{x}}, \hat{\mathbf{x}}) \leftarrow ((c'_1, c'_2, \ldots, c'_k), (c'_{k+1}, c'_{k+2}, \ldots, c'_m))$
3 $\bar{\mathbf{c}} \leftarrow (\bar{x}'_1, \bar{x}'_2, \ldots, \bar{x}'_n)$ `// (r̄, c̄) is bi-graphical`
4 $\hat{\mathbf{c}} \leftarrow (\hat{x}'_1, \hat{x}'_2, \ldots, \hat{x}'_n)$ `// (r̂, ĉ) is bi-graphical`
5 **return** $((\bar{\mathbf{r}}, \bar{\mathbf{c}}), (\hat{\mathbf{r}}, \hat{\mathbf{c}}))$

Observation If $\mathbf{r}$ and $\mathbf{c}$ are the row and column sums of an $m \times n$ binary matrix $M = (m_{ij})$, the partitioning of $(\mathbf{r}, \mathbf{c})$ via Lemma 4.2 divides M into two disjoint sub-matrices $\bar{M} = (\bar{m}_{ij})$ and $\hat{M} = (\hat{m}_{ij})$ of size $k \times n$ and $(m-k) \times n$ (see Fig. 4.17):

$$\begin{aligned}\bar{m}_{ij} &:= m_{ij}, && \text{for } 1 \leq i \leq k \text{ and } 1 \leq j \leq n \\ \hat{m}_{ij} &:= m_{(i+k)j}, && \text{for } 1 \leq i \leq m-k \text{ and } 1 \leq j \leq n.\end{aligned}$$

By construction, the row and column sums of $\bar{M}$ and $\hat{M}$ are identical to $(\bar{\mathbf{r}}, \bar{\mathbf{c}})$ and $(\hat{\mathbf{r}}, \hat{\mathbf{c}})$. As $(\bar{\mathbf{r}}, \bar{\mathbf{c}})$ and $(\hat{\mathbf{r}}, \hat{\mathbf{c}})$ are bi-graphical by Lemma 4.2, both sub-matrices can be processed independently from each other.

For simplicity, Lemma 4.2 has been defined asymmetrically with respect to the vectors $\mathbf{r}$ and $\mathbf{c}$. The following theorem corrects this flaw and generalizes the concept of partitioning to a set of so-called *composite* vector pairs.

Definition 4.13 (primitive, composite)**.** *Let* $\mathbf{r} = (r_1, r_2, \ldots, r_m)$ *and* $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ *be non-increasing integer vectors such that* $(\mathbf{r}, \mathbf{c})$ *is bi-graphical. The vector pair* $(\mathbf{r}, \mathbf{c})$ *is called* primitive *if and only if* $\Sigma_{\mathbf{c}'}^{k_1} > \Sigma_{\mathbf{r}}^{k_1}$ *and* $\Sigma_{\mathbf{r}'}^{k_2} > \Sigma_{\mathbf{c}}^{k_2}$ *hold for each* $k_1 \in \{1, \ldots, m-1\}$ *and* $k_2 \in \{1, \ldots, n-1\}$*. Otherwise, it is called* composite*.*

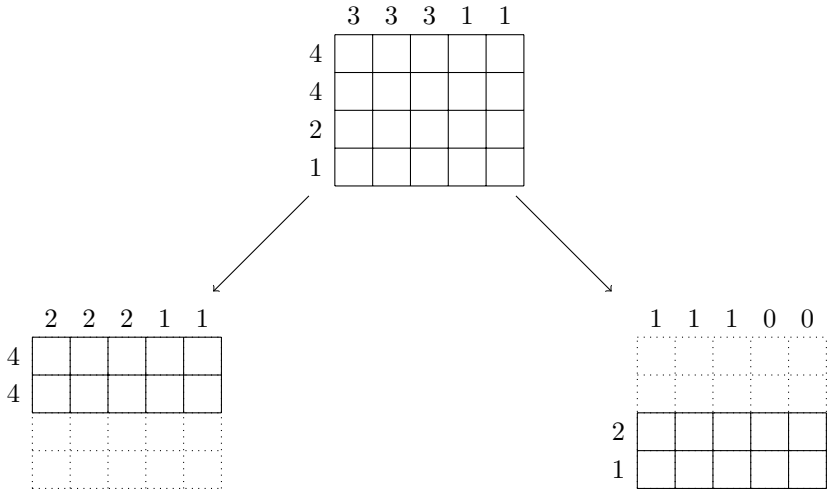

Figure 4.17: Partitioning of $\mathbf{r} = (4, 4, 2, 1)$ and $\mathbf{c} = (3, 3, 3, 1, 1)$ into $\bar{\mathbf{r}} = (4, 4)$, $\hat{\mathbf{r}} = (2, 1)$, $\bar{\mathbf{c}} = (2, 2, 2, 1, 1)$, and $\hat{\mathbf{c}} = (1, 1, 1, 0, 0)$.

Theorem 4.3. *Let* $\mathbf{r} = (r_1, r_2, \ldots, r_m)$ *and* $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ *be non-increasing integer vectors such that* $(\mathbf{r}, \mathbf{c})$ *is bi-graphical and composite. Then, we can partition* $(\mathbf{r}, \mathbf{c})$ *into* $((\bar{\mathbf{r}}, \bar{\mathbf{c}}), (\hat{\mathbf{r}}, \hat{\mathbf{c}}))$ *such that* $(\bar{\mathbf{r}}, \bar{\mathbf{c}})$ *and* $(\hat{\mathbf{r}}, \hat{\mathbf{c}})$ *are bi-graphical.*

Proof. As $(\mathbf{r}, \mathbf{c})$ is composite, at least one of two cases must occur:

a) $\exists k_1 \in \{1, 2, \ldots, m-1\}\colon \Sigma_{\mathbf{r}}^{k_1} = \Sigma_{\mathbf{c}'}^{k_1}$, or

b) $\exists k_2 \in \{1, 2, \ldots, n-1\}\colon \Sigma_{\mathbf{c}}^{k_2} = \Sigma_{\mathbf{r}'}^{k_2}$.

In the first case, we can directly apply Lemma 4.2 to create a partition $((\bar{\mathbf{r}}, \bar{\mathbf{c}}), (\hat{\mathbf{r}}, \hat{\mathbf{c}}))$ such that $(\bar{\mathbf{r}}, \bar{\mathbf{c}})$ and $(\hat{\mathbf{r}}, \hat{\mathbf{c}})$ are bi-graphical. In the latter case, we may switch the roles of $\mathbf{r}$ and $\mathbf{c}$ to create the bi-graphical vector pair $(\mathbf{c}, \mathbf{r})$. Now we can apply Lemma 4.2 to create the partition $((\bar{\mathbf{c}}, \bar{\mathbf{r}}), (\hat{\mathbf{c}}, \hat{\mathbf{r}}))$ such that $(\bar{\mathbf{c}}, \bar{\mathbf{r}})$ and $(\hat{\mathbf{c}}, \hat{\mathbf{r}})$ are bi-graphical. After switching back to the original roles, $((\bar{\mathbf{r}}, \bar{\mathbf{c}}), (\hat{\mathbf{r}}, \hat{\mathbf{c}}))$ is a partition such that $(\bar{\mathbf{r}}, \bar{\mathbf{c}})$ and $(\hat{\mathbf{r}}, \hat{\mathbf{c}})$ are bi-graphical. □

Remark If $(\mathbf{r}, \mathbf{c})$ is bi-graphical, it is possible that there are two integers $k_1 \in \{1, 2, \ldots, m-1\}$ and $k_2 \in \{1, 2, \ldots, n-1\}$ such that both $\Sigma_{\mathbf{r}}^{k_1} = \Sigma_{\mathbf{c}'}^{k_1}$ and $\Sigma_{\mathbf{c}}^{k_2} = \Sigma_{\mathbf{r}'}^{k_2}$ hold, as it is the case in the example above for $k_1 = 2$ and $k_2 = 3$. In such cases, we may apply Lemma 4.2 to either $(\mathbf{r}, \mathbf{c})$ or $(\mathbf{c}, \mathbf{r})$. Each way will produce a different set of bi-graphical vector pairs. In the example above, we may alternatively create the integer vectors $(\bar{\mathbf{r}}, \bar{\mathbf{c}})$ and $(\hat{\mathbf{r}}, \hat{\mathbf{c}})$ defined by

$$\bar{\mathbf{r}} = (3, 3, 2, 1) \qquad \hat{\mathbf{r}} = (1, 1, 0, 0) \qquad \bar{\mathbf{c}} = (3, 3, 3) \qquad \hat{\mathbf{c}} = (1, 1).$$

While in the previous case the associated binary matrix is partitioned vertically, the alternative choice results in a horizontal matrix partition.

4.4.2 Decomposition Algorithm

By iterated application of Theorem 4.3, we can decompose a bi-graphical pair of integer vectors into a set of primitive vector pairs. In this process, the associated binary matrix is partitioned into disjoint sub-matrices that correspond to independent instances of the sampling problem. The following algorithm (see Alg. 4.3) summarizes the ideas of this section and shows how a bi-graphical pair $(\mathbf{r}, \mathbf{c})$ can be decomposed into its primitive components. The algorithm assumes that $\mathbf{r}$ and $\mathbf{c}$ are already ordered non-increasingly.

Algorithm 4.3: DECOMPOSE

Input: bi-graphical pair $(\mathbf{r}, \mathbf{c})$ with $\mathbf{r} \in \mathbb{N}^m$ and $\mathbf{c} \in \mathbb{N}^n$
Output: decomposition of $(\mathbf{r}, \mathbf{c})$ into a set of primitive bi-graphical pairs

$k_1 \leftarrow \min\{i \in \{1, 2, \ldots, m\} \colon \Sigma^i_{\mathbf{r}} = \Sigma^i_{\mathbf{c}'}\}$
$k_2 \leftarrow \min\{j \in \{1, 2, \ldots, n\} \colon \Sigma^j_{\mathbf{c}} = \Sigma^j_{\mathbf{r}'}\}$
if $k_1 = m$ **and** $k_2 = n$ **then** // $(\mathbf{r}, \mathbf{c})$ is primitive
  return $\{(\mathbf{r}, \mathbf{c})\}$
else // $(\mathbf{r}, \mathbf{c})$ is composite
  if $k_1 < m$ **then** // $\Sigma^{k_1}_{\mathbf{r}} = \Sigma^{k_1}_{\mathbf{c}'}$
    $((\bar{\mathbf{r}}, \bar{\mathbf{c}}), (\hat{\mathbf{r}}, \hat{\mathbf{c}})) \leftarrow$ PARTITION$((\mathbf{r}, \mathbf{c}), k_1)$
  else // $\Sigma^{k_2}_{\mathbf{c}} = \Sigma^{k_2}_{\mathbf{r}'}$
    $((\bar{\mathbf{c}}, \bar{\mathbf{r}}), (\hat{\mathbf{c}}, \hat{\mathbf{r}})) \leftarrow$ PARTITION$((\mathbf{c}, \mathbf{r}), k_2)$
  end
  return DECOMPOSE$(\bar{\mathbf{r}}, \bar{\mathbf{c}}) \cup$ DECOMPOSE$(\hat{\mathbf{r}}, \hat{\mathbf{c}})$
end

Example Alg. 4.3 decomposes our exemplary instance $(\mathbf{r}, \mathbf{c})$ with $\mathbf{r} = (4, 4, 2, 1)$ and $\mathbf{c} = (3, 3, 3, 1, 1)$ into the following multi-set of twelve primitives vector pairs that can be divided into four groups:

$$
\begin{aligned}
4 \times \quad & \mathbf{r} = (0),\ \mathbf{c} = (0) & \qquad 1 \times \quad & \mathbf{r} = (1, 1),\ \mathbf{c} = (1, 1) \\
6 \times \quad & \mathbf{r} = (1),\ \mathbf{c} = (1) & \qquad 1 \times \quad & \mathbf{r} = (2, 1),\ \mathbf{c} = (1, 1, 1)
\end{aligned}
$$

In this process, the associated binary matrix is divided into twelve disjoint sub-matrices (see Fig. 4.18). From the example, we may derive several interesting observations:

- By decomposing the vector pair into primitives, we replace the original sampling problem by a set of twelve independent sub-problems.
- Ten out of twelve sub-problems are trivial as they possess just one realization. Consequently, we may fix several entries of the associated matrix to zero or one and exclude them from sampling.

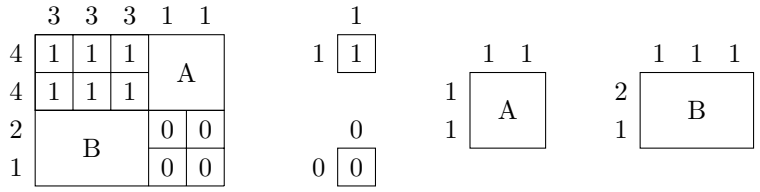

Figure 4.18: Decomposition of $\mathbf{r} = (4, 4, 2, 1)$ and $\mathbf{c} = (3, 3, 3, 1, 1)$.

- The remaining two non-trivial sub-problems are significantly smaller than the original one. We could easily enumerate all realizations of these tiny instances and select one randomly to produce exactly uniformly distributed samples of each sub-problem. As the sub-problems can be processed independently, we can combine the samples of the sub-problems to produce a uniformly distributed sample of the original problem.

In summary, this exemplary sampling problem could be significantly simplified.

4.4.3 Experimental Evaluation

To examine how useful this technique is for ecological real-world instances, we decomposed the 62 instances of the *web-of-life* data set (see Table B.2 in Appendix B) into their primitive components. We obtained the following observations:

- We found that most instances (52 of 62) are primitive and thus, cannot further be decomposed.
- In two of the remaining ten cases, the associated sampling problem could be replaced by a set of trivial sub-problems. As a consequence, the whole sampling problem is trivial in these two cases as there is just a single realization of the original problem.
- Each of the remaining eight instances could be replaced by a set of sub-problems containing exactly one non-trivial sub-problem. After omitting the trivial sub-problems, the remaining problem can be solved more efficiently due to its reduced instance size.

  For example, we could replace the instance `A_PH_007` of size 5×64 by a non-trivial sub-problem of size 4×19. Hence, almost 75% of the original matrix entries can be excluded from sampling. Consequently, the sampling methods perform more efficiently (see Fig. 4.19). As the decomposition can be carried out fast, the sampling process can be largely accelerated.

Summary Instance decomposition is a technique that can be used to pre-process integer vectors before starting the sampling algorithm. It is designed to detect

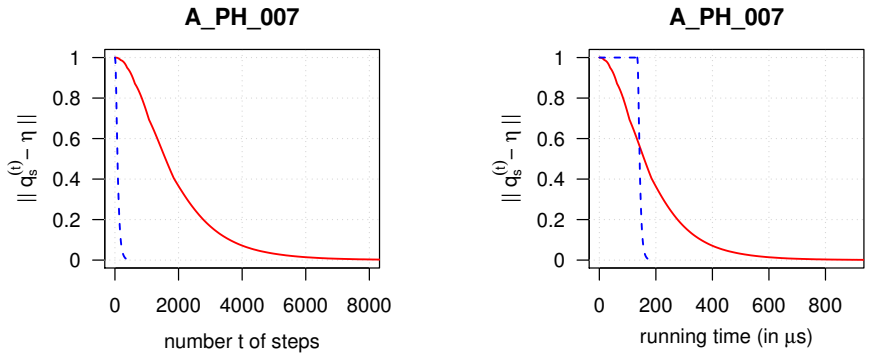

Figure 4.19: Distance to limiting distribution of the Hamming distance metric on the `A_PH_007` instance. Red line: classical switch chain applied to the original instance. Blue line: classical switch chain applied to the preprocessed instance. For this example, the empirical mixing time $\bar{\tau}_s(\varepsilon)$ of the classical switch chain decreased from 6460 to 325. The decomposition required 134 microseconds, after which the actual sampling process started. ($N = 10^6$, $\varepsilon = 0.01$.)

matrix entries that are fixed to either zero or one in *each* realization of the given integer vector. Such entries may be excluded from sampling. Hence, the original sampling problem may be replaced by a set of sub-problems that can be solved independently of each other in parallel.

In an experimental evaluation with 62 real-world instances from ecology, we found that we can simplify the sampling problem in ten out of 62 cases. In one case, instance decomposition could reduce the number of matrix entries by almost 75%. As the running time of the instance decomposition is small, we conclude that it is worth applying in practice.

4.5 Summary

In this chapter, we assessed the efficiency of three well-known sampling algorithms for the uniform sampling of bipartite graphs with fixed vertex degrees.

After summarizing established knowledge about the classical switch, the edge switch, and the curveball Markov chain, we experimentally analysed properties of the associated state graphs on various input instances. In doing so, we found that the curveball algorithm mixes significantly faster than both switch chain variants at the cost of a larger running time in each step. By experimentally determining the Markov chain's total mixing time on scalable instance classes, we established lower bounds on the total mixing time of the associated Markov chains. In particular, we showed that for several instance classes, the necessary number of steps is for both switch chain variants a quadratic function on the number $m + n$ of nodes.

In the following set of experiments, we further studied properties of the Markov

chains by assessing the influence of the average loop probability on the total mixing time. For this purpose, we artificially scaled the average loop probability of the Markov chain's state graphs to a common value, thereby eliminating the influence of the loop probabilities. In doing so, we found that the small loop probability of the curveball chain is an important but not exclusive reason for its efficiency.

By calculating lower and upper bounds on the total mixing time, we assessed how sharp such bounds may be if full information about state graphs was available. While we found that the quality of the upper congestion bound decreases fast, the lower and upper spectral bound remain tight for much longer. Our experiments suggest that the canonical path method is unlikely to gain sharp upper bounds on the total mixing time of a Markov chain.

As the construction of state graphs is infeasible if the number of states is large, we cannot calculate the total mixing time or its lower and upper spectral bounds in most practical sampling applications. To still derive some lower bound on the total mixing time, we showed that the empirical mixing time may be approximated efficiently even if the construction of state graphs is infeasible.

By assessing the running time spent in each step of the random walk on a set of ecological instances, we found that the edge switch chain is most efficient in the majority of cases. In contrast, the classical switch chain is least efficient.

We closed this chapter by showing how a bi-graphical pair of integer vectors can be preprocessed to accelerate the sampling process. For this purpose, we described an algorithm that decomposes a vector pair into its primitive components, which can subsequently be processed independently. Experimentally, we showed that our preprocessing method can largely accelerate the sampling process.

Chapter 5

Bipartite Graphs with Bounded Degrees

In this chapter, we further pursue our study of methods for the randomization of bipartite graphs. While in the previous chapter, we considered the degrees of a bipartite graph to be fixed, we now allow them to vary in prescribed intervals. This variant of the classical sampling problem is motivated by the work with ecological data.

In sciences like ecology, empirical networks are typically constructed by the observation and classification of species in the field. In this process, a group of ecologists patiently monitor a certain geographical spot and record the interaction of species. Due to limited resources however, ecologists can only spend a restricted amount on time to the data collection. Thus, despite all effort and carefulness, the collected data are typically imperfect [71]. Consequently, in present-absence tables like the one of "Darwin's finches" (see Table 4.1), rarely occurring species may falsely be declared absent in certain regions. On the other hand, potential misclassification of species may introduce interactions that are not present in reality. As a consequence, an observed ecological network will often differ at least slightly from the unknown "true" network.

One way to deal with such problems is to incorporate the imperfectness of the data in the network analysis [72, 73]. For example, by assuming that some edges of the true network remained unobserved, we can model the true network to be an unknown element of a set of graphs

a) with the same node set as the observed graph,

b) whose edge set includes the observed edges, and

c) whose degrees may exceed those of the observed graph by predefined constants.

Then, the expected structure of the true network can be approximated by drawing uniformly distributed samples from this set. Based on our confidence in the data,

we may chose the gap between the lower and upper bounds on the degrees smaller or larger. The application of such a procedure requires a method for the uniform sampling of bipartite graphs whose degrees lie in prescribed intervals. In this chapter, we focus on such sampling algorithms.

This chapter is based on joint work with Linda Strowick and Matthias Müller-Hannemann. Preliminary results presented in Ref. [74] were largely revised and enhanced. A research article based on this work was recently published in the Journal of Complex Networks.

> S. Rechner, L. Strowick, and M. Müller-Hannemann. *Uniform sampling of bipartite graphs with degrees in prescribed intervals.* Journal of Complex Networks (2017). DOI: `10.1093/comnet/cnx059` [26]

5.1 Definitions and Notation

As the sampling problem considered in this chapter is a variant of the previously discussed problem, we adopt and extend the notation of Chapter 4. For this purpose, let m and n be positive integers and let

$$\tilde{\mathbf{r}} = (\tilde{r}_1, \tilde{r}_2, \ldots, \tilde{r}_m) \qquad \tilde{\mathbf{c}} = (\tilde{c}_1, \tilde{c}_2, \ldots, \tilde{c}_n)$$
$$\bar{\mathbf{r}} = (\bar{r}_1, \bar{r}_2, \ldots, \bar{r}_m) \qquad \bar{\mathbf{c}} = (\bar{c}_1, \bar{c}_2, \ldots, \bar{c}_n)$$

be integer vectors of length m and n. We denote by $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$ the set of bipartite graphs whose degrees are bounded from below by $\tilde{\mathbf{r}}$ and $\tilde{\mathbf{c}}$, and which are bounded from above by $\bar{\mathbf{r}}$ and $\bar{\mathbf{c}}$, i.e.

$$\begin{aligned}\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}}) := \{G = (U, V, E) \colon\ & |U| = m \ \wedge\ |V| = n \\ & \wedge\ \forall i \in \{1, 2, \ldots, m\} \colon \tilde{r}_i \leq \delta_G(u_i) \leq \bar{r}_i \\ & \wedge\ \forall j \in \{1, 2, \ldots, n\} \colon \tilde{c}_j \leq \delta_G(v_j) \leq \bar{c}_j\}.\end{aligned}$$

The problem discussed in this chapter is to draw samples from $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$ uniformly at random. Exploiting the connection between bipartite graphs and binary matrices, the problem can equivalently be reformulated as the uniform sampling of an $m \times n$ binary matrix with $\tilde{\mathbf{r}}$ and $\bar{\mathbf{r}}$ being lower and upper bounds on its row sums, and $\tilde{\mathbf{c}}$ and $\bar{\mathbf{c}}$ being lower and upper bounds on its column sums.

Definition 5.1 (realizable, realization)**.** *We say that a four-tuple* $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$ *of integer vectors is* realizable *if and only if* $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}}) \neq \emptyset$*. If* $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$ *is realizable, we call any bipartite graph* $G \in \Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$ *a* realization*.*

In many cases we do not need to distinguish between vertices from U and V. For that reason, we define $\tilde{d} \colon U \cup V \to \mathbb{Z}$ and $\bar{d} \colon U \cup V \to \mathbb{Z}$ as functions of lower and upper bounds, i.e.

$$\tilde{d}(u_i) := \tilde{r}_i \quad \text{and} \quad \bar{d}(u_i) := \bar{r}_i \quad \text{for each } i \in \{1, 2, \ldots, m\},$$
$$\tilde{d}(v_j) := \tilde{c}_j \quad \text{and} \quad \bar{d}(v_j) := \bar{c}_j \quad \text{for each } j \in \{1, 2, \ldots, n\}.$$

When $G = (U, V, E)$ and $G' = (U, V, E')$ are bipartite graphs from $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$, we denote by $G \triangle G'$ the *symmetric difference* of the edge sets E and E', i.e.

$$G \triangle G' := (E \setminus E') \cup (E' \setminus E).$$

A sequence of nodes $p = (v_1, v_2, \ldots, v_k)$ is called *alternating path* (in $G \triangle G'$) if each unordered pair $\{v_i, v_{i+1}\}$ of consecutive vertices in p is an edge alternatingly included in $E \setminus E'$ and $E' \setminus E$. An alternating path p is called *non-extendable* if there is no alternating path in $G \triangle G'$ that contains p as a proper subpath. An alternating path is called *simple* if $v_i \neq v_j$ for $1 \leq i \neq j \leq k$.

A closed sequence of nodes $c = (v_1, v_2, \ldots, v_k, v_1)$ is called *alternating cycle* (in $G \triangle G'$) if $(v_1, v_2, \ldots, v_k)$ is an alternating path. An alternating cycle is called *simple* if it does not contain a smaller alternating cycle.

Related Work In contrast to the uniform sampling of bipartite graphs with fixed degrees, this variant of the classical sampling problem did not attract much attention in the past. We will still discuss some related literature.

In ecological null-network analysis, several methods have been discussed to create random bipartite graphs whose degrees are allowed to vary from those of an observed graph. As one of the simplest algorithms, the *EE-algorithm* [75] randomly distributes edges *equiprobably* between each pair of vertices. Thereby, it produces bipartite graphs that possess the same edge total as the observed network but whose vertex degrees may vary entirely at random.

To approximately preserve the degrees of the observed network, a family of so-called *PP-algorithms* can be used to produce bipartite graphs, in which the probability of an edge is *proportional* to the corresponding vertex degrees in the observed network [76]. Ulrich and Gotelli [6] present a PP-algorithm that produces random bipartite graphs, whose degrees may vary freely, but in which the expected values of the degrees are intended to match those of the observed network. Thus, the degrees of a sample will be distributed randomly around the observed ones.

In contrast, our methods are not intended to produce random samples with the same edge total as an observed network. Instead, we do allow the edge total to vary within the margins specified by the intervals.

5.2 Markov Chains

In this section, we present two Markov chains that are generalizations of the classical switch chain and the curveball chain discussed in Chapter 4. The state space of both Markov chains is the set $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$ from which we want do draw samples. In a series of theorems, we show that the stationary distribution of both chains is the uniform distribution on $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$. In contrast to previously discussed chains, each Markov chain presented in this section is based on a set of three operations.

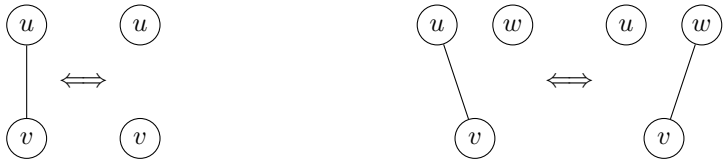

Figure 5.1: Visualization of a flip (left), and shift operation (right).

5.2.1 Simple Markov Chain

Our first Markov chain is based on the operations *classical switch* (here simply called *switch*), *flip*, and *shift*.

Definition 5.2 (flip)**.** *Let $G = (U, V, E)$ be a bipartite graph.*

1. *Choose two vertices $u \in U$ and $v \in V$ uniformly at random.*
2. *If $\{u, v\} \in E$, set $E' := E \setminus \{\{u, v\}\}$, otherwise set $E' := E \cup \{\{u, v\}\}$.*
3. *Set $G' := (U, V, E')$.*
4. *If $\tilde{d}(x) \leq \delta_{G'}(x) \leq \bar{d}(x)$ holds for each $x \in \{u, v\}$, return G'.*
5. *Otherwise, return G.*

Fig. 5.1 (left) visualizes the flip operation. Unlike the switch, a flip operation increases or decreases the total number of edges in G. In doing so, it increments or decrements the degrees of the vertices u and v and leaves the degrees of all other vertices unchanged.

Definition 5.3 (shift)**.** *Let $G = (U, V, E)$ be a bipartite graph.*

1. *Choose two vertices $u \in U$ and $v \in V$ uniformly at random.*
2. *Choose a vertex w from $(U \cup V) \setminus \{u, v\}$ uniformly at random.*
3. *If $w \in V$, switch the roles of u and v.*
4. *(a) If $\{u, v\} \notin E$ and $\{v, w\} \in E$, set $E' := (E \setminus \{\{v, w\}\}) \cup \{\{u, v\}\}$.*
   (b) Else, if $\{u, v\} \in E$ and $\{v, w\} \notin E$, set $E' := (E \setminus \{\{u, v\}\}) \cup \{\{v, w\}\}$.
   (c) In all other cases return G.
5. *Set $G' := (U, V, E')$.*
6. *If $\tilde{d}(x) \leq \delta_{G'}(x) \leq \bar{d}(x)$ holds for each $x \in \{u, w\}$ return G'.*
7. *Otherwise, return G.*

Fig. 5.1 (right) visualizes the shift operation. A shift operation does not change the total number of edges in G. However, it affects the degrees of the vertices u and w. In contrast, the degree of vertex v is unchanged.

We will now define the first Markov chain for the uniform sampling from $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$. It is based on an arbitrary probability function

$$q\colon \{\text{switch}, \text{flip}, \text{shift}\} \to (0, 1),$$

that assigns positive probability to each type of operation.

Definition 5.4 (simple chain)**.** *Let $G = (U, V, E) \in \Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$.*

1. *Randomly select a type of operation with respect to q.*
2. *Apply the operation to transform G into G'.*
3. *If the maximal number of steps has been reached, return G'.*
4. *Set $G \leftarrow G'$ and go to Step 1.*

Theorem 5.1. *The stationary distribution of the simple chain is the uniform distribution on $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$.*

In the remainder of this section, we give a proof for Theorem 5.1. By Corollary 2.4, we need to show that the simple chain is irreducible, aperiodic, and reversible with respect to the uniform distribution on $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$.

Irreducibility We start by proving the irreducibility of the Markov chain. For this purpose, we show that we can iteratively transform two bipartite graphs G and G' from $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$ into each other via a series switch, flip, and shift operations. More precisely, we show that we can strictly reduce the symmetric difference $G \triangle G'$ by iterated application of a switch, flip, or shift operation.

In the special case of $\tilde{\mathbf{r}} = \bar{\mathbf{r}}$ and $\tilde{\mathbf{c}} = \bar{\mathbf{c}}$ discussed in Chapter 4, it is well-known that the symmetric difference $G \triangle G'$ can be partitioned into a set of simple alternating cycles [59]. Then, the graphs G and G' can be transformed into each other by at most $|G \triangle G'|/2$ switch operations. We will now prove a similar result for our more general scenario.

For the following theorems, let $G = (U, V, E)$ and $G' = (U, V, E')$ be arbitrary bipartite graphs from $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$.

Lemma 5.2. *Let $c = (v_1, v_2, \ldots, v_k, v_1)$ be a simple alternating cycle in $G \triangle G'$. We can apply a switch operation to G or G' to reduce the size $|G \triangle G'|$ of the symmetric difference by two or more.*

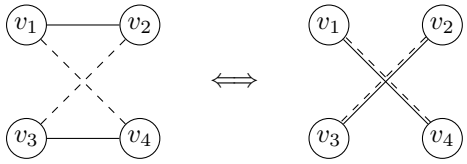

Figure 5.2: An alternating cycle of length $k = 4$. Straight lines are present in G, dashed lines are present in G'.

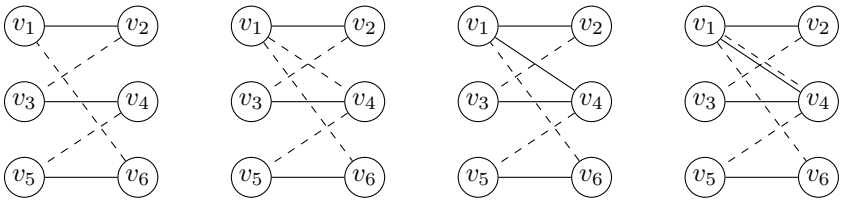

Figure 5.3: Example of an alternating cycle $c = (v_1, v_2, \ldots, v_6, v_1)$ of length $k = 6$. Straight lines are present in G, dashed lines are present in G'. From left to right: cases one to four of Lemma 5.2.

Proof. We prove the theorem inductively by the length k of the alternating cycle. For the matter of this proof, assume that $\{v_1, v_2\} \in E$ and $\{v_{k-1}, v_k\} \in E$, otherwise switch the roles of G and G'.

If $k = 4$ (see Fig. 5.2), we can apply a switch to G to gain $\bar{G} := (U, V, \bar{E})$ with

$$\bar{E} := (E \setminus \{\{v_1, v_2\}, \{v_3, v_4\}\}) \cup \{\{v_1, v_4\}, \{v_2, v_3\}\}.$$

As a consequence, $|G \triangle G'| - |\bar{G} \triangle G'| = 4$.

Now let $k > 4$ and assume the theorem to be proven for alternating cycles of length $i < k$. Depending on whether or not the edge $\{v_1, v_4\}$ exists in G and G', we discuss four cases (see Fig. 5.3).

1. If $\{v_1, v_4\} \notin E$ and $\{v_1, v_4\} \notin E'$, we can replace the edges $\{v_1, v_2\} \in E$ and $\{v_3, v_4\} \in E$ by $\{v_1, v_4\}$ and $\{v_2, v_3\}$. Thus, a switch operation may transform G into $\bar{G} = (U, V, \bar{E})$ with

   $$\bar{E} := (E \setminus \{\{v_1, v_2\}, \{v_3, v_4\}\}) \cup \{\{v_1, v_4\}, \{v_2, v_3\}\}).$$

   Consequently, $|G \triangle G'| - |\bar{G} \triangle G'| = 2$.

2. If $\{v_1, v_4\} \notin E$ and $\{v_1, v_4\} \in E'$, we can replace the edges $\{v_1, v_2\} \in E$ and $\{v_3, v_4\} \in E$ by $\{v_1, v_4\}$ and $\{v_2, v_3\}$. Thus, a switch operation may transform G into $\bar{G} = (U, V, \bar{E})$ with

   $$\bar{E} := (E \setminus \{\{v_1, v_2\}, \{v_3, v_4\}\}) \cup \{\{v_1, v_4\}, \{v_2, v_3\}\}).$$

   Consequently, $|G \triangle G'| - |\bar{G} \triangle G'| = 4$.

3. If $\{v_1, v_4\} \in E$ and $\{v_1, v_4\} \notin E'$, we do not directly find an applicable switch operation. However, there is an alternating cycle $(v_1, v_4, \ldots, v_k, v_1)$ of length $k-2$. Thus, by the induction hypothesis, we can find a switch operation that reduces the size of the symmetric difference $G \triangle G'$ by at least two.

4. If $\{v_1, v_4\} \in E$ and $\{v_1, v_4\} \in E'$, we can replace the edges $\{v_1, v_4\} \in E'$ and $\{v_2, v_3\} \in E'$ by $\{v_1, v_2\}$ and $\{v_3, v_4\}$. Thus, a switch operation may transform G' into $\bar{G}' = (U, V, \bar{E}')$ with

$$\bar{E}' := (E' \setminus \{\{v_1, v_4\}, \{v_2, v_3\}\}) \cup \{\{v_1, v_2\}, \{v_3, v_4\}\}.$$

Consequently, $|G \triangle G'| - |G \triangle \bar{G}'| = 2$. □

Lemma 5.3. *Let $(v_1, v_2, \ldots, v_k)$ be a simple non-extendable alternating path in $G \triangle G'$.*

a) If $\{v_1, v_2\} \in E$, we may remove an edge $\{v_1, \cdot\}$ from E or add a new edge $\{v_1, \cdot\}$ to E' without violating the lower and upper bounds on the degree of v_1.

b) If $\{v_1, v_2\} \in E'$, we may add a new edge $\{v_1, \cdot\}$ to E or remove an edge $\{v_1, \cdot\}$ from E' without violating the lower and upper bounds on the degree of v_1.

Proof. Assume $\{v_1, v_2\} \in E$, otherwise switch the roles of G and G'. As the path is simple and non-extendable, there is no edge $\{v_1, \cdot\} \in E' \setminus E$ that can be appended to p without violating its alternating structure. Thus, $\delta_G(v_1)$ exceeds $\delta_{G'}(v_1)$ by at least one. Since G and G' are valid realizations and so do not violate the lower and upper bounds on the degree of v_1, the inequalities

$$\tilde{d}(v_1) \leq \delta_{G'}(v_1) < \delta_G(v_1) \leq \bar{d}(v_1)$$

must hold. Hence, by adding an additional edge $\{v_1, \cdot\}$ to E' or removing an edge $\{v_1, \cdot\}$ from E, the lower and upper bounds on the degree of v_1 are not violated. □

Lemma 5.4. *Let $p = (v_1, v_2, \ldots, v_{k-1}, v_k)$ be a simple non-extendable alternating path in $G \triangle G'$. We can apply a series of at most two operations to G or G' to reduce the size $|G \triangle G'|$ of the symmetric difference by one or more.*

Proof. First, let p be of odd length, thus k is even. Assume that $\{v_1, v_2\} \in E$ and $\{v_{k-1}, v_k\} \in E$, otherwise switch the roles of G and G'. Depending on the existence of the edge $\{v_1, v_k\}$ in G and G' we discuss four cases (see Fig. 5.4).

1. If $\{v_1, v_k\} \notin E$ and $\{v_1, v_k\} \notin E'$, Lemma 5.3 allows to add the edge $\{v_1, v_k\}$ to E'. Thus, a flip operation may transform G' into $\tilde{G}' = (U, V, \tilde{E}')$ with

$$\tilde{E}' := E' \cup \{\{v_1, v_k\}\}.$$

In doing so, we temporarily create a situation with $|G \triangle G'| - |G \triangle \tilde{G}'| = -1$. However, we additionally construct the alternating cycle $c = (v_1, v_2, \ldots, v_k, v_1)$ in $G \triangle \tilde{G}'$. By Lemma 5.2, we can find a switch operation that transforms G into $\bar{G}$, or $\tilde{G}'$ into $\bar{G}'$ such that $|G \triangle G'| - |\bar{G} \triangle \bar{G}'| \geq 1$.

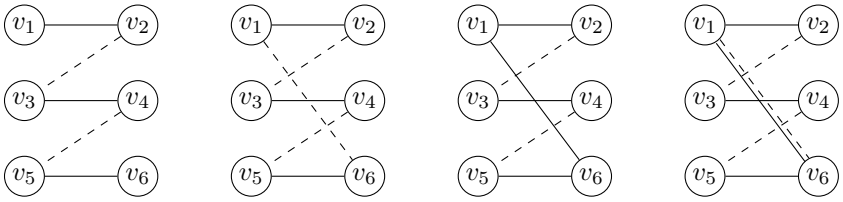

Figure 5.4: Example of a non-extendable alternating path $p = (v_1, v_2, \ldots, v_6)$ of odd length ($k = 6$). Straight lines are present in G, dashed lines are present in G'. From left to right: cases one to four of the first part of Lemma 5.4.

2. If $\{v_1, v_k\} \notin E$ and $\{v_1, v_k\} \in E'$ the closed path $(v_1, v_2, \ldots, v_k, v_1)$ is a simple alternating cycle in $G \triangle G'$. By Lemma 5.2, we can find a switch operation that transforms G into $\bar{G}$, or G' into $\bar{G}'$ such that $|G \triangle G'| - |\bar{G} \triangle \bar{G}'| \geq 2$.

3. If $\{v_1, v_k\} \in E$ and $\{v_1, v_k\} \notin E'$, Lemma 5.3 allows to remove the edge $\{v_1, v_k\}$ from E. Thus, a flip operation may transform G into $\bar{G} = (U, V, \bar{E})$ with

$$\bar{E} := E \setminus \{\{v_1, v_k\}\}.$$

   Consequently, $|G \triangle G'| - |\bar{G} \triangle G'| = 1$.

4. If $\{v_1, v_k\} \in E$ and $\{v_1, v_k\} \in E'$, Lemma 5.3 allows to remove the edge $\{v_1, v_k\}$ from E. Thus, a flip operation may transform G into $\tilde{G} = (U, V, \tilde{E})$ with

$$\tilde{E} := E \setminus \{\{v_1, v_k\}\}.$$

   In doing so, we temporarily create a situation in which $|G \triangle G'| - |\tilde{G} \triangle G'| = -1$. However, we also create the alternating cycle $c = (v_1, v_2, \ldots, v_k, v_1)$ in $\tilde{G} \triangle G'$. By Lemma 5.2, we can find a switch operation that transforms $\tilde{G}$ into $\bar{G}$, or G' into $\bar{G}'$ such that $|G \triangle G'| - |\bar{G} \triangle \bar{G}'| \geq 1$.

Now let p be of even length, thus k is odd. In this case, the edge $\{v_1, v_k\}$ can neither exist in E nor in E' since v_1 and v_k are in the same vertex set. However, it suffices to focus on the existence of the edge $\{v_1, v_{k-1}\}$ in G and G'. As before, we discuss four cases (see Fig. 5.5).

1. If $\{v_1, v_{k-1}\} \notin E$ and $\{v_1, v_{k-1}\} \notin E'$, Lemma 5.3 allows to replace $\{v_{k-1}, v_k\} \in E'$ by $\{v_{k-1}, v_1\}$. Thus, a shift operation may transform G' into $\tilde{G}' = (U, V, \tilde{E}')$ with

$$\tilde{E}' := (E' \setminus \{\{v_{k-1}, v_k\}\}) \cup \{\{v_{k-1}, v_1\}\}.$$

   While this leads to a situation where $|G \triangle G'| - |G \triangle \tilde{G}'| = 0$, we also create the alternating cycle $(v_1, \ldots, v_{k-1}, v_1)$ in $G \triangle \tilde{G}'$. By Lemma 5.2, we can find a switch operation that transforms G into $\bar{G}$, or $\tilde{G}'$ into $\bar{G}'$ such that $|G \triangle G'| - |\bar{G} \triangle \bar{G}'| \geq 2$.

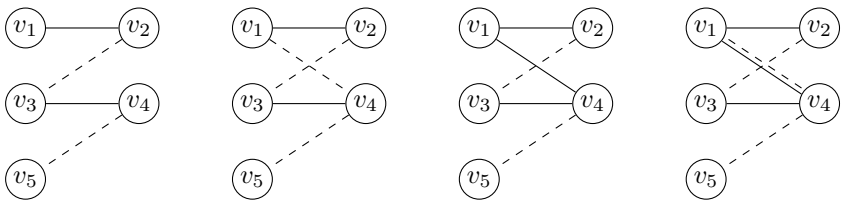

Figure 5.5: Example of a non-extendable alternating path $p = (v_1, v_2, \ldots, v_5)$ of even length ($k = 5$). Straight lines are present in G, dashed lines are present in G'. From left to right: cases one to four of the second part of Lemma 5.4.

2. If $\{v_1, v_{k-1}\} \notin E$ and $\{v_1, v_{k-1}\} \in E'$, the closed path $(v_1, v_2, \ldots, v_{k-1}, v_1)$ is a simple alternating cycle in $G \triangle G'$. By Lemma 5.2, we can find a switch operation that transforms G into $\bar{G}$, or G' into $\bar{G}'$ with $|G \triangle G'| - |\bar{G} \triangle \bar{G}'| \geq 2$.

3. If $\{v_1, v_{k-1}\} \in E$ and $\{v_1, v_{k-1}\} \notin E'$, Lemma 5.3 allows to replace $\{v_1, v_{k-1}\} \in E$ by $\{v_k, v_{k-1}\}$. Thus, a shift operation transforms G into $\bar{G} = (U, V, \bar{E})$ with

$$\bar{E} = (E \setminus \{\{v_1, v_{k-1}\}\}) \cup \{\{v_{k-1}, v_k\}\}.$$

   Consequently, $|G \triangle G'| - |\bar{G} \triangle G'| = 2$.

4. If $\{v_1, v_{k-1}\} \in E$ and $\{v_1, v_{k-1}\} \in E'$, Lemma 5.3 allows to replace $\{v_1, v_{k-1}\} \in E$ by $\{v_k, v_{k-1}\}$. Thus, a shift operation transforms G into $\tilde{G} = (U, V, \tilde{E})$ with

$$\tilde{E} = (E \setminus \{\{v_1, v_{k-1}\}\}) \cup \{\{v_k, v_{k-1}\}\}.$$

   While this leads to a situation where $|G \triangle G'| - |\tilde{G} \triangle G'| = 0$, we also create the alternating cycle $(v_1, \ldots, v_{k-1}, v_1)$ in $\tilde{G} \triangle G'$. By Lemma 5.2, we can find a switch operation that transforms $\tilde{G}$ into $\bar{G}$, or G' into $\bar{G}'$ such that $|G \triangle G'| - |\bar{G} \triangle \bar{G}'| \geq 2$. □

Lemma 5.5. *By a series of at most $2|G \triangle G'|$ switch, flip, and shift operations, we can transform G and G' into each other.*

Proof. We prove the theorem inductively by the size of $G \triangle G'$. If $|G \triangle G'| = 0$, the graphs must be identical. Otherwise, apply the following algorithm to construct either a simple non-extendable alternating path p or a simple alternating cycle c in $G \triangle G'$.

1. Choose an arbitrary edge $\{v, w\} \in G \triangle G'$ and let $p \leftarrow (v, w)$. Let $z \leftarrow w$ be the rightmost node of the path p.

2. While there is some edge $\{z, y\} \in G \triangle G'$ that can be used to extend p (preserving its alternating structure), append y to p. If this produces a cycle, terminate. Otherwise, let $z \leftarrow y$ and iterate until p cannot be extended.

3. Now reverse the path p, let $z \leftarrow v$ be the rightmost node and repeat the expanding of p.

The algorithm terminates with a simple non-extendable path or detects a simple alternating cycle. In the first case, we can reduce the size of $G \triangle G'$ via Lemma 5.4, while in the latter case, we can apply Lemma 5.2. Thus, we can inductively reduce $|G \triangle G'|$ to zero. The upper bound on the number of operations follows from Lemma 5.4 and Lemma 5.2 since we need at most two operations to reduce the symmetric difference by one. □

Aperiodicity As we showed that the state graph of the simple chain is strongly connected, the Markov chain is irreducible. In the next step, we show that the simple chain is aperiodic.

Lemma 5.6. *The simple chain is aperiodic.*

Proof. We prove the lemma by showing that the state graph of the simple chain contains loops. For this purpose, consider a bipartite graph $G = (U, V, E) \in \Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$. Depending on the degrees of G, one of three cases occurs.

1. If there is an isolated vertex in G, a shift operation may select three vertices $\{u, v, w\}$ from $U \cup V$ with $\delta_G(v) = 0$. Then, it will return $G' = G$ as neither $\{u, v\}$ nor $\{v, w\}$ can be included in E.

2. Otherwise, if there is a vertex $v \in U \cup V$ with $\delta_G(v) > 1$, then v has at least two adjacent vertices, say u and w. Whenever a shift operation selects $\{u, v, w\}$, it will return $G' = G$ as both $\{u, v\}$ and $\{v, w\}$ will be included in E.

3. If neither (a) nor (b) is true, then all vertices must have a degree of one. In such a case, a shift operation may very well select three vertices $\{u, v, w\}$ such that $\{u, v\} \notin E$ and $\{v, w\} \in E$. However, the shift operation then creates a bipartite graph $G' \neq G$ with $\delta_{G'}(u) = \delta_G(u) + 1$ and $\delta_{G'}(w) = \delta_G(w) - 1$. Consequently, the vertex u gains a degree of two or more. Thus, a shift operation applied to G' may return G' with nonzero probability. □

Symmetry By Lemma 5.5 and 5.6, the simple Markov chain is ergodic. Thus, it has a unique stationary distribution π. It remains to show that this stationary distribution is uniform.

Lemma 5.7. *The simple chain is reversible with respect to the uniform distribution on* $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$.

Proof. By Theorem 2.2, we have to show that the transition probability function of the simple chain is symmetric. For this purpose, we focus on the probability of transforming a state $G = (U, V, E)$ into $G' = (U, V, E') \neq G$ via a single operation.

1. As the probability of transforming G into G' via a switch operation is

$$p_{\text{switch}}(G, G') = \binom{m}{2}^{-1}\binom{n}{2}^{-1} = p_{\text{switch}}(G', G),$$

   another switch operation can reverse the transformation with identical probability.

2. The probability of transforming a graph G into $G' \neq G$ via a flip operation is

$$p_{\text{flip}}(G, G') = m^{-1}n^{-1} = p_{\text{flip}}(G', G).$$

   Thus, a second flip operation may reverse the transformation with identical probability.

3. Transforming G into $G' \neq G$ via a single shift operation has a probability of

$$p_{\text{shift}}(G, G') = m^{-1}n^{-1}(m + n - 2)^{-1} = p_{\text{shift}}(G', G).$$

   Thus, a second shift operation may reverse the transformation with identical probability.

As the transition probability $p\colon \Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}}) \times \Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}}) \to [0, 1]$ is composed by

$$\begin{aligned} p(G, G') = {} & q(\text{switch}) \cdot p_{\text{switch}}(G, G') \\ & + q(\text{flip}) \cdot p_{\text{flip}}(G, G') \\ & + q(\text{shift}) \cdot p_{\text{shift}}(G, G'), \end{aligned}$$

and q is constant, we conclude that $p(G, G')$ equals $p(G', G)$. By Theorem 2.2, the simple chain is reversible with respect to the uniform distribution on $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$. □

5.2.2 Informed Markov Chain

As shown in the previous section, the simple chain can be used for the uniform sampling of bipartite graphs from the set $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$. However, a potential problem of this Markov chain is its large loop probability that may arise from its uninformed nature of selecting random vertices. For that reason, we present a second Markov chain that is designed to avoid loop transitions by a more informed vertex selection. In addition, we increase the number of edges that are affected by a single transition. The price of this modification is an increased running time per step.

Inspired by the curveball algorithm [23], we replace the simple operations switch, shift, and flip by their more complex counterparts *trade*, *multi-shift*, and *multi-flip*. The main difference between the simple and complex operations is the number of affected edges per transition. Whereas the simple operations affect at most four edges per step, the complex operations affect $\mathcal{O}(|V|)$ edges. We start with defining two new operations.

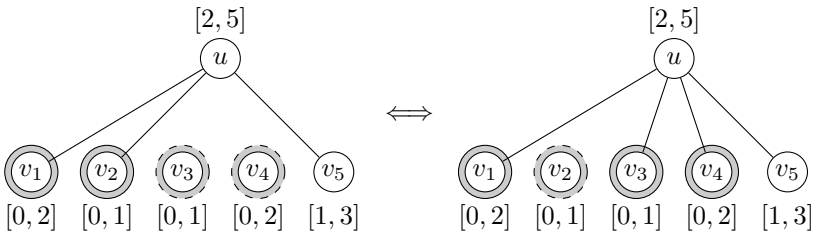

Figure 5.6: Visualization of a multi-flip operation. The numbers in brackets show the upper and lower bounds on the degrees. Transforming the left graph into the right: $V_0 = \{v_3, v_4\}$ (gray, dashed border), $V_1 = \{v_1, v_2\}$ (gray, solid border). Since the bounds on the degree of u differ by three, we randomly chose the set $S = \{v_2, v_3, v_4\}$ of size three from $V_0 \cup V_1$. Subsequently, we randomly chose $S = T$ as a subset of T.

Definition 5.5 (multi-flip)**.** *Let $G = (U, V, E)$ be a bipartite graph.*

1. *Choose a vertex $u \in U$ uniformly at random.*
2. *Determine two disjoint vertex sets $V_0 \subseteq V$ and $V_1 \subseteq V$ defined by*

$$V_0 := \left\{v \in V \colon \{u, v\} \notin E \wedge \delta_G(v) < \bar{d}(v)\right\},$$
$$V_1 := \left\{v \in V \colon \{u, v\} \in E \wedge \delta_G(v) > \tilde{d}(v)\right\}.$$

   Thus, V_0 is the set of vertices that are not adjacent to u and may be part of an additional edge without violating the upper bound on their degree. In contrast, V_1 is the set of vertices that are adjacent to u and may lose an incident edge without violating the lower bound on their degree.

3. *Let $s := \min\left(\bar{d}(u) - \tilde{d}(u), |V_0 \cup V_1|\right)$. If $s = 0$, return G.*
4. *Choose a subset $S \subseteq V_0 \cup V_1$ of size s uniformly at random.*
5. *Choose a subset $T \subseteq S$ of arbitrary size uniformly at random.*
6. *Create the edge sets E_0 and E_1 such that*

$$E_0 := \{\{u, v\} \colon v \in T \wedge \{u, v\} \in E\},$$
$$E_1 := \{\{u, v\} \colon v \in T \wedge \{u, v\} \notin E\}.$$

7. *Remove the edges E_0 from E and add the edges E_1, thus create the bipartite graph $G' = (U, V, E')$ with $E' := (E \setminus E_0) \cup E_1$.*
8. *If $\tilde{d}(u) \leq \delta_{G'}(u) \leq \bar{d}(u)$ return G', otherwise return G.*

Fig. 5.6 visualizes the multi-flip. The multi-flip contains the simple flip as a special case, when $|T| = 1$. In addition, the multi-flip may produce $G' = G$ when $T = \emptyset$.

Lemma 5.8. *The transition probability function of the multi-flip is symmetric.*

Proof. Consider two graphs $G = (U, V, E)$ and $G' = (U, V, E')$ that can be transformed into each other via a single multi-flip. To distinguish the sets V_0, V_1, and T in G and G', we affiliate each set with the corresponding graph. Following from its definition, the probability for transforming a graph G into $G' \neq G$ via a single multi-flip is

$$p_{\text{multi-flip}}(G, G') = m^{-1} \binom{|V_0(G) \cup V_1(G)|}{s}^{-1} 2^{-s},$$

where $s = \min(\bar{d}(u) - \tilde{d}(u), |V_0(G) \cup V_1(G)|)$. To prove that $p_{\text{multi-flip}}(G, G') = p_{\text{multi-flip}}(G', G)$, it suffices to show that

$$|V_0(G) \cup V_1(G)| = |V_0(G') \cup V_1(G')|,$$

since m and $\bar{d}(u) - \tilde{d}(u)$ are invariant constants. For this purpose, assume the contrary, i.e.

$$|V_0(G) \cup V_1(G)| > |V_0(G') \cup V_1(G')|.$$

Assume further that $v \in V_0(G)$ is a vertex that is neither in $V_0(G')$ nor in $V_1(G')$. Since $v \in V_0(G)$, it follows from the definition of V_0 that $\{u, v\} \notin E$. Now consider two cases that can occur while transforming G into G'.

1. If $v \in T(G)$, the edge $\{u, v\}$ must be included in E' by the definition of the multi-shift. Thus, $\delta_{G'}(v) = \delta_G(v) + 1$. Since $\{u, v\} \in E'$ and $\tilde{d}(v) < \delta_{G'}(v)$, it follows that $v \in V_1(G')$, which contradicts our assumption.

2. Otherwise, if $v \notin T(G)$, the degree of v is not changed by the multi-shift operation and thus, $v \in V_0(G')$. Again, this contradicts our assumption.

Thus, if $v \in V_0(G)$, either $v \in V_0(G')$ or $v \in V_1(G')$ must hold. As the symmetric case $v \in V_1(G)$ is discussed analogously, we conclude

$$|V_0(G) \cup V_1(G)| = |V_0(G') \cup V_1(G')|.$$

Hence, $p_{\text{multi-flip}}(G, G') = p_{\text{multi-flip}}(G', G)$. □

Definition 5.6 (multi-shift)**.** *Let $G = (U, V, E)$ be a bipartite graph.*

1. *Choose a random vertex $u \in U$.*

2. *Determine two disjoint vertex sets $V_0 \subseteq V$ and $V_1 \subseteq V$ defined by*

$$V_0 := \left\{ v \in V : \{u, v\} \notin E \wedge \delta_G(v) < \bar{d}(v) \right\},$$
$$V_1 := \left\{ v \in V : \{u, v\} \in E \wedge \delta_G(v) > \tilde{d}(v) \right\}.$$

Up to this point the multi-shift is defined identically to the multi-flip.

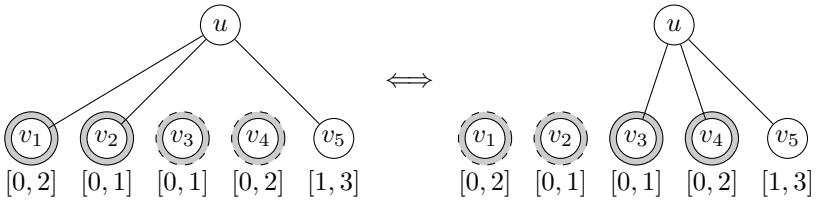

Figure 5.7: Visualization of a multi-shift operation. The numbers in brackets show the relevant upper and lower bounds on the degrees. Transforming the left graph into the right: $V_0 = \{v_3, v_4\}$ (gray, dashed border), $V_1 = \{v_1, v_2\}$ (gray, solid border). We randomly selected $S = \{v_3, v_4\}$ of size $|V_1|$ from $V_0 \cup V_1$.

3. *Choose a subset $S \subseteq V_0 \cup V_1$ of size $|V_1|$ uniformly at random.*

4. *Create the edge sets E_0 and E_1 such that*

$$E_0 := \{\{u, v\} \colon v \in V_0 \cup V_1\},$$
$$E_1 := \{\{u, v\} \colon v \in S\}.$$

5. *Remove the edges E_0 from E and add the edges E_1, thus create and return the bipartite graph $G' = (U, V, E')$ with $E' := (E \setminus E_0) \cup E_1$.*

Fig. 5.7 visualizes the multi-shift. In contrast to the multi-flip, the multi-shift operation does not change the degree of u since S is chosen to have the same cardinality as V_1. In addition, the upper or lower bound on the degree of a node $v \in V$ cannot be violated by the way we define V_0 and V_1. Consequently, $G' \in \Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$.

Like the other operations, the multi-shift contains the possibility of creating $G' = G$ when in step three S is chosen to be equal to V_1. In addition, it may simulate a simple shift if $|V_1 \setminus S| = 1$.

Lemma 5.9. *The transition probability function of the multi-shift is symmetric.*

Proof sketch. Following from its definition, the probability for transforming a graph G into $G' \neq G$ via a single multi-shift is

$$p_{\text{multi-shift}}(G, G') = m^{-1} \binom{|V_0(G) \cup V_1(G)|}{|V_1(G)|}^{-1}.$$

Since the degree of u is not changed by a multi-shift, $|V_1(G)|$ is equal to $|V_1(G')|$. In addition, a similar argument as used in the proof of Lemma 5.8 shows $|V_0(G) \cup V_1(G)| = |V_0(G') \cup V_1(G')|$. Thus, $p_{\text{multi-shift}}(G, G') = p_{\text{multi-shift}}(G', G)$. □

By the way we defined our operations, a single operation affects at most two vertices from U and up to $|V|$ vertices from V. Thus, we need at least $|U|/2$ operations until each vertex from U is affected by an operation. We intend to speed up the Markov

chain by occasionally switching the roles of U and V. However, if $|U| \ll |V|$ we do not like to switch the roles of U and V too often since the number of affected edges per operation is largely reduced in such cases. Thus, we switch the roles of U and V with probability $|U|/(|U| + |V|) = m/(m + n)$.

The improved Markov chain can now be stated as follows. Like the simple chain, it is based on an arbitrary probability function

$$q\colon \{\text{trade}, \text{multi-flip}, \text{multi-shift}\} \to (0, 1)$$

that assigns positive probability to each operation.

Definition 5.7 (informed chain)**.** *Let $G = (U, V, E) \in \Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$.*

1. *Select a real number $u \in [0, 1)$ uniformly at random.*
2. *If $u < m/(m + n)$, switch the roles of U and V.*
3. *Randomly select an operation type with respect to q.*
4. *Apply the selected operation to transform G into G'.*
5. *If the maximal number of steps has been reached, return G'.*
6. *Set $G \leftarrow G'$ and go to Step 1.*

Theorem 5.10. *The stationary distribution of the informed Markov chain is the uniform distribution on $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$.*

Proof. We first prove that the improved chain is irreducible. Since each of the three complex operations contains its simple counterpart as a special case, a state graph of the informed chain contains the corresponding state graph of the simple chain as a sub-graph. As the simple chain is irreducible, the informed chain must be irreducible, too. In addition, the improved chain is aperiodic since each operation may return G with non-zero probability. Thus, the improved chain possesses a unique stationary distribution. Since the transition probabilities of each operation type are symmetric, it follows from Corollary 2.2 that the stationary distribution is the uniform distribution on $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$. □

5.2.3 Dynamic Adjustment of Probability

The probability function q for selecting an operation type has a large effect on the efficiency of the sampling algorithms. A simple implementation is to select each operation with equal probability. However, this is obviously not a good choice in the special case of $\tilde{\mathbf{r}} = \bar{\mathbf{r}}$ and $\tilde{\mathbf{c}} = \bar{\mathbf{c}}$ since all flip and shift operations will fail and thus, at least 2/3 of all transitions will be loops.

It is not immediately clear how to choose q in a better way since the applicability of the operations depends on the given integer vectors. Thus, we implemented a

strategy that initially selects each type of operation with identical probability and dynamically adjusts q during the simulation of the Markov chain. For this purpose, we keep track of the number of simulated transitions for each type, and the number of non-loop transitions for each type. Every 100 steps, we re-balance the probability function q so that each type of operation gains a probability that is proportional to its success rate. Due to the dynamic adjustment of q, less time will be spent in operations that tend to have a large loop probability. To ensure positive probability for each type of operation, we slightly dampen the effect of the re-balancing by incorporating previous values.

Being specific, when $\mathbb{O}$ denotes a set of operations, we define a function $s\colon \mathbb{O}\times\mathbb{N} \to \mathbb{N}$ such that $s(o,t)$ denotes the number of non-loop transitions of an operation $o \in \mathbb{O}$ within the first t steps of the simulation. Similarly, we define $r\colon \mathbb{O} \times \mathbb{N} \to \mathbb{N}$ such that $r(o,t)$ denotes the total number of simulated operations of the operation type $o \in \mathbb{O}$ within the first t steps (including loop transitions). We define the *success rate* $\alpha\colon \mathbb{O} \times \mathbb{N} \to \mathbb{N}$ of each operation type after t steps by

$$\alpha(o,t) := \frac{s(o,t)}{r(o,t)},$$

and define $Z\colon \mathbb{N} \to \mathbb{Q}$ by $Z(t) := \sum_{o\in\mathbb{O}} \alpha(o,t)$. In doing so, the probability function $q\colon \mathbb{O} \times \mathbb{N} \to [0,1]$ depends on the current number of steps t, i.e.

$$q(o,t) := \begin{cases} 1/3, & \text{if } t = 0, \\ q(o,t-1), & \text{if } (t \bmod 100) \neq 0, \\ \left(q(o,t-100) + \frac{\alpha(o,t)}{Z(t)}\right)/2, & \text{if } t > 0 \text{ and } (t \bmod 100) = 0. \end{cases} \tag{5.1}$$

In the unlikely case that the calculations as described above require a division by zero, we simply set $q(o,t) := q(o,t-100)$. Using the operation sets

$$\mathbb{O}_1 := \{\text{switch}, \text{flip}, \text{shift}\} \quad \text{and} \quad \mathbb{O}_2 := \{\text{trade}, \text{multi-flip}, \text{multi-shift}\},$$

Eq. 5.1 defines the probability functions q for the simple and informed Markov chain.

By design, the Markov chains are no longer time-homogeneous if we dynamically adjust the transition probability every 100 steps. More precisely, the series of random variables $X = (X_0, X_1, \ldots)$ produced by our sampling algorithms is divided into batches of length 100, in which the Markov chains behave time-homogeneously. Thus, mathematically speaking, each batch of 100 random variables will define a new Markov chain that is different from the previous one. However, as the transition probability is designed to be symmetric regardless of the current number of steps, the stationary distribution of each chain will be the uniform distribution on $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$.

In the following discussion, we will call a Markov chain *static* when q is fixed to $1/3$ for each type of operation and *dynamic* when we use dynamic adjustment of the probability function via Eq. 5.1.

5.3 Experimental Results and Discussion

In this section, we present an experimental evaluation of the sampling algorithms introduced in the previous section. Whereas the first experiment is designed to gain insights into structural properties of the associated Markov chains, the second experiment is devoted to the number of steps required to produce random samples. Finally, we demonstrate how our sampling methods can be used to analyse structural properties of partially observed networks.

5.3.1 State Graph Analysis

In a first experiment, we assessed whether the informed Markov mixes faster than the simple chain, and if so, by which factor. For this purpose, we calculated the total mixing time of both chains on a common set of input instances. To create a set of scalable input instances, we re-used the instance classes from Chapter 4 as the vectors $\tilde{\mathbf{r}}$ and $\tilde{\mathbf{c}}$ of lower bounds. To each element of these vectors, we added a constant of $d = 1$ that simulates the imperfectness of the data collection process. In doing so, we created the following classes of integer vectors.

Type A:	$\tilde{\mathbf{r}} = (n-1, n-2, 1, 1, 1)$,	$\tilde{\mathbf{c}} = (2, 2, \ldots, 2)$
	$\bar{\mathbf{r}} = (n, n-1, 2, 2, 2)$,	$\bar{\mathbf{c}} = (3, 3, \ldots, 3)$
Type B:	$\tilde{\mathbf{r}} = (n-1, n-2, 2, 1)$,	$\tilde{\mathbf{c}} = (2, 2, \ldots, 2)$
	$\bar{\mathbf{r}} = (n, n-1, 3, 2)$,	$\bar{\mathbf{c}} = (3, 3, \ldots, 3)$
Type C:	$\tilde{\mathbf{r}} = (n-1, n-2, 3)$,	$\tilde{\mathbf{c}} = (2, 2, \ldots, 2)$
	$\bar{\mathbf{r}} = (n, n-1, 4)$,	$\bar{\mathbf{c}} = (3, 3, \ldots, 3)$
Type D:	$\tilde{\mathbf{r}} = (n-1, n-1, 1, 1)$,	$\tilde{\mathbf{c}} = (2, 2, \ldots, 2)$
	$\bar{\mathbf{r}} = (n, n, 2, 2)$,	$\bar{\mathbf{c}} = (3, 3, \ldots, 3)$
Type E:	$\tilde{\mathbf{r}} = (n-1, n-1, 1, 1)$,	$\tilde{\mathbf{c}} = (2, 2, \ldots, 2)$
	$\bar{\mathbf{r}} = (n, n, 2, 2)$,	$\bar{\mathbf{c}} = (3, 3, \ldots, 3)$
Type F:	$\tilde{\mathbf{r}} = (\lceil n/2 \rceil, \lfloor n/2 \rfloor)$,	$\tilde{\mathbf{c}} = (1, 1, \ldots, 1)$
	$\bar{\mathbf{r}} = (\lceil n/2 \rceil + 1, \lfloor n/2 \rfloor + 1)$,	$\bar{\mathbf{c}} = (\underbrace{2, 2, \ldots, 2}_{n \text{ times}})$

As the number $|\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})|$ grows fast, the calculation of the total mixing time is infeasible even for small instances. For that reason, we calculated the upper spectral bound. In doing so, we could process large state graphs up to a size of about 40 million states. Figs. 5.8 and 5.9 show how the upper spectral bound of both Markov chains depends on the size of the input instances. We highlight some important observations.

- The data suggests that the upper spectral bound of the simple chain grows more than one order of magnitude faster than that of the informed chain.

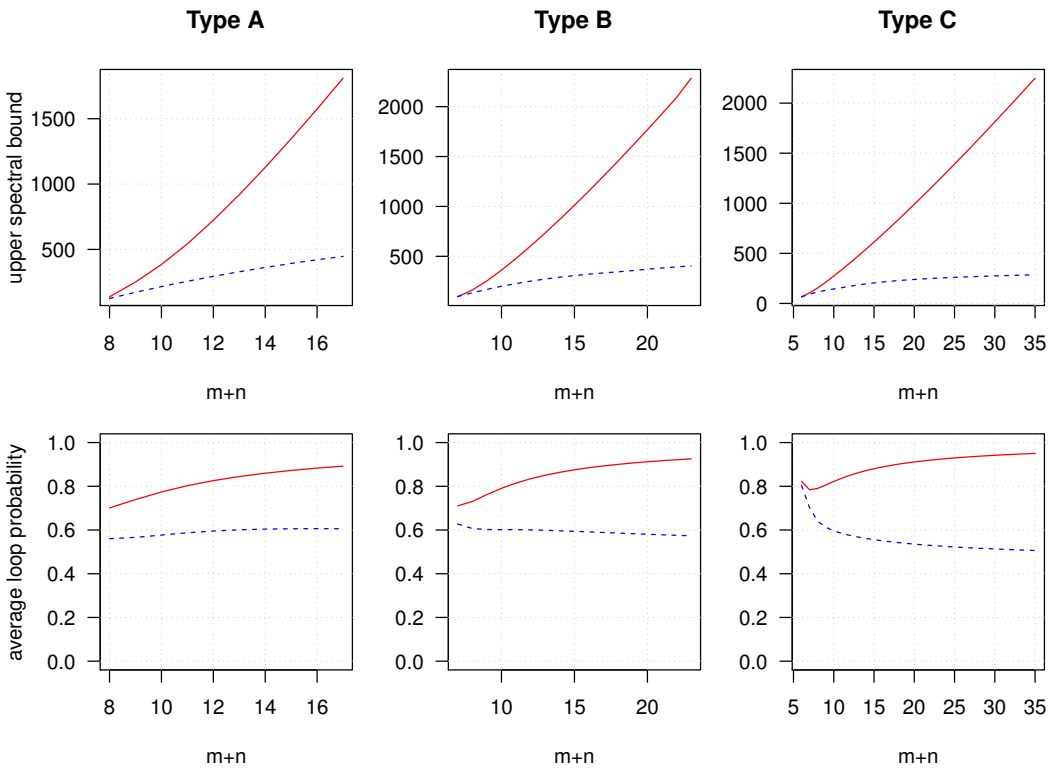

Figure 5.8: Upper spectral bound and expected loop probability of the simple (red, solid) and informed (blue, dashed) Markov chain on members of instance classes A, B, and C. ($\varepsilon = 0.01$.)

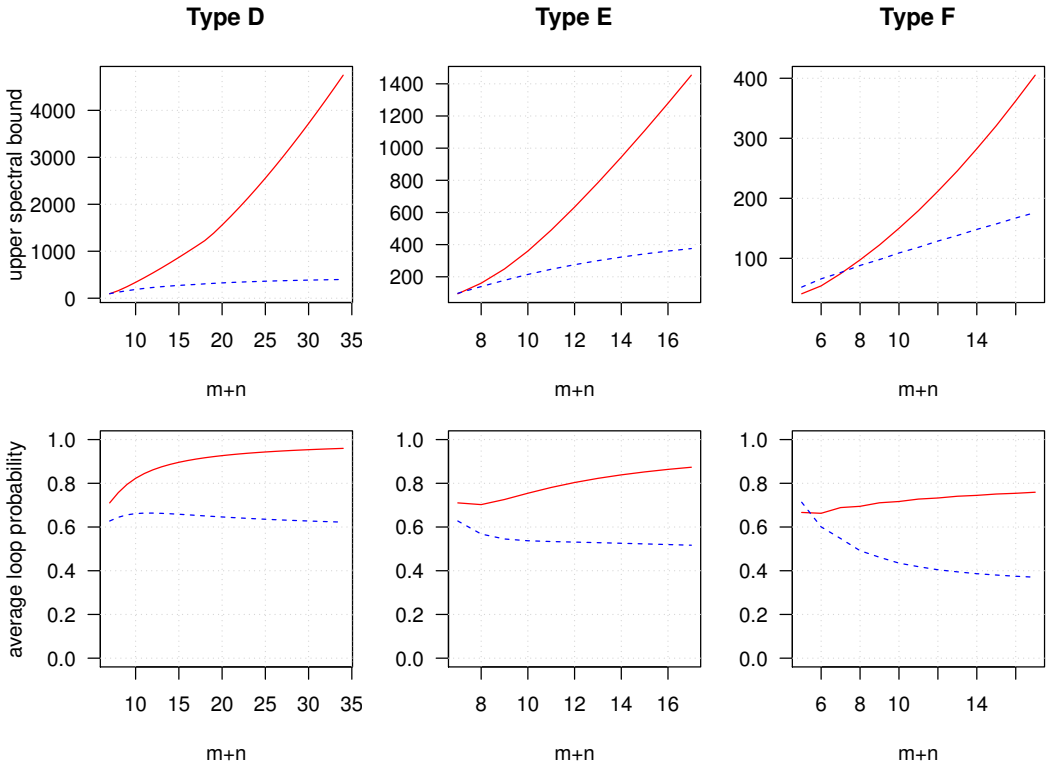

Figure 5.9: Upper spectral bound and expected loop probability of the simple (red, solid) and informed (blue, dashed) Markov chain on members of instance classes D, E, and F. ($\varepsilon = 0.01$.)

However, we should not jump to the conclusion that the informed is also faster in practice, as it requires a linear running time in each step. We will further investigate the practical efficiency of the sampling algorithms in another set of experiments.

- We observe that the loop probability of the simple chain increases with growing n and exceeds that of the informed chain when n is sufficiently large.

Our experiments showed that the informed Markov chain, as intended, mixes significantly faster than the simple chain. We conclude that the informed chain has the potential to outperform the simple one in practical applications.

5.3.2 Convergence of Sample Means

As the number of states grows fast, the calculation of the upper spectral bound becomes infeasible when the input instances grow larger. To still assess the efficiency of the Markov chains, we applied the technique described in Section 2.4 and observed the sample means of an auxiliary function $f\colon \Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}}) \to \mathbb{R}$. In doing so, we could easily process instances for which the construction of a state graph, and even the calculation of the empirical mixing time, would be too time-consuming.

Auxiliary Function The calculation of the sample means requires an auxiliary function $f\colon \Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}}) \to \mathbb{R}$ that is evaluated on a large set of random samples. In this experiment, we focused on several functions known from ecology. In ecological null-network analysis, the structure of bipartite networks can be quantified by several measures of *nestedness*, from which we considered the following.

- The $\bar{S}^2$ statistic introduced by Roberts and Stone [77].
- The *nested subset* statistic S_{nest} introduced by Patterson and Atmar [48].
- The *nestedness measure based on overlap and decreasing fill* (NODF) [78].
- The *spectral radius* of the bipartite graph's adjacency matrix [79].

The definitions of these metrics can be found in Appendix C.

Experimental Setup Starting at a fixed initial state $s \in \Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$, we simulated a random walk of length $t = 10^5$ for both the simple and the informed Markov chain. Every ten steps, we interrupted the simulation to evaluate the auxiliary function on the current state of the Markov chains. This process was repeated $N = 1000$ times to calculate the sample means $\mu_{p_s^{(t)}}[f(X)]$ for $t = 10, 20, \ldots, 100000$.

Initial State To observe the stabilization of the sample means, the value $f(s)$ at the initial state s should considerably differ from the expected value $E_\pi[f(X)]$. As most structural metrics considered in this experiment are well-known to be correlated with the edge total, we intentionally constructed the initial state s to possess as many edges as allowed by the upper bounds $\bar{\mathbf{r}}$ and $\bar{\mathbf{c}}$. For this purpose, we applied a special case of the realization algorithm described in Section 5.4.

Experiment 5.1 As many real-world networks are known to possess a scale-free degree distribution [80], we applied a Barabási-Albert model [81] to construct the degree sequence of a random, scale-free bipartite graph $G = (U, V, E)$.

Starting with a trivial graph consisting of two vertices connected by a single edge, we iteratively added new vertices to U and V. Each newly added vertex is connected to the existing vertices of the opposing vertex set with a probability that is proportional to the degree of the existing vertices. Thus, vertices that already have a high degree tend to gain more neighbors. After 100 vertices have been included in each U and V, we stopped the process and determined the degrees of the vertices in U and V as the integer vectors $\tilde{\mathbf{r}}$ and $\tilde{\mathbf{c}}$ (see Table B.1 in Appendix B).

To simulate the imperfectness of the data collection process, we added a constant d to each element of $\tilde{\mathbf{r}}$ and $\tilde{\mathbf{c}}$ to create the vectors of upper bounds $\bar{\mathbf{r}}$ and $\bar{\mathbf{c}}$. Based on these vectors, we estimated the efficiency of our Markov chains by observing the sample means of the $\bar{S}^2$ statistic. Fig. 5.10 shows the results for $d = 1$ and $d = 5$.

- Considering the case of $d = 1$, we observe that the sample mean of the informed chain stabilizes after very few steps. In contrast, the simple chain needs several thousand iterations to stabilize.

- By counting the number of loop transitions, we observe that about 94% of all transitions of the simple chain have been loops, in comparison to only 27% of the informed chain's transitions.

- Using dynamic probability adjustment, the simple chain starts to deviate from its static counterpart after the first re-balancing at $t = 100$. During the simulation, 11% of all operations have been switches, 76% were flips, and 13% have been shifts. Overall, the dynamic re-balancing reduced the simple chain's average loop probability by 4%.

- In contrast, the dynamic adjustment could not improve the informed chain as the necessary number of steps is already very small. As this observation will apply for each of the following experiments, we will not consider the dynamic adjustment for the informed chain from now on.

- Considering the case of $d = 5$, we observe that the sample means produced by the simple chain stabilize earlier than before. In addition, the loop rate of the simple chain drops to 85%.

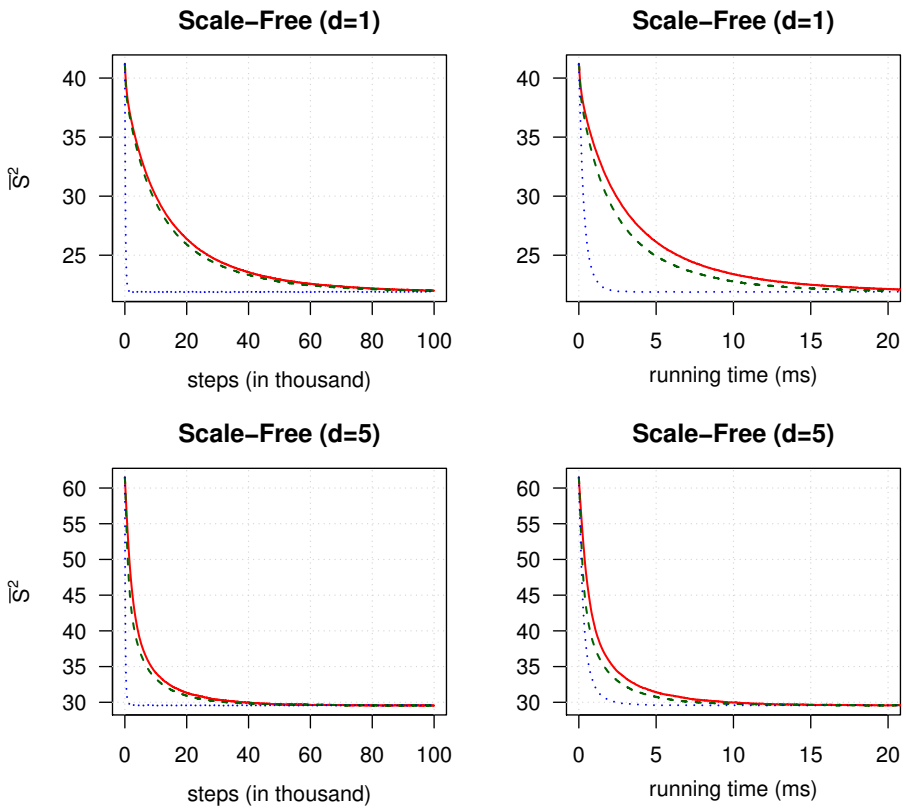

Figure 5.10: Sample means of the $\bar{S}^2$ metric for scale-free integer vectors. Red, solid line: static simple chain. Green, dashed line: dynamic simple chain. Blue, dotted line: static informed chain.

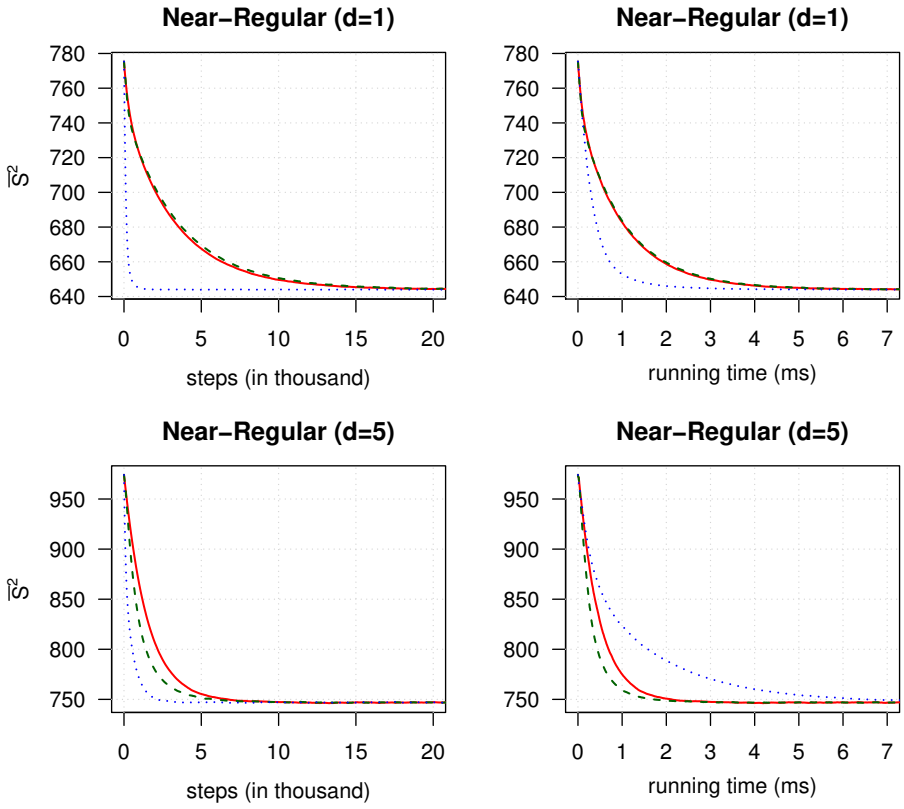

Figure 5.11: Sample means of the $\bar{S}^2$ metric for near-regular integer vectors. Red, solid line: static simple chain. Green, dashed line: dynamic simple chain. Blue, dotted line: static informed chain.

We conclude that the informed chain is properly suited for randomization of scale-free bipartite graphs, especially if the gap between the degree bounds is small.

Experiment 5.2 Next, we experimented with the degrees of near-regular bipartite graphs. For this purpose, we defined two integer vectors $\tilde{\mathbf{r}}$ and $\tilde{\mathbf{c}}$ of length $m = n = 100$ by setting $\tilde{r}_i := \tilde{c}_i := 50$ for each $i \in \{1, \ldots, 100\}$. As before, we added a constant d to each integer to produce the vectors of upper bounds $\bar{\mathbf{r}}$ and $\bar{\mathbf{c}}$. Fig. 5.11 shows the sample means of the $\bar{S}^2$ metric for $d = 1$ and $d = 5$.

- Considering $d = 1$, we observe that the informed chain is faster in terms of iterations but is in practice only fairly more efficient than the simple chain.
- This can be explained by the simple chain's rate of loop transitions, which is 82% and thus comparably small. In comparison, 24% of the informed chain's transitions have been loops.

- With dynamic probability adjustment, the simple chain selected switch operations with a probability of 25%, flip operations with 50% and shift operations with 25%, thereby effectively reducing the number of loops by 3%.

- Considering $d = 5$, we found that the simple chain is even more efficient than before, as its loop rate dropped to 59%. Although only 7% of the informed chain's transitions have been loops, the simple chain is superior in practice.

We conclude that the simple chain with dynamic adjustment is suited best for the randomization of near-regular bipartite graphs.

Experiment 5.3 In this experiment, we replaced the $\bar{S}^2$ metric by the other mentioned measures of nestedness (see Figs. D.8 to D.13 in Appendix D).

- The experiment confirms our previous observations. Although the sample means stabilize after a different number of steps, the qualitative results are mostly unchanged.

We conclude that our observations were not a side-effect of the metric used for convergence detection.

5.3.3 Sampling Application

In a final experiment, we demonstrate how our sampling methods can be applied to study properties of partially observed networks. In this experiment, we assumed that the observed "Darwin's Finches" network derived from Table 4.1 is part of some larger "true" network whose exact shape is unknown due to missing observations. To model this kind of uncertainty, we assumed that each island of the Galápagos archipelago may be inhabited by up to one finch species whose presence was not observed during the data collection. Symmetrically, we allowed each finch species to be present at one additional island. Mathematically speaking, we modeled the true "Darwin's Finches" network to be an unknown element of a set S that contains all bipartite graphs $G = (U, V, E)$,

(a) with $|U| = 13$ and $|V| = 17$ nodes,

(b) whose degrees may exceed those of the observed network by one, and

(c) which contain all edges of the observed network.

In this experiment, we approximated the expected structure of the true network by uniformly sampling from the set S.

Rejection Sampling Our methods can not directly be applied to uniformly sample from S as they do not incorporate requirement (c). However, as S is clearly a subset of $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$ with

$$\begin{aligned}
\tilde{\mathbf{r}} &= (14, 13, 14, 10, 12, 2, 10, 1, 10, 11, 6, 2, 17), \\
\bar{\mathbf{r}} &= (15, 14, 15, 11, 13, 3, 11, 2, 11, 12, 7, 3, 18), \\
\tilde{\mathbf{c}} &= (4, 4, 11, 10, 10, 8, 9, 10, 8, 9, 3, 10, 4, 7, 9, 3, 3), \\
\bar{\mathbf{c}} &= (5, 5, 12, 11, 11, 9, 10, 11, 9, 10, 4, 11, 5, 8, 10, 4, 4),
\end{aligned}$$

we can apply *rejection sampling*. By randomly sampling from $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$ and rejecting all samples that miss an observed edge, we assure that the non-rejected samples are uniformly distributed in S.

As we draw uniformly, we need to construct about $|\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})|/|S|$ random samples from $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$ to find a single sample from S. Unfortunately, as S is just a tiny subset of the much larger $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$, this strategy proved to be too inefficient to be of practical use. For that reason, we first transformed the sampling problem to an equivalent problem that can be solved more efficiently.

Problem Transformation By definition, a graph $G \in S$ contains all edges of the observed network. By removing these edges from G, we create a graph G' with reduced degrees. Being specific, as each node of G may posses at most one additional edge, the degrees of G' will be either zero or one. Consequently, we may replace the original sampling problem by the problem of uniform sampling from a set T that contains all bipartite graphs $G = (U, V, E)$,

a) with $|U| = 13$ and $|V| = 17$ nodes,

b) whose degrees are either zero or one, and

c) which do not contain edges that are present in the observed network.

In Fig. 5.12, we demonstrate the problem transformation in a small example. It is easy to see that the reduced sampling problem is equivalent to the original one, as we can easily transform an element of T into an element of S by adding the observed edges. Vice versa, an element of S can be converted into an element of T by removing all observed edges. Thus, there is a one-to-one correspondence between S and T. Consequently, we conclude that $|S| = |T|$.

Refined Sampling Strategy As T is a subset of the set $\Omega(\tilde{\mathbf{p}}, \bar{\mathbf{p}}, \tilde{\mathbf{q}}, \bar{\mathbf{q}})$ with

$$\tilde{\mathbf{p}} = [0]^{13}, \quad \bar{\mathbf{p}} = [1]^{13}, \quad \tilde{\mathbf{q}} = [0]^{17}, \quad \bar{\mathbf{q}} = [1]^{17},$$

we may produce uniform samples from T by uniformly sampling from $\Omega(\tilde{\mathbf{p}}, \bar{\mathbf{p}}, \tilde{\mathbf{q}}, \bar{\mathbf{q}})$ and rejecting all samples that contain an observed edge. After adding the observed

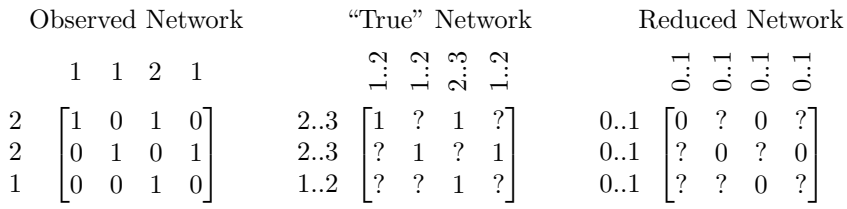

Figure 5.12: Illustration of the problem transformation. Let the network on the left be our observed network. We assume that the unknown "true" network may contain up to one additional edge per node, thus its degrees may exceed the observed ones by one. By removing the observed edges, we create a reduced network, whose degrees may be either zero or one.

edges to each non-rejected sample, we create a bipartite graph that is uniformly distributed in S.

To demonstrate that this refined strategy is superior to the basic rejection sampling strategy, we calculated the sizes of the sets $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$ and $\Omega(\tilde{\mathbf{p}}, \bar{\mathbf{p}}, \tilde{\mathbf{q}}, \bar{\mathbf{q}})$ by using an exact counting approach similar to that of Miller and Harrison [34]. In doing so, we found

$$|\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})| \approx 3.3 \times 10^{26} \quad \text{and} \quad |\Omega(\tilde{\mathbf{p}}, \bar{\mathbf{p}}, \tilde{\mathbf{q}}, \bar{\mathbf{q}})| \approx 1.2 \times 10^{14}.$$

Thus, the refined strategy outperforms the basic strategy by a factor of

$$\frac{|\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})| \cdot |T|}{|\Omega(\tilde{\mathbf{p}}, \bar{\mathbf{p}}, \tilde{\mathbf{q}}, \bar{\mathbf{q}})| \cdot |S|} = \frac{|\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})|}{|\Omega(\tilde{\mathbf{p}}, \bar{\mathbf{p}}, \tilde{\mathbf{q}}, \bar{\mathbf{q}})|} \approx \frac{3.3 \cdot 10^{26}}{1.2 \cdot 10^{14}} \approx 2.8 \cdot 10^{12}.$$

Experimental Setup To approximate the expected structure of the true network, we generated one million uniform samples from the set S and constructed the sampling histograms of the four metrics of nestedness. From these histograms, we approximated the expected structure of our assumed true network. To create a reference to which the structure of the true network can be compared with, we additionally calculated the corresponding sampling histograms for the set $\Omega(\tilde{\mathbf{r}}, \tilde{\mathbf{c}})$ of bipartite graphs with the same degrees as the observed "Darwin's Finches" network, and for the set $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$ of graphs with similar degrees as the true network but which do not need to contain the observed edges.

Results Figs. 5.13 and 5.14 show the calculated sampling histograms.

- Interestingly, the presence of the observed edges forces the true network to be denser than it would be expected from chance, as we approximated the expected number of edges of networks in $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$ to be 128.5. In contrast, we estimated the expected number of edges in the true network to be 130.5. The number of edges in the observed network is 122.

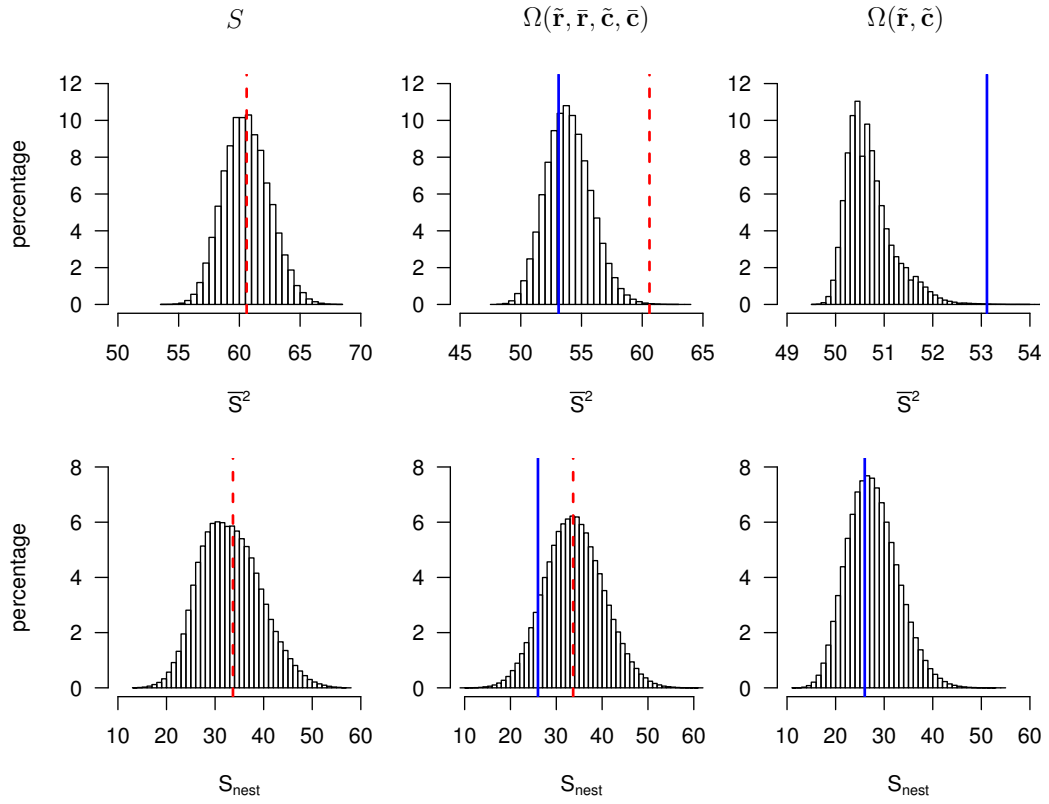

Figure 5.13: Sampling histograms obtained by the uniform sampling from three sets of graphs. Left column: the set S of potential "true" networks. Middle column: the set $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$. Right column: the set $\Omega(\tilde{\mathbf{r}}, \tilde{\mathbf{c}})$ of networks with the same degrees as "Darwin's Finches". While the blue, solid lines mark the structural properties of the observed network, the red, dashed lines mark the estimated expected values of our assumed "true" network.

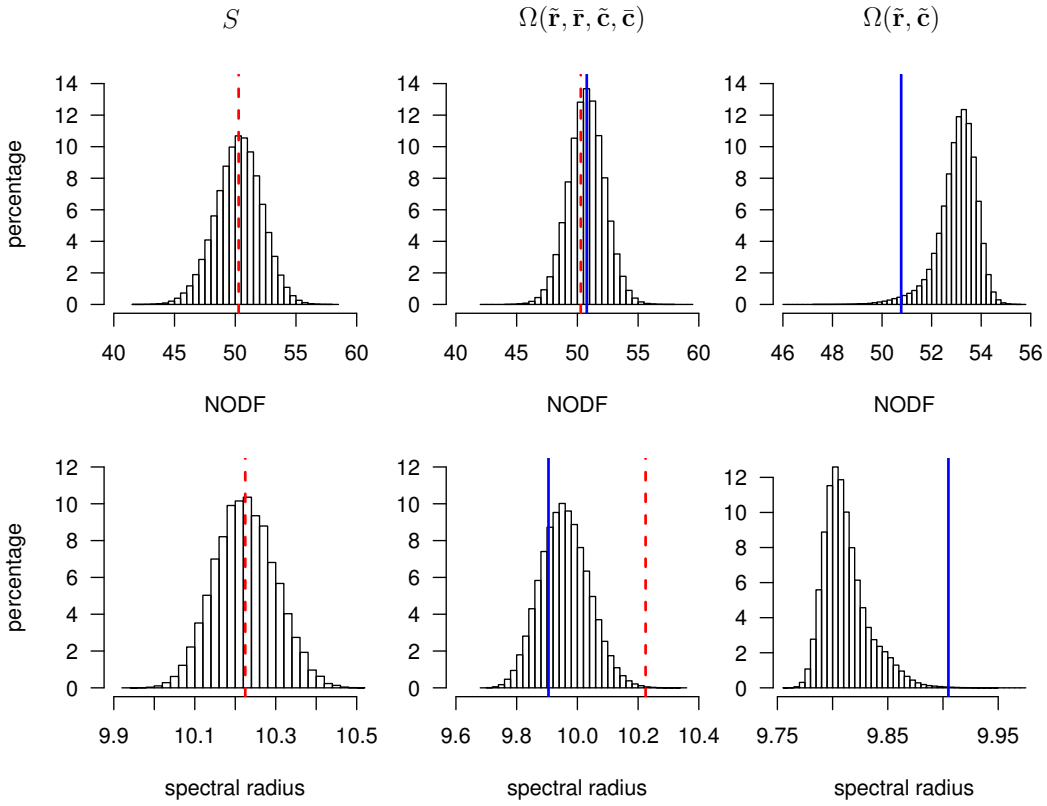

Figure 5.14: Sampling histograms obtained by the uniform sampling from three sets of graphs. Left column: the set S of potential "true" networks. Middle column: the set $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$. Right column: the set $\Omega(\tilde{\mathbf{r}}, \tilde{\mathbf{c}})$ of networks with the same degrees as "Darwin's Finches". While the blue, solid lines mark the structural properties of the observed network, the red, dashed lines mark the estimated expected values of our assumed "true" network.

- We observe that expected structural properties of the true network clearly exceed those of the observed network with respect to the $\bar{S}^2$, S_{nest}, and *spectral radius* metrics (see middle column). In contrast, the observed and the true network are structured very similarly with respect to NODF.

- To assess the influence of the observed edges, we compared the structure of the true network with the structure that would be expected if the observed edges were not required to be included in the true network (see middle column). In doing so, we found that the presence of the observed edges enforces the $\bar{S}^2$ and *spectral radius* metric of the true network to be significantly larger than it would be expected from chance alone. In contrast, the observed edges barely affect the S_{nest} and NODF metrics.

- In an analogous experiment, we evaluated the influence of the observed edges by comparing the structure of the observed network with that of null-networks with identical degrees but which do not need to contain the observed edges (see right column). In doing so, we found that the $\bar{S}^2$, NODF, and *spectral radius* metrics of the observed "Darwin's finches" notably differ from those of most random null-networks, whereas the values for S_{nest} are very similar.

We conclude that the structural properties of our imaginary true network significantly differ from the structure that would be expected if the observed edges were not required to exist.

5.4 An Optimal Realization Algorithm

In this section, we briefly move away from random sampling and consider the related problem of constructing an arbitrary bipartite graph whose degrees lie in prescribed intervals. This can be seen as a preceding sub-problem of the associated sampling problem, if no initial state is known in advance. An article based on this section is currently considered for publication in the Journal of Discrete Algorithms.

> S. Rechner. *An Optimal Realization Algorithm for Bipartite Graphs with Degrees in Prescribed Intervals.* arXiv preprint (2017). arXiv: `1708.05520v1 [cs.DS]` [27]

Problem Definition Let m and n be arbitrary integers and let $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$ be a four-tuple of integer vectors defined by

$$\tilde{\mathbf{r}} = (\tilde{r}_1, \tilde{r}_2, \ldots, \tilde{r}_m), \qquad \tilde{\mathbf{c}} = (\tilde{c}_1, \tilde{c}_2, \ldots, \tilde{c}_n),$$
$$\bar{\mathbf{r}} = (\bar{r}_1, \bar{r}_2, \ldots, \bar{r}_m), \qquad \bar{\mathbf{c}} = (\bar{c}_1, \bar{c}_2, \ldots, \bar{c}_n).$$

Then, $\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$ is the set of bipartite graphs whose degrees are bounded by the given integer vectors. The problem discussed in this section is how to construct an arbitrary bipartite graph $G \in \Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$.

Related Work The construction of bipartite graphs with prescribed degrees is a well-studied algorithmic problem with various applications in sciences. In the special case of $\tilde{\mathbf{r}} = \bar{\mathbf{r}}$ and $\tilde{\mathbf{c}} = \bar{\mathbf{c}}$, Ryser's algorithm [53] has been used for decades to construct a realization $G = (U, V, E)$ in $\mathcal{O}(|U| + |V| + |E|)$ time.

In contrast, the construction of bipartite graphs with bounded degrees attracted less attention. Although necessary and sufficient conditions for the existence of such graphs are well-known [82, 83, 84], no efficient realization algorithm has yet been suggested. We still discuss some approaches that can be used to find a valid realization.

One way to construct a realization $G = (U, V, E)$ is to compute an arbitrary (g, f)-factor in the complete bipartite graph K_{mn}, with $g := \tilde{\mathbf{r}}\tilde{\mathbf{c}}$ and $f := \bar{\mathbf{r}}\bar{\mathbf{c}}$ being integer vectors obtained by concatenating the vectors of lower and upper bounds. As a (g, f)-factor in an arbitrary graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ can be constructed in $\mathcal{O}(|\mathcal{V}|^3)$ time [85], this approach produces a realization in $\mathcal{O}((|U| + |V|)^3)$ time.

Another way of finding a realization is to construct an appropriate flow network $\mathcal{F} = (\mathcal{V}, \mathcal{A})$, and to calculate a maximal flow in $\mathcal{F}$ [82]. In our case, the flow network needs to have $|\mathcal{V}| = |U| + |V| + 2$ vertices and $|\mathcal{A}| = |U| \cdot |V| + |U| + |V|$ arcs. As calculating a maximal flow can be achieved in $\mathcal{O}(|\mathcal{V}| \cdot |\mathcal{A}|)$ time [86], this procedure gives a total running time of $\mathcal{O}(|U| \cdot |V| \cdot (|U| + |V|))$.

In contrast, we present a realization algorithm, whose running time is bounded from above by $\mathcal{O}(|U| + |V| + |E|)$. As we show that the bipartite graph produced by this algorithm is edge-minimal, this is asymptotically optimal.

Methodology Our algorithm is based on the following well-known theorem that shows under which conditions a four-tuple of integer vectors is realizable.

Theorem 5.11 (Fulkerson [82], Schocker [84])**.** *Let* $\tilde{\mathbf{r}} = (\tilde{r}_1, \tilde{r}_2, \ldots, \tilde{r}_m)$ *and* $\tilde{\mathbf{c}} = (\tilde{c}_1, \tilde{c}_1, \ldots, \tilde{c}_n)$ *be non-increasing integer vectors, and let* $\bar{\mathbf{r}} = (\bar{r}_1, \bar{r}_2, \ldots, \bar{r}_m)$ *and* $\bar{\mathbf{c}} = (\bar{c}_1, \bar{c}_2, \ldots, \bar{c}_n)$ *be integer vectors with* $\tilde{\mathbf{r}} \leq \bar{\mathbf{r}}$ *and* $\tilde{\mathbf{c}} \leq \bar{\mathbf{c}}$*. Then,* $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$ *is realizable if and only if* $\tilde{\mathbf{r}} \trianglelefteq \bar{\mathbf{c}}'$ *and* $\tilde{\mathbf{c}} \trianglelefteq \bar{\mathbf{r}}'$*.*

Our algorithm assumes that the four-tuple $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$ is realizable, which can be verified by Theorem 5.11 in $\mathcal{O}(m+n)$ time before the algorithm starts. The key idea is now to iteratively construct a bi-graphical pair of integer vectors $(\mathbf{r}, \mathbf{c})$ bounded by $\tilde{\mathbf{r}} \leq \mathbf{r} \leq \bar{\mathbf{r}}$ and $\tilde{\mathbf{c}} \leq \mathbf{c} \leq \bar{\mathbf{c}}$, which is afterwards realized via Ryser's algorithm.

For the sake of a simple presentation, we allow our algorithm to re-order the integer vectors. Consequently, the algorithm will produce a bipartite graph whose node labels will in general be permuted in a different order. We can easily re-order the nodes after the realization by keeping track of the original node labels during the execution of the algorithm.

Our algorithm assumes that the integer vectors $\tilde{\mathbf{r}}$ and $\tilde{\mathbf{c}}$ are already ordered non-increasingly, which can easily be arranged by sorting the pairs $(\tilde{\mathbf{r}}, \bar{\mathbf{r}})$ and $(\tilde{\mathbf{c}}, \bar{\mathbf{c}})$ in descending order by their first component. The algorithm is divided into two parts.

5.4.1 Phase One

As the four-tuple $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$ is realizable by assumption, there is a bipartite graph $G = (U, V, E)$ such that $\tilde{c}_j \leq \delta_G(v_j) \leq \bar{c}_j$ holds for each $v_j \in V$. As a consequence, there must be an integer vector $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ which describes the degrees of the vertex set V in G. In other words, there exists an integer vector $\mathbf{c}$ bounded by $\tilde{\mathbf{c}} \leq \mathbf{c} \leq \bar{\mathbf{c}}$ such that $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \mathbf{c}, \mathbf{c})$ is realizable. In its first phase (see Alg. 5.1), the algorithm constructs such an integer vector. For this purpose, $\mathbf{c}$ is initialized with $\tilde{\mathbf{c}}$. In a series of iterations, the algorithm identifies the right-most component c_i with $c_i < \bar{c}_i$ and increments the left-most component c_j identical to c_i. After a well-chosen number δ_1 of iterations, the algorithm returns the realizable four-tuple $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \mathbf{c}, \mathbf{c})$.

Algorithm 5.1: PHASE ONE

Input: realizable four-tuple $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$
Output: realizable four-tuple $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \mathbf{c}, \mathbf{c})$ with $\tilde{\mathbf{c}} \leq \mathbf{c} \leq \bar{\mathbf{c}}$

```
1  c ← c̃                                   // initialize c with lower bounds c̃
2  δ1 ← max{Σ^j_r̃ − Σ^j_c̃′ : 1 ≤ j ≤ m}      // calculate number of steps
3  i ← n                                   // right-most position such that c_i < c̄_i
4  for k = 1, 2, ..., δ1 do
5  |  while c_i = c̄_i do                   // proceed to next position with c_i < c̄_i
6  |  |  i ← i − 1
7  |  end
8  |  j ← min{ℓ : c_ℓ = c_i}               // identify left-most c_j with c_j = c_i
9  |  swap c̄_i and c̄_j
10 |  c_j ← c_j + 1
11 end
12 return (r̃, r̄, c, c)
```

Example Consider the following integer vectors.

$$\tilde{\mathbf{r}} = (4, 1, 0), \qquad \tilde{\mathbf{c}} = (2, 2, 0, 0, 0)$$
$$\bar{\mathbf{r}} = (4, 2, 3), \qquad \bar{\mathbf{c}} = (2, 3, 1, 2, 2).$$

By setting up at the corresponding conjugate sequences, we observe that the four-tuple $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$ is realizable via Theorem 5.11.

$$\begin{array}{llll} \tilde{\mathbf{r}} = (4,1,0) & \bar{\mathbf{r}}' = (3,3,2,1,0,\ldots) & \tilde{\mathbf{c}} = (2,2,0,0,0) & \bar{\mathbf{c}}' = (5,4,1,0,\ldots) \\ \Sigma_{\tilde{\mathbf{r}}} = (4,5,5) & \Sigma_{\bar{\mathbf{r}}'} = (3,6,8,9,9,\ldots) & \Sigma_{\tilde{\mathbf{c}}} = (2,4,4,4,4) & \Sigma_{\bar{\mathbf{c}}'} = (5,9,10,10,\ldots). \end{array}$$

The vector $\mathbf{c}$ is initialized with $\mathbf{c} := \tilde{\mathbf{c}} = (2, 2, 0, 0, 0)$. As

$$\tilde{\mathbf{c}}' = (2, 2, 0, \ldots) \quad \text{and} \quad \Sigma_{\tilde{\mathbf{c}}'} = (2, 4, 4, \ldots),$$

the number of steps of the outer loop is calculated by $\delta_1 := \max\{4-2, 5-4, 5-4\} = 2$.

1. The inner loop breaks with $i = 5$. The left-most component equal to $c_5 = 0$ is at position $j = 3$. Thus, the algorithm swaps $\bar{c}_5$ and $\bar{c}_3$ and increments c_3. We gain $\mathbf{c} = (2, 2, 1, 0, 0)$ and $\bar{\mathbf{c}} = (2, 3, 2, 2, 1)$.

2. The inner loop breaks again with $i = 5$. The left-most component equal to $c_5 = 0$ is now at position $j = 4$. Thus, the algorithm swaps $\bar{c}_5$ and $\bar{c}_4$ and increments c_4. We gain $\mathbf{c} = (2, 2, 1, 1, 0)$ and $\bar{\mathbf{c}} = (2, 3, 2, 1, 2)$.

The situation at the end of the first phase:

$$\begin{array}{llll} \tilde{\mathbf{r}} = (4, 1, 0) & \bar{\mathbf{r}} = (4, 2, 3) & \mathbf{c} = (2, 2, 1, 1, 0) & \mathbf{c}' = (4, 2, 0, \ldots) \\ \Sigma_{\tilde{\mathbf{r}}} = (4, 5, 5) & \Sigma_{\bar{\mathbf{r}}} = (4, 6, 9) & \Sigma_{\mathbf{c}} = (2, 4, 5, 6, 6) & \Sigma_{\mathbf{c}'} = (4, 6, 6, \ldots). \end{array}$$

We observe that $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \mathbf{c}, \mathbf{c})$ is realizable via Theorem 5.11.

Correctness The correctness of Alg. 5.1 follows from Lemmas 5.12 and 5.13. For the following theorems, let $\tilde{\mathbf{r}} = (\tilde{r}_1, \tilde{r}_2, \ldots, \tilde{r}_m)$ and $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ be non-increasing integer vectors and let $\bar{\mathbf{r}} = (\bar{r}_1, \bar{r}_2, \ldots, \bar{r}_m)$ and $\bar{\mathbf{c}} = (\bar{c}_1, \bar{c}_2, \ldots, \bar{c}_n)$ be integer vectors such that

(a) $\tilde{\mathbf{r}} \leq \bar{\mathbf{r}}$ and $\mathbf{c} \leq \bar{\mathbf{c}}$ hold,

(b) $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \mathbf{c}, \bar{\mathbf{c}})$ is realizable, and

(c) $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \mathbf{c}, \mathbf{c})$ is not realizable.

Lemma 5.12. *Let i be the right-most position such that $c_i < \bar{c}_i$, and let j be the left-most position with $c_j = c_i$. Let $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ and $\bar{\mathbf{a}} = (\bar{a}_1, \bar{a}_2, \ldots, \bar{a}_n)$ be integer vectors defined by*

$$a_k = \begin{cases} c_k + 1, & \text{if } k = j, \\ c_k, & \text{otherwise,} \end{cases} \qquad \text{and} \qquad \bar{a}_k = \begin{cases} \bar{c}_i, & \text{if } k = j, \\ \bar{c}_j, & \text{if } k = i, \\ \bar{c}_k, & \text{otherwise.} \end{cases}$$

Then, $\mathbf{a}$ is non-increasing, $\mathbf{a} \leq \bar{\mathbf{a}}$ holds, and $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \mathbf{a}, \bar{\mathbf{a}})$ is realizable.

Proof. As $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \mathbf{c}, \bar{\mathbf{c}})$ is realizable by assumption and $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \mathbf{c}, \mathbf{c})$ is not, $\mathbf{c}$ cannot be equal to $\bar{\mathbf{c}}$. Thus, there must be a position i such that $c_i < \bar{c}_i$. As $\mathbf{c}$ is non-increasing and j is chosen left-most with $c_j = c_i$, either $j = 1$ or $c_{j-1} > c_j$ must hold. In both cases, the integer vector $\mathbf{a}$ is non-increasing.

Next, we show that $\mathbf{a} \leq \bar{\mathbf{a}}$. As $c_i < \bar{c}_i$ is true by assumption, we may directly conclude that $a_j = c_j + 1 = c_i + 1 \leq \bar{c}_i = \bar{a}_j$. To see that $a_i \leq \bar{a}_i$ holds, consider two cases.

1. If $i = j$, the inequality $\bar{a}_i = \bar{c}_i \geq c_i + 1 = a_i$ holds as $c_i < \bar{c}_i$.

2. Otherwise, if $i \neq j$, we conclude that $\bar{a}_i = \bar{c}_j \geq c_j = c_i = a_i$.

Hence, $a_i \leq \bar{a}_i$ holds in both cases. Since $a_k = c_k$ and $\bar{a}_k = \bar{c}_k$ for all $k \neq i, j$, we conclude $\mathbf{a} \leq \bar{\mathbf{a}}$. It remains to show that $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \mathbf{a}, \bar{\mathbf{a}})$ is realizable. By Theorem 5.11, $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \mathbf{a}, \bar{\mathbf{a}})$ is realizable if and only if both $\tilde{\mathbf{r}} \trianglelefteq \bar{\mathbf{a}}'$ and $\mathbf{a} \trianglelefteq \bar{\mathbf{r}}'$ hold.

1. First, we show that $\tilde{\mathbf{r}} \trianglelefteq \bar{\mathbf{a}}'$ holds. As $\bar{\mathbf{a}}$ is a permutation of $\bar{\mathbf{c}}$, the sets

$$\{\ell \in \{1, 2, \ldots, n\} \colon \bar{a}_i \geq k\} \qquad \text{and} \qquad \{\ell \in \{1, 2, \ldots, n\} \colon \bar{c}_i \geq k\}$$

   are identical for all k. Thus, $\bar{\mathbf{a}}'$ equals $\bar{\mathbf{c}}'$. As $\tilde{\mathbf{r}} \trianglelefteq \bar{\mathbf{c}}'$ holds due to the realizability of $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \mathbf{c}, \bar{\mathbf{c}})$, the dominance relation $\tilde{\mathbf{r}} \trianglelefteq \bar{\mathbf{a}}'$ holds, too.

2. To show that $\mathbf{a} \trianglelefteq \bar{\mathbf{r}}'$ is true, assume the contrary. Thus, there is a right-most position k with $\Sigma^k_{\mathbf{a}} > \Sigma^k_{\bar{\mathbf{r}}'}$. Since $\Sigma^k_{\mathbf{c}} \leq \Sigma^k_{\bar{\mathbf{r}}'}$ must hold as $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \mathbf{c}, \bar{\mathbf{c}})$ is realizable and $\mathbf{a}$ differs from $\mathbf{c}$ only at position j, we conclude $\Sigma^k_{\mathbf{c}} = \Sigma^k_{\bar{\mathbf{r}}'}$ and $j \leq k$.

   Next, we show $i > k$. For this purpose, assume the contrary and let $i \leq k$. Since i is chosen right-most, it follows that $c_\ell = \bar{c}_\ell$ for each ℓ in range $i < \ell \leq n$. Thus, increasing an arbitrary c_ℓ in range $i < \ell \leq n$ by a positive amount would violate $\mathbf{c} \leq \bar{\mathbf{c}}$ whereas increasing an arbitrary c_ℓ in range $1 \leq \ell \leq i$ violates the realizability of $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \mathbf{c}, \bar{\mathbf{c}})$. Hence, $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \mathbf{c}, \bar{\mathbf{c}})$ can only be realizable if $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \mathbf{c}, \mathbf{c})$ is, which contradicts our assumption. Thus, $i > k$.

   Since j was chosen left-most with $c_j = c_i$, it follows from $j \leq k$ and $k < i$ that $c_j = \ldots = c_k = c_{k+1} = \ldots = c_i$. Since k is right-most and $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \mathbf{c}, \bar{\mathbf{c}})$ is realizable, we conclude that $\Sigma^{k+1}_{\mathbf{c}} < \Sigma^{k+1}_{\bar{\mathbf{r}}'}$ holds and thus, $\bar{r}'_{k+1} > c_{k+1}$.

   As $\bar{\mathbf{r}}'$ is non-increasing by definition, we further conclude

$$\bar{r}'_k \geq \bar{r}'_{k+1} > c_{k+1} = c_k.$$

   Consequently, $\Sigma^k_{\mathbf{c}} = \Sigma^k_{\bar{\mathbf{r}}'}$ holds if and only if $\Sigma^{k-1}_{\mathbf{c}} > \Sigma^{k-1}_{\bar{\mathbf{r}}'}$ holds. If $k > 1$, this contradicts the realizability of $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \mathbf{c}, \bar{\mathbf{c}})$. If $k = 1$, it is plainly wrong as $\Sigma^0_{\mathbf{c}} = \Sigma^0_{\bar{\mathbf{r}}'} = 0$. As a consequence, $\mathbf{a} \trianglelefteq \bar{\mathbf{r}}'$ must hold and thus, $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \mathbf{a}, \bar{\mathbf{a}})$ is realizable. □

Lemma 5.13. *Let $\mathbf{c}^{(k)}$ and $\bar{\mathbf{c}}^{(k)}$ be the state of the integer vectors $\mathbf{c}$ and $\bar{\mathbf{c}}$ after exactly k iterations of the outer loop in Alg. 5.1. Let $\delta_1 := \max\{\Sigma^j_{\tilde{\mathbf{r}}} - \Sigma^j_{\bar{\mathbf{c}}'} \colon 1 \leq j \leq m\}$. Then, $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \mathbf{c}^{(k)}, \mathbf{c}^{(k)})$ is realizable for $k \geq \delta_1$ and is not realizable if $k < \delta_1$.*

Proof. By Lemma 5.12, $\mathbf{c}^{(k)}$ is non-increasing and $\mathbf{c}^{(k)} \leq \bar{\mathbf{c}}^{(k)}$ holds. Hence, by Theorem 5.11, the four-tuple $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \mathbf{c}^{(k)}, \mathbf{c}^{(k)})$ is realizable if and only if $\tilde{\mathbf{r}} \trianglelefteq (\mathbf{c}^{(k)})'$ and $\mathbf{c}^{(k)} \leq \bar{\mathbf{r}}'$ hold. Whereas the latter domination relation is ensured by Lemma 5.12, the condition $\tilde{\mathbf{r}} \trianglelefteq (\mathbf{c}^{(k)})'$ will not hold if $k < \delta_1$.

To see why this is true, consider the incrementation of an arbitrary component $c^{(k)}_j$ to $c^{(k+1)}_j := c^{(k)}_j + 1$ and consider how this affects the conjugate sequence $\left(\mathbf{c}^{(k+1)}\right)'$. Whereas $(c^{(k+1)})'_\ell = (c^{(k)})'_\ell$ for each $\ell \neq c^{(k+1)}_j$, the component $(c^{(k+1)})'_\ell$ will be of

value $(c^{(k+1)})'_\ell = (c^{(k)})'_\ell + 1$ for $\ell = c_j^{(k+1)}$. As a consequence, the partial sums $\Sigma_{\mathbf{c}^{(k+1)}}$ have the value

$$\Sigma^i_{(\mathbf{c}^{(k+1)})'} = \begin{cases} \Sigma^i_{(\mathbf{c}^{(k)})'} + 1, & \text{if } c_j^{(k)} < i \leq m, \\ \Sigma^i_{(\mathbf{c}^{(k)})'}, & \text{otherwise.} \end{cases}$$

Now let p be an arbitrary position such that $\Sigma^p_{\tilde{\mathbf{r}}} > \Sigma^p_{\tilde{\mathbf{c}}'}$ holds at the beginning of the first phase. Since $\Sigma^p_{\tilde{\mathbf{r}}}$ stays constant, the inequality $\Sigma^p_{\tilde{\mathbf{r}}} \leq \Sigma^p_{(\mathbf{c}^{(k)})'}$ will be established as soon as a number of $k \geq \Sigma^p_{\tilde{\mathbf{r}}} - \Sigma^p_{\mathbf{c}'}$ components of value $c_j < p$ have been incremented. As Alg. 5.1 chooses c_j as small as possible, the domination relation $\tilde{\mathbf{r}} \trianglelefteq (\mathbf{c}^{(k)})'$ will hold after exactly $\delta_1 = \max\{\Sigma^j_{\tilde{\mathbf{r}}} - \Sigma^j_{\tilde{\mathbf{c}}'} : 1 \leq j \leq m\}$ iterations. □

Running Time Determining the quantity δ_1 requires a running time of $\mathcal{O}(m)$. The outer loop of Alg. 5.1 runs exactly δ_1 steps. In each step, the algorithm has to determine the position $j := \min\{\ell : c_\ell = c_i\}$ of the left-most occurrence of a component equal to c_i. This can be achieved in constant time if we use a pre-computed lookup-table which is updated after each incrementation. Fortunately, each increment operations only requires a table-update at the positions c_i and $c_i + 1$, which can be executed in constant time. Thus, disregarding the inner loop, each step of the outer loop is executed in constant time. In contrast, the inner loop may need linear time. However, the variable i can be decreased at most n times during the whole process. Thus, the running time of the outer loop is $\mathcal{O}(n + \delta_1)$. In summary, since $\delta_1 \leq \Sigma^m_{\tilde{\mathbf{r}}}$, the running time of the first phase can be described by $\mathcal{O}(m + n + \Sigma^m_{\tilde{\mathbf{r}}})$.

5.4.2 Phase Two

After the first phase has stopped, $\tilde{\mathbf{r}}$ and $\mathbf{c}$ are non-increasing integer vectors, and the four tuple $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \mathbf{c}, \mathbf{c})$ is realizable by Lemma 5.12 and 5.13. Hence, there is a bipartite graph $G = (U, V, E)$ with $\delta_G(v_j) = c_j$ for j in range $1 \leq j \leq n$ and $\tilde{r}_i \leq \delta_G(u_i) \leq \bar{r}_i$ for i in range $1 \leq i \leq m$. If we switch the roles of vertex sets U and V, we gain an instance of the realization problem in which the four-tuple $(\mathbf{c}, \mathbf{c}, \tilde{\mathbf{r}}, \bar{\mathbf{r}})$ is realizable, $\mathbf{c}$ and $\tilde{\mathbf{r}}$ are non-increasing, and $\tilde{\mathbf{r}} \leq \bar{\mathbf{r}}$ holds. Thus, we can re-apply the first phase to the modified instance to construct a suitable integer vector $\mathbf{r}$ such that $(\mathbf{c}, \mathbf{c}, \mathbf{r}, \mathbf{r})$ is realizable. After switching back the roles of U and V, we gain a bi-graphical pair $(\mathbf{r}, \mathbf{c})$ of integer vectors. As the single difference to the first phase, the number of steps of the outer for loop can be calculated more efficiently. Afterwards, a realization is constructed by Ryser's algorithm. Alg. 5.2 shows the second phase of the realization algorithm.

Example We start where the first phase stopped.

$$\begin{array}{cccc} \tilde{\mathbf{r}} = (4,1,0) & \bar{\mathbf{r}} = (4,2,3) & \mathbf{c} = (2,2,1,1,0) & \mathbf{c}' = (4,2,0) \\ \Sigma_{\tilde{\mathbf{r}}} = (4,5,5) & \Sigma_{\bar{\mathbf{r}}} = (4,6,9) & \Sigma_{\mathbf{c}} = (2,4,5,6,6) & \Sigma_{\mathbf{c}'} = (4,6,6). \end{array}$$

Algorithm 5.2: PHASE TWO

Input: realizable four-tuple $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \mathbf{c}, \mathbf{c})$
Output: bi-graphical pair $(\mathbf{r}, \mathbf{c})$

```
1  r ← r̃                                   // initialize r with lower bounds r̃
2  δ2 ← Σ_c^n − Σ_r̃^m                      // calculate number of steps
3  i ← m                                   // right-most position such that r̃_i < r̄_i
4  for k = 1, 2, ..., δ2 do
5  |  while r_i = r̄_i do                   // proceed to next position with r_i < r̄_i
6  |  |  i ← i − 1
7  |  end
8  |  j ← min{ℓ: r_ℓ = r_i}                // identify left-most r_j with r_j = r_i
9  |  swap r̄_i and r̄_j
10 |  r_j ← r_j + 1
11 end
12 return (r, c)
```

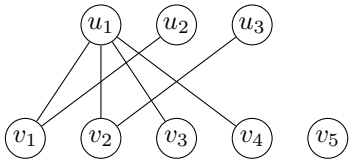

Figure 5.15: Realization of the sequence pair $\mathbf{r} = (4, 1, 1)$ and $\mathbf{c} = (2, 2, 1, 1, 0)$.

The number of iterations is determined by $\delta_2 \leftarrow 6 - 5 = 1$.

1. The inner loop breaks with $i = 3$. The left-most component equal to $r_3 = 0$ is at position $j = 3$. Thus, the algorithm switches $\bar{r}_3$ with itself and increments r_3. We gain $\mathbf{r} = (4, 1, 1)$ and $\bar{\mathbf{r}} = (4, 2, 3)$.

The situation at the end of the second phase:

$$\begin{array}{lll} \mathbf{r} = (4,1,1) & \mathbf{c} = (2,2,1,1,0) & \mathbf{c}' = (4,2,0) \\ \Sigma_{\mathbf{r}} = (4,5,6) & \Sigma_{\mathbf{c}} = (2,4,5,6,6) & \Sigma_{\mathbf{c}'} = (4,6,6) \end{array}$$

We verify by Theorem 4.1 that $(\mathbf{r}, \mathbf{c})$ is bi-graphical. Fig. 5.15 shows a realization.

Correctness The correctness of the second phase can be shown very similarly to phase one and follows directly from the following theorem.

Lemma 5.14. *Let $\mathbf{r}^{(k)}$ and $\bar{\mathbf{r}}^{(k)}$ be the state of the integer vectors $\mathbf{r}$ and $\bar{\mathbf{r}}$ after exactly k iterations of the outer loop in Alg. 5.2. Then, $(\mathbf{r}^{(k)}, \mathbf{r}^{(k)}, \mathbf{c}, \mathbf{c})$ is realizable if and only if $k = \delta_2 = \Sigma_{\mathbf{c}}^n - \Sigma_{\tilde{\mathbf{r}}}^m$.*

Proof. By Theorem 5.11, the four-tuple $(\mathbf{r}^{(k)}, \mathbf{r}^{(k)}, \mathbf{c}, \mathbf{c})$ is realizable if and only if $\mathbf{r}^{(k)} \leq \mathbf{c}'$ and $\mathbf{c} \trianglelefteq (\mathbf{r}^{(k)})'$ hold. In particular, $\Sigma^m_{\mathbf{r}^{(k)}} = \Sigma^n_{\mathbf{c}}$ must hold. Since $\Sigma^m_{\tilde{\mathbf{r}}^{(k)}} \neq \Sigma^n_{\mathbf{c}}$ for $k \neq \Sigma^n_{\mathbf{c}} - \Sigma^m_{\tilde{\mathbf{r}}}$, the four-tuple $(\tilde{\mathbf{r}}^{(k)}, \tilde{\mathbf{r}}^{(k)}, \mathbf{c}, \mathbf{c})$ cannot be realizable if $k \neq \delta_2$. On the other hand, as $(\mathbf{r}^{(k)}, \bar{\mathbf{r}}^{(k)}, \mathbf{c}, \mathbf{c})$ is realizable and thus $\mathbf{r}^{(k)} \trianglelefteq \mathbf{c}'$ holds after each iteration by Lemma 5.12, the four-tuple $(\mathbf{r}^{(k)}, \mathbf{r}^{(k)}, \mathbf{c}, \mathbf{c})$ will be realizable as soon as $\Sigma^m_{\mathbf{r}^{(k)}} = \Sigma^n_{\mathbf{c}}$ holds, which is true after exactly δ_2 iterations. □

Running Time By similar arguments as before, the running time of the second phase can be described by $\mathcal{O}(m + n + \Sigma^n_{\mathbf{c}})$. As Ryser's algorithm produces a realization $G = (U, V, E)$ of the bi-graphical pair $(\mathbf{r}, \mathbf{c})$ in $\mathcal{O}(|U| + |V| + |E|)$ time and $|E| = \Sigma^n_{\mathbf{c}} \geq \Sigma^m_{\tilde{\mathbf{r}}}$, the total running time of the realization algorithm is $\mathcal{O}(|U| + |V| + |E|)$.

5.4.3 Edge-Minimality

Lemma 5.15. *The bipartite graph produced by our algorithm is edge-minimal.*

Proof. Let $(\mathbf{r}, \mathbf{c})$ be a bi-graphical pair of integer vectors associated to an arbitrary realization $G = (U, V, E)$ of the four-tuple $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$. As $\tilde{\mathbf{c}} \leq \mathbf{c}$ must hold, the number of edges $|E|$ is bounded from below by $|E| \geq \Sigma^n_{\tilde{\mathbf{c}}}$. In addition, if $(\mathbf{r}, \mathbf{c})$ is bi-graphical, the four-tuple $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \mathbf{c}, \mathbf{c})$ must be realizable. Hence, the inequalities

$$\Sigma^j_{\tilde{\mathbf{r}}} \leq \Sigma^j_{\mathbf{c}'} = \Sigma^j_{\tilde{\mathbf{c}}'} + x_j$$

must hold for each j in range $1 \leq j \leq m$. Thus, $x_j = \Sigma^j_{\tilde{\mathbf{r}}} - \Sigma^j_{\tilde{\mathbf{c}}'}$ is the minimal number of edges that G needs to possess *in addition to* the $\Sigma^n_{\tilde{\mathbf{c}}}$ edges, so that the inequality $\Sigma^j_{\tilde{\mathbf{r}}} \leq \Sigma^j_{\mathbf{c}'}$ can hold. Hence, the total number of edges is bounded from below by

$$|E| \geq \Sigma^n_{\tilde{\mathbf{c}}} + \max\{x_j \colon 1 \leq j \leq n\} = \Sigma^n_{\tilde{\mathbf{c}}} + \delta_1.$$

Since our algorithm produces a bipartite graph $G = (U, V, E)$ with exactly $|E| = \Sigma^n_{\tilde{\mathbf{c}}} + \delta_1$ edges, G is edge-minimal. □

Remark Our algorithm can easily be used to construct *edge-maximal* realizations. For this purpose, consider an arbitrary edge-minimal realization $G = (U, V, E)$ of the four-tuple $(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$. The associated *complement graph* $G^* = (U, V, E^*)$ is defined by $E^* := (U \times V) \setminus E$. By construction, the graph G is edge-minimal if and only if G^* is edge-maximal. In addition, it follows from definition that the degrees of G^* are bounded from below and above by the *complementary four-tuple* $(\tilde{\mathbf{r}}^*, \bar{\mathbf{r}}^*, \tilde{\mathbf{c}}^*, \bar{\mathbf{c}}^*)$ with

$$\begin{aligned}
\tilde{r}^* &= (n - \bar{r}_1, n - \bar{r}_2, \ldots, n - \bar{r}_m) \\
\bar{r}^* &= (n - \tilde{r}_1, n - \tilde{r}_2, \ldots, n - \tilde{r}_m) \\
\tilde{c}^* &= (m - \bar{c}_1, m - \bar{c}_2, \ldots, m - \bar{c}_n) \\
\bar{c}^* &= (m - \tilde{c}_1, m - \tilde{c}_2, \ldots, m - \tilde{c}_n).
\end{aligned}$$

Thus, we can find an edge-maximal realization by first determining an edge-minimal realization of the complementary four-tuple $(\tilde{\mathbf{r}}^*, \bar{\mathbf{r}}^*, \tilde{\mathbf{c}}^*, \bar{\mathbf{c}}^*)$ and creating the associated complement graph.

Summary In this section, we gave a description of an algorithm that constructs an edge-minimal bipartite graph $G = (U, V, E)$ whose degrees lie in prescribed intervals. Subsequently, we showed that this algorithm has a running time of $\mathcal{O}(|U|+|V|+|E|)$. Since we cannot hope to construct a bipartite graph in sub-linear time, our algorithm is asymptotically optimal.

5.5 Summary

In this chapter, we introduced two Markov chains designed for the uniform sampling of bipartite graphs whose degrees lie in prescribed intervals. By a series of theorems, we proved the ergodicity of both Markov chains and showed that their stationary distribution is uniform.

Afterwards, we experimentally assessed the efficiency of both Markov chains and found that the informed chain outperforms the simple chain when the prescribed degrees are nearly scale-free, whereas the simple chain is superior for near-regular vertex degrees.

Subsequently, we demonstrated how our sampling algorithms can be used to study the properties of partially observed networks and found that the existence of a few unobserved edges may crucially influence the structure of an ecological network.

We closed this chapter by presenting an algorithm that constructs a bipartite graph $G = (U, V, E)$ whose degrees lie in prescribed intervals. This can be seen as a subproblem of the associated sampling problem if no initial state is known in advance. After proving the correctness of our algorithm, we showed that its running time is bounded by $\mathcal{O}(|U| + |V| + |E|)$ and is thus optimal.

Chapter 6

Perfect and Near-Perfect Matchings in Bipartite Graphs

In this chapter, we briefly focus on another classical sampling application. To demonstrate *marathon*'s wide range of application, we discuss the uniform sampling of *perfect matchings* of a bipartite graph $G = (U, V, E)$.

Definition 6.1 (perfect matching)**.** *Let $G = (U, V, E)$ be an arbitrary bipartite graph. A subset $M \subseteq E$ of edges is called* matching *if and only if no pair of its elements share a common vertex. The matching is called* perfect *if $2|M| = |U \cup V|$. We call a vertex* matched *if it is part of some matching edge. Otherwise, it is called* unmatched*.*

As a bipartite graph $G = (U, V, E)$ cannot possess a perfect matching if U and V are of different size, we focus on bipartite graphs with $|U| = |V|$ and define $n := |U|$ to be the number of vertices in each vertex set. In addition, we denote by $m := |E|$ the number of edges in G.

The construction of an arbitrary perfect matching is a fundamental graph algorithm that has been studied for decades. Hopcroft and Karp [87] showed that a perfect matching of a bipartite graph can be constructed in $\mathcal{O}(m\sqrt{n})$ time. If G is dense, a randomized algorithm based on fast matrix multiplication can be used to find a perfect matching in $\mathcal{O}(n^\omega)$ time, where ω is the exponent of the best known matrix multiplication algorithm [88]. Recently, Borradaile et al. presented an algorithm to find a perfect matching in $\mathcal{O}(n \log^3 n)$ time if G is planar [89].

In this chapter, we focus on the uniform sampling of a perfect matching from the set $\mathcal{M}(G)$ of all perfect matchings of a bipartite graph G. Originating from statistical physics, this sampling problem has been extensively studied due to its close connection to the associated counting problem. The number $|\mathcal{M}(G)|$ of perfect matchings of a bipartite graph G is equivalent to the *permanent* of G's bi-adjacency matrix. As shown by Valiant [90], the computation of the permanent is a #P-complete problem. Thus, an efficient algorithm for the counting of a bipartite graph's perfect matchings is unlikely to exist. As there is a close relationship between exact counting and

sampling [9], an efficient algorithm for the exact sampling of perfect matchings in bipartite graphs is therefore out of reach.

In 2004, Jerrum et al. [19] presented a fully polynomial-time randomized approximation scheme (FPRAS) for the number of perfect matchings of a bipartite graph. This algorithm is based on the rapidly mixing JSV Markov chain that will be specified soon. Although the approximation algorithm is of high theoretical relevance, it is fairly complicated and may still be too inefficient to be used in practical applications.

In the recent past, research interest shifted to the uniform sampling of perfect matchings in general, non-bipartite graphs. Interestingly, it is unclear whether an FPRAS for the number of perfect matchings may exist in this scenario. Recent work showed that the JSV chain is rapidly mixing for several classes of non-bipartite graphs, and *not* to be rapidly mixing for others [91]. Similar results were obtained for related Markov chains [92]. Thus, the classical sampling problem is still of high interest.

Parts of this chapter are based on joint work with Annabell Berger.

> A. Berger and S. Rechner. *Broder's Chain Is Not Rapidly Mixing.* arXiv preprint (2014). arXiv: `1404.4249v1 [cs.DM]` [25]
>
> S. Rechner and A. Berger. *Marathon: An open source software library for the analysis of Markov-Chain Monte Carlo algorithms.* PLOS ONE **11** (2016). DOI: `10.1371/journal.pone.0147935` [20]

6.1 Markov Chains

As an exact sampling algorithm of polynomial running time is unlikely to exist, several MCMC sampling algorithms have been proposed, from which we describe two important ones. Both MCMC algorithms use the set $\mathcal{N}(G)$ of *near-perfect matchings* in G as auxiliary states.

Definition 6.2 (near-perfect matching)**.** *Let $G = (U, V, E)$ be a bipartite graph such that $|U| = |V|$. A matching $M \subseteq E$ is called* near-perfect *if and only if $|M| = |U|-1$.*

Thus, the state space $\Omega(G)$ of the following Markov chains is the set of perfect and near-perfect matchings of a bipartite graph G, i.e.

$$\Omega(G) := \mathcal{M}(G) \cup \mathcal{N}(G).$$

Using rejection sampling, both algorithms can be used for the uniform sampling of perfect matchings. For this purpose, a Markov chain is simulated to produce a random sample $x \in \Omega(G)$. If x is a near-perfect matching, the sampling algorithm is restarted until a perfect matching is found. The main challenge of this approach is to guarantee that near-perfect matchings do not occur too often.

6.1.1 Broder's Chain

In 1986, Broder [24] introduced the following Markov chain and showed its ergodicity.

Definition 6.3 (Broder's chain)**.** *Let $M \in \Omega(G)$.*

1. *Choose an edge $e = \{u, v\}$ uniformly at random from E.*
2. *(a) If M is perfect and $e \in M$, set $M' := M \setminus \{e\}$.*
   (b) Otherwise, if u and v are unmatched, set $M' := M \cup \{e\}$.
   (c) Otherwise, if u is unmatched and v is matched, then determine the matching edge $e' = \{w, v\} \in M$ and set $M' := (M \setminus \{e'\}) \cup \{e\}$.
   (d) Otherwise, if u is matched and v is unmatched, then determine the matching edge $e' = \{u, z\} \in M$ and set $M' := (M \setminus \{e'\}) \cup \{e\}$.
   (e) Otherwise, set $M' := M$.
3. *If the maximal number of steps has been reached, return M'.*
4. *Set $M \leftarrow M'$ and go to Step 1.*

By Corollary 2.4, the stationary distribution of Broder's chain is uniform as the transition probability $p(x, y) = m^{-1} = p(y, x)$ of transforming a state x into $y \neq x$ via a single operation is symmetric. Jerrum and Sinclair [18] showed that Broder's chain is rapidly mixing when the ratio of near-perfect matchings $\mathcal{N}(G)$ and perfect matchings $\mathcal{M}(G)$ in G can be bounded from above by a polynomial function of the number m of edges. Being precise, the total mixing time can be bounded from above by

$$\tau_{\max}(\varepsilon) \leq 16^2 m^2 \left(\frac{|\mathcal{N}(G)|}{|\mathcal{M}(G)|} \right)^4 \ln(|\Omega| \cdot \varepsilon^{-1}). \tag{6.1}$$

As a special case, Jerrum and Sinclair showed that the ratio $|\mathcal{N}(G)|/|\mathcal{M}(G)|$ is bounded polynomially if the minimal vertex degree of a bipartite graph is at least $n/2$. Then, the total mixing time of Broder's chain is bounded from above by [93]

$$\mathcal{O}\left(n^7 \ln(|\Omega(G)| \cdot \varepsilon^{-1})\right).$$

Independently from the question of whether or not Broder's chain is rapidly mixing, an additional problem arises when the ratio $|\mathcal{N}(G)|/|\mathcal{M}(G)|$ is an exponential function on the number n of vertices. In such cases, the associated sampling algorithm is expected to need an exponential number of trials to find a single perfect matching.

6.1.2 JSV chain

To address this problem, Jerrum, Sinclair and Vigoda [19] presented an MCMC algorithm that has been proven to find a perfect matching after an expected polynomial number of trials. The sampling algorithm is based on a carefully chosen weight function $w\colon \Omega(G) \to \mathbb{R}^+$ that will be specified soon. The transition rules of this Markov chain can be stated as follows.

Definition 6.4 (JSV chain)**.** *Let $M \in \Omega(G)$.*

1. *If M is perfect, choose a matching edge $e = \{u, v\} \in M$ uniformly at random and set $M' := M \setminus \{e\}$.*
2. *If M is near-perfect, determine the unmatched vertices $u \in U$ and $v \in V$ and choose a vertex z uniformly at random from $U \cup V$.*
   - *2a) If z is unmatched and $\{u, v\} \in E$, then set $M' := M \cup \{\{u, v\}\}$.*
   - *2b) Otherwise, if z is matched and $\{u, z\} \in E$, then determine the matching edge $\{x, z\} \in M$ and set $M' := (M \setminus \{\{x, z\}\}) \cup \{\{u, z\}\}$.*
   - *2c) Otherwise, if z is matched and $\{z, v\} \in E$, then determine the matching edge $\{z, y\} \in M$ and set $M' := (M \setminus \{\{z, y\}\}) \cup \{\{z, v\}\}$.*
   - *2d) Otherwise, set $M' := M$.*
3. *Select a real number $u \in [0, 1)$ uniformly at random.*
4. *If $u < w(M')/w(M)$, set $M \leftarrow M'$.*
5. *If the maximal number of steps has been reached, return M.*
6. *Go to Step 1.*

By construction, the proposal probability $\kappa(x, y)$ of moving between a perfect matching and an adjacent near-perfect matching is

$$\kappa(x, y) = n^{-1} = \kappa(y, x).$$

In contrast, the proposal probability $\kappa(x, y)$ of moving between adjacent near-perfect matchings is

$$\kappa(x, y) = (2n)^{-1} = \kappa(y, x).$$

As the proposal probability is symmetric, Theorem 2.3 shows that the stationary distribution of the JSV chain is proportional to the weight function w. Jerrum et al. suggested to define w by

$$w(x) = \begin{cases} 1, & x \in \mathcal{M}(G) \\ |\mathcal{M}(G)|/|\mathcal{N}_{u,v}(G)|, & x \in \mathcal{N}_{u,v}(G), \end{cases}$$

where $\mathcal{N}_{u,v}(G) \subseteq \mathcal{N}(G)$ is the subset of near-perfect matchings where $u \in U$ and $v \in V$ are unmatched. Consequently, the accumulated weight of all perfect matchings is

$$w(\mathcal{M}(G)) := \sum_{x \in \mathcal{M}(G)} w(x) = |\mathcal{M}(G)|.$$

Furthermore, for all $(u, v) \in U \times V$,

$$w(\mathcal{N}_{u,v}(G)) := \sum_{x \in \mathcal{N}_{u,v}(G)} w(x) = \begin{cases} 0, & \text{if } \mathcal{N}_{u,v}(G) = \emptyset, \\ \frac{|\mathcal{N}_{u,v}(G)| \cdot |\mathcal{M}(G)|}{|\mathcal{N}_{u,v}(G)|} = |\mathcal{M}(G)|, & \text{else.} \end{cases}$$

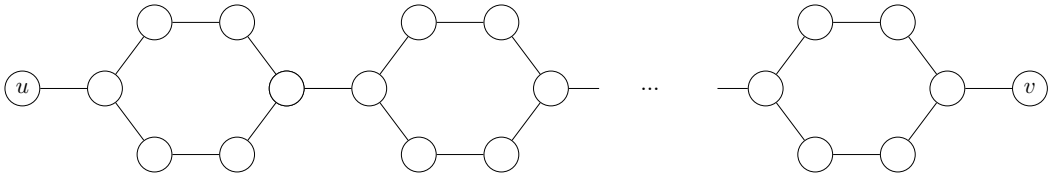

Figure 6.1: Hexagon graph class.

Thus, the state space $\Omega(G)$ is partitioned into at most $n^2 + 1$ subsets of identical positive weight. As the JSV chain produces samples from each of these subsets equiprobably, the sampling algorithm will produce a perfect matching with a probability of at least $(n^2 + 1)^{-1}$. Hence, the sampling algorithm is expected to produce a perfect matching after a polynomial number of trials.

Bezáková et al. [94] showed that, knowing the values of $|\mathcal{M}(G)|$ and $|\mathcal{N}_{u,v}(G)|$, the total mixing time $\tau_{\max}(\varepsilon)$ of the JSV chain can be bounded from above by

$$\mathcal{O}(n^4 \ln(\pi_{min} \cdot \varepsilon)^{-1}), \tag{6.2}$$

where $\pi_{\min} := \min_{x \in \Omega(G)}\{\pi(x)\}$. Thus, the JSV chain is a rapidly mixing Markov chain for the set of all bipartite graphs. Unfortunately, $|\mathcal{M}(G)|$ and $|\mathcal{N}_{u,v}(G)|$ are usually not known in practice. For that reason, Jerrum et al. gave a description of a procedure that approximates these quantities in polynomial time [19]. In the following experiments though, we calculated the exact values by a recursive counting approach based on dynamic programming.

6.2 Experiments on Mixing Time

In this chapter, we experimentally analyse the total mixing time of Broder's chain and the JSV chain. In contrast to the Markov chains discussed in the previous chapters, Broder's chain is known *not* to be rapidly mixing for several classes of bipartite graphs. In particular, Berger and Rechner [25] showed that the total mixing time of Broder's chain cannot be bounded from above by a polynomial function on n and ε^{-1} for several graph classes, from which we will briefly address two.

Hexagon Graphs Jerrum et al. [19] presented the class of *hexagon graphs*. Such graphs are composite of so-called hexagons, i.e. cycles of length six. The k-th hexagon graph H_k is a bipartite graph consisting of k hexagons $c_1, c_2, \ldots, c_k$. Each pair of consecutive hexagons is connected via a single edge (see Fig. 6.1). In addition, two exposed vertices u and v are connected to the left- and rightmost hexagon. By construction, each hexagon graph possesses exactly one perfect matching. In contrast, the number of near-perfect matchings in H_k is bounded from below by

$$|\mathcal{N}(H_k)| > |\mathcal{N}_{u,v}(H_k)| = 2^k.$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 6.2: Bi-adjacency matrices of T_1 and T_2.

As the total number of vertices in H_k is $2n = 6k + 2$, the number $|\mathcal{N}(H_k)|$ of near-perfect matchings exceeds

$$\begin{aligned} |\mathcal{N}_{u,v}(H_k)| &= 2^{(n-1)/3} \\ &= 2^{n/3} \cdot 2^{-1/3} = (2^{1/3})^n \cdot 2^{-1/3} \\ &\approx 0.79 \cdot 1.26^n. \end{aligned}$$

Hence, the ratio $|\mathcal{N}(G)|/|\mathcal{M}(G)|$ is an exponential function on n. As shown by Berger and Rechner [25], Broder's chain is not rapidly mixing for the hexagon graph class.

Odd Triangle Graphs Introduced by Berger and Rechner [25], the k-th *odd triangle graph* $T_k = (U, V, E)$ is a bipartite graph whose vertex sets contain $n = 2k+1$ vertices, and whose edge set is defined by

$$E := \{\{u_i, v_j\} \colon i + j \leq n + 1\}.$$

By construction, the bi-adjacency matrix of an odd triangle graph is a square triangular matrix, which gave the graph class its name. Fig 6.2 shows the bi-adjacency matrices of some small odd triangle graphs.

It is straight-forward to see that each odd triangle graph possesses exactly one perfect matching. The following lemma shows that the number $|\mathcal{N}(T_k)|$ of near-perfect matchings in T_k is an exponential function on the number n of nodes.

Lemma 6.1. *Let $T_k = (U, V, E)$ be the k-th odd triangle graph and let $n = 2k + 1$ the number of nodes in each vertex set. Then, $|\mathcal{N}(T_k)| \geq 2^{n-2}$.*

Proof. Consider the set $\mathcal{N}_{u_n,v_n}(T_k)$ of near-perfect matchings with $u_n \in U$ and $v_n \in V$ being unmatched. The number $|\mathcal{N}_{u_n,v_n}(T_k)|$ is equal to the number $|\mathcal{M}(T'_k)|$ of perfect matchings of a bipartite graph $T'_k = (U', V', E')$ with

$$\begin{aligned} U' &:= U \setminus \{u_n\}, \\ V' &:= V \setminus \{v_n\}, \\ E' &:= \{\{u, v\} \in E \colon u \neq u_n \wedge v \neq v_n\}. \end{aligned}$$

The bi-adjacency matrix of T'_k is the $(n-1)\times(n-1)$ sub-matrix created by omitting the last row and the last column from that of T_k. For example, the bi-adjacency matrix of T'_2 is

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}.$$

We can construct a perfect matching in T'_k by iteratively selecting a positive matrix entry in row $i = n-1, n-2, \ldots, 1$. For $i = n-1$, there are two options to select a positive matrix entry. After fixing the associated edge as a matching edge, there are two ways left to choose a positive entry in row $n-2$. Iterating this argument, there are 2^{n-2} ways to construct a perfect matching in T'_k. Consequently, we conclude $|\mathcal{N}(T_k)| \geq |\mathcal{N}_{u_n,v_n}(T_k)| = 2^{n-2}$. □

A more refined argument shows that the total number $|\mathcal{N}(T_k)|$ of near-perfect matchings in T_k equals $2^n - 1$. Following from Lemma 6.1, the ratio $|\mathcal{N}(T_k)|/|\mathcal{M}(T_k)|$ is an exponential function on n. Berger and Rechner [25] showed that Broder's chain is not rapidly mixing for the class of odd triangle graphs.

6.2.1 Total Mixing Time

We started our experiments by assessing the total mixing time of both Markov chains for the classes of hexagon and odd triangle graphs. For this purpose, we constructed the associated state graphs of several members of each class. By fixing ε to a constant of $\varepsilon = 0.01$ and calculating the associated total mixing time $\tau_{\max}(\varepsilon)$, we experimentally determined how the total mixing time depends on the number n of vertices. From our observations, we derived a function $f\colon \mathbb{N} \to \mathbb{N}$ that describes how the total mixing time depends on n. For this purpose, we evaluated three models.

1. First, we assumed the total mixing time to be an exponential function on n, i.e.

$$\begin{aligned} & f(n) = e^{c \cdot n^k} \\ \Leftrightarrow \quad & \ln f(n) = c \cdot n^k \\ \Leftrightarrow \quad & \ln \ln f(n) = \ln c + k \cdot \ln n. \end{aligned}$$

2. Next, we assumed that the total mixing time is a polynomial on n, i.e.

$$\begin{aligned} & f(n) = c \cdot n^k \\ \Leftrightarrow \quad & \ln f(n) = \ln c + k \cdot \ln n. \end{aligned}$$

3. Finally, the total mixing time may be a poly-logarithmic function on n, i.e.

$$\begin{aligned} & f(n) = c \cdot (\ln n)^k \\ \Leftrightarrow \quad & \ln f(n) = \ln c + k \cdot \ln \ln n. \end{aligned}$$

Similar as in Section 4.3, we used a two-step procedure to determine which model is suited best to describe our observations.

1. First, we transformed the data according to each model by taking the natural logarithms of n and $\tau_{\max}(\varepsilon)$. If the transformed data points appear to relate linearly, we conclude that the associated model fits well to our data.

2. In the second step, we used a conjugate-gradient method to estimate the model parameters c and k that minimize the associated sum of squared errors. For this purpose, we applied the general-purpose optimization function `optim` available in the standard R software framework. In doing so, we could quantitatively assess which model is suited best to describe the relation between n and the associated total mixing time.

Experiment 6.1 We started our experiments by testing our methodology on the total mixing time of Broder's chain. As the hexagon and odd triangle graphs are known to possess an exponential total mixing time, we expect that our experiment should clearly suggest the exponential model. Figs. 6.3 and 6.4 show how the total mixing time relates to the size n of the vertex sets of hexagon and odd triangle graphs.

- For both graph classes, we observe that the exponential model describes our observations best. The data suggests that the total mixing time of both graph classes is an exponential function on n. This coincides with the theoretical results presented by Berger and Rechner [25].

- We estimated the model parameters $c \approx 2.12$, $k \approx 0.47$ for the hexagon graphs, and $c \approx 1.71$, $k \approx 0.68$ for the class of odd triangle graphs. For the constant of $\varepsilon = 0.01$, we may thus describe the total mixing time of Broder's chain as a function $f(n) \approx 8.33^{n^{0.47}}$ in case of the hexagon graphs, and as $f(n) \approx 5.53^{n^{0.68}}$ for the odd triangle graph class.

Experiment 6.2 Next, we assessed the total mixing time of the JSV chain on the classes of hexagon and odd triangle graphs. In contrast to Broder's chain, the JSV chain is known to be rapidly mixing for all bipartite graphs. Thus, we expect that our experiments clearly reject the exponential model. Figs. 6.5 and 6.6 show the results of this experiment.

- Calculating the sum of squared errors, we observe that the polynomial model fits best to the total mixing time of the JSV chain for both classes of bipartite graphs. This agrees with our expectations.

- We estimated the model parameters to be $c \approx 0.36$, $k \approx 2.92$ in case of the hexagon graphs, and $c \approx 0.84$, $k \approx 2.66$ for the odd-triangle graphs. Our experiments show that the upper bound provided by Bezáková et al. (see Eq. 6.2) is not sharp for both graph classes considered in this experiment.

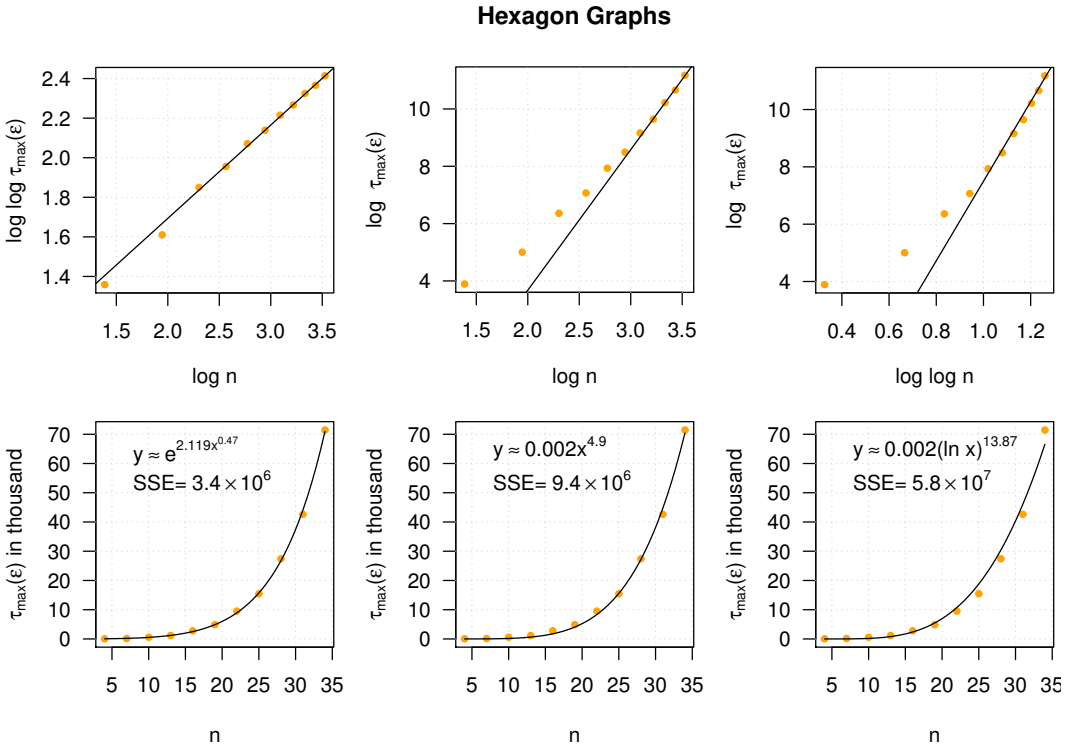

Figure 6.3: Relationship between Broder's chain's total mixing time $\tau_{\max}(\varepsilon)$ and the size n of hexagon graphs. Left column: exponential model. Middle column: polynomial model. Right column: poly-logarithmic model. The black lines indicate the estimated fit. ($\varepsilon = 0.01$)

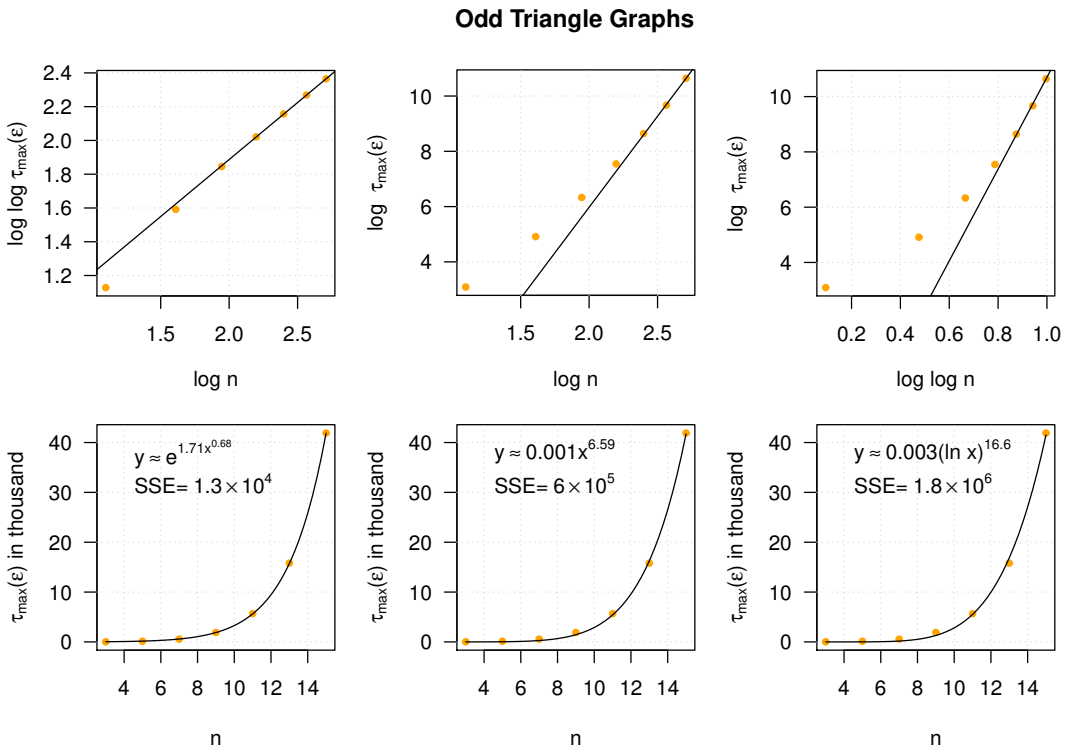

Figure 6.4: Relationship between Broder's chain's total mixing time $\tau_{\max}(\varepsilon)$ and the size n of odd triangle graphs. Left column: exponential model. Middle column: polynomial model. Right column: poly-logarithmic model. The black lines indicate the estimated fit. ($\varepsilon = 0.01$)

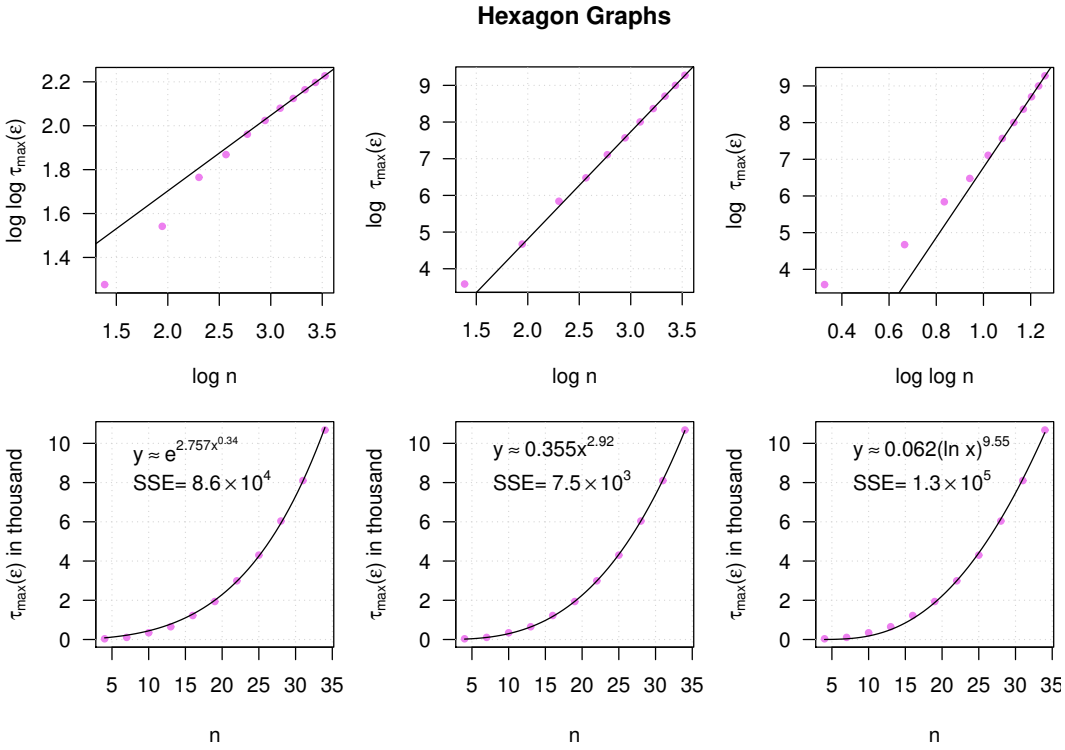

Figure 6.5: Relationship between the JSV chain's total mixing time $\tau_{\max}(\varepsilon)$ and the size n of hexagon graphs. Left column: exponential model. Middle column: polynomial model. Right column: poly-logarithmic model. The black lines indicate the estimated fit. ($\varepsilon = 0.01$)

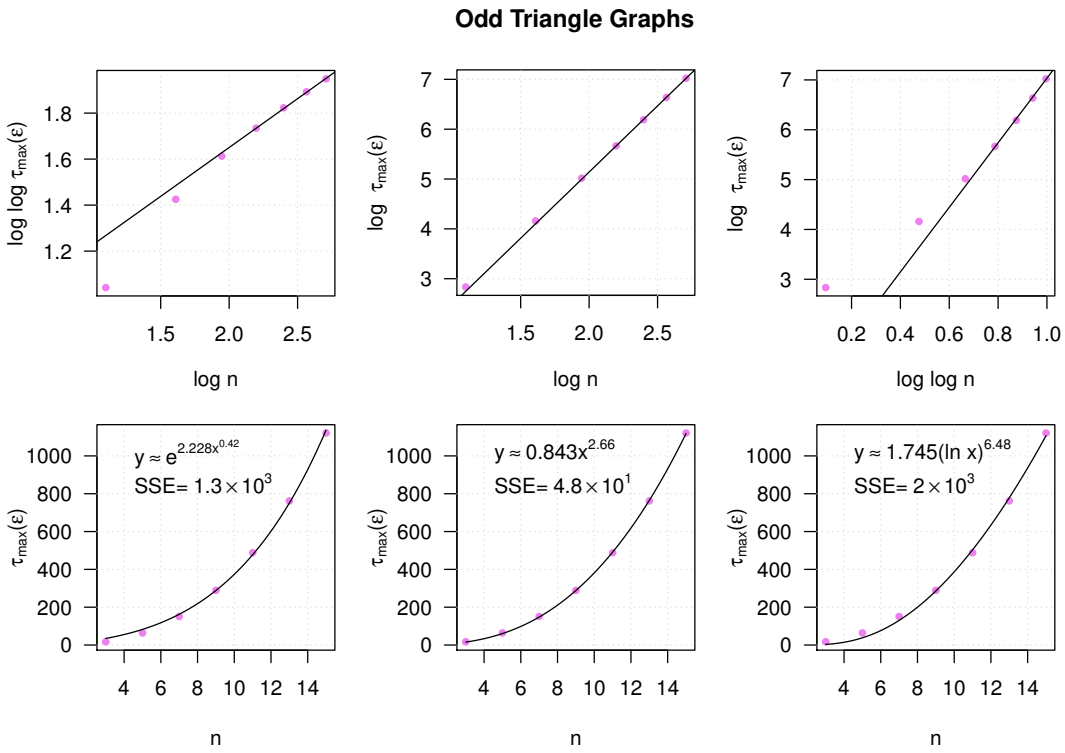

Figure 6.6: Relationship between the JSV chain's total mixing time $\tau_{\max}(\varepsilon)$ and the size n of odd triangle graphs. Left column: exponential model. Middle column: polynomial model. Right column: poly-logarithmic model. The black lines indicate the estimated fit. ($\varepsilon = 0.01$)

Our experiments provide a proof of concept that we can apply our methodology to experimentally assess the asymptotic behavior of the total mixing time. In particular, we demonstrated that our approach can correctly separate an exponential from a polynomial growth.

6.2.2 Influence of Initial State

In the next set of experiments, we assessed the influence of the initial state on the empirical mixing time $\bar{\tau}_s(\varepsilon)$ (introduced in Sec. 2.3). As the empirical mixing time does not require the construction of a state graph, we can process bipartite graphs for which the construction of state graphs is infeasible.

Auxiliary Function The empirical mixing time $\bar{\tau}_s(\varepsilon)$ requires an auxiliary function $f\colon \Omega \to \mathbb{R}$. As we saw before that the Hamming distance metric worked well in Chapter 4, we used a related metric here. Interpreting a perfect or near-perfect matching as a bipartite graph on $2n$ nodes, the Hamming distance between the associated bi-adjacency matrices is equivalent to the size

$$|M_1 \triangle M_2| := (M_1 \setminus M_2) \cup (M_2 \setminus M_1)$$

of the symmetric difference of both matchings. Denoting by $s \in \Omega(G)$ the initial state of each Markov chain, we define $f(x) = |x \triangle s|$ as our auxiliary function.

Experiment 6.3 In a first experiment, we approximated the empirical mixing time of Broder's chain while using the unique perfect matching of each graph as the initial state. To determine the empirical mixing time, we constructed the limiting distribution η of the auxiliary function f from $N = 10^6$ random samples that have been generated by an unbiased sampling method based on the exact counting of perfect and near-perfect matchings. To approximate the t-step distribution functions $q_s^{(t)}$, we constructed $N = 10^6$ random samples by simulating t steps of Broder's chain. Similar to the previous experiment, we evaluated whether the exponential, polynomial, or the poly-logarithmic model describes our observations best. Figs. 6.7 and 6.8 show the results of this experiment.

- Considering the class of hexagon graphs, we observe that the exponential model clearly fits worst to the data. Interestingly, our observations disagree with the outcome of our previous experiment, where we found the associated total mixing time to be an exponential function on n. We will further study this deviation in the following experiment.
- In contrast, the empirical mixing time of the odd triangle graphs is described best by the exponential model. We estimated the model parameter $c \approx 1.39, k \approx 0.72$. This is close to our previous approximation of the associated total mixing time function. We conclude that Broder's chain requires an exponential number of steps when starting from the perfect matching of an odd triangle graph.

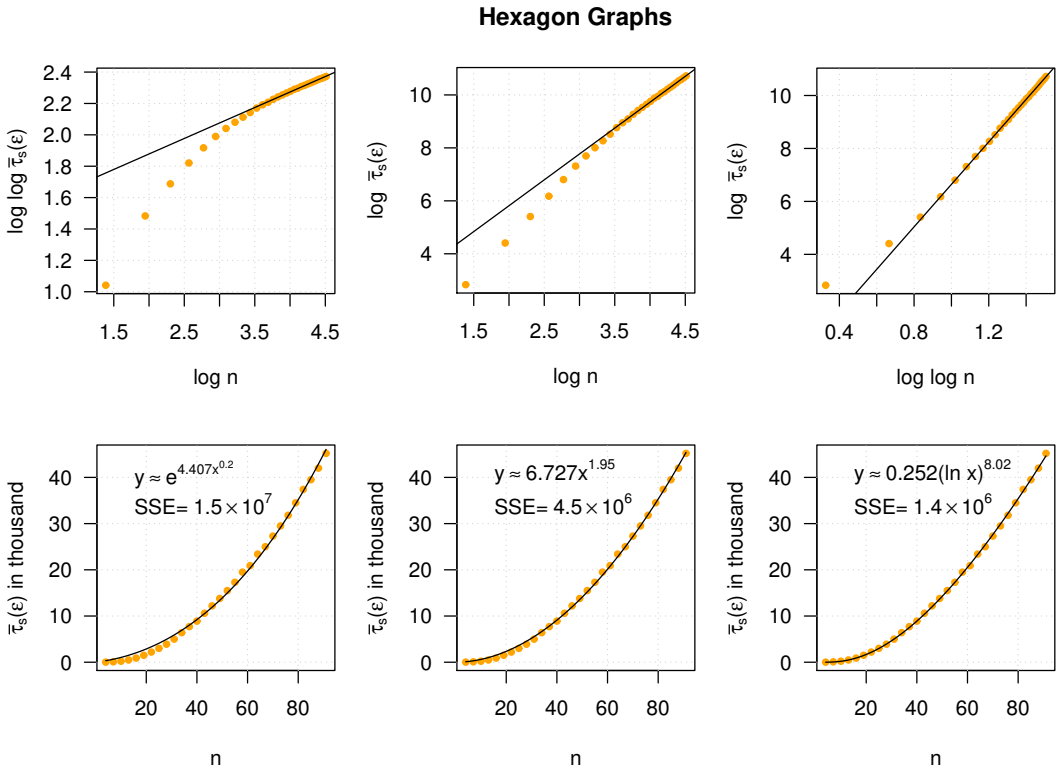

Figure 6.7: Relationship between the number n and the empirical mixing time $\bar{\tau}_s(\varepsilon)$ of Broder's chain on hexagon graphs. Left column: exponential model. Middle column: polynomial model. Right column: poly-logarithmic model. The black lines indicate the estimated fit. ($N = 10^6$, $\varepsilon = 0.01$, $s \in \mathcal{M}(G)$)

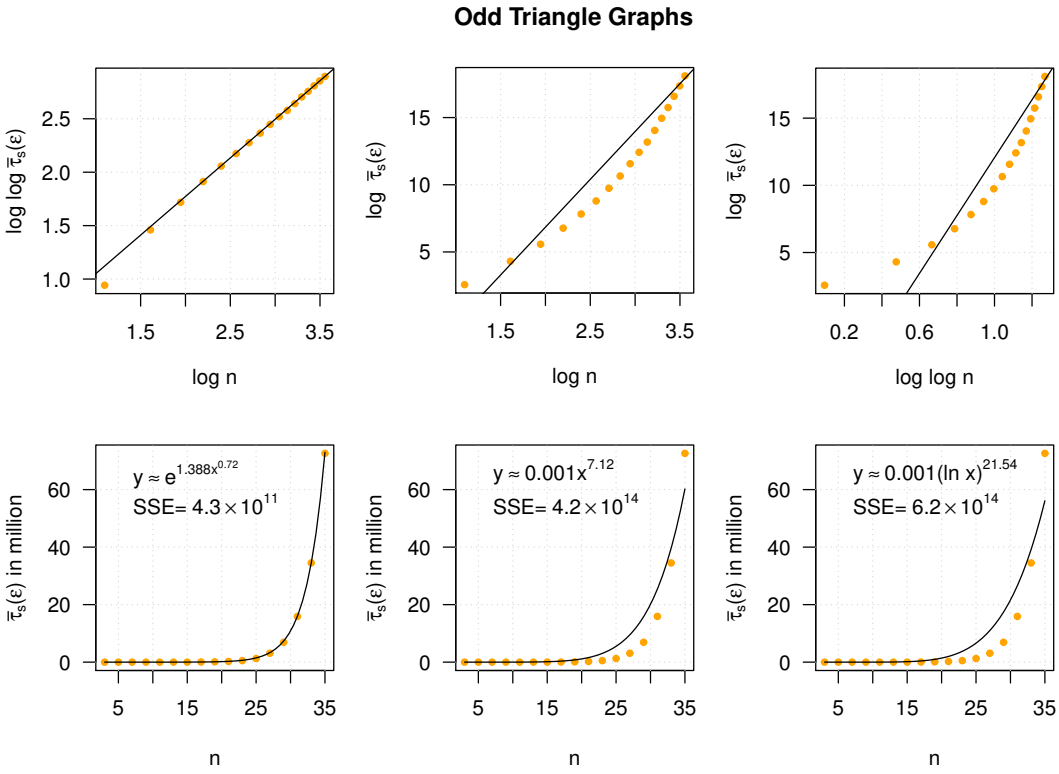

Figure 6.8: Relationship between the number n and the empirical mixing time $\bar{\tau}_s(\varepsilon)$ of Broder's chain on odd triangle graphs. Left column: exponential model. Middle column: polynomial model. Right column: poly-logarithmic model. The black lines indicate the estimated fit. ($N = 10^6$, $\varepsilon = 0.01$, $s \in \mathcal{M}(G)$)

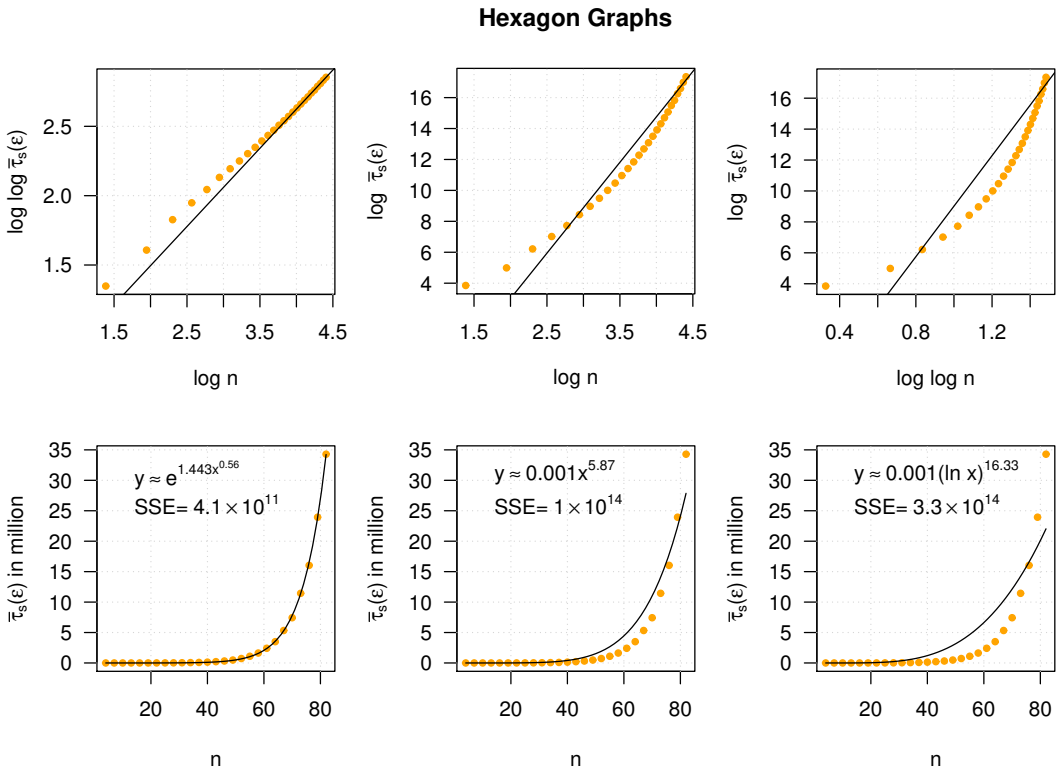

Figure 6.9: Relationship between the number n and the empirical mixing time $\bar{\tau}_s(\varepsilon)$ of Broder's chain on hexagon graphs. Left column: exponential model. Middle column: polynomial model. Right column: poly-logarithmic model. The black lines indicate the estimated fit. ($N = 10^6$, $\varepsilon = 0.01$, $s \in \mathcal{N}_{u,v}(G)$)

Experiment 6.4 Next, we evaluated the effect of the initial state s on the empirical mixing time $\bar{\tau}_s(\varepsilon)$ of Broder's chain. For this purpose, we focused on the class of hexagon graphs. Instead of using a perfect matching as initial state, we used one of the 2^k near-perfect matchings in which the two exposed vertices u and v are unmatched. Fig. 6.9 shows the results of this experiment.

- Now, we observe that the exponential model fits well to our observations. Thus, our experiments suggests that Broder's chain requires an exponential number of steps when starting from a near-perfect matching $s \in \mathcal{N}_{u,v}$.

Our experiments highlight the importance of the initial state s. In general graphs however, it is hard to tell which state is suited best as an initial state. The question on how to choose a good initial state is an interesting topic for further theoretical studies.

6.2.3 Quality of Bounding Techniques

In Sec. 4.3, we calculated lower and upper bounds on the total mixing time of the Markov chains for the uniform sampling of bipartite graphs with fixed vertex degrees. In doing so, we found that the lower and upper spectral bound are very close to the total mixing time, while the congestion bound exceeds the total mixing time by a large factor. In this set of experiments, we assessed whether we could reproduce these observations with Broder's and the JSV chain.

For this purpose, we processed the first 11 hexagon and the first seven odd triangle graphs. For each bipartite graph, we constructed the associated state graph of both Markov chains, from which we calculated the following properties

- the total mixing time $\tau_{\max}(\varepsilon)$ with $\varepsilon = 0.01$,
- its lower and upper spectral bound via Ineq. 2.6 and 2.7, and
- the canonical path congestion bound via Ineq. 2.10.

For the latter one, we implemented Jerrum and Sinclair's [18] path construction scheme, which was used to derive the upper bound of Ineq. 6.1. By applying the path construction scheme to specific state graphs, we assessed how precise the upper congestion bound could be if full information about state graphs were given. We quantified the quality of the lower or upper bounds by the quotient of each bound and the associated total mixing time $\tau_{\max}(\varepsilon)$. Fig. 6.10 shows how the quality of the bounds depend on the size n of the bipartite graphs.

- We observe very similar results as before in Chapter 4. In particular, we confirm that the lower and upper spectral bound is very tight to the total mixing time, while the quality of the congestion bound exceeds the total mixing time by multiple orders of magnitude.
- Considering Broder's chain, the data suggests that quality of the congestion bound decreases almost exponentially. In contrast, the congestion bound decreases more slowly in case of the JSV chain.

Our observations qualitatively agree with those of Chapter 4. Thus, we conclude that the canonical path method may not lead to sharp bounds on the total mixing time, even if full information about state graphs was available.

6.2.4 Induced Subgraphs

As the size of the sample space Ω is typically an exponential function on the input length, the construction of state graphs is infeasible in nearly all practical applications. Unfortunately, as we require a state graph for the calculation of the total mixing time and its lower and upper spectral bounds, we can rarely apply these

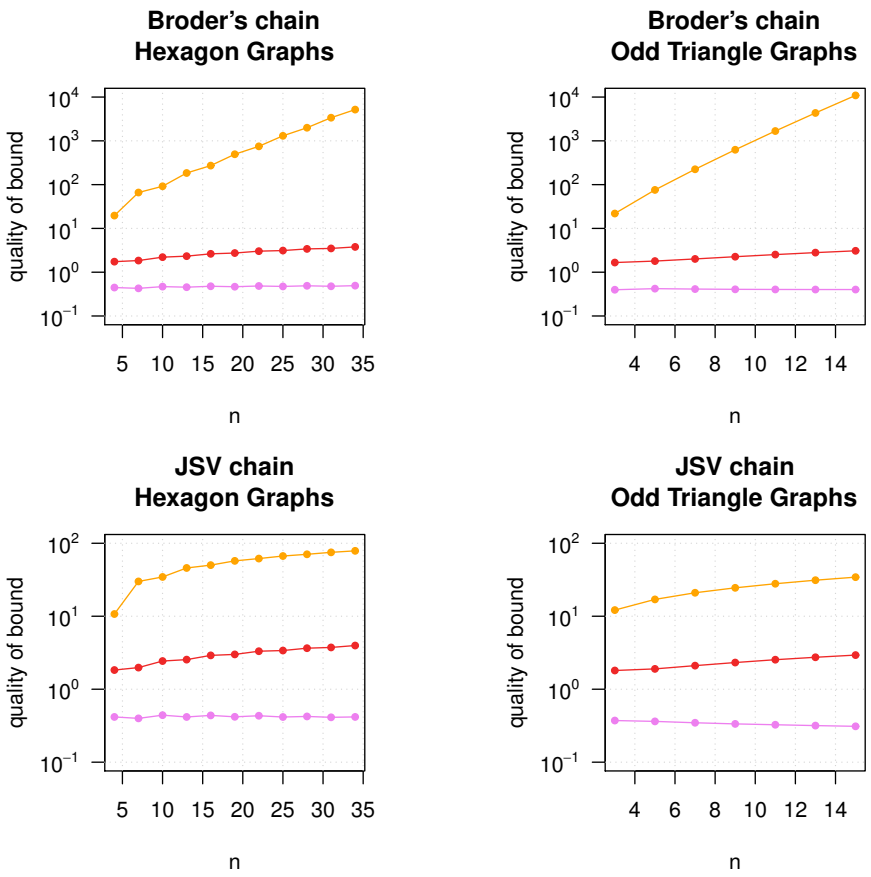

Figure 6.10: Quotient of congestion bound (orange), upper spectral bound (red), and lower spectral bound (violet) with total mixing time. ($\varepsilon = 0.01$.)

techniques in practical sampling applications. In such cases however, we can approximate the empirical mixing time as a lower bound on the mixing time $\tau_s(\varepsilon)$ of the initial state s.

In this section, we present an alternative method to determine a lower bound on the total mixing time of a Markov chain when the construction of a state graph is infeasible. It is based on the following theorem.

Theorem 6.2 (interlacing theorem (Corollary 2.2 from Ref. [95]))**.** *Let A be a real symmetric $n \times n$ matrix with eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$ and let B be a principal sub-matrix of A. Then, the eigenvalues $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_k$ of B* interlace *those of A, i.e.*

$$\forall i \in \{1, 2, \ldots, k\} \colon \lambda_i \geq \mu_i \geq \lambda_{n-k+i}.$$

Corollary 6.3. *Let P be the transition matrix of an ergodic Markov chain and let P' be a principal sub-matrix of P. Then, the total mixing time $\tau_{\max}(\varepsilon)$ of the Markov chain can be bounded from below by*

$$\tau_{\max}(\varepsilon) \geq \frac{1}{2} \cdot \frac{\mu_2}{1 - \mu_2} \cdot \ln(2\varepsilon)^{-1},$$

where μ_2 is the second largest eigenvalue of P'.

Proof. Let $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_{|\Omega|}$ be the real eigenvalues of the transition matrix P. By Theorem 2.5, the total mixing time is bounded from below by

$$\tau_{\max}(\varepsilon) \geq \frac{1}{2} \cdot \frac{\lambda_{\max}}{1 - \lambda_{\max}} \cdot \ln(2\varepsilon)^{-1},$$

where $\lambda_{\max} := \max\{\lambda_2, |\lambda_{|\Omega|}|\}$. Following from Theorem 6.2, we infer

$$\begin{aligned}
& \lambda_{\max} \geq \lambda_2 \geq \mu_2 \\
\Leftrightarrow \quad & \lambda_{\max} - \mu_2 \lambda_{\max} \geq \mu_2 - \mu_2 \lambda_{\max} \\
\Leftrightarrow \quad & \lambda_{\max} \cdot (1 - \mu_2) \geq \mu_2 \cdot (1 - \lambda_{\max}) \\
\Leftrightarrow \quad & \frac{\lambda_{\max}}{1 - \lambda_{\max}} \geq \frac{\mu_2}{1 - \mu_2}.
\end{aligned}$$

Consequently,

$$\tau_{\max}(\varepsilon) \geq \frac{1}{2} \cdot \frac{\mu_2}{1 - \mu_2} \cdot \ln(2\varepsilon)^{-1}.$$

□

Corollary 6.3 shows that we can lower bound the total mixing time of a Markov chain by the eigenvalues of any principal sub-matrix P' of the associated transition matrix P. Interpreting P as the weighted adjacency matrix of a state graph Γ, a principal sub-matrix P' corresponds to the weighted adjacency matrix of an induced subgraph $\Gamma' \subseteq \Gamma$. In summary, we can lower bound the total mixing time of a Markov chain by the eigenvalues of an arbitrary induced subgraph of a potentially large state graph.

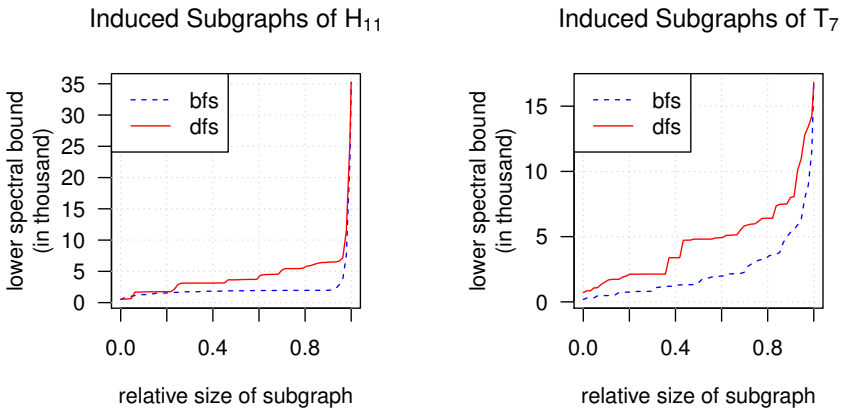

Figure 6.11: Lower bound on the total mixing time of Broder's chain derived from Corollary 6.3. ($\varepsilon = 0.01$, $s \in \mathcal{M}(G)$.)

Construction Strategies To construct an induced subgraph, we may simply stop our construction algorithm (see Alg. 3.1) after a certain number of states has been processed. In our default implementation, we iteratively explore the unknown state graph via breadth first search (bfs), starting at the initial state of the Markov chain. Alternatively, we could easily apply a modified construction algorithm based on depth first search (dfs). While the bfs strategy will produce subgraphs with a small diameter, the diameter of an induced subgraph produced by the dfs strategy will tend to be large. In addition, a subgraph created by bfs will only contain states that are close to the initial state, while the dfs strategy will produce a subgraph that contains states with a large distance to the initial state. As it is unclear which strategy will be lead to better lower bounds, we evaluated both exploration strategies.

Experiment 6.5 In a first experiment, we assessed how sharp the lower bounds derived from an induced subgraph can be. For this purpose, we exemplarily considered Broder's chain on the hexagon graph H_{11}, and the odd triangle graph T_7. Both bipartite graphs possess exactly one perfect matching, and about 32.000 near-perfect matchings. Starting with the single perfect matching as the initial state, we ran breadth first search, respective depth first search to iteratively explore further regions of the state graph. Every 500 steps, we interrupted the state graph construction to calculate the lower bound via Corollary 6.3. Fig. 6.11 shows how the lower bound on the total mixing time depends on the size of the induced subgraph.

- We observe that the lower bound gained by depth first search is superior to that gained by breadth first search for both exemplary graphs.
- Considering the hexagon graph H_{11}, we observe that the lower bound on the total mixing time increases very slowly until almost the full state graph has

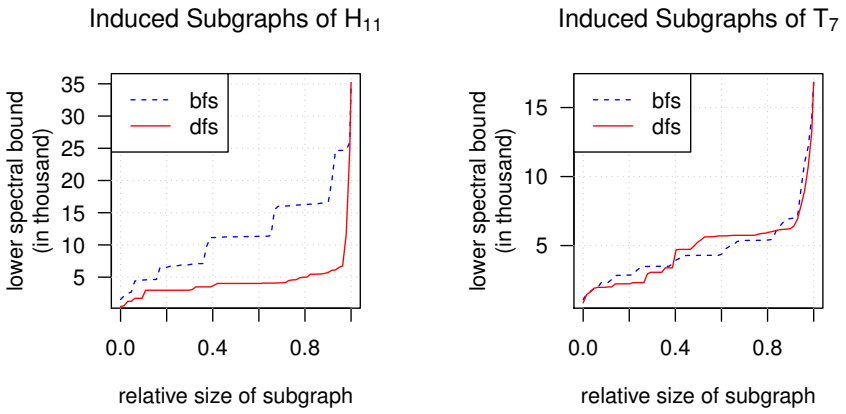

Figure 6.12: Lower bound on the total mixing time of Broder's chain derived from Corollary 6.3. ($\varepsilon = 0.01$, $s \in \mathcal{N}(G)$.)

been constructed. After about 90% of the state graphs full size has been reached, the lower bound increases erratically.

- Considering the odd triangle graph T_7, we observe that the erratic increase of the lower bound is less steep. In addition, we find the depth first search strategy to be significantly superior to breadth first search here.

From our experiment, we conclude that the first states added to the subgraph apparently play an unimportant role for the total mixing time of the Markov chain. In contrast, the states that are included last have a strong impact on the lower bound and thus, might have a large mixing time. As we observed in a previous experiment that the mixing time of Broder's chain depends on the initial state s, we assume that a similar effect may be the reason for the effects observed here. We will further investigate the effect of the initial state s in the next experiment.

Experiment 6.6 To evaluate the influence of the initial state s on the quality of the lower bound, we repeated the experiment while starting the state graph construction with a near-perfect matching of maximal distance to the perfect matching. In doing so, we early process the states we assume to have a large mixing time. Fig. 6.12 shows the results of this experiment.

- Considering the hexagon graph H_{11}, we observe that the lower bound derived from breadth first search is now significantly larger than before, while the bound gained by depth first search is very similar to the previous one. In addition, we can now clearly observe that the lower bound increases in a stepwise manner. After periods in which the lower bound stays almost constant, it increases erratically after certain states have been included in the subgraph.

- We observe similar effects when considering the odd triangle graph T_7. While the lower bound gained by depth first search is nearly identical to that gained in the previous experiment, breadth first search produces a lower bound which is significantly larger than the one observed before.
- Unfortunately, we still observe that the lower bound increases erratically after about 90% of the full state graph has been constructed. Thus, we still have to construct almost the entire state graph until the lower bound is approximately as large as the spectral bound of the full state graph.

Our experiments highlight the influence of the initial state s on the quality of the lower bound gained from an induced subgraph. In particular, the quality of the breadth first search strategy highly depends on the initial state, while depth first search seems to be more robust to the choice of the initial state. A further project for future studies would be to find an explanation for the stepwise growth of the lower bound. First experiments showed no obvious connection to properties of the induced subgraph.

Experiment 6.7 In a final experiment, we assessed how sharp a lower bound gained by Corollary 6.3 may be in practical sampling scenarios. In practical applications, we neither know the size $|\Omega(G)|$ of a state graph, nor is it feasible to fully construct the state graph. (Otherwise, the sampling problem would be trivial.) However, we may very well construct an induced subgraph $\Gamma' = (\Omega', \Psi')$ of limited size, say of size $|\Omega'| := \min\{|\Omega(G)|, 10000\}$. This would be a reasonable limit for which the construction of an induced subgraph and the calculation of the lower bound via Corollary 6.3 can be executed within a few seconds.

To assess the quality of the thereby gained bound, we determined a posteriori how large the lower spectral bound of the full state would have been. The ratio between both quantities shows by how much our method underestimates the (in practical applications unknown) lower spectral bound, and can therefore be used to assess the quality of the bound. Figs. 6.13 and 6.14 show how the quality decreases for the classes of hexagon and odd-triangle graphs.

- We observe that the quality of the lower bound is trivially equal to one when the size $|\Omega(G)|$ of a state graph is not larger than our upper limit of 10000. In contrast, the quality of the lower bound decreases rapidly as soon as the size of the full state graph exceeds 10000.
- Considering the breadth first search strategy, we observe that the quality decreases exponentially (note the logarithmic scale of the y-axis) when we let the size n of a bipartite graph grow. This observation holds for both graph classes.
- In contrast, the depth first search strategy is slightly superior to breadth first search as the quality of the lower bound decreases more slowly. However, its quality is still too poor to be applicable in practical applications.

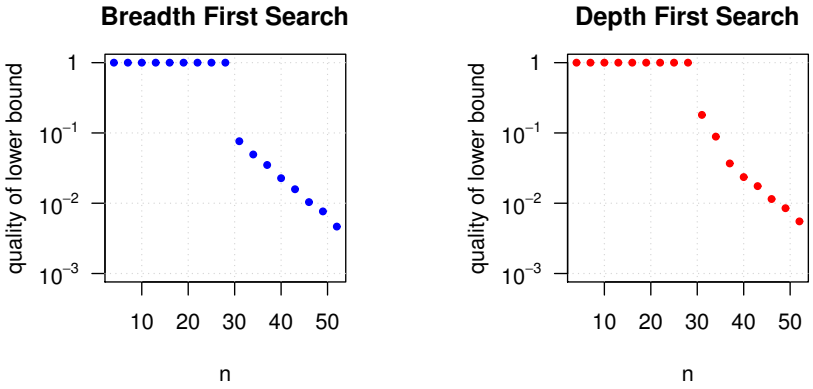

Figure 6.13: Quality of the lower bound on the total mixing time of Broder's chain for hexagon graphs. The lower bounds were calculated via Corollary 6.3 from induced subgraphs of size $\min\{|\Omega(G)|, 10000\}$. ($\varepsilon = 0.01$.)

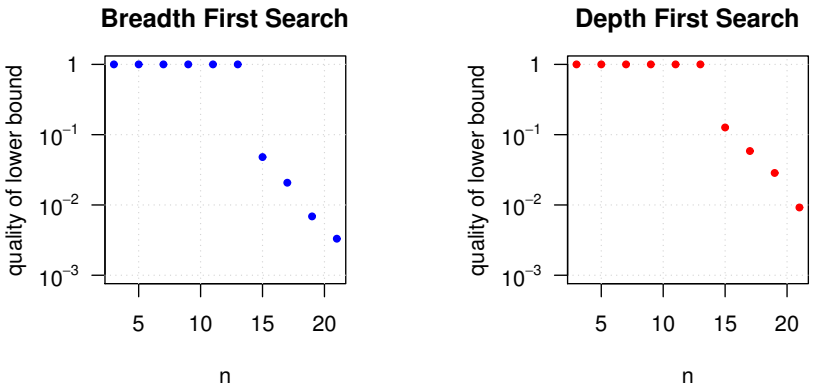

Figure 6.14: Quality of the lower bound on the total mixing time of Broder's chain for odd triangle graphs. The lower bounds were calculated via Corollary 6.3 from induced subgraphs of size $\min\{|\Omega(G)|, 10000\}$. ($\varepsilon = 0.01$.)

Our experiment showed that the lower bound gained by an induced subgraph of constant size may largely underestimate the unknown lower spectral bound. As this observations holds for both exploration strategies, we conclude that our method is unlikely to produce suitable lower bounds in practical applications like the one outlined in this experiment.

Summary In this section, we presented a method that is designed to derive a lower bound on the total mixing time of a Markov chain when the construction of the associated state graph is infeasible. By constructing an induced subgraph of a potentially large state graph, we may lower bound the total mixing time by calculating the eigenvalues of the sub-graph's weighted adjacency matrix. In a set of experiments, we assessed the quality of the lower bound on the total mixing time of Broder's chain. In doing so, we found that the quality of the lower bound highly depends on the initial state of the Markov chain. However, determining an appropriate initial state is a problem that seems hard to address in practical applications. Finally, we showed that the lower bound calculated from an induced subgraph of constant size underestimates the lower spectral bound of the full state graph by a high factor. Thus, we conclude that our method is of very limited use in practical scenarios.

6.3 Summary

In this chapter, we considered the uniform sampling of perfect and near-perfect matchings in bipartite graphs. This is a classical sampling problem with interesting connections to complexity theory. To demonstrate how *marathon* may be used to support the analysis of MCMC algorithms, we experimentally investigated the total mixing time of two well-known Markov chains. While the total mixing time of Broder's chain is known to be an exponential function on the size of a bipartite graph for several graph classes, the JSV chain is proven to be rapidly mixing for bipartite graphs in general.

By experimentally determining the relation between the size of a bipartite graph and the total mixing time of the Markov chains for two classes of bipartite graph, we confirmed the theoretical results elaborately shown by Rechner and Berger [25]. Our findings demonstrate how *marathon* can be applied to determine properties of Markov chains that are usually hard to find in theory.

In an additional set of experiments, we showed that the initial state s has a large influence on the mixing time of Broder's chain and may even decide between a polynomial and exponential mixing time. The question of what makes a good initial state is an interesting topic for further studies.

In a final set of experiments, we evaluated a strategy to calculate a lower bound on the total mixing time of a Markov chain when the construction of the associated state graph is infeasible. By constructing an induced subgraph of a (potentially much

larger) state graph, we showed how to lower bound the total mixing time using the eigenvalues of the induced sub-graph's weighted adjacency matrix. Unfortunately, we found that the lower bound gained by this method may largely underestimate the total mixing time of the Markov chain. Thus, it may be of very limited usefulness in practical applications.

Chapter 7

Conclusion and Future Work

In this thesis, we discussed Markov chain Monte Carlo algorithms for three sampling problems. Such an algorithm can be seen as a random walk on the state graph of an associated Markov chain. By starting at an arbitrary initial state, the algorithm iteratively moves to an adjacent state. After a certain number of steps, the algorithm returns the final state as the random sample. In our discussion, we laid a particular focus on the running time of such methods. The number of steps required to sample from a distribution that is "close" to the target distribution π is known as the total mixing time of the associated Markov chain. A sharp bound the total mixing time is of large interest in many applications.

7.1 Conclusion

As the total mixing time of non-trivial Markov chains is hard to assess analytically, the number of steps of a random walk is typically chosen empirically. Thus, our main motivation was the question whether we might provide a more reliable suggestion on the number of steps in practical applications. We briefly summarize our key results.

- In Chapter 2, we laid the mathematical groundwork for our following discussion. After summarizing well-known theorems on Markov chains and mixing time, we introduced the concept of empirical mixing time, and showed that it can be used to gain a lower bound on the total mixing time of a Markov chain. As the empirical mixing time can efficiently be approximated by the simulation of the associated Markov chain, it is applicable in many practical sampling applications.

- In the following chapter, we presented the *marathon* software, a C++ library designed to support the analysis of MCMC algorithms. This software package has two main applications. First, it contains highly efficient sampling routines that are discussed in Chapters 4 to 6. Second, the software supports the construction and analysis of a Markov chain's state graph. From such a state

graph, properties like the total mixing time, the expected loop probability, its diameter, its second largest eigenvalue, and many more can be calculated that would otherwise be hard to find in theory. As a key aspect of *marathon* is extensibility, various MCMC algorithms of diverse sampling applications can easily be integrated and analysed.

- In Chapter 4, we addressed the uniform sampling of bipartite graphs with fixed degrees. This is a classical sampling problem of large practical relevance in network analysis and other fields. In a set of experiments, we assessed the efficiency of the classical switch chain, the edge switch chain, and the curveball chain on a large set of instances.

  By calculating the total mixing time of each Markov chain, we found that both switch chain variants require a quadratic number of steps to produce a random sample for several instance classes. In an additional set of experiments, we assessed the quality of some well-known lower and upper bounds on the total mixing time of a Markov chain. Thereby, we found that the famous canonical path method is unlikely to produce sharp bounds and thus, existing upper bounds derived from this method are most likely too pessimistic. To determine the efficiency of the Markov chains on real-world instances, we measured the running time of each sampling algorithm on a large set of ecological instances. In doing so, we found that the edge switch chain is most efficient in the majority of cases.

  We closed this chapter by showing how the sampling process can be accelerated by an efficient preprocessing of the vertex degrees. For this purpose, we presented an algorithm that decomposes a bi-graphical vector pair into a set of primitive components that can be processed independently of each other. Experimentally, we showed that the sampling process can thereby be significantly accelerated.

- In Chapter 5, we discussed the uniform sampling of bipartite graphs whose degrees lie in prescribed intervals. This is a variant of the previously assessed sampling problem motivated by the work with incomplete data. We presented two Markov chains whose stationary distribution is designed to be the uniform distribution on the set of bipartite graphs from which we want to draw samples. Via a series of theorems, we showed that both Markov chains are ergodic, thereby proving the correctness of the associated sampling algorithms.

  Afterwards, we experimentally assessed the efficiency of both Markov chains. In doing so, we found the simple Markov chain superior if the degrees of a bipartite graph are near-regular, while the informed chain is suited best for the degrees of scale-free graphs. Subsequently, we demonstrated how our methods can be used to approximate expected network measures of partially observed networks.

  We finished this chapter by discussing the problem of constructing a bipartite

graph whose degrees lie in prescribed intervals. This is a sub-problem of the associated sampling problem if no initial state is known. After presenting a realization algorithm, we gave a proof of its correctness and showed that its running time is asymptotically optimal, thereby beating straight-forward approaches.

- In Chapter 6, we discussed the uniform sampling of perfect matchings in bipartite graphs. As it seems to be hard to sample from the set of perfect matchings alone, various MCMC sampling algorithms use near-perfect matchings as auxiliary states. Whereas Broder's chain is known to be rapidly mixing for some classes of bipartite graphs, its total mixing time grows exponentially for others. In contrast, the JSV chain is a rapidly mixing Markov chain that uses a well-chosen weight function to produce random samples according to a non-uniform distribution. By experimentally assessing the total mixing time of both Markov chains, we evaluated the influence of the initial state on the efficiency of Broder's chain. In doing so, we showed that the right choice of the initial state may decide between an exponential or polynomial running time.

  Finally, we showed how an induced subgraph of a much larger state graph may be used to derive a lower bound on the total mixing time of an associated Markov chain. Unfortunately, our experiments showed that this method is unlikely to be of high value for practical applications as the quality of the lower bound decreases fast.

Coming back to our main motivation, our findings confirmed the difficulty of providing a reliable suggestion for the length of a random walk in practical applications. By the construction of a state graph and the computation of the associated total mixing time, *marathon* can be used to calculate the exact number of steps required to sample from a probability distribution that is arbitrary close to the target distribution π. Unfortunately, the construction of state graphs is infeasible in nearly all practical applications, which limits the applicability of this approach.

However, we can still approximate the empirical mixing time when the construction of a state graph is infeasible. In doing so, we gain a lower bound on the unknown total mixing time and thus, on the necessary number of steps of the random walk. Alas, the quality of the lower bound depends on the input instance and can be very small, leaving the open question of how reliable this lower bound really is. Thus, the right choice of the parameter t still remains a difficult problem.

7.2 Open Problems and Future Work

Our findings leave room for several future projects.

- A general limitation of our approach is the necessity of constructing a state graph if we want to calculate a sharp bound on the total mixing time. Unfortunately, this is infeasible in many practical applications as the number of states

is too large to be enumerated. Thus, a well-rewarding future project might be to establish further bounding methods that do not require state graphs.

- An interesting open problem is the choice of the initial state s. As the total mixing time is defined as the largest mixing time of any initial state, such a definition seems too pessimistic in practice. In Chapter 6, we showed that the choice of the initial state may induce an exponential or polynomial mixing time. Thus, further research on the influence of the initial state seems promising.

- Further developing the question of what makes a good initial state, we could discuss randomization heuristics. Such sampling algorithms have no guarantee of correctness but can nonetheless be used to efficiently create a random initial state for an MCMC sampling algorithm. In such cases, an MCMC algorithm does not start with an initial state but rather with an initial distribution of states. As it is unclear by which amount this strategy reduces the mixing time of a Markov chain, the discussion of randomization heuristics seem to be an interesting future project.

- To eliminate the question of how many steps are required to produce a random sample, an integrated convergence detection mechanism would be a great advance in practical sampling applications. Such a mechanism is supposed to monitor the evolution of a Markov chain during a simulation and decides when the current state is sufficiently random.

- To further assess the quality of upper bounds on the total mixing time, we could implement methods to evaluate a *multi commodity congestion scheme* [16]. This generalization of the canonical path method may lead to sharper upper bounds on the total mixing time. To evaluate the potential of the canonical path and the multi commodity flow method, we might further calculate an *optimal* set $\mathcal{P}$ of paths in a certain state graph. Such a set of paths minimizes the maximal congestion $\rho(\mathcal{P})$, thereby resulting in an upper bound that is as close to the total mixing time as the method is possible to derive. As the construction of an optimal set of paths is $\mathcal{NP}$-hard, formulating the path construction problem as an integer linear program seems to be a promising approach. Preliminary results [96] show that the multi commodity bound can be very tight to the spectral bound for small state graphs.

- A potential candidate for future projects is the generalization of the Markov chains presented in Chapter 5 for non-bipartite graphs, directed graphs, and multi-graphs. The main challenge of such a project would be to establish a proof of ergodicity. In particular, proving the irreducibility of the Markov chains seems challenging.

- An ongoing project is the development of an R interface to *marathon*. As the R programming language is widely accepted in many life sciences, an R-package containing the sampling methods provided by *marathon* would be

usable by a large group of scientists. In doing so, we combine the efficiency of the programming language C++ with the simple usability and the widespread acceptance of R.

- Further investigating the properties of state graphs, we may apply methods from network analysis to the state graphs of Markov chains. For example, we can calculate measures of network centrality to determine states with an important role for the connectivity of state graphs. Such an analysis may help us to better understand the nature of the associated sampling algorithms. Preliminary results [97, 98], however, show that is hard to find a connection between such network measures and the total mixing time of the associated Markov chain.

Bibliography

[1] J. S. Liu. *Monte Carlo Strategies in Scientific Computing (Springer Series in Statistics)*. Springer (2008). DOI: 10.1007/978-0-387-76371-2.

[2] C. P. Robert. *Monte Carlo Statistical Methods (Springer Texts in Statistics)*. Springer (2010). DOI: 10.1007/978-1-4757-4145-2.

[3] J. M. Hammersley and D. C. Handscomb. *Monte Carlo Methods*. Springer (1964). DOI: 10.1007/978-94-009-5819-7.

[4] N. J. Gotelli and G. R. Graves. *Null models in ecology*. Smithsonian (1996). DOI: 10.2307/2265928.

[5] N. J. Gotelli. *Null model analysis of species co-occurrences patterns*. Ecology **81** (2000), 2606–2621. DOI: 10.2307/177478.

[6] W. Ulrich and N. J. Gotelli. *A null model algorithm for presence–absence matrices based on proportional resampling*. Ecological Modelling **244** (Oct. 2012), 20–27. DOI: 10.1016/j.ecolmodel.2012.06.030.

[7] O. J. Heilmann and E. H. Lieb. *Theory of monomer-dimer systems*. Communications in Mathematical Physics **25** (1972), 190–232. DOI: 10.1007/978-3-662-10018-9_7.

[8] L. G. Valiant. *The Complexity of Enumeration and Reliability Problems*. SIAM Journal on Computing **8** (Aug. 1979), 410–421. DOI: 10.1137/0208032.

[9] M. R. Jerrum, L. G. Valiant, and V. V. Vazirani. *Random generation of combinatorial structures from a uniform distribution*. Theoretical Computer Science **43** (1986), 169–188. DOI: 10.1016/0304-3975(86)90174-x.

[10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. *Optimization by simulated annealing*. Science **220** (1983), 671–680. DOI: 10.1126/science.220.4598.671.

[11] V. Černý. *Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm*. Journal of Optimization Theory and Applications **45** (1985), 41–51. DOI: 10.1007/bf00940812.

[12] R. Y. Rubinstein and D. P. Kroese. *Simulation and the Monte Carlo Method*. Wiley-Interscience (2007). DOI: 10.1002/9780470230381.

[13] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press (2005). DOI: 10.1017/cbo9780511813603.

[14] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press (1995).

[15] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. *Equation of state calculations by fast computing machines*. The Journal of Chemical Physics **21** (1953), 1087–1092. DOI: 10.2172/4390578.

[16] A. Sinclair. *Improved Bounds for Mixing Rates of Markov Chains and Multicommodity Flow*. Combinatorics, Probability and Computing **1** (Dec. 1992), 351–370. DOI: 10.1017/s0963548300000390.

[17] R. Kannan, P. Tetali, and S. Vempala. *Simple Markov-chain algorithms for generating bipartite graphs and tournaments*. Random Structures & Algorithms **14** (1997), 293–308. DOI: 10.1002/(sici)1098-2418(199907)14:4<293::aid-rsa1>3.0.co;2-g.

[18] M. Jerrum and A. Sinclair. *Approximating the permanent*. SIAM Journal on Computing **18** (1989), 1149–1178. DOI: 10.1137/0218077.

[19] M. Jerrum, A. Sinclair, and E. Vigoda. *A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries*. Journal of the ACM **51** (2004), 671–697. DOI: 10.1145/1008731.1008738.

[20] S. Rechner and A. Berger. *Marathon: An open source software library for the analysis of Markov-Chain Monte Carlo algorithms*. PLOS ONE **11** (2016). DOI: 10.1371/journal.pone.0147935.

[21] A. R. Rao, R. Jana, and S. Bandyopadhyay. *A Markov chain Monte Carlo method for generating random (0,1)-matrices with given marginals*. Sankhyā: The Indian Journal of Statistics, Series A (1961-2002) **58** (1996), 225–242.

[22] A. Berger and M. Müller-Hannemann. "Uniform Sampling of Digraphs with a Fixed Degree Sequence". *Graph Theoretic Concepts in Computer Science*. Ed. by D. M. Thilikos. Springer Berlin Heidelberg (2010), 220–231. DOI: 10.1007/978-3-642-16926-7_21.

[23] G. Strona, D. Nappo, F. Boccacci, S. Fattorini, and J. San-Miguel-Ayanz. *A fast and unbiased procedure to randomize ecological binary matrices with fixed row and column totals*. Nature communications **5** (2014). DOI: 10.1038/ncomms5114.

[24] A. Z. Broder. *How hard is it to marry at random? (On the approximation of the permanent)*. Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing (1986), 50–58. DOI: 10.1145/12130.12136.

[25] A. Berger and S. Rechner. *Broder's Chain Is Not Rapidly Mixing*. arXiv preprint (2014). arXiv: 1404.4249v1 [cs.DM].

[26] S. Rechner, L. Strowick, and M. Müller-Hannemann. *Uniform sampling of bipartite graphs with degrees in prescribed intervals*. Journal of Complex Networks (2017). DOI: 10.1093/comnet/cnx059.

[27] S. Rechner. *An Optimal Realization Algorithm for Bipartite Graphs with Degrees in Prescribed Intervals*. arXiv preprint (2017). arXiv: 1708.05520v1 [cs.DS].

[28] R. Diestel. *Graph theory*. 5th edition. Springer Publishing Company (2017). DOI: 10.1007/978-3-662-53622-3.

[29] W. Feller. *An Introduction to Probability Theory and Its Applications, Vol. 1, 3rd Edition*. Wiley (1968). DOI: 10.2307/1267002.

[30] D. A. Levin, Y. Peres, and E. L. Wilmer. *Markov chains and mixing times*. American Mathematical Society (2009). DOI: 10.1090/mbk/058.

[31] W. K. Hastings. *Monte Carlo sampling methods using Markov chains and their applications*. Biometrika **57** (1970), 97–109. DOI: 10.2307/2334940.

[32] O. Perron. *Zur Theorie der Matrices*. Mathematische Annalen **64** (1907), 248–263. DOI: 10.1007/bf01449896.

[33] G. F. Frobenius. *Über Matrizen aus nicht negativen Elementen*. Königliche Akademie der Wissenschaften (1912). DOI: 10.3931/e-rara-18865.

[34] J. W. Miller and M. T. Harrison. *Exact sampling and counting for fixed-margin matrices*. The Annals of Statistics **41** (2013), 1569–1592. DOI: 10.1214/13-aos1131.

[35] M. T. Harrison and J. W. Miller. *Importance sampling for weighted binary random matrices with specified margins*. arXiv preprint (2013). arXiv: 1301.3928v1 [stat.CO].

[36] C. Gkantsidis, M. Mihail, and E. W. Zegura. *The Markov Chain Simulation Method for Generating Connected Power Law Random Graphs*. Proceedings of the Fifth Workshop on Algorithm Engineering and Experiments (2003), 16–25.

[37] I. Miklós and J. Podani. *Randomization of presence-absence matrices: Comments and new algorithms*. Ecology **85** (2004), 86–92. DOI: 10.1890/03-0101.

[38] R. Milo, N. Kashtan, S. Itzkovitz, M. E. Newman, and U. Alon. *On the uniform generation of random graphs with prescribed degree sequences*. arXiv preprint: arXiv:cond-mat/0312028v2 (2004). arXiv: cond-mat/0312028v2 [cond-mat.stat-mech].

[39] Y. Artzy-Randrup and L. Stone. *Generating uniformly distributed random networks*. Physical Review E **72** (2005), 056708. DOI: 10.1103/physreve.72.056708.

[40] D. Aldous. *Random walks on finite groups and rapidly mixing Markov chains*. Séminaire de Probabilités XVII 1981/82 (1983), 243–297. DOI: 10.1007/bfb0068322.

[41] C. Sanderson and R. Curtin. *Armadillo: a template-based C++ library for linear algebra*. Journal of Open Source Software **1** (2016), 26. DOI: 10.21105/joss.00026.

[42] A. S. Asratian, T. M. J. Denley, and R. Häggkvist. *Bipartite graphs and their applications*. Cambridge Tracts in Mathematics. Cambridge University Press (1998). DOI: 10.1017/cbo9780511984068.

[43] D. J. Watts and S. H. Strogatz. *Collective dynamics of 'small-world' networks*. Nature **393** (1998), 440. DOI: 10.1515/9781400841356.301.

[44] L. A. N. Amaral, A. Scala, M. Barthélémy, and H. E. Stanley. *Classes of small-world networks*. Proceedings of the National Academy of Sciences **97** (2000), 11149–11152. DOI: 10.1073/pnas.200327197.

[45] J. Scott. *Social Network Analysis*. SAGE Publications Ltd (2017).

[46] J. M. Montoya, S. L. Pimm, and R. V. Solé. *Ecological networks and their fragility*. Nature **442** (2006), 259. DOI: 10.1038/nature04927.

[47] T. C. Ings, J. M. Montoya, J. Bascompte, N. Blüthgen, L. Brown, C. F. Dormann, F. Edwards, D. Figueroa, U. Jacob, J. I. Jones, R. B. Lauridsen, M. E. Ledger, H. M. Lewis, J. M. Olesen, F. F. Van Veen, P. H. Warren, and G. Woodward. *Review: Ecological networks – beyond food webs*. Journal of Animal Ecology **78** (2009), 253–269. DOI: 10.1111/j.1365-2656.2008.01460.x.

[48] B. D. Patterson and W. Atmar. *Nested subsets and the structure of insular mammalian faunas and archipelagos*. Biological Journal of the Linnean Society **28** (1986), 65–82. DOI: 10.1111/j.1095-8312.1986.tb01749.x.

[49] Y. Chen, P. Diaconis, S. P. Holmes, and J. S. Liu. *Sequential Monte Carlo methods for statistical analysis of tables*. Journal of the American Statistical Association **100** (2005), 109–120. DOI: 10.1198/016214504000001303.

[50] E. F. Connor and D. Simberloff. *The assembly of species communities: Chance or competition?* Ecology **60** (1979), 1132–1140. DOI: 10.2307/1936961.

[51] U. Brandes and T. Erlebach (Eds.) *Network Analysis: Methodological Foundations (Lecture Notes in Computer Science)*. Springer (2005). DOI: 10.1007/b106453.

[52] D. Gale. *A theorem on flows in networks.* Pacific Journal of Mathematics **7** (1957), 1073–1082. DOI: 10.2140/pjm.1957.7.1073.

[53] H. J. Ryser. *Combinatorial properties of matrices of zeros and ones*. Canadian Mathematical Society **9** (1957), 371–377. DOI: 10.1007/978-0-8176-4842-8_18.

[54] R. B. Holmes and L. K. Jones. *On uniform generation of two-way tables with fixed margins and the conditional volume test of Diaconis and Efron*. The Annals of Statistics **24** (1996), 64–68. DOI: 10.1214/aos/1033066199.

[55] C. I. Del Genio, H. Kim, Z. Toroczkai, and K. E. Bassler. *Efficient and exact sampling of simple graphs with given arbitrary degree sequence*. PLOS ONE **5** (2010), 1–7. DOI: 10.1371/journal.pone.0010012.

[56] I. Bezáková, A. Sinclair, D. Štefankovič, and E. Vigoda. *Negative examples for sequential importance sampling of binary contingency tables*. Algorithmica **64** (2012), 606–620. DOI: 10.1007/s00453-011-9569-3.

[57] I. Bezáková, N. Bhatnagar, and E. Vigoda. *Sampling binary contingency tables with a greedy start*. Random Structures & Algorithms **30** (2007), 168–205. DOI: 10.1002/rsa.20155.

[58] W. Tutte. *The factors of graphs*. Canadian Journal of Mathematics **4** (1952), 314–328. DOI: 10.1007/978-0-8176-4842-8_13.

[59] H. J. Ryser. *Matrices of zeros and ones*. Bulletin of the American Mathematical Society **66** (1960), 442–464. DOI: 10.1090/s0002-9904-1960-10494-6.

[60] I. Miklós, P. L. Erdős, and L. Soukup. *Towards random uniform sampling of bipartite graphs with given degree sequence*. The Electronic Journal of Combinatorics **20** (2013).

[61] P. L. Erdős, T. R. Mezei, and I. Miklós. *Efficiently sampling the realizations of irregular, but linearly bounded bipartite and directed degree sequences*. arXiv preprint (2017). arXiv: 1712.01709v1 [math.CO].

[62] P. L. Erdős, I. Miklós, and Z. Toroczkai. *New classes of degree sequences with fast mixing swap Markov chain sampling*. Combinatorics, Probability and Computing (2017), 1–22. DOI: 10.1017/s0963548317000499.

[63] C. Cooper, M. Dyer, and C. Greenhill. *Sampling regular graphs and a peer-to-peer network*. Combinatorics, Probability and Computing **16** (2007), 557–593. DOI: 10.1017/s0963548306007978.

[64] C. Greenhill and M. Sfragara. *The switch Markov chain for sampling irregular graphs and digraphs*. Theoretical Computer Science (2017). DOI: 10.1016/j.tcs.2017.11.010.

[65] C. J. Carstens and P. Kleer. *Comparing the Switch and Curveball Markov Chains for Sampling Binary Matrices with Fixed Marginals*. arXiv preprint (2017). arXiv: 1709.07290v3 [cs.DM].

[66] R. A. Fisher and F. Yates. *Statistical tables for biological, agricultural and medical research*. Vol. 3. Oliver and Boyd, Edinburgh (1949). DOI: 10.2307/2985258.

[67] N. D. Verhelst. *An Efficient MCMC Algorithm to Sample Binary Matrices with Fixed Marginals*. Psychometrika **73** (Apr. 2008), 705–728. DOI: 10.1007/s11336-008-9062-3.

[68] C. J. Carstens. *Proof of uniform sampling of binary matrices with fixed row sums and column sums for the fast curveball algorithm*. Physical Review E **91** (2015), 042812. DOI: 10.1103/physreve.91.042812.

[69] C. J. Carstens, A. Berger, and G. Strona. *Curveball: A new generation of sampling algorithms for graphs with fixed degree sequence*. arXiv preprint (2016). eprint: 1609.05137v2.

[70] Bascompte Lab. *Web of Life*. Sept. 2017. URL: http://www.web-of-life.es.

[71] A. Kodric-Brown and J. H. Brown. *Incomplete Data Sets in Community Ecology and Biogeography: A Cautionary Tale*. Ecological Applications **3** (Nov. 1993), 736–742. DOI: 10.2307/1942104.

[72] T. E. Raghunathan. *What Do We Do with Missing Data? Some Options for Analysis of Incomplete Data*. Annual Review of Public Health **25** (2004). PMID: 15015914, 99–117. DOI: 10.1146/annurev.publhealth.25.102802.124410. eprint: https://doi.org/10.1146/annurev.publhealth.25.102802.124410.

[73] P. M. Schlüter and S. A. Harris. *Analysis of multilocus fingerprinting data sets containing missing data*. Molecular Ecology Notes **6** (June 2006), 569–572. DOI: 10.1111/j.1471-8286.2006.01225.x.

[74] L. Strowick. *Erzeugen zufälliger bipartiter Graphen mit vorgegebenen Knotengradintervallen*. Master's thesis. Martin Luther University Halle-Wittenberg, (2017).

[75] D. Schluter. *A variance test for detecting species associations, with some example applications*. Ecology **65** (1984), 998–1005. DOI: 10.2307/1938071.

[76] M. E. Gilpin and J. M. Diamond. *Factors contributing to non-randomness in species co-occurrences on islands*. Oecologia **52** (1982), 75–84. DOI: 10.1007/bf00349014.

[77] A. Roberts and L. Stone. *Island-sharing by archipelago species*. Oecologia **83** (1990), 560–567. DOI: 10.1007/bf00317210.

[78] M. Almeida-Neto, P. Guimarães, P. R. Guimarães, R. D. Loyola, and W. Ulrich. *A consistent metric for nestedness analysis in ecological systems: reconciling concept and measurement*. Oikos **117** (2008), 1227–1239. DOI: 10.1111/j.0030-1299.2008.16644.x.

[79] P. P. Staniczenko, J. C. Kopp, and S. Allesina. *The ghost of nestedness in ecological networks*. Nature Communications **4** (Jan. 2013), 1391. DOI: 10.1038/ncomms2422.

[80] A. Clauset, C. R. Shalizi, and M. E. J. Newman. *Power-law distributions in empirical data*. SIAM Review **51** (2009), 661–703. DOI: 10.1137/070710111.

[81] A.-L. Barabási and R. Albert. *Emergence of scaling in random networks*. Science **286** (1999), 509–512. DOI: 10.1515/9781400841356.349.

[82] D. R. Fulkerson. *A network flow feasibility theorem and combinatorial applications*. Canadian Journal of Mathematics **11** (1959), 440–451. DOI: 10.4153/cjm-1959-045-1.

[83] É. Komáromi. *Matrices with restricted elements, row sums and column sums.* Acta Mathematica Academiae Scientiarum Hungarica **31** (1978), 349–354. DOI: 10.1007/bf01901983.

[84] M. Schocker. *On graphs with degrees in prescribed intervals.* Algebraic Combinatorics and Applications (2001), 307–315. DOI: 10.1007/978-3-642-59448-9_20.

[85] R. Anstee. *An algorithmic proof of Tutte's f-factor theorem.* Journal of Algorithms **6** (1985), 112–131. DOI: 10.1016/0196-6774(85)90022-7.

[86] J. B. Orlin. *Max flows in O(nm) time, or better.* Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing (2013), 765–774. DOI: 10.1145/2488608.2488705.

[87] J. E. Hopcroft and R. M. Karp. *An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs.* SIAM Journal on Computing **2** (1973), 225–231. DOI: 10.1137/0202019.

[88] M. Mucha and P. Sankowski. *Maximum matchings via Gaussian elimination.* 45th Annual IEEE Symposium on Foundations of Computer Science (Oct. 2004), 248–255. DOI: 10.1109/focs.2004.40.

[89] G. Borradaile, P. N. Klein, S. Mozes, Y. Nussbaum, and C. Wulff-Nilsen. *Multiple-Source multiple-sink maximum flow in directed planar graphs in near-linear time.* SIAM Journal on Computing **46** (2017), 1280–1303. DOI: 10.1109/focs.2011.73.

[90] L. Valiant. *The complexity of computing the permanent.* Theoretical Computer Science **8** (1979), 189–201. DOI: 10.1016/0304-3975(79)90044-6.

[91] D. Štefankovič, E. Vigoda, and J. Wilmes. *On Counting Perfect Matchings in General Graphs.* arXiv preprint (2017). arXiv: 1712.07504v1 [cs.DS].

[92] M. E. Dyer and H. Müller. *Counting perfect matchings and the switch chain.* arXiv preprint (2018). arXiv: 1705.05790v3 [cs.DM].

[93] P. Diaconis, R. Graham, and S. P. Holmes. *Statistical problems involving permutations with restricted positions.* IMS Lecture Notes-Monograph Series (1999), 195–222. DOI: 10.1214/lnms/1215090070.

[94] I. Bezáková, D. Štefankovič, V. V. Vazirani, and E. Vigoda. *Accelerating simulated annealing for the permanent and combinatorial counting problems.* Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm (2006), 900–907. DOI: 10.1145/1109557.1109656.

[95] W. H. Haemers. *Interlacing eigenvalues and graphs.* Linear Algebra and its Applications **226** (1995), 593–616. DOI: 10.1016/0024-3795(95)00199-2.

[96] L. Molitor. *Pfadbasierte Schranken für Mischzeiten von Markov-Ketten mit ganzzahligen linearen Programmen.* Master's thesis. Martin Luther University Halle-Wittenberg, (2017).

[97] H. Boulouednine. *Experimentelle Strukturanalyse des Zustandsgraphen von Markovketten mit Hilfe von Clusteringalgoritmen.* Bachelor's thesis. Martin Luther University Halle-Wittenberg, (2016).

[98] B. Schmidt. *Untersuchung der Zustandsgraphen von Markovketten anhand verschiedener Zentralitätsmaße.* Bachelor's thesis. Martin Luther University Halle-Wittenberg, (2016).

Appendix A

Index to Notations

Notation	Meaning	Page
Ω	set $\Omega = \{x_1, x_2, \ldots, x_{\lvert\Omega\rvert}\}$ of combinatorial objects	1
$\pi(x)$	probability distribution function $\pi\colon \Omega \to [0,1]$	1
X	random variable from the set Ω	1
$E_\pi[f(X)]$	expected value of a function $f\colon \Omega \to \mathbb{R}$ according to the probability distribution function π	1
$\mu_\pi[f(X)]$	sample mean associated to $E_\pi[f(X)]$	1
N	number of samples used to calculate sample mean	1
s	initial state $s \in \Omega$	3
$w(x)$	weight function $w\colon \Omega \to \mathbb{R}^+$	3
$\pi(x) \propto w(x)$	$\pi(x)$ is proportional to $w(x)$, i.e. $\exists c \in \mathbb{R}\colon \forall x \in \Omega\colon \pi(x) = c \cdot w(x)$.	3
$p(x,y)$	transition probability function $p\colon \Omega \times \Omega \to [0,1]$. $p(x_i, x_j) := \Pr[X_t = x_j \mid X_{t-1} = x_i]$	10
P	transition matrix $P = (p_{ij})$ with $p_{ij} := p(x_i, x_j)$	10
Γ	state graph $\Gamma = (\Omega, \Psi)$ of an associated Markov chain	10
$\kappa(x,y)$	proposal probability function $\kappa\colon \Omega \times \Omega \to [0,1]$	10
$p_s^{(t)}(x)$	t-step probability distribution function $p_s^{(t)}\colon \Omega \to [0,1]$ $p_s^{(t)}(x) := \Pr[X_t = x \mid X_0 = s]$	10
$\lvert\lvert\mu - \eta\rvert\rvert$	total variation distance. $\lvert\lvert\mu - \eta\rvert\rvert := \frac{1}{2}\sum_{x\in\Omega} \lvert\mu(x) - \eta(x)\rvert$	13
ε	distance to target distribution ($\varepsilon \in [0,1]$)	
$\tau_s(\varepsilon)$	mixing time $\tau_s(\varepsilon) := \min\{t \in \mathbb{N}\colon \lvert\lvert p_s^{(t)} - \pi\rvert\rvert \leq \varepsilon\}$ of initial state $s \in \Omega$	13
$\tau_{\max}(\varepsilon)$	total mixing time. $\tau_{\max}(\varepsilon) := \max_{s\in\Omega}\{\tau_s(\varepsilon)\}$	13
$\lambda_{\max}$	second-largest eigenvalue of a Markov chain's transition matrix. $\lambda_{\max} := \max\{\lambda_2, \lvert\lambda_{\lvert\Omega\rvert}\rvert\}$	14
$\pi_{\min}$	$\pi_{\min} := \min_{x\in\Omega}\{\pi(x)\}$	14
$\rho(\mathcal{P})$	maximal load congestion of a system $\mathcal{P}$ of paths	15

Notation	Meaning	Page
$X \sim \pi$	random variable $X \in \Omega$ is distributed according to the probability distribution function π	15
$\eta(y)$	$\eta(y) := \Pr[f(X) = y \mid X \sim \pi]$ (defined with respect to an auxiliary function $f\colon \Omega \to \mathbb{R}$.)	15
$q_s^{(t)}(y)$	$q_s^{(t)}(y) := \Pr[f(X) = y \mid X \sim p_s^{(t)}]$ (defined with respect to an auxiliary function $f\colon \Omega \to \mathbb{R}$.)	15
$\bar{\tau}_s(\varepsilon)$	empirical mixing time of state $s \in \Omega$. $\bar{\tau}_s(\varepsilon) := \min\{t \in \mathbb{N}\colon \|q_s^{(t)} - \eta\| \leq \varepsilon\}$	16
G	bipartite graph $G = (U, V, E)$ with disjoint vertex sets $U = \{u_1, \ldots, u_{\|U\|}\}$ and $V = \{v_1, \ldots, v_{\|V\|}\}$, and edge set $E \subseteq U \times V$	33
$\mathbf{r}, \mathbf{c}$	integer vectors $\mathbf{r} = (r_1, \ldots, r_m)$ and $\mathbf{c} = (c_1, \ldots, c_n)$	34
m, n	length of integer vectors $\mathbf{r}$ and $\mathbf{c}$	34
$\delta_G(v)$	node degree of a vertex $v \in U \cup V$ in the bipartite graph $G = (U, V, E)$	34
$\Omega(\mathbf{r}, \mathbf{c})$	set of bipartite graphs whose degrees match $\mathbf{r}$ and $\mathbf{c}$	34
$\mathbf{r} \leq \bar{\mathbf{r}}$	$\forall i \in \{1, \ldots, m\}\colon r_i \leq \bar{r}_i$	35
$\mathbf{c}'$	conjugate sequence $\mathbf{c}' = (c'_1, c'_2, \ldots)$ of integer vector $\mathbf{c}$. $\forall j \in \mathbb{N}\colon c'_j := \|\{i \in \{1, 2, \ldots, n\}\colon c_i \geq j\}\|$	35
$\mathbf{c}''$	double conjugate sequence $\mathbf{c}'' = (c''_1, c''_2, \ldots)$ of integer vector $\mathbf{c}$. $\forall j \in \{1, \ldots, n\}\colon c''_j = c_j$	35
$\Sigma_{\mathbf{r}}^k$	short for $\sum_{i=1}^{k} r_i$	35
$\Sigma_{\mathbf{r}}$	sequence $\Sigma_{\mathbf{r}} := (\Sigma_{\mathbf{r}}^1, \Sigma_{\mathbf{r}}^2, \ldots)$ of partial sums	35
$\mathbf{r} \trianglelefteq \mathbf{c}'$	$\mathbf{c}'$ dominates $\mathbf{r}$. $\forall k \in \{1, \ldots, m\}\colon \Sigma_{\mathbf{r}}^k \leq \Sigma_{\mathbf{c}'}^k$	35
$T_s(\varepsilon)$	running time for $\tau_s(\varepsilon)$ steps of an associated Markov chain	65
$\overline{T}_s(\varepsilon)$	running time for $\bar{\tau}_s(\varepsilon)$ steps of an associated Markov chain	66
$\mathbf{vw}$	concatenation of integer vectors $\mathbf{v}$ and $\mathbf{w}$	67
$\tilde{\mathbf{r}}, \tilde{\mathbf{c}}$	integer vectors of length m and n (lower bounds)	77
$\bar{\mathbf{r}}, \bar{\mathbf{c}}$	integer vectors of length m and n (upper bounds)	77
$\Omega(\tilde{\mathbf{r}}, \bar{\mathbf{r}}, \tilde{\mathbf{c}}, \bar{\mathbf{c}})$	set of bipartite graphs whose degrees are bounded from below by $\tilde{\mathbf{r}}$ and $\tilde{\mathbf{c}}$, and from above by $\bar{\mathbf{r}}$ and $\bar{\mathbf{c}}$	77
$\tilde{d}(v)$	lower bound on the degree of vertex $v \in U \cup V$	77
$\bar{d}(v)$	upper bound on the degree of vertex $v \in U \cup V$	77
$G \triangle G'$	symmetric difference of edge sets. $G \triangle G' := (E \backslash E') \cup (E' \backslash E)$	78
$\mathcal{M}(G)$	set of perfect matchings in a bipartite graph G	113
$\mathcal{N}(G)$	set of near-perfect matchings in a bipartite graph G	114
$\Omega(G)$	set of perfect and near-perfect matchings in G	114
$\mathcal{N}_{u,v}(G)$	set of near-perfect matchings in $G = (U, V, E)$ with $u \in U$ and $v \in V$ being unmatched	116
H_k	k-th hexagon graph	117
T_k	k-th odd triangle graph	118

Appendix B

Data Sets

The following tables contain the instances that were used in Chapters 4 and 5. Among other information, the tables show the row and column sums of each instance in non-increasing order. We use a run length encoding to shorten the row and column sums. Whenever a number occurs more than twice, we replace all following repetitions by the number of occurrences (see Table B.1 for two examples). The columns labeled by "m" and "n" state the number of rows and columns, "$|E|$" shows the sum of the row and column sums, and "Prim." displays whether or not $(\mathbf{r}, \mathbf{c})$ is a primitive vector pair in the sense of Definition 4.13.

Table B.1: Bi-Graphical Integer Vectors

Identifier	Row Sums $\mathbf{r}$	Column Sums $\mathbf{c}$	m	n	$\lvert E\rvert$	Prim.
Darwin's Finches	$17, 14, 14, 13, 12, 11, 10(\times 3), 6, 2, 2, 1$	$11, 10(\times 4), 9(\times 3), 8, 8, 7, 4(\times 3), 3(\times 3)$	13	17	122	✗
Barabási-Albert	$56, 48, 38, 36, 36, 33, 31, 31, 29, 24(\times 6), 23(\times 3), 22, 22, 21(\times 3), 20, 20, 19(\times 3), 18, 17, 16(\times 3), 15, 14(\times 5), 13(\times 9), 12(\times 10), 11(\times 6), 10(\times 5), 9(\times 5), 8(\times 7), 7(\times 4), 6(\times 4), 5(\times 6), 3, 3, 2, 2, 1$	$56, 49, 49, 47, 47, 45, 44, 40, 36, 30, 30, 29, 28, 28, 25, 24, 23, 22, 21, 21, 19(\times 3), 17, 16, 16, 15(\times 5), 14(\times 3), 13(\times 4), 12(\times 11), 11(\times 5), 10(\times 4), 9(\times 7), 8(\times 7), 7(\times 11), 6(\times 4), 5(\times 6), 4(\times 4), 3, 3, 2$	100	100	1470	✓

Table B.2 contains the sampling instances derived from 62 binary ecological matrices in Bascompte's web-of-life [70]. This data set is a collection of ecological networks collected by scientists from all over the world. Among others, seed-dispersal, plant-pollinator and plant-herbivore networks are included within the data set.

Table B.2: Web-Of-Life

Identifier	Row Sums $\mathbf{r}$	Column Sums $\mathbf{c}$	m	n	$\lvert E\rvert$	Prim.
A_PH_004	$22, 14(\times 3), 13, 11, 10, 9, 8, 7, 6(\times 7), 5(\times 3), 4, 1$	$19, 15, 13, 11, 7(\times 3), 6, 6, 5, 5, 4(\times 6), 3(\times 7), 2(\times 10), 1(\times 18)$	22	52	184	✓
A_PH_005	$23, 20, 17, 12, 12, 10, 7, 6(\times 5), 5(\times 4), 4, 3(\times 4), 2(\times 3)$	$17, 13, 11, 10, 8, 6(\times 3), 5(\times 3), 4(\times 5), 3(\times 5), 2(\times 13), 1(\times 20)$	24	54	173	✓
A_PH_006	$72, 31, 5, 4, 3, 1$	$3(\times 3), 2(\times 22), 1(\times 63)$	6	88	116	✓
A_PH_007	$64, 12, 9, 5, 5$	$4(\times 4), 3(\times 4), 2(\times 11), 1(\times 45)$	5	64	95	✗
M_PL_001	$25, 22, 14, 13(\times 3), 10, 10, 9, 8, 8, 7(\times 4), 6(\times 7), 5(\times 8), 4(\times 5), 3(\times 14), 2(\times 9), 1(\times 26)$	$34, 22, 21, 15, 13, 13, 11, 9, 9, 8(\times 3), 7(\times 4), 6, 6, 5(\times 3), 4(\times 6), 3(\times 9), 2(\times 19), 1(\times 46)$	84	101	361	✓
M_PL_002	$14, 13, 12, 11, 10(\times 4), 6(\times 4), 5(\times 7), 4(\times 3), 3(\times 3), 2(\times 8), 1(\times 10)$	$16, 15, 9, 8, 8, 7, 6, 6, 5, 5, 4(\times 7), 3(\times 13), 2(\times 10), 1(\times 24)$	43	64	196	✓
M_PL_003	$15, 9, 8, 7, 7, 6, 3, 3, 2(\times 6), 1(\times 11)$	$7, 6, 5, 5, 4(\times 4), 3(\times 4), 2(\times 6), 1(\times 18)$	25	36	81	✓
M_PL_005	$62, 59, 51, 49, 47, 39, 39, 31, 29, 28, 28, 22, 18, 17, 16, 15, 14, 14, 13, 13, 12, 11(\times 5), 10, 10, 9(\times 3), 7(\times 8), 6(\times 3), 5(\times 5), 4(\times 7), 3(\times 11), 2(\times 14), 1(\times 17)$	$37, 32, 30, 27, 22, 21, 17, 16, 15, 15, 14, 13, 13, 12(\times 3), 11(\times 4), 10(\times 4), 9(\times 3), 8(\times 3), 7(\times 4), 6(\times 5), 5(\times 15), 4(\times 11), 3(\times 26), 2(\times 41), 1(\times 143)$	96	275	923	✓
M_PL_008	$18, 14, 14, 12, 10, 9, 8, 7, 7, 5, 2$	$6(\times 5), 5, 5, 4(\times 5), 3(\times 6), 2(\times 8), 1(\times 12)$	11	38	106	✓
M_PL_009	$28, 25, 25, 24, 15, 14, 13, 13, 9, 8(\times 4), 6, 6, 5, 5, 4(\times 4), 3, 2, 1$	$10, 9(\times 4), 8, 6, 6, 5(\times 3), 4(\times 7), 3(\times 9), 2(\times 15), 1(\times 76)$	24	118	242	✓
M_PL_010	$32, 29, 28, 26, 25, 23, 23, 20, 19, 17, 16, 16, 15, 14, 14, 13, 12, 11, 11, 10(\times 3), 9, 9, 8, 8, 7, 6(\times 3), 3$	$20, 19, 18, 16(\times 5), 15, 15, 14, 13, 12(\times 3), 11(\times 3), 10, 9, 8, 8, 7, 6(\times 4), 5(\times 6), 4(\times 8), 3(\times 6), 2(\times 18), 1(\times 11)$	31	76	456	✓
M_PL_011	$12, 10, 6, 5, 3(\times 4), 2, 2, 1(\times 3)$	$8, 6, 6, 5, 4(\times 3), 3, 3, 2(\times 4), 1$	13	14	52	✓
M_PL_012	$24, 19, 15, 13, 11, 10, 9, 7, 6, 3, 3, 2(\times 7), 1(\times 11)$	$16, 14, 8, 7, 6(\times 3), 5, 4, 4, 3(\times 8), 2(\times 8), 1(\times 29)$	29	55	145	✓

Identifier	Row Sums $\mathbf{r}$	Column Sums $\mathbf{c}$	m	n	$\lvert E\rvert$	Prim.
M_PL_014	$40, 27, 24, 14, 8, 7, 7, 5, 5, 4(\times 5), 3(\times 3), 2, 1(\times 11)$	$10, 9, 6, 5, 5, 4(\times 8), 3(\times 9), 2(\times 26), 1(\times 33)$	29	81	179	✓
M_PL_015	$124, 69, 63, 63, 62, 58, 55, 55, 54, 53, 51, 50, 50, 49, 48, 48, 47, 45, 43, 43, 42, 41, 41, 39, 39, 36, 35, 35, 34, 34, 33, 32(\times 3), 31(\times 4), 28, 28, 27(\times 3), 25, 25, 24, 24, 22(\times 3), 21, 20, 20, 19(\times 4), 18(\times 9), 17, 15(\times 3), 14(\times 7), 13(\times 4), 12, 12, 11(\times 4), 10(\times 9), 9(\times 7), 8(\times 4), 7(\times 6), 6, 6, 5(\times 5), 4(\times 3), 3(\times 4), 2, 2, 1, 1$	$104, 38, 38, 33(\times 3), 32, 31, 28, 26, 24, 24, 22, 22, 21(\times 3), 20, 19(\times 3), 18(\times 4), 17(\times 4), 16(\times 4), 15(\times 3), 14(\times 8), 13(\times 5), 12(\times 4), 11(\times 15), 10(\times 11), 9(\times 9), 8(\times 23), 7(\times 11), 6(\times 35), 5(\times 26), 4(\times 40), 3(\times 76), 2(\times 119), 1(\times 248)$	131	666	2933	✓
M_PL_016	$86, 28, 26, 23, 22, 21, 20, 19, 19, 18, 17, 16, 16, 15, 13, 12, 7, 6, 5(\times 3), 4, 4, 3, 1, 1$	$17, 14, 11, 10, 9, 9, 8, 7(\times 6), 6, 6, 5(\times 5), 4(\times 10), 3(\times 16), 2(\times 34), 1(\times 99)$	26	179	412	✓
M_PL_018	$31, 24, 22, 20, 17, 17, 15, 15, 14(\times 4), 13, 11, 11, 9(\times 3), 8, 7(\times 4), 6(\times 7), 5, 5, 4, 3, 3, 2, 2, 1, 1$	$25, 16, 15, 12, 10(\times 5), 9(\times 4), 8(\times 4), 7(\times 3), 6(\times 3), 5(\times 5), 4(\times 7), 3(\times 9), 2(\times 17), 1(\times 44)$	39	105	383	✓
M_PL_020	$60, 27, 24, 12, 11, 10, 9, 9, 7, 6, 4, 2, 2, 1(\times 7)$	$7(\times 3), 6(\times 3), 5(\times 3), 4(\times 3), 3(\times 14), 2(\times 17), 1(\times 48)$	20	91	190	✓
M_PL_021	$188, 75, 66, 50, 45, 45, 42, 41, 38, 34, 30, 28, 28, 25, 21(\times 3), 20, 19, 18, 18, 17(\times 3), 15, 12(\times 3), 11, 10(\times 6), 9, 8(\times 3), 7, 7, 6, 6, 5, 5, 4(\times 3), 3(\times 6), 2(\times 11), 1(\times 26)$	$25, 25, 23, 21, 17, 14, 11(\times 3), 10, 9, 9, 8(\times 6), 7(\times 3), 6(\times 6), 5(\times 15), 4(\times 12), 3(\times 33), 2(\times 90), 1(\times 500)$	91	677	1193	✓
M_PL_022	$21, 12, 6(\times 3), 5, 5, 3, 2(\times 6), 1(\times 7)$	$7, 4(\times 5), 3, 3, 2(\times 13), 1(\times 24)$	21	45	83	✓
M_PL_023	$34, 28, 11, 6, 5(\times 3), 4, 3(\times 4), 2(\times 4), 1(\times 7)$	$9, 6, 4(\times 3), 3(\times 7), 2(\times 17), 1(\times 43)$	23	72	125	✓

Identifier	Row Sums $\mathbf{r}$	Column Sums $\mathbf{c}$	m	n	$\lvert E\rvert$	Prim.
M_PL_026	$80, 14, 9, 8, 7(\times 3), 4, 4, 3(\times 5), 2(\times 9), 1(\times 31)$	$23, 11, 9, 8, 7, 5(\times 4), 4, 3(\times 6), 2(\times 15), 1(\times 74)$	54	105	204	✓
M_PL_027	$12, 11, 10, 10, 9, 9, 8, 7(\times 4), 6, 5, 4, 3, 2, 2, 1$	$9, 7, 6, 4(\times 5), 3(\times 6), 2(\times 14), 1(\times 32)$	18	60	120	✓
M_PL_028	$43, 31, 25, 17, 16(\times 4), 14, 13, 12(\times 3), 10(\times 3), 9, 8, 8, 7, 7, 6(\times 4), 5, 5, 4, 4, 3(\times 3), 2, 2, 1(\times 7)$	$16, 14, 14, 11, 10, 9, 8(\times 4), 7, 6(\times 5), 5, 5, 4(\times 14), 3(\times 15), 2(\times 28), 1(\times 64)$	41	139	374	✓
M_PL_029	$43, 22, 14, 12(\times 4), 10(\times 5), 9, 8, 8, 7(\times 6), 6(\times 4), 5(\times 5), 4(\times 8), 3(\times 3), 2(\times 4), 1(\times 4)$	$26, 26, 17, 11(\times 3), 9(\times 3), 8, 7, 7, 6(\times 3), 5(\times 4), 4(\times 5), 3(\times 13), 2(\times 17), 1(\times 64)$	49	118	346	✓
M_PL_030	$16, 10, 6, 6, 5, 5, 4(\times 5), 3(\times 9), 2(\times 6), 1, 1$	$8, 7, 6, 4(\times 4), 3(\times 6), 2(\times 14), 1(\times 26)$	28	53	109	✓
M_PL_031	$11, 8, 7(\times 6), 6, 5, 4(\times 5), 3(\times 7), 2(\times 17), 1(\times 9)$	$17, 13, 10, 9, 8, 7, 7, 6, 5(\times 3), 4, 4, 3(\times 7), 2(\times 6), 1(\times 23)$	48	49	156	✓
M_PL_032	$23, 20, 11, 6, 2, 2, 1$	$6, 4(\times 3), 3(\times 7), 2(\times 4), 1(\times 18)$	7	33	65	✓
M_PL_034	$47, 38, 34, 29, 22, 18, 17, 16, 15, 11, 8, 8, 7, 7, 5, 5, 4, 4, 3, 2(\times 7)$	$21, 18, 13, 10, 9, 7, 7, 6(\times 4), 5(\times 5), 4(\times 5), 3(\times 11), 2(\times 29), 1(\times 67)$	26	128	312	✓
M_PL_035	$31, 19, 17, 8, 8, 7, 6, 6, 5, 5, 4(\times 5), 3(\times 10), 2(\times 5), 1(\times 6)$	$13, 12, 11, 9, 8, 7, 6, 6, 5, 4(\times 9), 3(\times 6), 2(\times 10), 1(\times 27)$	36	61	178	✓
M_PL_036	$8, 4, 3(\times 3), 2(\times 4), 1$	$6, 4, 4, 3(\times 3), 2, 1(\times 5)$	10	12	30	✓
M_PL_037	$17, 9, 9, 8, 6(\times 3), 5, 5, 1$	$7, 4(\times 4), 3(\times 4), 2(\times 6), 1(\times 25)$	10	40	72	✓
M_PL_038	$17, 15, 13, 11, 9, 6, 6, 2$	$5, 4(\times 3), 3(\times 7), 2(\times 10), 1(\times 21)$	8	42	79	✓
M_PL_039	$17, 14, 12(\times 3), 10, 9, 6, 6, 5(\times 3), 4, 4, 3, 3, 2$	$13, 13, 8, 6, 6, 5(\times 3), 4, 4, 3(\times 5), 2(\times 9), 1(\times 27)$	17	51	129	✓
M_PL_042	$11, 6, 5, 1(\times 3)$	$4, 3(\times 5), 1(\times 6)$	6	12	25	✓
M_PL_043	$31, 17, 15, 15, 14(\times 4), 13, 11, 10, 9, 9, 7(\times 4), 6(\times 3), 4, 3(\times 3), 2, 1(\times 3)$	$18, 13, 10, 9, 7(\times 3), 6(\times 8), 5(\times 4), 4(\times 6), 3(\times 8), 2(\times 14), 1(\times 35)$	28	82	250	✓

Identifier	Row Sums $\mathbf{r}$	Column Sums $\mathbf{c}$	m	n	$\lvert E\rvert$	Prim.
M_PL_046	$30, 27, 25, 25, 23, 20, 20, 18, 18, 15, 13, 12, 11, 10, 7, 4$	$16, 15, 15, 14, 14, 12, 11, 10, 10, 9(\times 4), 8, 7(\times 5), 6(\times 6), 5, 5, 4(\times 4), 3, 2(\times 5), 1(\times 7)$	16	44	278	✗
M_PL_047	$75, 70, 54, 42, 32, 29, 22, 17, 17, 15, 11, 10, 8, 8, 6, 3, 3, 2, 1$	$13, 11, 9(\times 5), 8(\times 4), 7(\times 3), 6(\times 4), 5(\times 6), 4(\times 8), 3(\times 12), 2(\times 39), 1(\times 103)$	19	186	425	✓
M_PL_048	$75, 74, 73, 63, 48, 44, 31, 26, 26, 25, 23, 20, 20, 19, 18, 17, 15, 11, 10, 7, 5, 4(\times 3), 3, 2, 1(\times 4)$	$20, 16, 15, 13(\times 3), 11, 10(\times 4), 9, 9, 8, 7(\times 7), 6(\times 10), 5(\times 11), 4(\times 22), 3(\times 20), 2(\times 40), 1(\times 112)$	30	236	671	✓
M_PL_049	$80, 53, 34, 32, 27, 25, 23, 23, 21, 21, 19, 16(\times 4), 14, 14, 12(\times 3), 11, 10, 9, 9, 8, 8, 7(\times 4), 6, 5, 4, 2, 2, 1, 1$	$24, 24, 22, 18, 13, 12, 11, 11, 10, 9, 9, 8, 7(\times 7), 6(\times 5), 5(\times 6), 4(\times 13), 3(\times 19), 2(\times 38), 1(\times 125)$	37	225	590	✓
M_PL_050	$17, 11, 10, 10, 8, 8, 6, 5, 4, 2, 2, 1(\times 3)$	$8, 7, 7, 6, 5, 4, 3(\times 6), 2(\times 8), 1(\times 15)$	14	35	86	✓
M_PL_052	$20, 12, 10, 8, 6, 6, 5, 5, 4, 4, 3(\times 3), 2, 1$	$9, 9, 6(\times 4), 3(\times 4), 2(\times 9), 1(\times 20)$	15	39	92	✓
M_PL_053	$43, 31, 28, 26, 19, 16, 15, 15, 13, 12(\times 5), 11(\times 3), 10, 10, 9(\times 4), 8(\times 5), 7(\times 4), 6(\times 4), 5(\times 5), 4(\times 9), 3(\times 11), 2(\times 10), 1(\times 28)$	$16, 15, 15, 13, 11, 10, 10, 9, 8(\times 3), 7(\times 4), 6(\times 4), 5(\times 10), 4(\times 12), 3(\times 13), 2(\times 37), 1(\times 203)$	99	294	589	✓

Identifier	Row Sums $\mathbf{r}$	Column Sums $\mathbf{c}$	m	n	$\lvert E\rvert$	Prim.
M_PL_062	$58, 54, 54, 53, 53, 52, 51(\times 6), 50(\times 4), 49(\times 5), 48(\times 3), 47(\times 6), 46(\times 8), 45(\times 7), 44(\times 7), 43(\times 8), 42(\times 9), 41(\times 18), 40(\times 9), 39(\times 11), 38(\times 16), 37(\times 20), 36(\times 15), 35(\times 19), 34(\times 21), 33(\times 26), 32(\times 24), 31(\times 32), 30(\times 26), 29(\times 27), 28(\times 24), 27(\times 23), 26(\times 19), 25(\times 11), 24(\times 14), 23(\times 5), 22(\times 7), 21(\times 11), 20(\times 4), 18(\times 3), 17, 17$	$157, 151, 121, 119, 104, 102, 102, 100, 99, 97, 96, 95, 91, 90, 90, 86, 85, 83, 82, 80, 80, 78(\times 3), 76, 76, 74, 74, 71, 71, 70(\times 3), 69, 67, 67, 66, 66, 65, 65, 63, 62, 61, 61, 60, 60, 59(\times 4), 57, 57, 56(\times 3), 55, 55, 54, 53, 50, 50, 49(\times 7), 48(\times 5), 47, 47, 46(\times 3), 45(\times 3), 44, 43(\times 3), 42(\times 3), 41(\times 6), 40(\times 3), 39(\times 5), 38(\times 4), 37(\times 6), 36(\times 7), 35(\times 4), 34(\times 10), 33, 33, 32(\times 7), 31(\times 6), 30(\times 9), 29(\times 5), 28(\times 9), 27(\times 6), 26(\times 9), 25(\times 8), 24(\times 7), 23(\times 8), 22(\times 11), 21(\times 14), 20(\times 6), 19(\times 8), 18(\times 16), 17(\times 21), 16(\times 14), 15(\times 22), 14(\times 23), 13(\times 27), 12(\times 18), 11(\times 20), 10(\times 29), 9(\times 29), 8(\times 31), 7(\times 43), 6(\times 43), 5(\times 59), 4(\times 63), 3(\times 66), 2(\times 94), 1(\times 178)$	456	1044	15255	✓
M_SD_007	$48, 48, 22, 12, 8, 4, 1$	$5, 4(\times 5), 3(\times 17), 2(\times 18), 1(\times 31)$	7	72	143	✓
M_SD_011	$12, 5(\times 4), 4, 3, 3, 2, 2, 1$	$7, 7, 4(\times 3), 3(\times 4), 2(\times 4), 1$	11	14	47	✓
M_SD_013	$17, 16, 16, 15, 15, 14, 12, 11, 10, 10, 9(\times 3), 8, 8, 7, 5, 3, 3$	$16, 15, 11, 10, 10, 9, 9, 8(\times 3), 7(\times 3), 6, 6, 5, 5, 4(\times 7), 3, 3, 2(\times 6), 1(\times 4)$	19	36	197	✓
M_SD_014	$17, 13, 12, 11, 10, 9, 7, 7, 6, 5(\times 3), 4, 4, 3, 3$	$15, 15, 13, 13, 10, 9, 7, 6, 5, 5, 4(\times 3), 3(\times 3), 2$	16	17	121	✗
M_SD_015	$25, 19, 18, 16, 8$	$5(\times 4), 4(\times 6), 3(\times 9), 2(\times 7), 1$	5	27	86	✗

Identifier	Row Sums $\mathbf{r}$	Column Sums $\mathbf{c}$	m	n	$\lvert E\rvert$	Prim.
M_SD_016	$34, 32, 30, 30, 29, 29, 27, 26, 25, 25, 24, 23, 22, 22, 16, 16, 15, 14, 13, 11(\times 3), 9, 6$	$21(\times 3), 20(\times 3), 19, 18, 17(\times 3), 16, 14(\times 3), 13, 12(\times 4), 11(\times 3), 9(\times 4), 8, 7(\times 3), 6(\times 3), 5(\times 3), 3(\times 4), 2(\times 7), 1(\times 13)$	24	61	500	✓
M_SD_017	$12, 11, 11, 10, 9, 7, 7, 5$	$7, 7, 6, 6, 5(\times 4), 4(\times 3), 3(\times 4), 2$	8	16	72	✓
M_SD_018	$7, 7, 6, 6, 5, 3(\times 3), 2(\times 5), 1(\times 16)$	$8, 7, 6, 4, 3, 3, 2(\times 9), 1(\times 17)$	29	32	66	✓
M_SD_019	$96, 51, 44, 40, 38, 31, 30, 29, 26, 20, 16, 14(\times 3), 13(\times 4), 12, 11, 10, 10, 9, 9, 8, 8, 7(\times 4), 6(\times 3), 4(\times 7)$	$30, 21, 20, 19, 17, 16, 15, 14, 14, 12, 12, 11, 11, 10(\times 3), 9, 9, 8(\times 4), 7(\times 5), 6(\times 6), 5(\times 14), 4(\times 19), 3(\times 14), 2(\times 26), 1(\times 63)$	40	169	666	✓
M_SD_021	$22, 20, 14, 9, 9, 8, 8, 7, 5(\times 3), 4, 3(\times 3), 2, 1, 1$	$12, 12, 10, 10, 8, 7, 7, 6, 6, 5(\times 4), 4, 3(\times 4), 2(\times 5), 1(\times 5)$	18	28	129	✓
M_SD_022	$57, 48, 45, 41, 38, 37, 37, 36, 29, 24, 24, 23, 23, 22(\times 4), 19, 18(\times 3), 17, 16, 16, 15, 15, 14(\times 6), 13(\times 3), 12, 12, 11, 11, 10(\times 3), 9, 8(\times 4), 7(\times 3), 6(\times 4), 5(\times 8), 4(\times 6), 3(\times 10), 2(\times 8), 1(\times 24)$	$36, 32, 29, 29, 27, 26, 26, 22, 20, 19, 19, 18, 17, 17, 16(\times 3), 14, 14, 13(\times 3), 12, 12, 11(\times 5), 10(\times 3), 9(\times 4), 8(\times 7), 7(\times 8), 6(\times 10), 5(\times 20), 4(\times 13), 3(\times 28), 2(\times 31), 1(\times 54)$	110	207	1121	✓
M_SD_024	$10, 7, 6, 6, 4, 4, 3$	$5(\times 3), 4(\times 3), 3(\times 3), 2, 1, 1$	7	12	40	✓
M_SD_025	$7, 4, 3(\times 3), 2$	$5, 4(\times 3), 2, 2, 1$	6	7	22	✗
M_SD_026	$3, 2, 1$	$3, 2, 1$	3	3	6	✗
M_SD_027	$11, 8, 6, 6$	$4(\times 4), 3, 3, 2(\times 3), 1(\times 3)$	4	12	31	✗
M_SD_028	$8, 6, 6, 4, 2$	$5, 5, 4, 4, 3, 2, 2, 1$	5	8	26	✗
M_SD_029	$5, 3, 1, 1$	$4, 2, 2, 1, 1$	4	5	10	✗
M_SD_030	$4, 3, 2, 2$	$4, 4, 1(\times 3)$	4	5	11	✗

Appendix C

Auxiliary Functions

Here, we define the auxiliary functions used in Chapters 4 and 5. While the first two metrics are used in a multitude of applications, the latter functions are measures of *nestedness*. In ecology, such metrics are used to quantify the structure of a network.

Definition C.1 (Hamming distance)**.** *If $A = (a_{ij})$ and $B = (b_{ij})$ are binary matrices of size $m \times n$, the* Hamming distance $h\colon \{0,1\}^{m\times n} \times \{0,1\}^{m\times n} \to \mathbb{N}$ *is defined as*

$$h(A,B) := \sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij} \oplus b_{ij},$$

where $\oplus$ is the exclusive or *operator.*

Definition C.2 (`std::hash`)**.** *If $M = (m_{ij})$ is a binary matrix of size $m \times n$, we define $v = (v_1, v_2, \ldots, v_{mn}) \in \{0,1\}^{mn}$ to be a 0-1-vector defined by*

$$v_k := m_{ij},$$

where $i := \lfloor k/m \rfloor$ and $j := k \mod m$. Then, `std::hash`(M) returns the hash value of the standard C++ hash function applied to v.

Definition C.3 ($\bar{S}^2$ statistic [77])**.** *Let M be a binary matrix of size $m \times n$ and let $S := (s_{ij}) = MM^T$. Then,*

$$\bar{S}^2(M) := \binom{m}{2}^{-1} \sum_{i<j} s_{ij}^2.$$

Definition C.4 (nested subset statistic [48])**.** *Let M be a binary matrix of size $m\times n$. Then, $S_{\text{nest}}(G) = \sum_{i,j} x_{ij}$, where*

$$x_{ij} = \begin{cases} 1, & \text{if } \{u_i, v_j\} \notin E \wedge \delta_G(v_j) > \min\{\delta_G(v_k)\colon \{u_i, v_k\} \in E\} \\ 0, & \text{otherwise.} \end{cases}$$

Definition C.5 (NODF [78])**.** *Let M be a binary matrix of size $m \times n$ with row sums $\mathbf{r} = (r_1, r_2, \ldots, r_m)$ and column sums $\mathbf{c} = (c_1, c_2, \ldots, c_n)$. Then,*

$$NODF(M) := \frac{p+q}{\binom{m}{2} + \binom{n}{2}} = \frac{2(p+q)}{m(m-1) + n(n-1)},$$

where p and q are real numbers calculated by the following formulae.

$$p := \sum_{i=1}^{m} \sum_{k=i+1}^{m} f(i,k) \qquad \text{and} \qquad q := \sum_{j=1}^{n} \sum_{l=j+1}^{n} g(j,l),$$

where $f\colon \mathbb{N} \times \mathbb{N} \to \mathbb{R}$ and $g\colon \mathbb{N} \times \mathbb{N} \to \mathbb{R}$ are functions defined by

$$f(i,k) := \begin{cases} 0, & \text{if } r_i \leq r_k \\ \frac{1}{r_k} \sum_{j=1}^{n} m_{ij} m_{kj}, & \text{else.} \end{cases}$$

and

$$g(j,l) := \begin{cases} 0, & \text{if } c_j \leq c_l \\ \frac{1}{c_l} \sum_{i=1}^{m} m_{ij} m_{il}, & \text{else.} \end{cases}$$

Definition C.6 (checker board score)**.** *Let $M = (m_{ij})$ be a binary matrix of size $m \times n$. Then, the* checker board score *$c\colon \{0,1\}^{m \times n} \to \mathbb{N}$ is defined as the number of 2×2 sub-matrices of type*

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad \text{or} \qquad \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad \text{i.e.}$$

$$c(M) := \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{m} \sum_{l=1}^{n} f(i,j,k,l),$$

where $f\colon \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \{0,1\}$ is defined by

$$f(i,j,k,l) := \begin{cases} 1, & \text{if } m_{ij} = m_{kl} \neq m_{kj} = m_{il}, \\ 0, & \text{else.} \end{cases}$$

Definition C.7 (Spectral Radius [79])**.** *Let $M = (m_{ij})$ be a binary matrix of size $m \times n$ and let $k := m + n$. Then, the $k \times k$ matrix*

$$M' := \begin{pmatrix} 0 & M \\ M^T & 0 \end{pmatrix}$$

is a symmetric matrix with real eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_k$. Then, the spectral radius *$sr\colon \{0,1\}^{m \times n} \to \mathbb{R}^+$ is defined by*

$$sr(M) := \max\{|\lambda_1|, |\lambda_k|\}.$$

Appendix D

Additional Figures

This appendix contains additional figures from Chapters 4 and 5. Most of these figures contain additional information that is less essential for our findings, but is still relevant for our conclusions. To improve the readability of the thesis, we decided to list such figures here.

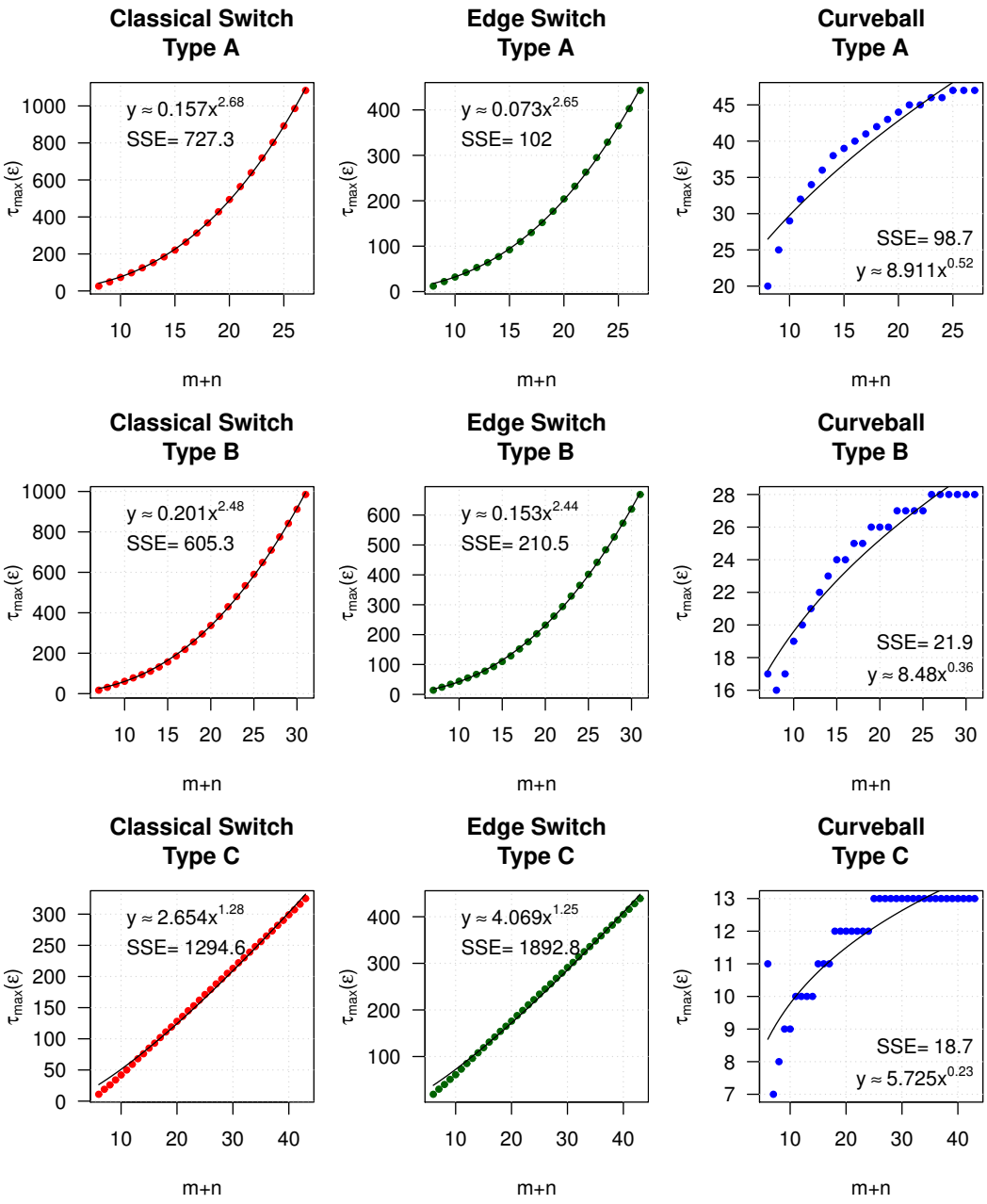

Figure D.1: Estimated polynomials for the total mixing time of the classical switch chain (red), edge switch chain (green), and the curveball chain (blue) on the instance classes A, B, and C. ($\varepsilon = 0.01$)

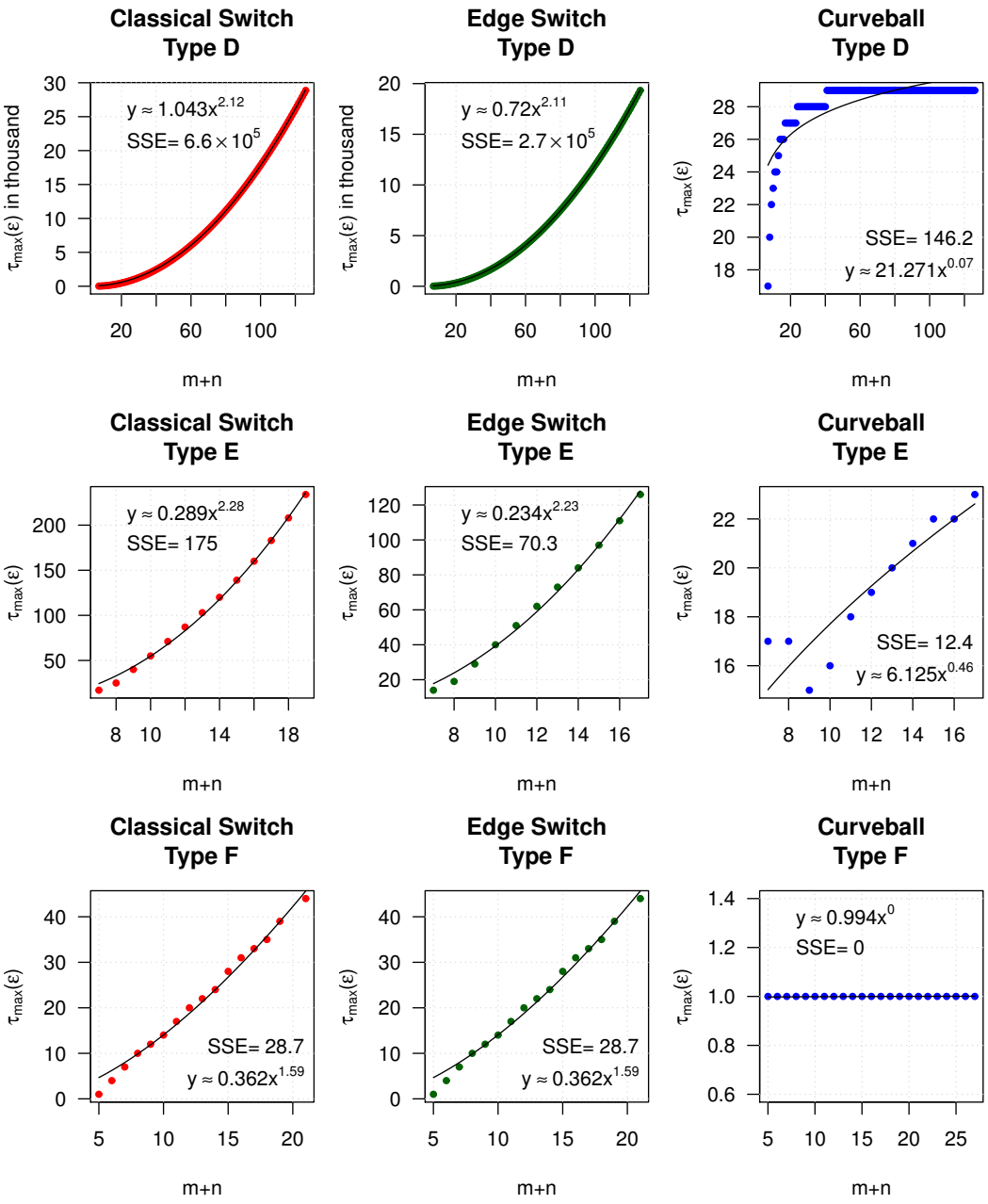

Figure D.2: Estimated polynomials for the total mixing time of the classical switch chain (red), edge switch chain (green), and the curveball chain (blue) on the instance classes D, E, and F. ($\varepsilon = 0.01$)

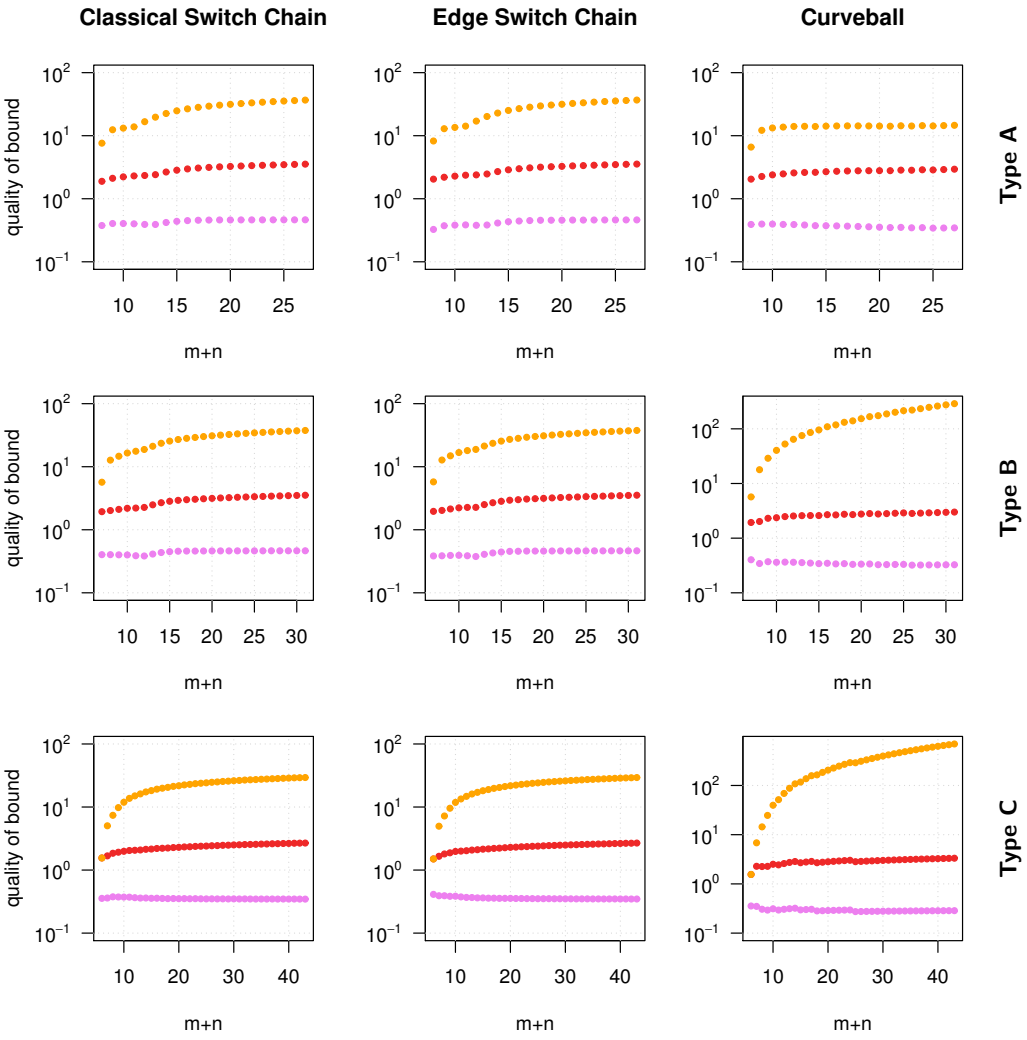

Figure D.3: Quotient of congestion bound (orange), upper spectral bound (red), and lower spectral bound (violet) with total mixing time for the *scalable* instance classes A,B, and C.

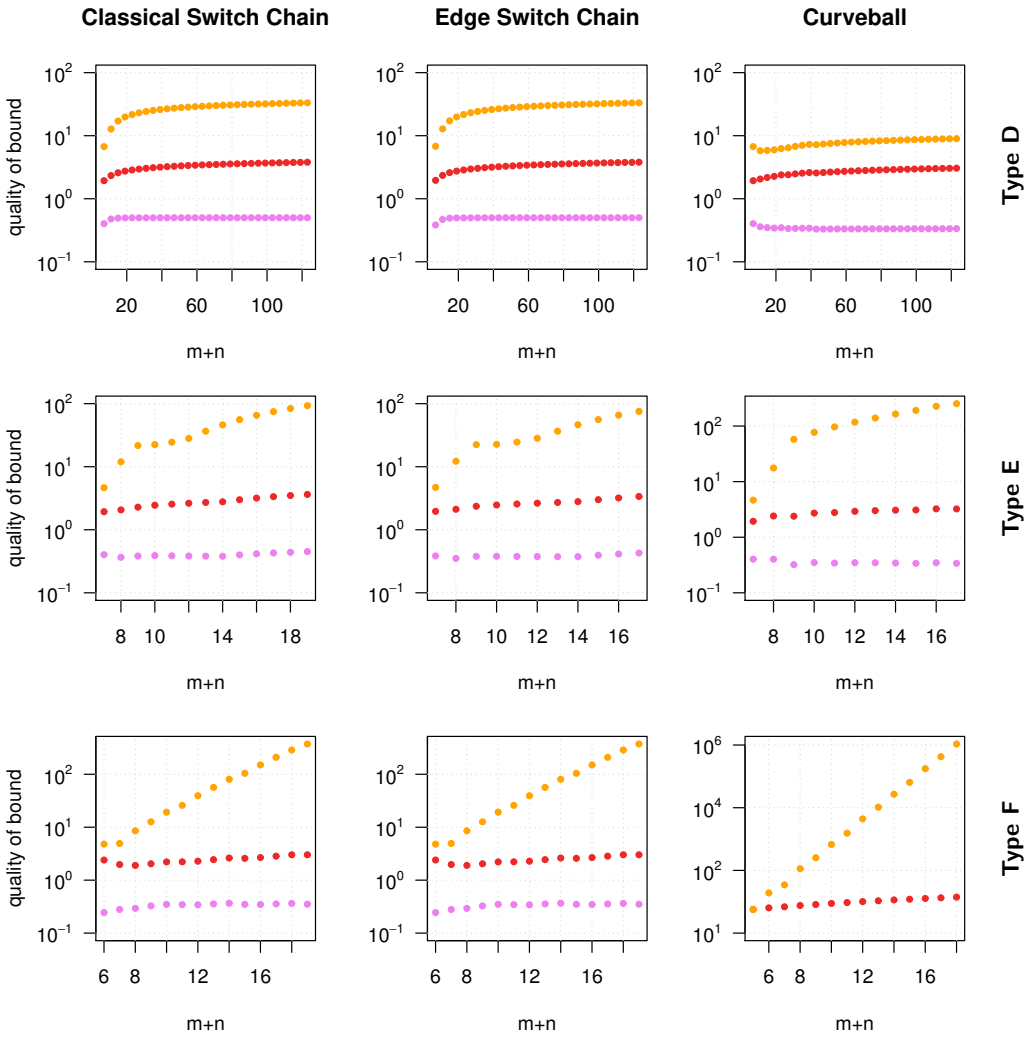

Figure D.4: Quotient of congestion bound (orange), upper spectral bound (red), and lower spectral bound (violet) with total mixing time for the *scalable* instance classes D,E, and F.

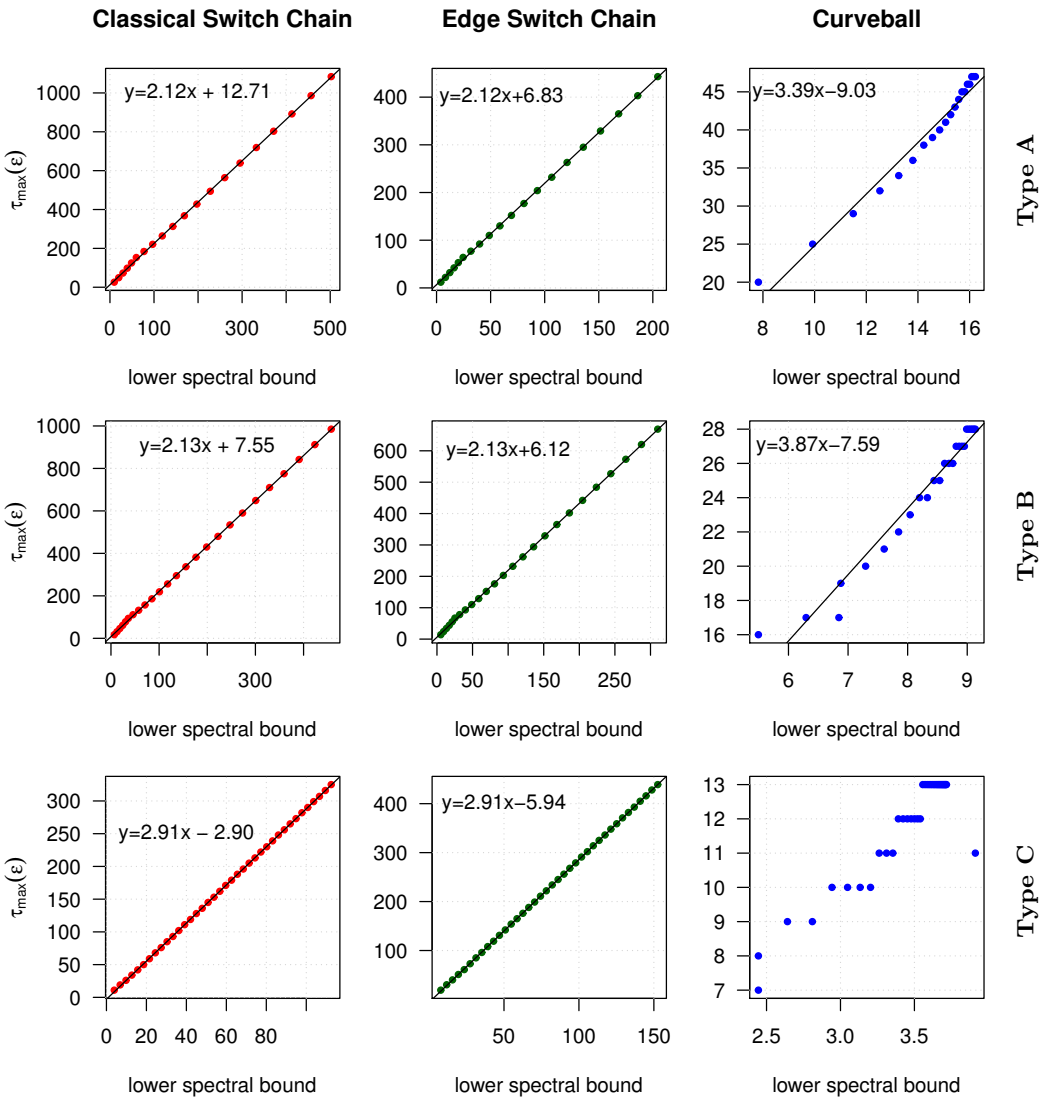

Figure D.5: Relationship between the lower spectral bound and the total mixing time for the *scalable* instance classes A,B, and C. ($\varepsilon = 0.01$.)

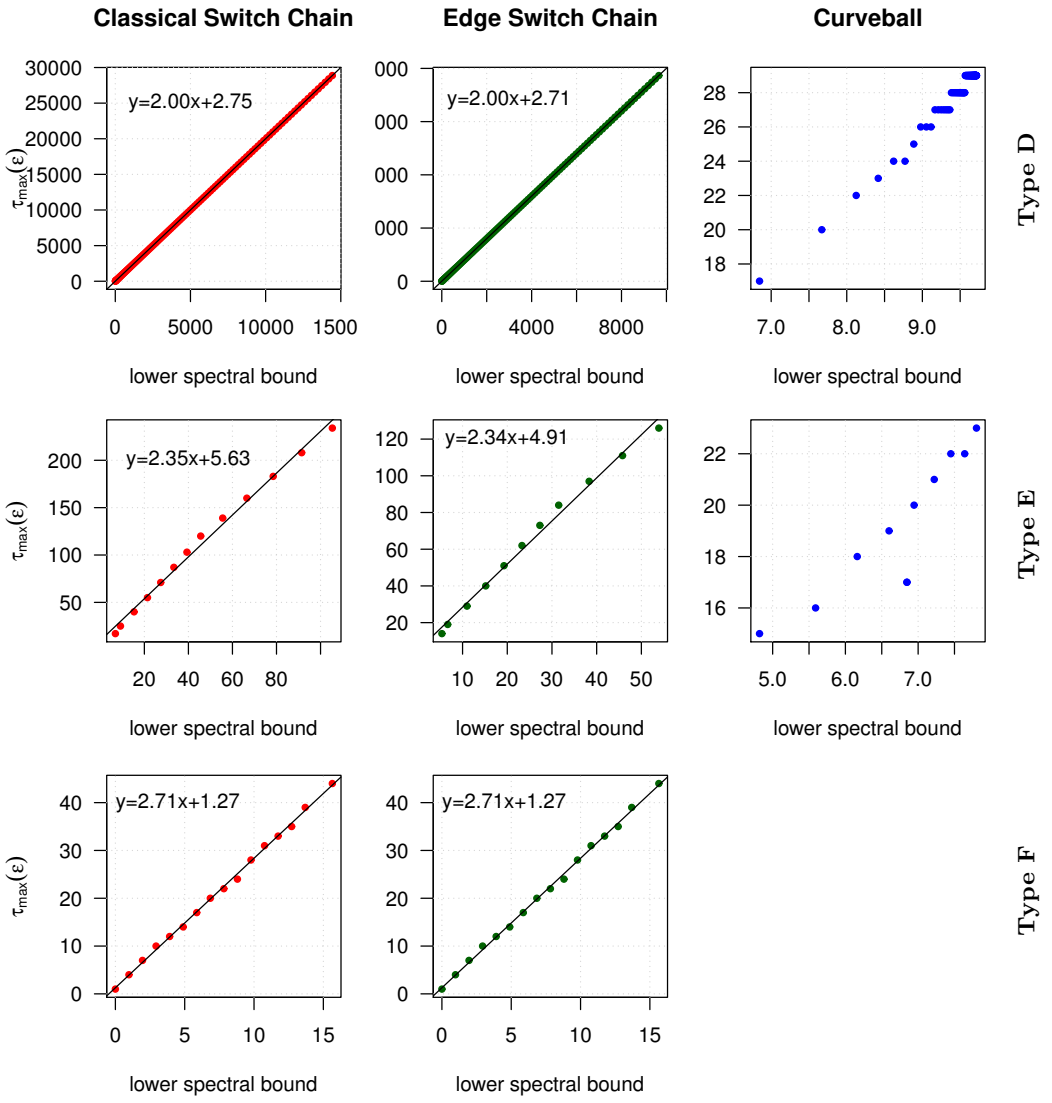

Figure D.6: Relationship between the lower spectral bound and the total mixing time for the *scalable* instance classes D,E, and F. The figure for the curveball Markov chain is missing for class F, as the associated total mixing time is one. ($\varepsilon = 0.01$.)

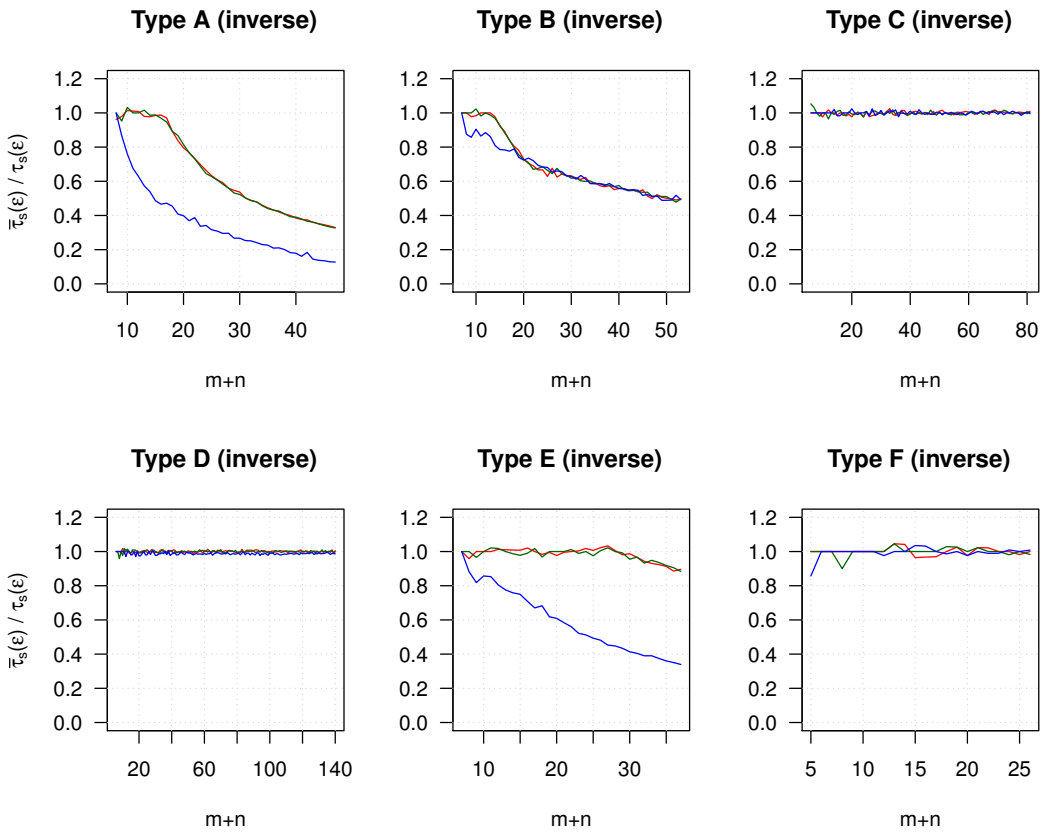

Figure D.7: Quality of applied mixing time $\bar{\tau}_s(\varepsilon)$ for the scalable instance classes in which the roles of **r** and **c** are inversed with respect to the Hamming distance metric. Red: classical switch chain. Green: edge switch chain. Blue: curveball chain. ($\varepsilon = 0.01$, $N = 10^6$ samples).

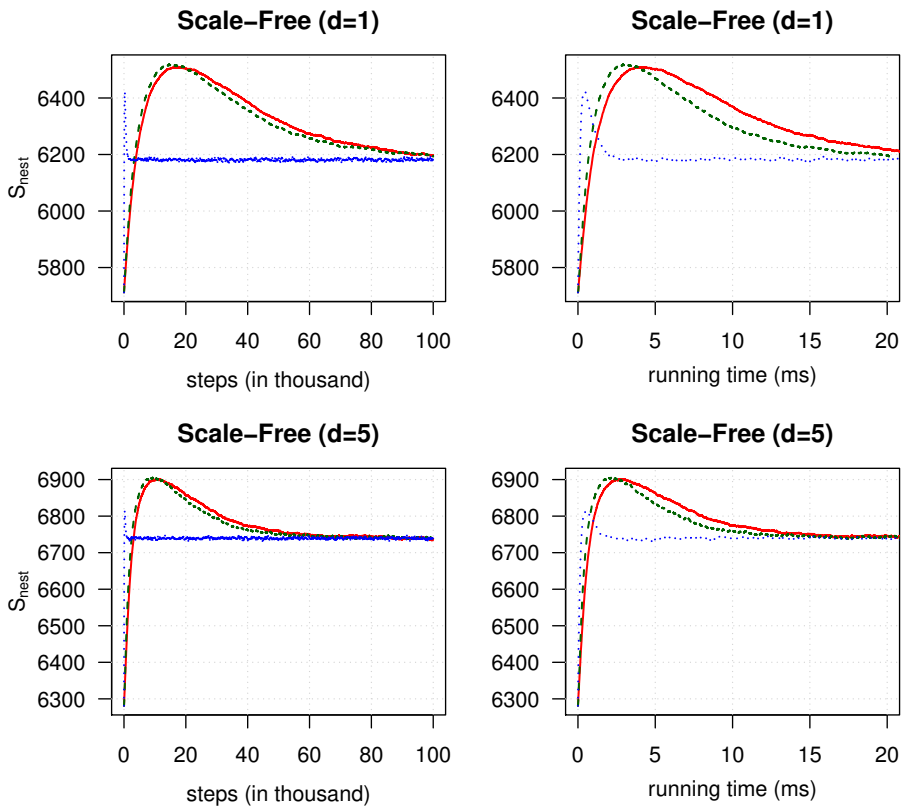

Figure D.8: Sample means of the S_{nest} metric for scale-free integer vectors. Red, solid line: static simple chain. Green, dashed line: dynamic simple chain. Blue, dotted line: static informed chain.

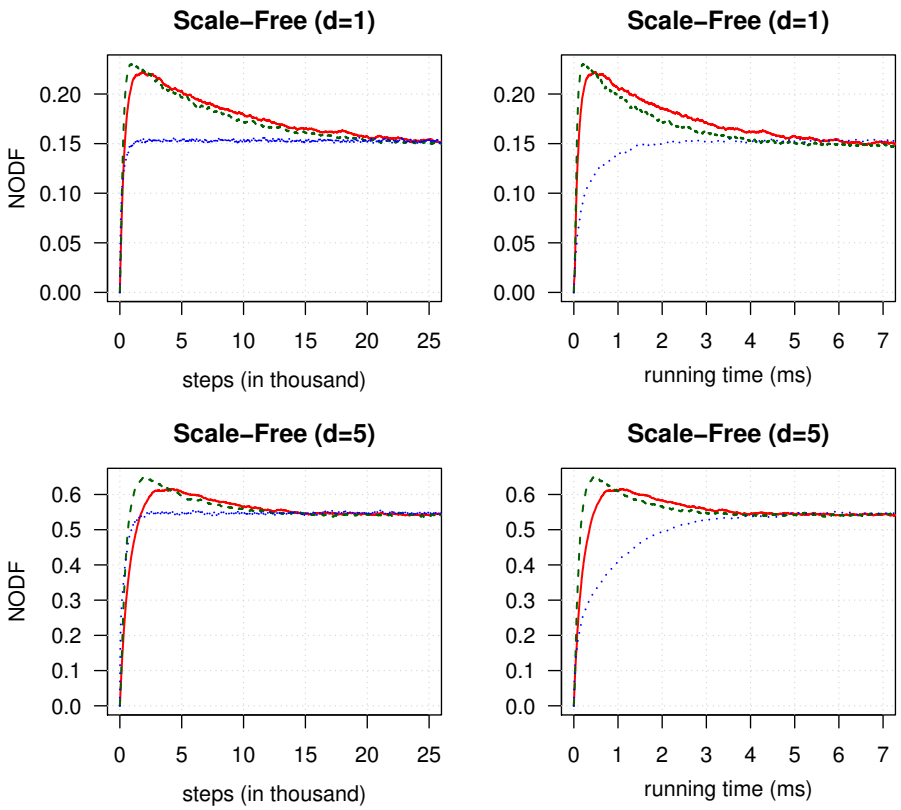

Figure D.9: Sample means of the NODF metric for scale-free integer vectors. Red, solid line: static simple chain. Green, dashed line: dynamic simple chain. Blue, dotted line: static informed chain.

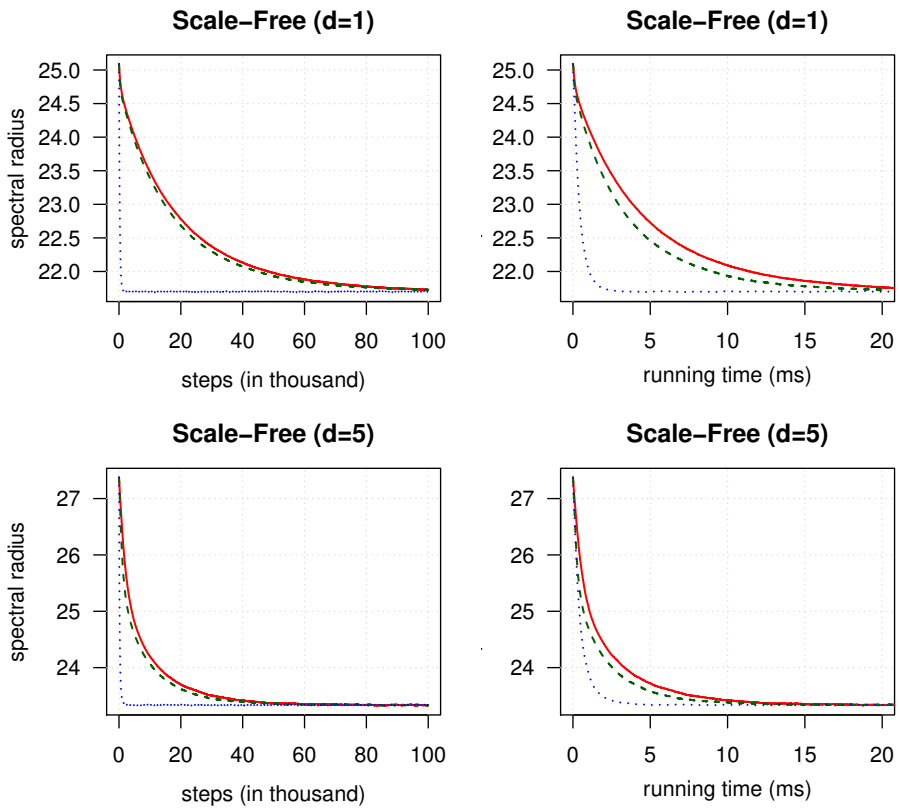

Figure D.10: Sample means of the *spectral radius* metric for scale-free integer vectors. Red, solid line: static simple chain. Green, dashed line: dynamic simple chain. Blue, dotted line: static informed chain.

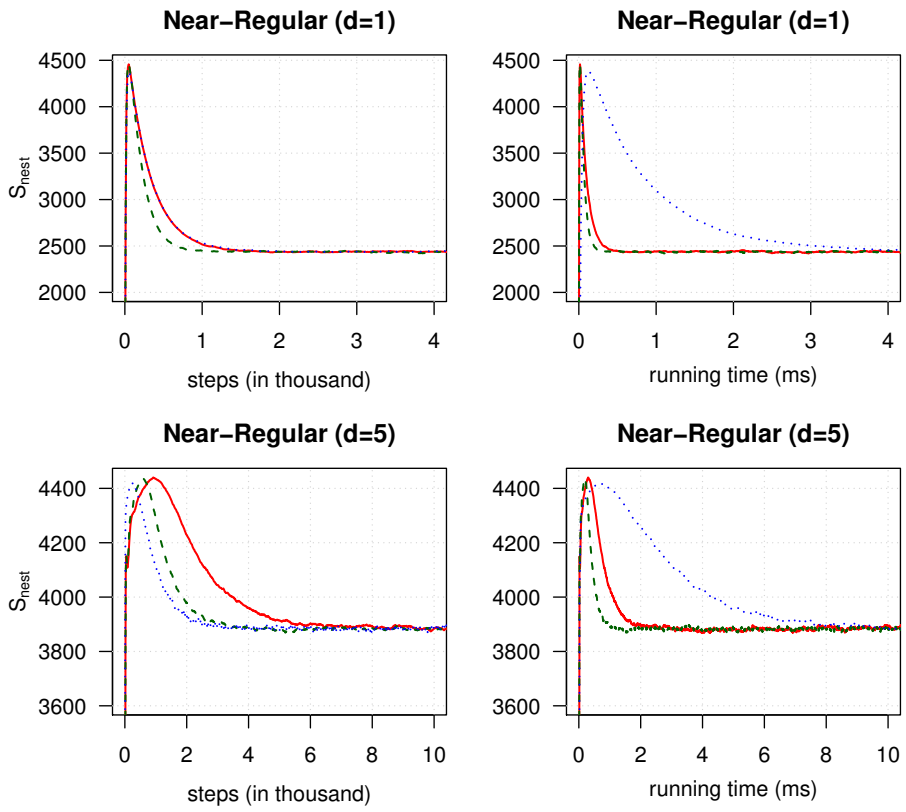

Figure D.11: Sample means of the S_{nest} metric for scale-free integer vectors. Red, solid line: static simple chain. Green, dashed line: dynamic simple chain. Blue, dotted line: static informed chain.

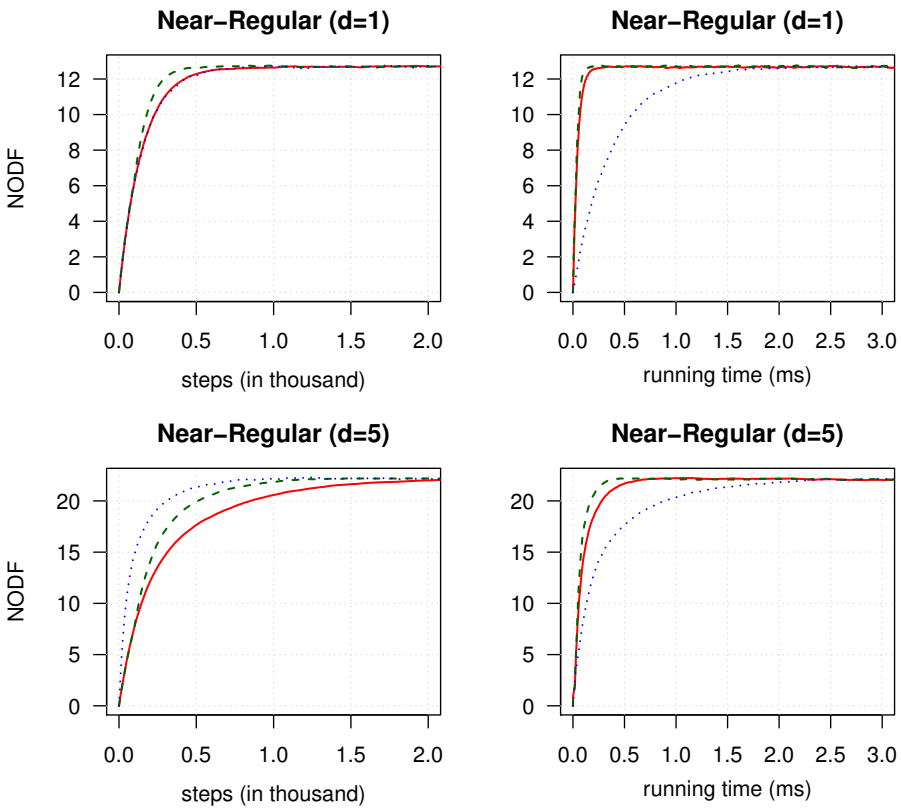

Figure D.12: Sample means of the NODF metric for scale-free integer vectors. Red, solid line: static simple chain. Green, dashed line: dynamic simple chain. Blue, dotted line: static informed chain.

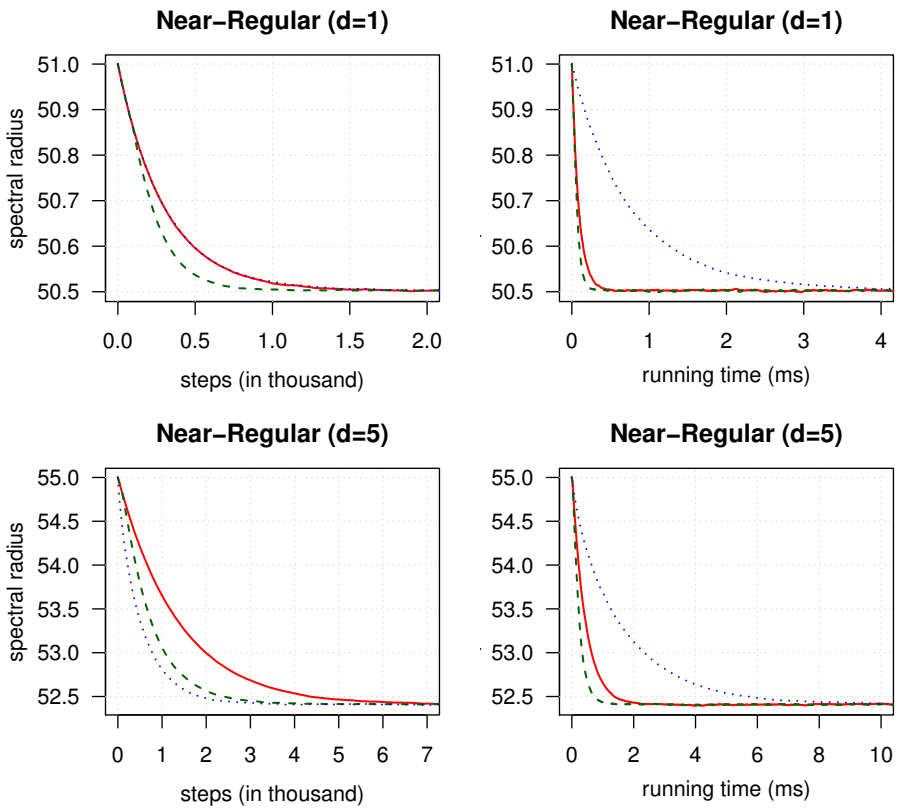

Figure D.13: Sample means of the *spectral radius* metric for scale-free integer vectors. Red, solid line: static simple chain. Green, dashed line: dynamic simple chain. Blue, dotted line: static informed chain.

Eidesstattliche Erklärung / *Declaration under Oath*

Ich erkläre an Eides statt, dass ich die Arbeit selbstständig und ohne fremde Hilfe verfasst, keine anderen als die von mir angegebenen Quellen und Hilfsmittel benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

I declare under penalty of perjury that this thesis is my own work entirely and has been written without any help from other people. I used only the sources mentioned and included all the citations correctly both in word or content.

Halle, den 14.06.2018

Steffen Rechner

Datum / *Date*

Unterschrift des Antragstellers / *Signature of the applicant*

Steffen Rechner

Persönliche Daten

Wohnsitz	Wilhelm-Schrader-Straße 25 06120 Halle (Saale)
Geburtsdatum	02.12.1986
Geburtsort	Bad Frankenhausen
Nationalität	Deutsch

Bildungsweg

1999-2006	**Allgemeine Hochschulreife** Geschwister-Scholl-Gymnasium Sangerhausen
2007-2010	**Bachelor of Science** (Informatik) Martin-Luther-Universität Halle-Wittenberg Halle (Saale)
2010-2013	**Master of Science** (Informatik) Martin-Luther-Universität Halle-Wittenberg Halle (Saale)

Bachelorarbeit

Titel	*Der globale Rang eines Flughafens im Wandel der Zeit: Experimentelle Analyse der tageszeitabhängigen Zentralitäten im Flugverkehr*
Betreuer	Prof. Dr. rer. nat. Matthias Müller-Hannemann Annabell Berger

Masterarbeit

Titel	*Vergleich und experimentelle Analyse von Verfahren zum Erzeugen zufälliger perfekter Matchings*
Betreuer	Prof. Dr. rer. nat. Matthias Müller-Hannemann Dr. rer. nat. Annabell Berger

Publikationen

2011 A. Berger, M. Müller-Hannemann, S. Rechner, and A. Zock. "Efficient Computation of Time-Dependent Centralities in Air Transportation Networks". *WALCOM: Algorithms and Computation*. Ed. by N. Katoh and A. Kumar. Springer Berlin Heidelberg (2011), 77–88. DOI: 10.1007/978-3-642-19094-0_10.

2014 A. Berger and S. Rechner. *Broder's Chain Is Not Rapidly Mixing*. arXiv preprint (2014). arXiv: 1404.4249v1 [cs.DM].

2014 M. Lemnian, R. Rückert, S. Rechner, C. Blendinger, and M. Müller-Hannemann. "Timing of Train Disposition: Towards Early Passenger Rerouting in Case of Delays". *14th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*. Ed. by S. Funke and M. Mihalák. Vol. 42. OpenAccess Series in Informatics (OASIcs). Schloss Dagstuhl–Leibniz-Zentrum für Informatik (2014), 122–137. DOI: 10.4230/OASIcs.ATMOS.2014.122.

2016 S. Rechner and A. Berger. *Marathon: An open source software library for the analysis of Markov-Chain Monte Carlo algorithms*. PLOS ONE **11** (2016). DOI: 10.1371/journal.pone.0147935.

2017 R. Rückert, M. Lemnian, C. Blendinger, S. Rechner, and M. Müller-Hannemann. *PANDA: a software tool for improved train dispatching with focus on passenger flows*. Public Transport **9** (July 2017), 307–324. DOI: 10.1007/s12469-016-0140-0.

2017 M. Erbert, S. Rechner, and M. Müller-Hannemann. *Gerbil: a fast and memory-efficient k-mer counter with GPU-support*. Algorithms for Molecular Biology **12** (Mar. 2017), 9. DOI: 10.1186/s13015-017-0097-9.

2017 S. Rechner, L. Strowick, and M. Müller-Hannemann. *Uniform sampling of bipartite graphs with degrees in prescribed intervals*. Journal of Complex Networks (2017). DOI: 10.1093/comnet/cnx059.

2017 S. Rechner. *An Optimal Realization Algorithm for Bipartite Graphs with Degrees in Prescribed Intervals*. arXiv preprint (2017). arXiv: 1708.05520v1 [cs.DS].

Halle, den 14.06.2018

Steffen Rechner

Datum / *Date*

Unterschrift des Antragstellers / *Signature of the applicant*