Exact Sampling with Markov Chains

by

David Bruce Wilson

S.B., Electrical Engineering, Massachusetts Institute of Technology (1991)
S.B., Mathematics, Massachusetts Institute of Technology (1991)
S.B., Computer Science, Massachusetts Institute of Technology (1991)

Submitted to the Department of Mathematics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Mathematics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1996

Authoı . . . .
Department of Mathematics
May 28, 1996

Certified b ....
James G. Propp
Assistant Professor
Thesis Supervisor

Accepted t ....
Richard P. Stanley
Chairman of the Applied Mathematics Committee

Accepted by ....
David A. Vogan
C mitte on Graduate Students

Exact Sampling with Markov Chains

by

David Bruce Wilson

Submitted to the Department of Mathematics
on May 28, 1996, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Mathematics

Abstract

Random sampling has found numerous applications in computer science, statistics, and physics. The most widely applicable method of random sampling is to use a Markov chain whose steady state distribution is the probability distribution π from which we wish to sample. After the Markov chain has been run for long enough, its state is approximately distributed according to π. The principal problem with this approach is that it is often difficult to determine how long to run the Markov chain. In this thesis we present several algorithms that use Markov chains to return samples distributed exactly according to π. The algorithms determine on their own how long to run the Markov chain. Two of the algorithms may be used with any Markov chain, but are useful only if the state space is not too large. Nonetheless, a spin-off of these two algorithms is a procedure for sampling random spanning trees of a directed graph that runs more quickly than the Aldous/Broder algorithm. Another of the exact sampling algorithms presented here is much more efficient and may be used on Markov chains with huge state spaces, provided that they have a suitable special structure. Many Markov chains of practical interest have this structure, including some from statistical mechanics, and this exact sampling algorithm has already been implemented by the author and others.

One example from physics is the random cluster model, which generalizes the Ising and Potts models. Of particular interest are the so-called critical exponents, which characterize the singular behavior of quantities such as the heat capacity and spontaneous magnetization at a certain critical temperature. In this thesis we also see how to use these sampling algorithms to generate (perfectly random) "omnithermal" random cluster samples. Given a single such sample, one can quickly read off a random Potts configuration at any prescribed temperature. We include preliminary results on the use of omnithermal samples to estimate the critical exponents.

Thesis Supervisor: James G. Propp
Title: Assistant Professor

Dedicated to the memories of Helen Elizabeth Hughes Wilson and Carrie Lee.

For Mom, Dad, and Janet.

Credits

Much of this thesis represents work that has appeared in conference proceedings and/or will appear in journal articles. Much of this research was done jointly with Jim Propp. Some of this research was done at Sandia National Laboratories, and it was supported in part by an ONR-NDSEG fellowship, NSF grant 9500936-DMS, NSA grant MDA904-92-H-3060, and Lockheed Martin-Marietta Corporation subcontract ST30G4230R. Chapter 2 comes from [1] and [2], chapter 3 and 4 come from [1], chapter 6 comes from [2], and chapter 7 comes from [3]. Conference articles [2] and [3] are being combined for a special issue of the Journal of Algorithms. Chapter 8 will be incorporated into an article by Rick Kenyon, Jim Propp, and myself. The larger simulations described in chapter 5 were done with the synthetic imagery group's SGIs.

Thanks to Persi Diaconis for useful advice — the suggestion that Jim and I work together on Markov chains seems to have been well worth taking. Thanks go to David Aldous, Henry Cohn, Persi Diaconis, Jim Fill, John Mount, Richard Tweedie, and Peter Winkler for interesting conversations. Thanks also to my thesis committee members David Karger and Persi Diaconis for taking an interest and reading prior drafts, and of course to Jim Propp for being pleasant to work with.

Bibliography

[1] James G. Propp and David B. Wilson. Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Structures and Algorithms*, 1996. To appear.

[2] David B. Wilson and James G. Propp. How to get an exact sample from a generic Markov chain and sample a random spanning tree from a directed graph, both within the cover time. In *Symposium on Discrete Algorithms*, pages 448–457, 1996.

[3] David B. Wilson. Generating random spanning trees more quickly than the cover time. In *Symposium on the Theory of Computing*, pages 296–303, 1996.

You need to study probability. It is your weak area.

— Charles E. Leiserson, summer 1991.

```
sinclair@icsi.berkeley.edu
random sample from Ising (hard?)
  sample subgraphs, works for many functionals
  connected component M.C. (lattice?)
pick random base from matroid, oracle for independence
random tree from undirected graph
random directed spanning tree from directed graph
```

— notes taken some years ago after discussing with Alistair Sinclair possible research topics.

Contents

List of Figures

Chapter 1

Introduction to Exact Sampling

1.1 Introduction

Random sampling of combinatorial objects has numerous applications in computer science and statistics. Random sampling algorithms often make use of Markov chains, that is they repeatedly apply randomizing operations that preserve the probability distribution π from which one wishes to sample. After enough randomizing operations, under mild assumptions, the probability distribution of the state of the system will approach the desired distribution. Despite much work at determining how many randomizing operations are "enough" [27] [47] [29] [62] [70] [28], this task remains quite difficult if not impossible.

In this thesis we give a number of novel random sampling algorithms that make use of Markov chains, run in finite time, and then output a state distributed exactly according to the desired distribution — the steady state distribution of the Markov chain. The principle advantage of this approach is that, when applicable, one can obtain the desired random samples, and be confident that they are indeed properly distributed, without first having to solve difficult math problems.

The first technique, called coupling from the past, works for any *monotone* Markov chain. When viewing a Markov chain as a sequence of randomizing operations, one can apply a particular randomizing operation to any number of states. Typically a monotone Markov chain will have two extremal states, extremal in the sense that if the same sequence of randomizing moves is applied to all possible initial states, and the two extremal states get mapped to the same value, then any other state necessarily gets mapped to this same value. (Refer to the glossary for a more detailed definition.) A surprisingly large number of interesting Markov chains satisfy this monotonicity condition, including several that have origins in statistical mechanics. One example is the Ising model, which was introduced to model magnetic substances (see Figure 1-1). In truth, coupling from the past (CFTP) works for *any* Markov chain, but the monotonicity condition makes it particularly efficient.

The second technique is called cycle popping, and it too works for any Markov chain. While cycle popping does not take advantage of monotonicity like CFTP does, for general Markov chains it is faster.

A generic exact sampling algorithm takes a Markov chain as its input, meaning that it is able to observe the Markov chain in action or simulate it, and generates a sample from the steady state distribution of the Markov chain. A few years ago this was shown to be possible,

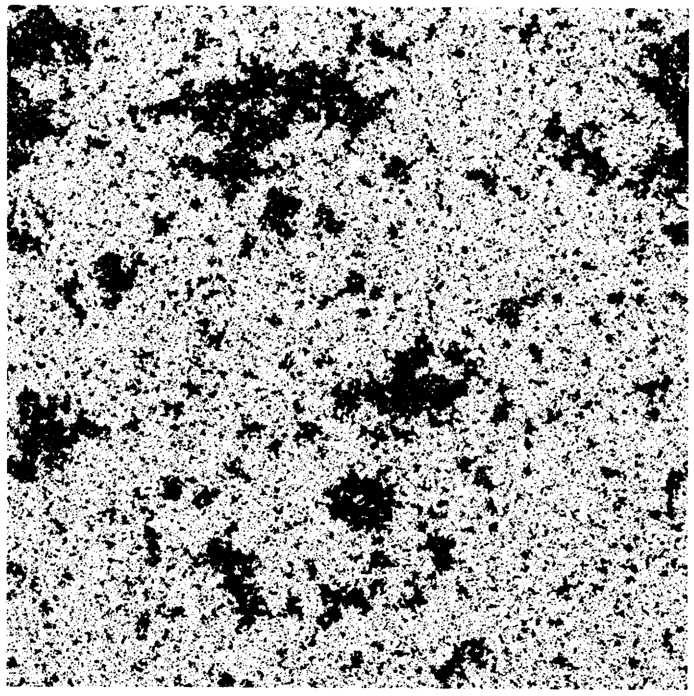

Figure 1-1: An equilibrated Ising state at the critical temperature on a 2100×2100 toroidal grid, generated using monotone coupling from the past (Chapters 2 and 3).

provided the algorithm is told how many states the Markov chain has, but the algorithms given here are much better. In the *passive setting* the algorithm is given a `NextState()` procedure which allows to observe the state of a Markov chain as it evolves through time. We give a version of CFTP that for the passive setting is within a constant factor of optimal. In the *active setting* the algorithm is given a `RandomSuccessor()` procedure, which is similar to `NextState()`, but first sets the state of the Markov chain to some specified value before letting it run one step and returning the random successor state. Clearly any passive case algorithm will work in the active setting, but the cycle popping algorithm is more efficient than any passive case algorithm. We don't give lower bounds on active case algorithms, but cycle popping is so natural that it is hard to imagine a better algorithm. See section 1.3 for comparisons with previous algorithms.

Both CFTP and cycle popping are intimately connected with random spanning trees, and both algorithms may be used to efficiently generate random spaning trees of directed weighted graphs. Figure 1-2 shows a random spanning tree of an undirected unweighted graph. More generally we are given a weighted directed graph G and wish to output an in-directed spanning tree of G such that the probability of outputting a particular tree is proportional to the product of the weights of the edges in the tree. Without going into details, applications of random spanning trees include

- the generation of random Eulerian tours for
  - de Bruijn sequences in psychological tests [32]
  - evaluating statistical significance of DNA patterns [7] [34][1] [10] [51]

[1][34] does not use random spanning trees, and generates biased Eulerian tours

- Monte Carlo evaluation of coefficients of the network reliability polynomial [22] [65] [20]
- generation of random tilings of regions[77] [18] (see Figure 1-3 and also Chapter 8)

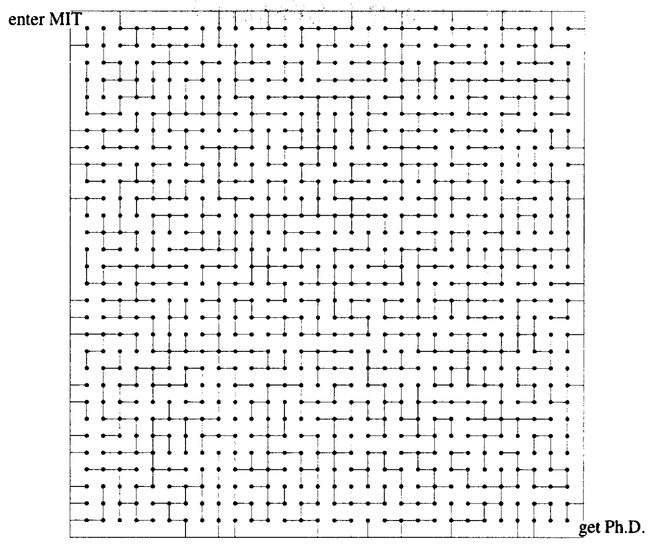

Figure 1-2: A spanning tree of the 30×30 grid with a 901st outer vertex adjoined (the boundary), chosen randomly from all such spanning trees.

There is a natural Markov chain on these spanning trees, and while the chain appears to have no monotone structure, the CFTP algorithm may still be efficiently applied to this Markov chain. Here too one can consider in turn the passive and active cases of tree sampling with Markov chains. In the passive case (Chapter 6), the tree sampling algorithm first calls the general Markov chain sampling algorithm to pick the root of the tree, and then does some additional coupling from the past to pick the rest of the tree. In the active case (Chapter 7), on the other hand, the general Markov chain sampling algorithm calls the tree sampling algorithm, and simply returns the root of the tree.

Random spanning tree generation algorithms have been studied quite a lot. These algorithms may be placed into two categories, those based on computing determinants, and those based on random walks. Of the random walk based algorithms, the ones given here are both the most general and most efficient. No direct comparison is possible with the determinant based algorithms, as the relative efficiency would depend on the input. Section 1.4 compares these tree algorithms at greater length.

As mentioned earlier, some of the examples of monotone Markov chains come from statistical mechanics. One such example is a Markov chain for sampling from the random cluster model introduced by Fortuin and Kasteleyn. The random cluster model is closely connected with the Ising and Potts models, and this connection is utilized in most Monte Carlo studies of these models. (The Ising state shown in Figure 1-1 was generated using a random cluster sample.) It turns out that this random cluster model Markov chain is monotone in two different ways. The first monotonicity allows CFTP to be applied, while the second makes it possible to generate samples at not just one temperature, but essentially all temperatures simultaneously. Figure 1-4 shows two physical parameters (energy and magnetization) plotted as a function of temperature using an *omnithermal* random cluster sample. Chapter 5 discusses how omnithermal sampling might be used to estimate physical parameters.

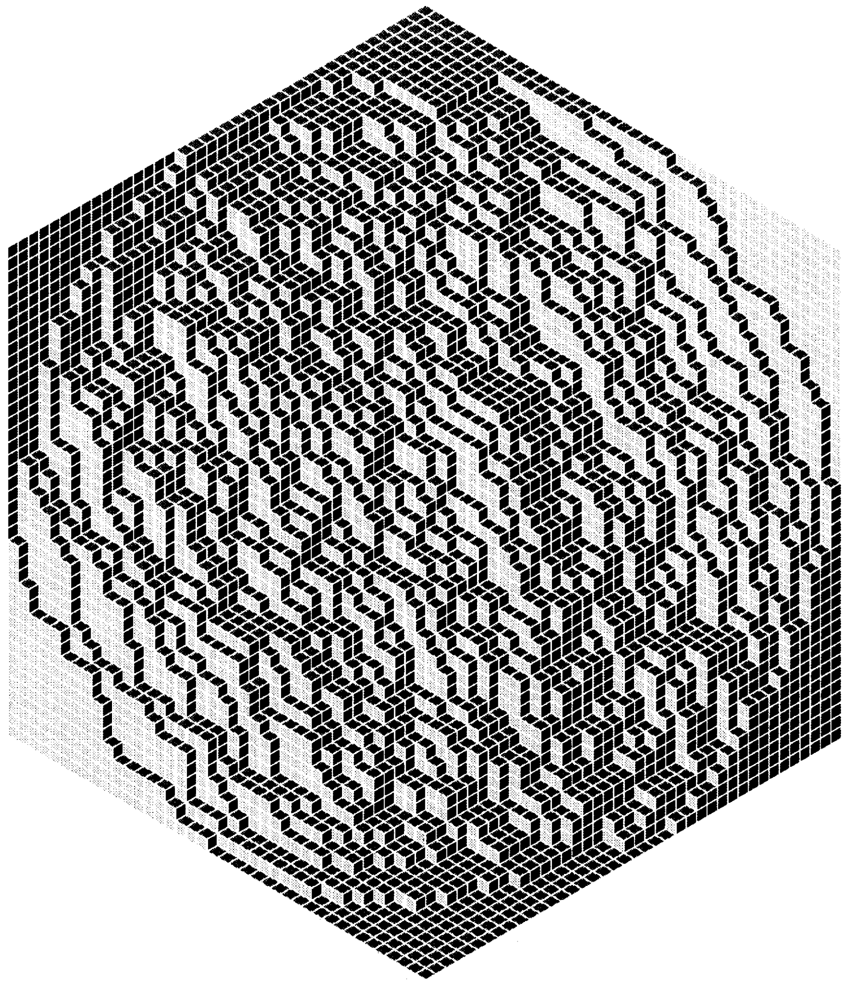

Figure 1-3: A random tiling of a hexagonal region by lozenges, generated using both the directed random spanning tree generator and coupling from the past (see Chapter 8).

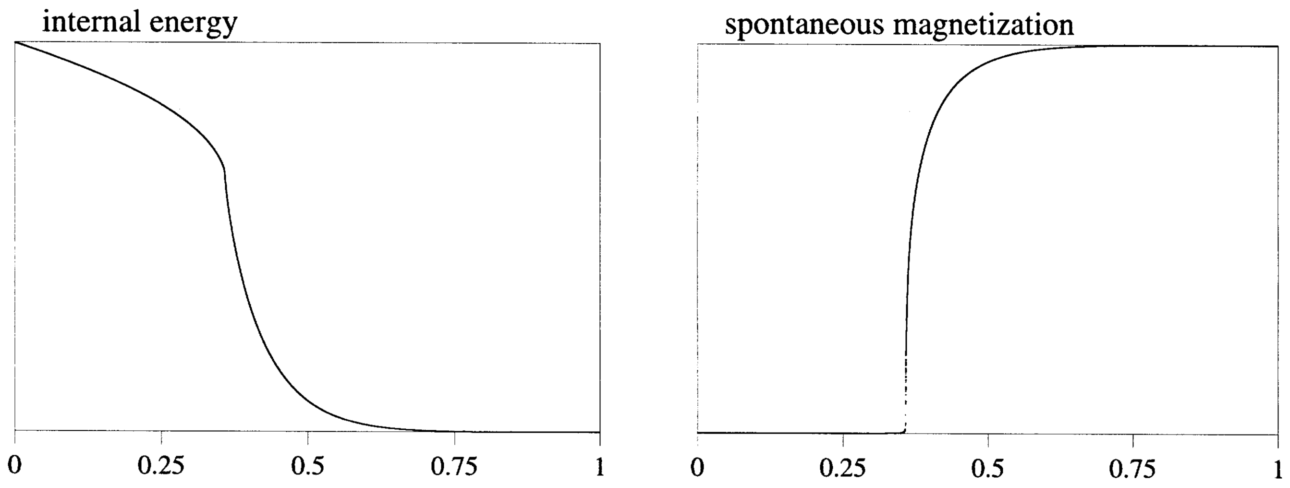

Figure 1-4: The internal energy and spontaneous magnetization of an Ising system as a function of $p = 1 - e^{-\Delta E/kT}$ on the $100 \times 100 \times 100$ grid with periodic boundary conditions. These plots were made from a single omnithermal sample.

1.2 A Simple Exact Sampling Algorithm

To illustrate the simplicity of the sampling algorithms in this thesis, we give an example here. Assume that we have a Markov chain with states numbered 1 through n, and a randomized procedure `RandomSuccessor()`, which when given a state of the Markov chain, returns a random successor state with appropriate probabilities (the active setting). Define a two-dimensional array M with the rules

$$M_{0,i} = i \quad (i = 1 \dots n)$$

and

$$M_{t,i} = M_{t+1,\texttt{RandomSuccessor}(i)} \quad (t < 0, i = 1 \dots n)$$

If the Markov chain is ergodic, then with probability 1, eventually for some T all the entries of the array $M_{T,*}$ will have the same value. In Chapter 2 we show that this value is a perfectly random sample from the steady state distribution of the Markov chain.

1.3 History of Exact Sampling Algorithms

Just a few years ago Asmussen, Glynn, and Thorisson [8] gave the first algorithm for sampling from the stationary distribution π of a Markov chain without knowledge of the mixing time of the Markov chain. They give a general procedure, which given n and a Markov chain on n states, simulates the Markov chain for a while, stops after finite time, and then outputs a random state distributed exactly according to π. However, their procedure is complicated, no bounds on the running time were given, and the authors described it as more of a possibility result than an algorithm to run. Aldous [1] then described an efficient procedure for sampling, but with some bias ε in the samples. Let τ denote the *mean hitting time* of the Markov chain, i.e. the expected number of Markov chain steps to travel from one state i distributed according to π to another state j distributed according to π but independent of i. Aldous's procedure runs within $O(\tau/\varepsilon^2)$ time. Intrigued by these results, Lovász and Winkler [63] found the first exact sampling algorithm that provably runs in time polynomial in certain natural parameters associated with the Markov chain. Let E_iT_j denoted the expected number of steps for the Markov chain to reach j starting from i. The *maximum hitting time* h is the maximum over all pairs of states i and j of E_iT_j. A *(randomized) stopping time* is a rule that decides when to stop running a Markov chain, based on the moves that it has made so far (and some additional random variables). Let T'_{mix} be the optimal expected time of any randomized stopping stop that leaves a particular Markov chain in the stationary distribution (the parameter T'_{mix} is one measure of how long the Markov chain takes to randomize, and is defined as $\tau_1^{(2)}$ in [5]). The Lovász-Winkler algorithm is a randomized stopping rule (i.e. it works by simply observing the Markov chain) and runs in $O(hT'_{\text{mix}}n\log n) \leq O(h^2 n \log n)$ time.

Meanwhile, Jim Propp had been studying random domino tilings of regions. After discussing the problem of how long to run an associated Markov chain, we devised monotone coupling from the past. In contrast with the previous algorithms, this one relies on monotonicity to deal with Markov chains with huge state spaces (e.g. $2^{35,000,000}$ states). Upon learning of the previous work on exact sampling, we went about optimizing the coupling-from-the-past technique as applied to general Markov chains. The result was an algorithm that ran within the *cover time* of the Markov chain — the cover time is the expected time required by the Markov chain to visit all states, starting from the worst possible start state. If the running time of this algorithm were squared, the result would *still* be smaller than the previous fastest running time.

Exact Sampling Algorithm	Expected running time
Asmussen, Glynn, Thorisson [8]	finite time
Aldous [1] (ε bias in sample)	$81\tau/\varepsilon^2$
Lovász, Winkler [63]	$O(hT'_{\text{mix}}n\log n)$
new (requires monotonicity)	$O(T_{\text{mix}}\log l)$
new	$15T_c$ or $O(T_{\text{mix}}n\log n)$
Fill [33]	(not yet analyzed)
new	22τ

n = number of states

l = length of longest chain (monotone Markov chains only). Usually $\log l = O(\log\log n)$.

π = stationary probability distribution

E_iT_j = expected time to reach j starting from i

τ = mean hitting time $= \sum_{i,j}\pi(i)\pi(j)E_iT_j \quad (\tau \le h \le T_c)$

h = maximum hitting time $= \max_{i,j} E_iT_j$

E_iC = expected time to visit all states starting from i

T_c = cover time $= \max_i E_iC$

T_{mix} = mixing time threshold; time for Markov chain to "get within $1/e$ of random"

T'_{mix} = optimal expected stationary stopping time

Table 1.1: Summary of exact sampling algorithms. See [5] for background on the Markov chain parameters. In all the cases for which the expected running time is given, the probability of the actual running time deviating by more than a constant factor from its expected value will decay exponentially fast. Perhaps "self-verifying sampling algorithms" would have been a more inclusive title for this table, but only one of these algorithms is approximate rather than exact.

It has been noted that any passive exact sampling algorithm that works for any Markov chain must necessarily visit all the states. If not, then the algorithm can't output the unvisited state (for all we know this state might be almost transient), and it can't output any other state (for all we know the unvisited state is almost a sink). Therefore, of the passive sampling algorithms, the cover time algorithm is within a constant factor of optimal. But in the active setting, where the algorithm is also allowed to reset the state of the Markov chain, Aldous's $O(\tau/\varepsilon^2)$ approximate sampling algorithm suggested that an $O(\tau)$ time active exact sampling algorithm should be possible. After making several changes in Aldous's algorithm, and completely redoing the analysis, the result is not only an $O(\tau)$ exact sampling algorithm, but also a random spanning tree algorithm faster than the Aldous/Broder algorithm. The running time compares favorably with all the previous algorithms, except the ones that rely on the Markov chain having special structure.

Jim Fill [33] has found another exact sampling method which may be applied to either moderate-sized general Markov chains or huge Markov chains with special structure. His method requires the ability to simulate the reverse Markov chain. Because the algorithm is new, the running time has not yet been determined.

It is worth mentioning that in the physics community it's not uncommon to use a Markov chain to do sampling while having no rigorous bounds on its convergence rate. In these cases experimenters often heuristically estimate the rate of convergence by using auxillary procedures that attempt to measure the correlation between sample points separated by some number of random steps. This approach will not rule out the possibility that the system is in some metastable state [71]. Independently of this work Johnson [50] proposed using monotonicity when testing for convergence, but didn't notice how to bring the bias to zero in finite time.

1.4 History of Random Tree Generation

There is a long history of algorithms to generate random spanning trees from a graph. We are given a weighted directed graph G on n vertices with edge weights that are nonnegative real numbers. The weight of a spanning tree, with edges directed towards its root, is the product of the weights of its edges. Let $\Upsilon_r(G)$ be the probability distribution on spanning trees rooted at vertex r such that the probability of a tree is proportional to its weight, and let $\Upsilon(G)$ be the probability distribution on all spanning trees, with probabilities proportional to the weights of the trees. The goal is to sample a random spanning tree (according to $\Upsilon(G)$), or a random spanning tree with fixed root r (according to $\Upsilon_r(G)$).

The first algorithms for generating random spanning trees were based on the Matrix Tree Theorem, which allows one to compute the number of spanning trees by evaluating a determinant (see [16, ch. 2, thm. 8]). Guénoche [42] and Kulkarni [57] gave one such algorithm that runs in $O(n^3m)$ time[2], where n is the number of vertices and m is the number of edges. This algorithm was optimized for the generation of many random spanning trees to make it more suitable for Monte Carlo studies [22]. Colbourn, Day, and Nel [21] reduced the time spent computing determinants to get an $O(n^3)$ algorithm for random spanning trees. Colbourn, Myrvold, and Neufeld [23] simplified this algorithm, and showed how to sample random arborescences in the time required to multiply $n \times n$ matrices, currently $O(n^{2.376})$ [25].

A number of other algorithms use random walks, Markov chains based on the graph G. For some graphs the best determinant algorithm will be faster, but Broder argues that for most graphs, the random-walk algorithms will be faster [17]. Say G is *stochastic* if for each

[2]Guénoche used $m \leq n^2$ and stated the running time as $O(n^5)$.

Tree Algorithm	Expected running time
Guénoche [42] / Kulkarni [57]	$O(n^3 m)$
Colbourn, Day, Nel [21]	$O(n^3)$
Colbourn, Myrvold, Neufeld [23]	$M(n) = O(n^{2.376})$
Aldous [4] / Broder [17]	$O(\bar{T}_c)$ (undirected)
Kandel, Matias, Unger, Winkler [51]	$O(\bar{T}_c)$ (Eulerian)
new	$O(\bar{T}_c)$ (any graph)
new	$O(\bar{\tau})$ (undirected)
new	$O(\min(\bar{h}, \bar{\tau}))$ (Eulerian)
new	$O(\tilde{\tau})$ (any graph)

n = number of vertices

m = number of edges

$M(n)$ = time to multiply two $n \times n$ matrices

π = stationary probability distribution

$E_i T_j$ = expected time to reach j starting from i

τ = mean hitting time = $\sum_{i,j} \pi(i)\pi(j) E_i T_j$

h = maximum hitting time = $\max_{i,j} E_i T_j$

$E_i C$ = expected time to visit all states starting from i

T_c = cover time = $\max_i E_i C$

Table 1.2: Summary of algorithms for generating random spanning trees of a graph. The top three are based on computing determinants, the bottom six are based on random walks. Of the random walk algorithms, the top three are in the passive setting, and the bottom three are in the active setting.

vertex the weighted outdegree, i.e. the sum of weights of edges leaving the vertex, is 1. If G is stochastic, then in a sense it already is a Markov chain — the state space is the set of vertices, and the probability of a transition from i to j is given by the weight of the edge from i to j. Otherwise, we can define two stochastic graphs $\bar{G}$ and $\tilde{G}$ based on G. To get $\bar{G}$, for each vertex we normalize the weights of its outdirected edges so that its weighted outdegree is 1. To get $\tilde{G}$, first add self-loops until the weighted outdegree of each vertex is the same, and then normalize the weights. Markov chain parameters written with overbars refer to $\bar{G}$, and parameters written with tildes refer to $\tilde{G}$. Running time bounds give in terms of $\bar{G}$ will be better than similar bounds given in terms of $\tilde{G}$.

Broder [17] and Aldous [4] found the first random-walk algorithm for randomly generating spanning trees after discussing the Matrix Tree Theorem with Diaconis. The algorithm works for undirected graphs and runs within the cover time $\bar{T}_c$ of the random walk. The cover time $\bar{T}_c$ of a simple undirected graph is bounded by $O(n^3)$, and is often as small as $O(n \log n)$; see [17] and references contained therein. Broder also described an algorithm for the *approximate* random generation of arborescences from a directed graph. It works well when the out-degree is regular, running in time $O(\tilde{T}_c)$, but for general simple directed graphs, Broder's algorithm takes $O(n\bar{T}_c)$ time. Kandel, Matias, Unger, and Winkler [51] extended the Aldous-Broder algorithm to sample arborescences of a directed Eulerian graph (i.e., one in which in-degree equals out-degree at each node) within the cover time $\bar{T}_c$. Chapter 6 shows how to use coupling from the past to sample random arborescences from a *general* directed graph within 18 cover times of $\bar{G}$. Most of this running time is spent just picking the root, which must be distributed according to the stationary distribution of the random walk of $\tilde{G}$. (A sample from the stationary distribution of $\tilde{G}$ may be obtained with the cover time of $\bar{G}$ using a continuous-time simulation.)

All of these random-walk algorithms run within the cover time of the graph $\bar{G}$ — the expected time it takes for the random walk to reach all the vertices. Chapter 7 gives another class of tree algorithms that are generally better than the previous random walk algorithms (except possibly the one given in chapter 6). The time bounds are $O(\bar{\tau})$ for undirected graphs, $O(\min(\bar{h}, \tilde{\tau}))$ for Eulerian graphs, and $O(\tilde{\tau})$ for general graphs. One wonders whether or not a $O(\bar{\tau})$ algorithm exists for general graphs.

The mean and maximum hitting times are always less than the cover time. Broder described a simple directed graph on n vertices which has an exponentially large cover time [17]. It is noteworthy that the mean hitting time of Broder's graph is linear in n. Even for undirected graphs these times can be substantially smaller than the cover time. For instance, the graph consisting of two paths of size $n/3$ adjoined to a clique of size $n/3$ will have a cover time of $\Theta(n^3)$ but a mean hitting time of $\Theta(n^2)$. Broder notes that most random graphs have a cover time of $\Theta(n \log n)$; since most random graphs are expanders and have a mixing time of $O(1)$, their maximum hitting time will be $\Theta(n)$. Thus these times will usually be much smaller than the cover time, and in some cases the difference can be quite striking.

Chapter 2

Coupling From The Past

2.1 General Coupling From the Past

Suppose that we have an ergodic (irreducible and aperiodic) Markov chain with n states, numbered 1 through n, where the probability of going from state i to state j is $p_{i,j}$. Ergodicity implies that there is a unique stationary probability distribution π, with the property that if we start the Markov chain in some state and run the chain for a long time, the probability that it ends up in state i converges to $\pi(i)$. We have access to a randomized subroutine `RandomSuccessor()` which given some state i as input produces some state j as output, where the probability of observing `RandomSuccessor(`i`)` $= j$ is equal to $p_{i,j}$; we will assume that the outcome of a call to `RandomSuccessor()` is independent of everything that precedes the call. This subroutine gives us a method for approximate sampling from the probability distribution π, in which our Markov-transition oracle `RandomSuccessor()` is iterated M times (with M large) and then the resulting state is output. For convenience, we assume that the start and finish of the simulation are designated as time $-M$ and time 0; of course this notion of time is internal to the simulation being performed, and has nothing to do with the external time-frame in which the computations are taking place. To start the chain, we use an arbitrarily chosen initial state i^*: We call this *fixed-time forward simulation.*

```
$i_{-M} \leftarrow i^*$    (start chain in state $i^*$ at time $-M$)
for $t = -M$ to $-1$
        $i_{t+1} \leftarrow$ RandomSuccessor($i_t$)
return $i_0$
```

Figure 2-1: Fixed time forward simulation.

Unfortunately, it can be difficult to determine how big M needs to be before the probability distribution of the output gets close to π.

We will describe a procedure for sampling with respect to π that does not require foreknowledge of how large the cutoff M needs to be. In essence, to pick a random element with respect to the stationary distribution, we run the Markov chain from the indefinite past until the present, where the distance into the past we have to look is determined dynamically, and more particularly, is determined by how long it takes for n runs of the Markov chain (starting in each of the n possible states, at increasingly remote earlier times) to coalesce.

We start by describing an approximate sampling procedure whose output is governed by the same probability distribution as M-step forward simulation, but which starts at time 0 and moves into the past; this procedure takes fewer steps than fixed-time forward simulation when M is large. Then, by removing the cutoff M — by effectively setting it equal to infinity — we will see that one can make the simulation output state i with probability exactly $\pi(i)$, and that the expected number of simulation steps will nonetheless be finite. (Issues of efficiency will be dealt with in section 2.2 for general Markov chains, and in Chapter 3 in the case where the Markov chain is monotone.)

To run fixed-time simulation backwards, we start by running the chain from time -1 to time 0. Since the state of the chain at time -1 is determined by its history from time $-M$ to time -1, it is unknown to us when we begin our backwards simulation; therfore we run the chain from time -1 to time 0 not just once but n times, once for each of the n states of the chain that might occur at time -1. That is, we can define a map f_{-1} from the state space to itself, by putting $f_{-1}(i) =$ `RandomSuccessor(`i`)` for $i = 1, \ldots, n$. Similarly, for all time t with $-M \leq t < -1$, we can define a random map f_t by putting $f_t(i) =$ `RandomSuccessor(`i`)` (using separate calls to `RandomSuccessor()` for each time t); in fact, we can suppress the details of the construction of the f_t's and imagine that each successive f_t is obtained by calling a randomized subroutine `RandomMap()` whose *values* are actually *functions* from the state space to itself. The output of fixed-time simulation is given by $F_{-M}^{0}(i^*)$, where $F_{t_1}^{t_2}$ is defined as the composition $f_{t_2-1} \circ f_{t_2-2} \circ \cdots \circ f_{t_1+1} \circ f_{t_1}$.

If this were all there were to say about backward simulation, we would have incurred a substantial computational overhead (vis-a-vis forward simulation) for no good reason. Note, however, that under backward simulation there is no need to keep track of all the maps f_t individually; rather, one need only keep track of the compositions F_t^0, which can be updated via the rule $F_t^0 = F_{t+1}^0 \circ f_t$. More to the point is the observation that if the map F_t^0 ever becomes a constant map, with $F_t^0(i) = F_t^0(i')$ for all i, i', then this will remain true from that point onward (that is, for all earlier t's), and the value of $F_{-M}^0(i^*)$ must equal the common value of $F_t^0(i)$ $(1 \leq i \leq n)$; there is no need to go back to time $-M$ once the composed map F_t^0 has become a constant map. When the map F_{-t}^0 is a constant map, we say coalescence occurs from time $-t$ to time 0, or more briefly that coalescence occurs from time $-t$. Backwards simulation is the procedure of working backward until t is sufficiently large that F_{-t}^0 is a constant map, and then returning the unique value in the image of this constant map.

We shall see below that as t goes to $-\infty$, the probability that the map F_t^0 is a constant map increases to 1. Let us suppose that the $p_{i,j}$'s are such that this typically happens with $t \approx -1000$. As a consequence of this, backwards simulation with M equal to one million and backwards simulation with M equal to one billion, which begin in exactly the same way, nearly always turn out to involve the exact same simulation steps (if one uses the same random numbers); the only difference between them is that in the unlikely event that $F_{-1{,}000{,}000}^0$ is not a constant map, the former algorithm returns the sample $F_{-1{,}000{,}000}^0(i^*)$ while the latter does more simulations and returns the sample $F_{-1{,}000{,}000{,}000}^0(i^*)$.

We now see that by removing the cut-off M, and running the backwards simulation into the past until F_t^0 is constant, we are achieving an output-distribution that is equal to

```
$t \leftarrow 0$
$F_t^0 \leftarrow$ the identity map
while $F_t^0$ is not collapsing
    $t \leftarrow t - 1$
    $f_t \leftarrow$ RandomMap()
    $F_t^0 \leftarrow F_{t+1}^0 \circ f_t$
return value to which $F_t$ collapses
```

Figure 2-2: Coupling From the Past (CFTP)

the limit, as M goes to infinity, of the output-distributions that govern fixed-time forward simulation for M steps. However, this limit is equal to π. Hence backwards simulation, with no cut-off, gives a sample whose distribution is governed by the steady-state distribution of the Markov chain. This algorithm is summarized in Figure 2-2.

We remark that the procedure above can be run with $O(n)$ memory, where n is the number of states. In the next section we will give another implementation of the `RandomMap()` procedure that can significantly reduce the running time as well.

Theorem 1 *With probability 1 the coupling-from-the-past protocol returns a value, and this value is distributed according to the stationary distribution of the Markov chain.*

Proof: Since the chain is ergodic, there is an L such that for all states i and j, there is a positive chance of going from i to j in L steps. Hence for each t $F_{t-L}^t(\cdot)$ has a positive chance of being constant. Since each of the maps $F_{-L}^0(\cdot), F_{-2L}^{-L}(\cdot), \ldots$ has some positive probability $\varepsilon > 0$ of being constant, and since these events are independent, it will happen with probability 1 that one of these maps is constant, in which case F_{-M}^0 is constant for all sufficiently large M. When the algorithm reaches back M steps into the past, it will terminate and return a value that we may as well call $\overline{F}_{-\infty}^0$. Note that $\overline{F}_{-\infty}^1$ is obtained from $\overline{F}_{-\infty}^0$ by running the Markov chain one step, and that $\overline{F}_{-\infty}^1$ and $\overline{F}_{-\infty}^0$ have the same probability distribution. Together these last two assertions imply that the output $\overline{F}_{-\infty}^0$ is distributed according to the unique stationary distribution π. □

In this set-up, the idea of simulating from the past up to the present is crucial; indeed, if we were to change the procedure and run the chain from time 0 into the future, finding the smallest M such that the value of $F_0^M(x)$ is independent of x and then outputting that value, we would obtain biased samples. To see this, imagine a Markov chain in which some states have a unique predecessor; it is easy to see such states can never occur at the exact instant when all n histories coalesce.

It is sometimes desirable to view the process as an iterative one, in which one successively starts up n copies of the chain at times -1, -2, etc., until one has gone sufficiently far back in the past to allow the different histories to coalesce by time 0. However, when one adopts this point of view (as we will do in Chapter 3), it is important to bear in mind that the random bits that one uses in going from time t to time $t+1$ must be the same for the many sweeps one might make through this time-step. If one ignores this requirement, then there will in general be bias in the samples that one generates. The curious reader may verify this by considering the Markov chain whose states are 0, 1, and 2, and in which transitions are implemented using a fair coin, by the rule that one moves from state i to state $\min(i+1,2)$ if the coin comes up heads and to state $\max(i-1,0)$ otherwise. It is simple to check that if one runs an incorrect version of our scheme in which entirely new random bits are used every time the chain gets restarted further into the past, the samples one gets will be biased in favor of the extreme states 0 and 2.

We now address the issue of independence of successive calls to `RandomSuccessor()` (implicit in our calls to the procedure `RandomMap()`). Independence was assumed in the proof of Theorem 1 in two places: first, in the proof that the procedure eventually terminates, and second, in the proof that the distribution governing the output of the procedure is the steady-state distribution of the chain. We can relax this independence condition, but successive calls to `RandomMap()` need to remain independent, and we still need some guarantee that the process will terminate with probability 1.

To treat possible dependencies, we adopt the point of view that $f_t(i)$ is given by $\phi_t(i, U_t)$ where $\phi_t(\cdot,\cdot)$ is a deterministic function and U_t is a random variable associated with time t. One might have several different Markovian update-rules ϕ on a state-space, and cycle among them, as long as each one of them preserves the distribution π. We assume that the random variables $\ldots, U_{-2}, U_{-1}$ are independent, and we assume that the random process given by the U_t's is the source of all the randomness used by our algorithm, since the other random variables are deterministic functions of the U_t's. Note that this framework includes the case of full independence discussed earlier, since for example one could let the U_t's be i.i.d. variables taking their values in the n-cube $[0,1]^n$ with uniform distribution, and let $\phi_t(i,u)$ be the smallest j such that $p_{i,1} + p_{i,2} + \cdots + p_{i,j}$ exceeds the ith component of the vector u.

Theorem 2 *Let $\ldots, U_{-3}, U_{-2}, U_{-1}$ be independent random variables and let $\phi_t(\cdot,\cdot)$ $(t < 0)$ be a sequence of deterministic functions. Define $f_t(i) = \phi_t(i, U_t)$ and $F_t^0 = f_{-1} \circ f_{-2} \circ \cdots \circ f_t$. If we assume (1) for all t and j,*

$$\sum_i \pi(i) \Pr[\phi_t(i, U_t) = j] = \pi(j),$$

and (2) with probability 1 there exists T for which the map F_T^0 is constant, then the constant value of F_T^0 is a random variable with probability distribution governed by π.

Proof: Let X be a random variable on the state-space of the Markov chain governed by the steady-state distribution π, and for all $t \leq 0$ let Y_t be the random variable $F_t^0(X)$. Each Y_t has distribution π, and the sequence $Y_{-1}, Y_{-2}, Y_{-3}, \ldots$ converges almost surely to some state $Y_{-\infty}$, which must also have distribution π. But the constant value of F_T^0 is $Y_{-\infty}$. □

Usually it is not hard to show condition (2) of Theorem 2. For instance, if all the functions ϕ_t are identical, and the U_t are identically distributed, and there is a nonzero probability that some F_T^0 is constant, then with probability 1 some F_T^0 is constant.

2.2 Random Maps for Passive Coupling From the Past

Here we give a RandomMap() procedure which makes the coupling-from-the-past protocol of the previous section run in time proportional to the cover time — the expected time for the Markov chain to visit all the states. This procedure outputs a random map from $\mathcal{X}$ to $\mathcal{X}$ satisfying conditions 1 and 2 of Theorem 2 after observing the Markov chain for some number of steps.

Recall that if $f =$ RandomMap(), we don't need $f(i_1)$ and $f(i_2)$ to be independent. So we will use the Markov chain to make a suitable random map, yet contrive to make the map have small image. Then the algorithm will terminate quickly. The RandomMap() procedure consists of two phases. The first phase estimates the cover time from state 1. The second phase starts in state 1, and actually constructs the map.

Initialization phase (Expected time $\leq 3T_c$)

- Wait until we visit state 1.
- Observe the chain until all states are visited, and let C be the number of steps required.
- Wait until we next visit state 1.

Construct Map phase (Expected time $\leq 3T_c$)

- Randomly set an alarm clock that goes off every $4C$ steps.
- When we next visit some state i for the first time, commit to setting $f(i)$ to be the state when the alarm clock next goes off.

The following two lemmas show that this implementation of RandomMap() (see Figure 2-3) satisfies the two necessary conditions.

Lemma 3 *If X is a random state distributed according to π, and $f =$* RandomMap()*, then $f(X)$ is a random state distributed according to π.*

Proof: Let c be the value of *clock* when state X is encountered for the first time in the Construct Map phase. Since *clock* was randomly initialized, c itself is a random variable uniform between 1 and $4C$. The value $f(X)$ was obtained by starting the Markov chain in the random state X and then observing the chain after some number $(4C - c)$ of steps, where the distribution of the number of steps is independent of X. Hence $f(X)$ is also distributed according to π. $\square$

Lemma 4 *The probability that the output of* RandomMap() *is a constant map is at least* $3/8$.

Proof: Let C' be the number of steps it takes the Markov chain to visit all the states in the Construct Map phase, and let A denote the number of steps in the Construct Map phase before the alarm clock first goes off. If $C' \leq A$, then the output of RandomMap() will be constant.

Since C and C' are independent identically distributed random variables,

$$\Pr[C' \leq C] \geq 1/2.$$

```
Wait until we visit state 1
Observe the chain until all states are visited, and let $C$ be the number of steps required
Wait until we next visit state 1

$clock$ ← random number between 1 and $4C$
for $i \leftarrow 1$ to $n$
  $status[i] \leftarrow$ UNSEEN
$num\_assigned \leftarrow 0$
$status[1] \leftarrow$ SEEN
$ptr \leftarrow 0$
$stack[ptr++] \leftarrow 1$
while $num\_assigned < n$
  while $clock < 4C$
    $clock \leftarrow clock + 1$
    $s \leftarrow$ NextState()
    if $status[s] =$ UNSEEN then
      $status[s] \leftarrow$ SEEN
      $stack[ptr++] \leftarrow s$
  $clock \leftarrow 0$
  $num\_assigned \leftarrow num\_assigned + ptr$
  while $ptr > 0$
    $f[stack[--ptr]] \leftarrow s$
return $f$
```

Figure 2-3: Pseudocode for `RandomMap()`. The Construct Map phase starts out looking at state 1. The variable *status*[i] indicates whether or not state i has been seen yet. When i is first seen, *status*[i] gets set to `SEEN` and i is put on the stack. When the alarm clock next goes off, for each state i in the stack, $f[i]$ gets set to be the state at that time.

On the other hand, since the alarm clock was randomly initialized

$$\Pr[A \geq C] = 3/4.$$

Since A and C' are independent (even when conditioning on C),

$$\Pr[C' \leq C \leq A] = \Pr[C' \leq C] \cdot \Pr[C \leq A] \geq 3/8.$$

$\square$

Theorem 5 *Using the above* `RandomMap()` *procedure with the coupling-from-the-past protocol yields an unbiased sample from the steady-state distribution. On average the Markov chain is observed for* $\leq 15T_c$ *steps. The expected computational time is also* $O(T_c)$*, the memory is* $O(n)$*, and the expected number of external random bits used is* $O(\log T_c)$.

Proof: Since the odds are $\geq 3/8$ that a given output of `RandomMap()` is constant, the expected number of calls to `RandomMap()` before one is constant is $\leq 8/3$. Each call to `RandomMap()` observes the chain an average of $\leq 6T_c$ steps. Before sampling we may arbitrarily label the current state to be state 1 and thereby reduce the expected number of steps from $16T_c$ to $15T_c$. □

The algorithm is surely better than the above analysis suggests, and if the Markov chain may be reset, even faster performance is possible. Also note that in general some external random bits will be required, as the Markov chain might be deterministic or almost deterministic. When the Markov chain is not deterministic, Lovász and Winkler [63] discuss how to make do without external random bits by observing the random transitions.

2.3 The Voter Model

The CFTP procedure in Figure 2-2 is closely related to two stochastic processes, known as *the voter model*, and *the coalescing random walk model*. Both of these models are based on a Markov chain, either discrete or continuous time. In this section we bound the running time of these two processes, and then give a variation on the CFTP procedure (in the active setting) and bound its running time. This section is not essential to the remainder of this thesis, except that here we define the variation distance and the mixing time of a Markov chain, which will be used in one section of Chapter 3.

Given a (discrete time) Markov chain on n states, in which state i goes to state j with probability $p_{i,j}$, one defines a "coalescing random walk" by placing a pebble on each state and decreeing that the pebbles must move according to the Markov chain statistics; pebbles must move independently unless they collide, at which point they must stick together. (For ease of exposition we work with discrete time, but there are no difficulties in generalizing to continuous time. One could adjoin self loops to the Markov chain and pass to a limit to get the continuous time case.)

The original Markov chain also determines a (generalized finite discrete time) voter model, in which each of n voters starts by wanting to be be the leader. At each time step each of the voters picks a random neighbor (voter i picks neighbor j with probability $p_{i,j}$), asks who that neighbor will vote for, and at the next time step changes his choice to be that candidate. (These updates are done in parallel.) The voter model has been studied quite a lot in continuous time case, and usually on a grid where $p_{i,j}$ is nonzero only if i and j are neighbors on the grid. See [45], [39], and [61]) for background on the voter model.

These two models are dual to one another, in the sense that each can be obtained from the other by simply reversing the direction of time (see [5]). Specifically, suppose that for each t between t_1 and t_2, for each state we choose in advance a successor state according to the rules of the Markov chain. We can run the coalescing random walk model from time t_1 to t_2 with each pebble following the choices made in advance. Or we can run the voter model from time t_2 to time t_1 using the choices made in advance. The pebble that started at state i ends up at state j if and only if voter i plans to vote for j.

Note that a simulation of the voter model is equivalent to running the CFTP procedure in Figure 2-2. Since the CFTP algorithm returns a random state distributed according to the steady state distrubtion π of the Markov chain, or else runs forever, it follows that if all the voters ever agree on a common candidate, their choice will be distributed according to the steady state distribution of the Markov chain. This result holds for both discrete and continuous time versions of the voter model. (For certain special cases, such as the grid, this result can also be derived using other techniques.)

In this section we analyze the running time of this process. We will show

Theorem 6 *The expected time before all the voters agree on a common candidate is at most $O(nT_{mix})$ steps, where T_{mix} is the "mixing time" of the associated Markov chain.*

Up until now we have not found it necessary to formally define the mixing time, the time it takes for the state of a Markov chain to become approximately randomly distributed, but we will do so in a moment.

Since every step of the CFTP algorithm, or a simulation of the voter model, takes $\Theta(n)$ computer time, this implies that one can obtain random samples within $\Theta(n^2 T_{\text{mix}})$ time. However, we give a variation of the algorithm that runs in $\Theta(n \log n T_{\text{mix}})$ time. We can do this by taking advantage of the connection with the coalescing random walk model. As time moves forward, the pebbles tend to get glued together into pebble piles, and less computational work is needed to update their positions. Of course simply waiting until all the pebbles get glued together will in general result in a biased sample — we give the actual variation on the CFTP algorithm after proving the following theorem about the coalescing random walk model.

Theorem 7 *The expected time before all the pebbles coalesce is at most $O(nT_{mix})$ steps, where T_{mix} is the "mixing time" of the associated Markov chain. The number of pebble piles added up for each time until coalescence is at most $O(n \log n T_{mix})$.*

Theorem 6 is immediate from theorem 7.

The time T_{mix} is defined using the variation distance between probability distributions. Let μ and ρ be two probability distributions on a space $\mathcal{X}$, the variation distance between them, $\|\mu - \rho\|$, is given by

$$\|\mu - \rho\| \equiv \max_{A \subseteq \mathcal{X}} |\mu(A) - \rho(A)| = \frac{1}{2} \sum_x |\mu(x) - \rho(x)|.$$

Let ρ_x^{*k} be the probability distribution of the state of the Markov chain when started in state x and run for k steps. Define

$$\bar{d}(k) = \max_{x,y} \|\rho_x^{*k} - \rho_y^{*k}\|.$$

Then the variation distance threshold time T_{mix} is the smallest k for which $\bar{d}(k) \leq 1/e$. It is not hard to show that $\bar{d}(k)$ is submultiplicative, and that for any starting state x, $\|\rho_x^{*k} - \pi\| \leq \bar{d}(k)$.

To prove Theorem 7 we make use of the following lemma.

Lemma 8 *Let ρ_x denote the probability distribution given by $\rho_x(y) = p_{x,y}$. Suppose that for each x the variation distance of ρ_x from π is at most $\varepsilon < 1/2$. Then the expected time before all the pebbles are in the same pile is $O(n)$, and the sum of the number of pebble piles at each time step, until they're all in the same pile, is $O(n \log n)$.*

Proof: Suppose that there are $m > 1$ piles. Let $0 < \beta < 1$ and $1/2 < \alpha < 1$. Say that x avoids y if $\rho_x(y) < \beta\pi(y)$. Call a state lonely if it is avoided by at least $(1-\alpha)m$ piles. We have

$$\begin{aligned}
(1-\beta)(1-\alpha)m \sum_{\text{lonely } y} \pi(y) &\leq (1-\beta) \sum_{\substack{x,y \\ \text{pile at } x \\ x \text{ avoids } y}} \pi(y) \\
&\leq \sum_{\substack{x,y \\ \text{pile at } x \\ x \text{ avoids } y}} \pi(y) - \rho_x(y) \\
&\leq \sum_{\substack{x \\ \text{pile at } x}} \|\pi - \rho_x\| \\
&\leq m\varepsilon
\end{aligned}$$

Let

$$\gamma = \sum_y \bar{\pi}(y) \geq 1 - \frac{1}{1-\alpha}\frac{\varepsilon}{1-\beta},$$

where $\bar{\pi}(y) = \pi(y)$ if y is good, and 0 if y is lonely. Since $\varepsilon < 1/2$, we may choose α and β so that $\gamma > 0$.

Let A_y be the number pile mergers that happen at state y — A_y is the number of piles that get mapped to y, minus one if a pile gets mapped to y.

Suppose y is good. There are at least $\lceil \alpha m \rceil$ piles which have at least a $\beta\pi(y)$ chance of being mapped to y. If we suppose that there are exactly $\lceil \alpha m \rceil$ piles that get mapped to y with probability $\beta\pi(y)$, and no other piles get mapped to y, we can only decrease A_y. Thus we find

$$E[A_y] \geq \lceil \alpha m \rceil \beta\bar{\pi}(y) - 1 + (1-\beta\bar{\pi}(y))^{\lceil \alpha m \rceil},$$

which is also true even if y is lonely.

Since $\lceil \alpha m \rceil \geq 1$, the above expression for $E[A_y]$ is convex in $\bar{\pi}(y)$, and we get

$$\begin{aligned}
\sum_y E[A_y] &\geq n\left[\lceil \alpha m \rceil \beta\gamma/n - 1 + (1-\beta\gamma/n)^{\lceil \alpha m \rceil}\right] \\
&\geq \lceil \alpha m \rceil \beta\gamma - n + n\left[1 - (\beta\gamma/n)\binom{\lceil \alpha m \rceil}{1} + (\beta\gamma/n)^2\binom{\lceil \alpha m \rceil}{2} - (\beta\gamma/n)^3\binom{\lceil \alpha m \rceil}{3}\right] \\
&\geq \frac{(\beta\gamma)^2(\lceil \alpha m \rceil)(\lceil \alpha m \rceil - 1)}{3n}
\end{aligned}$$

as $(\lceil \alpha m \rceil - 2) < n$ and $\beta\gamma \leq 1$.

Group the time steps into phases so that in phase k the number of piles m satisfies $n/2^k + 1/\alpha > m \geq n/2^{k+1} + 1/\alpha$. All the pebbles are in one pile when phase $\lceil \lg[n/(2-1/\alpha)] \rceil$ starts. Let $R = n(\beta\gamma\alpha)^2/2^{2k+2}/3$. For each step i after phase k begins and before phase $k+1$, let V_i be the random variable denoting the reduction in the number of piles. After phase $k+1$ begins, let $V_i = R$. Then for all i $E[V_i] \geq R$. Let $s = 12/(\beta\gamma\alpha)^2 \cdot 2^k$, and $V = \sum_{i=1}^{\lfloor s \rfloor} V_i + (s - \lfloor s \rfloor)V_{\lceil s \rceil}$. $E[V] \geq n/2^k$, and $V \leq 2n/2^k + 1$, so by Markov's inequality, $\Pr[V \geq (1/2)n/2^k] \geq \Theta(1)$. There is at least a constant chance that the next phase has started, so the expected number of steps in phase k is at most $O(s) = O(2^k)$. The expected sum over phase k of the number of piles is $O(n)$, completing the proof.

□

Proof of Theorem 7: Following Lovász, Winkler, and Aldous [63] [3], we work with a Markov chain derived from the original Markov chain. We can let one step in the derived Markov chain be T steps (say $T = T_{\text{mix}}$) in the original chain. If two pebble piles coalesce in the derived chain, they must have coalesced in the original chain, so the theorem is a straightforward consequence of the above lemma.

We remark there is some flexibility in choosing T, we could for instance let $T = 2T_{\text{mix}}$, and because of submultiplicativity, we would get $\varepsilon \leq e^{-2}$. If $\varepsilon < 1/8$, the optimal choice of the constants in the lemma is about $\alpha = \beta = \gamma = 1 - \varepsilon^{1/3}$. Optimizing $(1 - \varepsilon^{1/3})^{-6} \ln 1/\varepsilon$ yields $\varepsilon \doteq 1.5769 \times 10^{-4}$. □

For the variation on coupling from the past, we want to spend most of the time following the pebbles forward in time, because they will tend to coalesce, and less work will be required at subsequent time steps. Initialize T to be zero. Then if F^0_{-T} is not collapsing, run the coalescing random walk process and count the number of steps before all the pebbles are in the same pile. Increment T by this amount, and try again. The expected number of Markov simulation steps is at most six times the number of steps required to simulate the coalescing random walk process until all the pebbles are in the same pile — $O(n \log n T_{\text{mix}})$. The computational overhead can be kept similarly small.

Chapter 3

Monotone Coupling From the Past

The coupling-from-the-past algorithm using the `RandomMap()` procedure given in section 2.2 works fine if the Markov chain is not too large, but it is not practical for Markov chains with $2^{35,000,000}$ states, since it needs to visit all of the states. However, many Markov chains from which we wish to sample have a special property, called monotonicity. One nice example is the Ising model, which we will discuss further in section 4.2.1. In the Ising model there are a collection of sites, usually on a grid, each site is either spin up or spin down, and neighboring sites prefer to be aligned with one another. Given two spin configurations, if every site in the first that is spin up is also spin up in the second configuration, then we may simultaneously update the two spin configurations with a Markov chain, preserving this domination property. In this chapter we give a variation of the coupling-from-the-past algorithm that takes advantage of this monotonicity condition, and derive bounds on the running time. In Chapter 4 we give a number of examples of monotone Markov chains.

3.1 Monotone Markov Chains

Suppose now that the (possibly huge) state space S of our Markov chain admits a natural partial ordering $\leq$, and that our update rule ϕ has the property that $x \leq y$ implies $\phi(x, U_0) \leq \phi(y, U_0)$ almost surely with respect to U_0. Then we say that our Markov chain gives a *monotone Monte Carlo algorithm* for approximating π. We will suppose henceforth that S has elements $\hat{0}$, $\hat{1}$ with $\hat{0} \leq x \leq \hat{1}$ for all $x \in S$.

Define $\Phi_{t_1}^{t_2}(x, u) = \phi_{t_2-1}(\phi_{t_2-2}(\ldots(\phi_{t_1}(x, u_{t_1}), u_{t_1+1}), \ldots, u_{t_2-2}), u_{t_2-1})$, where u is short for $(\ldots, u_{-1}, u_0)$. If $u_{-T}, u_{-T+1}, \ldots, u_{-2}, u_{-1}$ have the property that $\Phi^0_{-T}(\hat{0}, u) = \Phi^0_{-T}(\hat{1}, u)$, then the monotonicity property assures us that $\Phi^0_{-T}(x, u)$ takes on their common value for all $x \in S$. This frees us from the need to consider trajectories starting in all $|S|$ possible states; two states will suffice. Indeed, the smallest T for which $\Phi^0_{-T}(\cdot, u)$ is constant is equal to the smallest T for which $\Phi^0_{-T}(\hat{0}, u) = \Phi^0_{-T}(\hat{1}, u)$.

Let T_* denote this smallest value of T. It would be possible to determine T_* exactly by a bisection technique, but this would be a waste of time: an overestimate for T_* is as good as the correct value, for the purpose of obtaining an unbiased sample. Hence, we successively try $T = 1, 2, 4, 8, \ldots$ until we find a T of the form 2^k for which $\Phi^0_{-T}(\hat{0}, u) = \Phi^0_{-T}(\hat{1}, u)$. The number of simulation-steps involved is $2(1 + 2 + 4 + \cdots + 2^k) < 2^{k+2}$, where the factor of

2 in front comes from the fact that we are simulating two copies of the chain (one from $\hat{0}$ and one from $\hat{1}$). However, this is close to optimal, since T_* must exceed 2^{k-1} (otherwise we would not have needed to go on to try $T = 2^k$); that is, the number of simulation steps required merely to *verify* that $\Phi^0_{-T_*}(\hat{0}, u) = \Phi^0_{-T_*}(\hat{1}, u)$ is greater than $2 \cdot 2^{k-1} = 2^k$. Hence our double-until-you-overshoot procedure comes within a factor of 4 of what could be achieved by a clairvoyant version of the algorithm in which one avoids overshoot.

Here is the pseudocode for monotone coupling from the past. Implicit in this pseudocode

```
$T \leftarrow 1$
repeat
   $upper \leftarrow \hat{1}$
   $lower \leftarrow \hat{0}$
   for $t = -T$ to $-1$
      $upper \leftarrow \phi_t(upper, u_t)$
      $lower \leftarrow \phi_t(lower, u_t)$
   $T \leftarrow 2T$
until $upper = lower$
return $upper$
```

Figure 3-1: Monotone coupling from the past.

is the random generation of the u_t's. Note that when the random mapping $\phi_t(\cdot, u_t)$ is used in one iteration of the repeat loop, for any particular value of t, it is essential that the same mapping be used in all subsequent iterations of the loop. We may accomplish this by storing the u_t's; alternatively, if (as is typically the case) our u_t's are given by some pseudo-random number generator, we may simply suitably reset the random number generator to some specified seed $seed(i)$ each time t equals -2^i.

In the context of monotone Monte Carlo, a hybrid between fixed-time forward simulation and coupling from the past is a kind of adaptive forward simulation, in which the monotone coupling allows the algorithm to check that the fixed time M that has been adopted is indeed large enough to guarantee mixing. This approach was foreshadowed by the work of Valen Johnson [50], and it has been used by Kim, Shor, and Winkler [56] in their work on random independent subsets of certain graphs; see section 4.3. Indeed, if one chooses M large enough, the bias in one's sample can be made as small as one wishes. However, it is worth pointing out that for a small extra price in computation time (or even perhaps a saving in computation time, if the M one chose was a very conservative estimate of the mixing time), one can use coupling from the past to eliminate the initialization bias entirely.

In what sense does our algorithm solve the dilemma of the Monte Carlo simulator who is not sure how to balance his need to get a reasonably large number of samples and his need to get unbiased samples? We will see below that the expected run time of the algorithm is not much larger than the mixing time of the Markov chain (which makes it fairly close to optimal), and that the tail distribution of the run time decays exponentially quickly. If one is willing to wait for the algorithm to return an answer, then the result will be an unbiased sample. More generally, if one wants to generate several samples, then as long as one completes every run that gets started, all of the samples will be unbiased as well as independent of one another. Therefore, we consider our approach to be a practical solution to the Monte Carlo simulator's problem of not knowing how long to run the Markov chain.

We emphasize that the experimenter must not simply interrupt a run and discard its results, retaining only those samples obtained prior to this interruption; the experimenter must either allow the run to terminate or else regard the final sample as indeterminate (or only partly determined) — or resign himself to contaminating his samples with bias. To reduce the likelihood of ending up in this dilemma, the experimenter can use the techniques of section 3.2 to estimate in advance the average number of Markov chain steps that will be required for each sample.

Recently Jim Fill has found an *interruptible* exact sampling protocol. Such a protocol, when allowed to run to completion, returns a sample distributed according to π, just as coupling from the past does. Additionally, if an impatient user interrupts some runs, then rather than regarding the samples as indeterminate, the experimenter can actually throw them out without introducing bias. This is because the run time of an interruptible exact sampling procedure is independent of the sample returned, and therefore independent of whether or not the user became impatient and aborted the run.

As of yet there remain a number of practical issues that need to be resolved before a truly interruptible exact sampling program can be written. We expect that Fill and others will discuss these issues more thoroughly in future articles.

3.2 Time to Coalescence

In this section we bound the coupling time of a monotone Markov chain, i.e. a chain with partially ordered state space whose moves preserve the partial order. These bounds directly relate to the running time of our exact sampling procedure. We bound the expected run time, and the probability that a run takes abnormally long, in terms of the mixing time of the Markov chain. If the underlying monotone Markov chain is rapidly mixing, then it is also rapidly coupling, so that there is essentially no reason not to apply coupling from the past when it is possible to do so. If the mixing time is unknown, then in it may be estimated from the coupling times. We also bound the probability that a run takes much longer than these estimates.

Recall that the random variable T_* is the smallest t such that $F^0_{-t}(\hat{0}) = F^0_{-t}(\hat{1})$. Define T^*, the time to coalescence, to be the smallest t such that $F^t_0(\hat{0}) = F^t_0(\hat{1})$. Note that $\Pr[T_* > t]$, the probability that $F^0_{-t}(\cdot)$ is not constant, equals the probability that $F^t_0(\cdot)$ is not constant, $\Pr[T^* > t]$. The running time of the algorithm is linear in T_*, but since T_* and T^* are governed by the same probability distribution, in this section we focus on the conceptually simpler T^*.

In order to relate the coupling time to the mixing time, we will consider three measures of progress towards the steady state distribution π: $\mathrm{Exp}[T^*]$, $\Pr[T^* > K]$ for particular or random K, and $\bar{d}(k) = \max_{\mu_1,\mu_2} \|\mu_1^k - \mu_2^k\|$ for particular k, where μ^k is the distribution governing the Markov chain at time k when started at time 0 in a random state governed by the distribution μ. Let ρ_0 and ρ_1 be the distributions on the state space S that assign probability 1 to $\hat{0}$ and $\hat{1}$, respectively.

Theorem 9 *Let l be the length of the longest chain (totally ordered subset) in the partially ordered state-space S. Then*

$$\frac{\Pr[T^* > k]}{l} \leq \overline{d}(k) \leq \Pr[T^* > k].$$

Proof: If x is an element of the ordered state-space S, let $h(x)$ denote the length of the longest chain whose top element is x. Let the random variables X_0^k and X_1^k denote the states of (the two copies of) the Markov chain after k steps when started in states $\hat{0}$ and $\hat{1}$, respectively. If $X_0^k \neq X_1^k$ then $h(X_0^k) + 1 \leq h(X_1^k)$ (and if $X_0^k = X_1^k$ then $h(X_0^k) = h(X_1^k)$). This yields

$$\begin{aligned}
\Pr[T^* > k] &= \Pr[X_0^k \neq X_1^k] \\
&= \Pr[h(X_0^k) - h(X_1^k) \geq 1] \\
&\leq E[h(X_1^k) - h(X_0^k)] \\
&= \left| E_{\rho_1^k}[h(X)] - E_{\rho_0^k}[h(X)] \right| \\
&\leq ||\rho_1^k - \rho_0^k|| \left[\max_x h(x) - \min_x h(x) \right] \\
&\leq \overline{d}(k)\, l,
\end{aligned}$$

proving the first inequality. To prove the second, consider a coupling in which one copy of the chain starts in some distribution μ_1 and the other starts in distribution μ_2. By the monotonicity of the coupling, the probability that the two copies coalesce within k steps is at least $\Pr[T^* \leq k]$. Hence our coupling achieves a joining of μ_1^k and μ_2^k such that the two states disagree only on an event of probability at most $\Pr[T^* > k]$. It follows that the total variation distance between the two distributions is at most $\Pr[T^* > k]$. □

Next we show that $\Pr[T^* > k]$ is submultiplicative. At this point we will assume that the randomization operations at each time step are the same. So for instance, if a random process operates on red vertices and then on blue vertices, these two operations together are considered one step.

Theorem 10 *Let K_1 and K_2 be nonnegative random variables (which might be constant). Then*

$$\Pr[T^* > K_1 + K_2] \leq \Pr[T^* > K_1] \cdot \Pr[T^* > K_2].$$

Proof: The event that $F_0^{K_1}$ is constant and the event that $F_{K_1}^{K_1+K_2}$ is constant are independent, and if either one is constant, then $F_0^{K_1+K_2}$ is constant. □

Next we estimate tail-probabilities for T^* in terms of the expected value of T^*, and vice versa.

Lemma 11

$$k \Pr[T^* > k] \leq \operatorname{Exp}[T^*] \leq k / \Pr[T^* \leq k].$$

Proof: The first inequality follows from the non-negativity of T^*. To prove the second, note that if we put $\varepsilon = \Pr[T^* > k]$, then by submultiplicativity, $\Pr[T^* > ik] \leq \varepsilon^i$. Hence $\operatorname{Exp}[T^*] \leq k + k\varepsilon + k\varepsilon^2 + \cdots = k/\Pr[T^* \leq k]$. □

Now we can say what we meant by "if the Markov chain is rapidly mixing then it is rapidly coupling" in the first paragraph of this section. The mixing time threshold T_{mix} is defined to be the smallest k for which $\bar{d}(k) \leq 1/e$ [5]. Let l be the length of the longest chain in the partially ordered state space. Since $\bar{d}(k)$ is submultiplicative (see [2]), after $k = T_{\text{mix}}(1+\ln l)$ steps, $\bar{d}(k) \leq 1/el$, so $\Pr[T^* > k] \leq 1/e$ by Theorem 9, and by Lemma 11,

$$\text{Exp}[T^*] \leq k/(1-1/e) < 2k = 2T_{\text{mix}}(1+\ln l).$$

It has been noted [27] that many Markov chains exhibit a sharp threshold phenomenon: after $(1-\varepsilon)T_{\text{mix}}$ steps the state is very far from being random, but after $(1+\varepsilon)T_{\text{mix}}$ steps the state is very close to being random (i.e. $\bar{d}(k)$ is close to 0). In such chains the coupling time will be less than $O(T_{\text{mix}} \log l)$.

In addition to being useful as a source of random samples, our method can also be used as a way of estimating the mixing time of a random walk. One application to which this might be put is the retrospective analysis of someone else's (or one's own) past simulations, which were undertaken with only the experimenter's intuitive sense to guide the choice of how long the chain needed to be run before the initialization bias would be acceptably small. Using our technique, one can now assess whether the experimenter's intuitions were correct.

For instance, suppose one takes ten independent samples of the coupling time random variable T^*, and obtains $10T_{\text{est}} = T_1 + \cdots + T_{10} = 997$. Then one can be fairly confident that if one were to run the Markov chain from an arbitrary starting point for 1000 steps (or if someone had done so in the past), the residual initialization bias would be less than 2^{-10}. Assertions of this kind can be made rigorous. Specifically, we can argue that if one treats T_{est} as a random variable, then the initialization bias of the Markov chain when run for random time $10T_{\text{est}}$ is at most 2^{-10}. By symmetry we have $\Pr[T^* > T_i] \leq 1/2$, and then by submultiplicativity we get

$$\Pr[T^* > T_1 + \cdots + T_{10}] \leq 2^{-10}.$$

Since $\bar{d}(k)$ is bounded by $\Pr[T^* > k]$, the initialization bias is at most 2^{-10}. (We cannot make any such rigorous assertion if we condition on the event $10T_{\text{est}} = 997$, nor should we expect to be able to do so in the absence of more detailed information about the nature of the Markov chain.)

Another approach would be to consider the maximum coupling time of a number of runs.

Theorem 12 *Let $T_1, \dots, T_m$ and T^* be independent samples of the coupling time. Then*

$$\Pr[T^* > j \max(T_1, \dots, T_m)] \leq \frac{j!m!}{(j+m)!}.$$

Proof: Let $T_{\max}$ denote $\max(T_1, \dots, T_m)$. We will construct coupling times $S_1, \dots, S_j$ and T^* such that $T^*, T_1, \dots, T_m$ are mutually independent and $T_1, \dots, T_m, S_1, \dots, S_j$ are mutually independent, but $S_1, \dots, S_j$ and T^* are dependent. Each coupling time is determined by the random variables (earlier called U_t) that are used in simulating the Markov chain; call these random variables "moves". Let the first $T_{\max}$ of the moves for the time S_i be used for moves $(i-1)T_{\max}$ through $iT_{\max} - 1$ of time T^*, and let the remaining random moves for S_i be independent of the other random moves. If for any i we have $S_i \leq T_{\max}$, then moves $(i-1)T_{\max}$ through $iT_{\max}-1$ of time T^* are coalescing, whence $T^* \leq jT_{\max}$. Thus, $\Pr[T^* > jT_{\max}] \leq \Pr[S_i > T_{\max},\ i = 1,\dots,j] = \Pr[S_i > T_k,\ i = 1,\dots,j,\ k = 1,\dots,m]$. But if we have $j+m$ i.i.d. random variables, the probability that the last j are strictly larger than the first first m is at most $1/\binom{j+m}{j}$. □

In the above example with $m = 10$, if we take $j = 6$ then $\Pr[T^* > 6T_{\max}] \leq 1/\binom{16}{6} = 1/8008$. If the longest of the ten runs takes 150 steps, then the randomized upper bound $6T_{\max}$ is 900. Calculations of this sort clearly can help an experimenter determine when her initialization bias is likely to be acceptably small.

3.3 Optimizing Performance

As we mentioned in section 3.1, when applying coupling from the past in the context of monotone Monte Carlo algorithms, it would be grossly inefficient to consider F_{-T}^0 for each positive integer T. We may restrict T to take on the values $T_0 < T_1 < \cdots$. Earlier we recommended taking the simulation start-times to be $-T_i = -2^i$. We shall see that this choice is close to optimal.

Let T_* denote the minimum value of T for which $F_{-T}^0(\hat{0}) = F_{-T}^0(\hat{1})$. One natural choice is to take $T_1 = rT_0$, $T_2 = rT_1$, etc., for some initial trial value T_0 and some ratio r. Then the number of simulation steps required to find the value of $F_{-T_*}^0(\hat{0}) = F_{-T_*}^0(\hat{1})$ is $2T_0 + 2rT_0 + 2r^2T_0 + \cdots + 2r^kT_0$, where k is the least k such that $r^kT_0 \geq T_*$. (The factor of 2 comes from the fact that we must simulate using both $\hat{0}$ and $\hat{1}$ as initial states.) The number of required steps is

$$\frac{r^{k+1}-1}{r-1}2T_0 < \frac{r^2}{r-1}r^{k-1}2T_0 \leq \frac{r^2}{r-1}2T_*$$

where in the second inequality we assumed $T_0 \leq T_*$. On the other hand, if one could magically guess T_*, then computing $F_{-T_*}^0(\hat{0})$ and $F_{-T_*}^0(\hat{1})$ (and in so doing verifying that the value of T_* is no greater than claimed) would take a full $2T_*$ simulation steps. The ratio between the two — worst-case and best-case — is at most $r^2/(r-1)$, which is minimized at $r = 2$, where it takes the value 4. Hence, if our goal is to minimize the worst-case number of steps, we should take $r = 2$.

One might be concerned that it is better to minimize the expected number of steps, rather than the worst case. Indeed, we argue that to minimize the expected number of steps one should pick $r = e$ $(= 2.71828\cdots)$. Let u be the fractional part of $\log_r(T_0/T_*)$. It is a reasonable heuristic to suppose that u is approximately uniformly distributed. In fact, by randomizing the choice of T_0, we can force this assumption to be true. In any case, the number of simulation steps needed to find the unbiased sample is

$$2r^uT_* + 2r^{u-1}T_* + 2r^{u-2}T_* + \cdots < 2\frac{r^u}{1-1/r}T_*.$$

The expected value of r^u is $(r-1)/\ln r$, so the expected number of steps is bounded above by $2T_*r/\ln r$. To minimize $r/\ln r$ we set $r = e$, with the expected number of steps equal to $2eT_* \approx 2.72(2T_*)$.

In practice, we do not expect to make great savings in time from this randomization strategy. Indeed, using $r = 2$ we find that the expected number of steps required is approximately 2.89 times $2T_*$, which is quite close to the optimal achieved by $r = e$. Hence we have adopted the simpler doubling-procedure.

Chapter 4

Examples of Monotone Markov Chains

4.1 Attractive Spin Systems and Distributive Lattices

Define a *spin system* on a vertex set V as the set of all ways of assigning a spin $\sigma(i)$ ("up"=$\uparrow$ or "down"=$\downarrow$) to each of the vertices $i \in V$, together with a probability distribution π on the set of such assignments $\sigma(\cdot)$. We order the set of configurations by putting $\sigma \geq \tau$ iff $\sigma(i) \geq \tau(i)$ for all $i \in V$ (where $\uparrow > \downarrow$), and we say that π is *attractive* if the conditional probability of the event $\sigma(i) = \uparrow$ is a monotone increasing function of the values of $\sigma(j)$ for $j \neq i$. (In the case of the Ising model, this corresponds to the ferromagnetic situation.)

More formally, given $i \in V$ and configurations σ, τ with $\sigma(j) \leq \tau(j)$ for all $j \neq i$, define configurations $\sigma_\uparrow$, $\sigma_\downarrow$, $\tau_\uparrow$, and $\tau_\downarrow$ by putting $\sigma_\uparrow(i) = \uparrow$, $\sigma_\uparrow(j) = \sigma(j)$ for all $j \neq i$, and so on. We say π is monotone iff $\pi(\sigma_\downarrow)/\pi(\sigma_\uparrow) \geq \pi(\tau_\downarrow)/\pi(\tau_\uparrow)$, or rather, if $\pi(\sigma_\downarrow)/\pi(\tau_\downarrow) \geq \pi(\sigma_\uparrow)/\pi(\tau_\uparrow)$, for all $\sigma \leq \tau$ and all $i \in V$.

A *heat bath algorithm* on a spin-system is a procedure whereby one cycles through the vertices i (using any mixture of randomness and determinacy that guarantees that each i almost surely gets chosen infinitely often) and updates the value at site i in accordance with the conditional probability for π. One may concretely realize this update rule by putting

$$f_t(\sigma, u_t) = \begin{cases} \sigma_\downarrow & \text{if } u_t < \pi(\sigma_\downarrow)/(\pi(\sigma_\downarrow) + \pi(\sigma_\uparrow)) \\ \sigma_\uparrow & \text{if } u_t \geq \pi(\sigma_\downarrow)/(\pi(\sigma_\downarrow) + \pi(\sigma_\uparrow)) \end{cases}$$

where t is the time, u_t is some random variable distributed uniformly in $[0, 1]$ (with all the u_t's independent of each other), and σ is some configuration of the system. If π is attractive, then this realization of π gives rise to a coupling-scheme that preserves the ordering (just use the same u_t's in all copies of the chain). To see why this is true, notice that if $\sigma < \tau$, then the u_t-threshold for σ is higher than for τ, so that the event $f_t(\sigma, u_t)(i) = \uparrow$, $f_t(\tau, u_t)(i) = \downarrow$ is impossible.

We now may conclude:

Theorem 13 *If one runs the heat bath for an attractive spin-system under the coupling-from-the-past protocol, one will generate states of the system that are exactly governed by the* *target distribution π.*

In certain cases this method is slow (for instance, if one is sampling from the Gibbs distribution for the Ising model below at the critical temperature), but in practice we find that it works fairly quickly for many attractive spin-systems of interest. In any case, as shown in section 3.2, in a certain sense the coupled-from-the-past version of the heat bath is no slower than heat bath.

We point out that, like the standard heat bath algorithm, ours can be accelerated if one updates many sites in parallel, provided that these updates are independent of one another. For instance, in the case of the Ising model on a square lattice, one can color the sites so that black sites have only white neighbors and vice versa, and it then becomes possible to alternate between updating all the white sites and updating all the black sites. In effect, one is decomposing the configuration σ as a pair of configurations $\sigma_{\mathbf{white}}$ and $\sigma_{\mathbf{black}}$, and one alternates between randomizing one of them in accordance with the conditional distribution determined by the other one.

4.1.1 Omnithermal sampling

In many cases, one is studying a one-parameter family of spin systems in which some quantity is varying. For instance, in the Ising model one can vary the strength of an external field, or in the random cluster model (see subsection 4.2.2) one can vary the temperature. If some parameter (say the temperature T) affects the ratio $\pi(\sigma_\uparrow) : \pi(\sigma_\downarrow)$ in a monotone (say increasing) way, then it is possible to make an "omnithermal" heat bath Markov chain, one that in effect generates simultaneous random samples for *all* values of the temperature. An omnithermal state σ assigns to each vertex i the set $c(i)$ of temperatures for which site i is spin-down. The sets $c(i)$ are "monotone" in the sense that if a temperature is in $c(i)$, so is every lower temperature; that is, when the temperature is raised, spin-down sites may become spin-up, but not vice versa. Given a site i of configuration σ and a random number u between 0 and 1, the heat bath update rule defines $c(i)$ to be the set of T for which $u_t < \pi_T(\sigma_{T,\downarrow})/(\pi_T(\sigma_{T,\downarrow}) + \pi_T(\sigma_{T,\uparrow}))$, where σ_T denotes σ at temperature T. For each T this update rule is just the ordinary heat bath, and monotonicity ensures that the new set $c(i)$ is monotone in the aforementioned sense. The omnithermal heat bath Markov chain is monotone with a maximum and minimum state, so monotone coupling from the past may be applied.

In the case where all the spin sites are independent of one another, the trick of simultaneously sampling for all values of a parameter has been used for some time in the theory of random graphs [6, chapter 10] and percolation [41]. Holley [44] used an omnithermal Markov chain with two temperatures to give an alternate proof of the FKG inequality [36] governing attractive spin systems. More recently Grimmett has given a monotone omnithermal Markov chain for the bond-correlated percolation model [40] (another name for the random cluster model described in subsection 4.2.2), and used it to derive a number of properties about these systems. Interestingly, Grimmett's Markov chain is different from the omnithermal heat bath chain. See subsection 4.2.2 and Chapter 5 for further discussion of omnithermal sampling and its uses.

4.2 Applications to Statistical Mechanics

When a physical system is in thermodynamic equilibrium, the probability that the system is in a given state σ is proportional to $e^{-E_\sigma/kT}$ where E_σ is the energy of state σ, T is the absolute temperature, and k is Boltzmann's constant. This probability distribution is known as the Gibbs distribution. In effect, kT is the standard unit of energy; when kT is large, the energies of the states are not significant, and all states are approximately equally likely, but when kT is small, the system is likely to be in a low-energy state. In some cases, a very small change in some parameter (such as the temperature) causes a significant change in the physical system. This phenomenon is called a *phase transition.* If changing the temperature caused the phase transition, then the temperature at the phase transition is called the *critical temperature.*

In the study of phase transitions, physicists often consider idealized models of substances. Phase-transition phenomena are thought to fall into certain *universality classes*, so that if a substance and a model belong to the same universality class, the global properties of the model correspond well to those of the real-world substance. For example, carbon dioxide, xenon, and brass are thought to belong to the same universality class as the three-dimensional Ising model (see [9] and [13], and references contained therein). Other models that have received much attention include the Potts model [82] (which generalizes the Ising model) and the related random cluster model [35].

In Monte Carlo studies of phase transitions, it is essential to generate many random samples of the state of a system. Sampling methods include the Metropolis algorithm and multi-grid versions of it. However, even after the Markov chain has run for a long time, it is not possible to tell by mere inspection whether the system has converged to a steady-state distribution or whether it has merely reached some metastable state. In many cases, we can use the method of coupled Markov chains to eliminate this problem and provide samples precisely according to the steady-state distribution.

In subsection 4.2.1 we review the definition of the Ising model and describe a simple algorithm for obtaining unbiased Ising samples that works well if the system is above (and not too close to) the critical temperature. We also define the Potts model, which generalizes the Ising model to the situation in which there are more than two possible spins.

In subsection 4.2.2 we consider the random cluster model. This model turns out to be very useful, in large part because random Ising and Potts states can be derived from random-cluster states. In many cases the best way to get Ising or Potts states is via the random cluster model. We show how to apply the method of monotone coupling from the past to get unbiased samples, and include a picture of a perfectly equilibrated Ising state obtained by this method (Figure 1-1).

Finally, in subsection 4.2.3 we show how to apply our methods to the square ice model and the dimer model on the square and honeycomb grids.

We also mention that interest in these models is not restricted to physicists. For instance, in image processing, to undo the effects of blurring and noise one may place a Gibbs distribution on the set of possible images. The energy of a possible image depends on the observed pixel values, and nearby pixels tend to have similar values. Sampling from this Gibbs distribution can be effective in reducing noise. See [38] and [11] for more information.

4.2.1 The Ising model

Here we introduce the Ising model, and the single-site heat bath algorithm for sampling from it. The efficiency of the heat bath algorithm has been the object of much study [78] [73] [37] [46] [69] [64]. To summarize, it runs quickly at temperatures above the critical temperature, but below this temperature it takes an enormously long time to randomize. (In subsection 4.2.2 we will describe a different algorithm which does not suffer from this "critical slowing down"; however, that algorithm does not apply when different parts of the substance are subjected to magnetic fields of different polarity, which makes that algorithm less suitable for some applications, such as image processing.)

The Ising model was introduced to model ferromagnetic substances; it is also equivalent to a lattice gas model [9]. An Ising system consists of a collection of n small interacting magnets, possibly in the presence of an external magnetic field. Each magnet may be aligned up or down. (In general there are more directions that a magnet may point, but in crystals such as $FeCl_2$ and $FeCO_3$ there are in fact just two directions [9].) Magnets that are close to each other prefer to be aligned in the same direction, and all magnets prefer to be aligned with the external magnetic field (which sometimes varies from site to site, but is often constant). These preferences are quantified in the total energy E of the system

$$E = -\sum_{i<j} \alpha_{i,j}\sigma_i\sigma_j - \sum_i B_i\sigma_i,$$

where B_i is the strength of the external field as measured at site i, σ_i is 1 if magnet i is aligned up and -1 if magnet i is aligned down, and $\alpha_{i,j} \geq 0$ represents the interaction strength between magnets i and j.

Often the n magnets are arranged in a 2D or 3D lattice, and $\alpha_{i,j}$ is 1 if magnets i and j are adjacent in the lattice, and 0 otherwise.

Characterizing what the system looks like at a given temperature is useful in the study of ferromagnetism. To study the system, we may sample a random state from the Gibbs distribution with a Markov chain. The single-site heat bath algorithm, also known as Glauber dynamics, iterates the following operation: Pick a magnet, either in sequence or at random, and then randomize its alignment, holding all of the remaining magnets fixed. There are two possible choices for the next state, denoted by $\sigma_\uparrow$ and $\sigma_\downarrow$, with energies $E_\uparrow$ and $E_\downarrow$. We have $\Pr[\sigma_\uparrow]/\Pr[\sigma_\downarrow] = e^{-(E_\uparrow - E_\downarrow)/kT} = e^{-(\Delta E)/kT}$. Thus a single update is simple to perform, and it is easy to check that this defines an ergodic Markov chain for the Gibbs distribution.

To get an exact sample, we make the following observation: If we have two spin-configurations σ and τ with the property that each spin-up site in σ is also spin-up in τ, then we may evolve both configurations simultaneously according to the single-site heat bath algorithm, and this property is maintained. The all spin-up state is "maximal", and the all spin-down state is "minimal", so we have a monotone Markov chain to which we can apply the method of coupling from the past. Indeed, this is just a special case of our general algorithm for attractive spin-systems.

It is crucial that the $\alpha_{i,j}$'s be non-negative; if this were not the case, the system would not be attractive, and our method would not apply. (A few special cases, such as a paramagnetic system on a bipartite lattice, reduce to the attractive case.) However, there are no additional constraints on the $\alpha_{i,j}$'s and the B_i's; once the system is known to be attractive, we can be sure that our method applies, at least in a theoretical sense.

Also note that we can update two spins in parallel if the corresponding sites are non-adjacent (i.e., if the associated $\alpha_{i,j}$ vanishes), because the spin at one such site does not affect the conditional distribution for the spins at another. For instance, on a square grid, we can update half of the spins in parallel in a single step, and then update the other half at the next step. Despite the speed-up available from parallelization, there is no guarantee that the heat bath Markov chain will be a practical one. Indeed, it is well-known that it becomes disastrously slow near the critical temperature.

An important generalization of the Ising model is the q-state Potts model, in which each site may have one of q different "spins". Wu [82] gives a survey describing the physical significance of the Potts model. In a Potts configuration σ, each site i has spin σ_i which is one of $1, 2, 3, \ldots, q$. The energy of a Potts configuration is

$$E = \sum_{i<j} \alpha_{i,j}(1 - \delta_{\sigma_i,\sigma_j}) + \sum_i B_i(1 - \delta_{\sigma_i,e_i}),$$

where δ is the Kronecker delta-function, equal to 1 if its subscripts are equal and 0 otherwise, $\alpha_{i,j} \geq 0$ is the interaction strength between sites i and j, B_i is the strength of the magnetic field at site i, and e_i is the polarity of the magnetic field at site i. As before, adjacent sites prefer to have the same spin. When $q = 2$, the Potts-model energy reduces to the Ising-model energy aside from an additive constant (which does not affect the Gibbs distribution) and a scaling factor of two (which corresponds to a factor of two in the temperature).

4.2.2 Random cluster model

The random cluster model was introduced by Fortuin and Kasteleyn [35] and generalizes the Ising and Potts models. (The random cluster model is also closely related to the Tutte polynomial of a graph (see [12]).) The Ising state shown in Figure 1-1 was generated with the methods described here.

In the random cluster model we have an undirected graph G, and the states of the system are subsets H of the edges of the graph. Often G is a two- or three-dimensional lattice. Each edge $\{i, j\}$ has associated with it a number p_{ij} between 0 and 1 indicating how likely the edge is to be in the subgraph. There is a parameter q which indicates how favorable it is for the subgraph to have many connected components. In particular, the probability of observing a particular subgraph $H \subseteq G$ is proportional to

$$\left(\prod_{\{i,j\}\in H} p_{ij}\right)\left(\prod_{\{i,j\}\notin H} (1 - p_{ij})\right) q^{\mathcal{C}(H)},$$

where $\mathcal{C}(H)$ is the number of connected components of H (isolated vertices count as components of size 1). To derive a random q-spin Potts state from a random H, one assigns a common random spin to all the vertices lying in any given connected component (see Sokal's survey [71] for more information on this).

Sweeny [74] used the results of Fortuin and Kasteleyn to generate random q-spin Potts states near the critical temperature. He used the "single-bond heat bath" algorithm (i.e.

Glauber dynamics) to sample from the random cluster model, and then converted these samples into Potts states. The single-bond heat bath algorithm for sampling from the random cluster model is a Markov chain which focuses on a single edge of G at a time and, conditioning on the rest of H, randomly determines whether or not to include this edge in the new state H'. It turns out that the random clusters are in some ways more directly informative than the Potts states themselves, since for instance they can be used to obtain more precise information on spin correlations.

Swendsen and Wang [75] proposed a different Markov chain based on the relation between the random cluster and Potts models. Given a random-cluster sample, they compute a random Potts state consistent with the clusters. Given a Potts state, they compute a random subgraph of G consistent with the Potts state. Their chain alternates between these two phases. In another variation due to Wolff [81], a single cluster is built from the spin states and then flipped. One major advantage of these approaches over the single-bond heat bath algorithm is that they obviate the need to determine connectivity for many pairs of vertices adjacent in V. Determining the connectivity can be computationally expensive, so Sweeny gave an algorithm for dynamically maintaining which vertices remain connected each time an edge is added or deleted. The dynamic connectivity algorithm is limited to planar graphs, but the Markov chain itself is general.

We will now argue that the single-bond heat bath algorithm is monotone for $q \geq 1$, so that the method of monotone coupling from the past applies to it. Here the states of the Markov chain are partially ordered by subgraph-inclusion, the top state is G, and the bottom state is the empty graph. We claim that when $q \geq 1$, this partial order is preserved by the Markov chain. At each step we pick an edge (either randomly or in sequence) and apply the heat bath Markov chain to this particular edge. If the two sites connected by this edge are in separate clusters in both states, or in the same cluster in both states, then the probability of including the edge is the same, so the partial order is preserved. If in one state they are in the same cluster while in the second state they are in separate clusters, then 1) the first state is the larger state, and 2) the probability of putting a bond there is larger for the first state. Hence the partial order is preserved, and our approach can be applied.

We have been unable to find monotonicity in the Swendsen-Wang or Wolff algorithms. Recent developments in dynamic connectivity algorithms [43] may make the single-bond heat bath algorithm a viable option, given that one can obtain exact samples at effectively all temperatures simultaneously using the heat bath algorithm. (See the discussion at the end of section 4.1, and also Chapter 5.)

While we know of no good rigorous bounds on how long monotone coupling from the past will take with the random cluster, in practice it runs fairly quickly. Jerrum and Sinclair solved the related problem of approximating the partition function (defined as the normalizing constant in the Gibbs distribution) in polynomial time [48].

4.2.3 Ice and dimer models

In the square ice (or "six-vertex") model whose residual entropy was determined by Lieb [60], states are assignments of orientation to the edges of a square grid, such that at each internal vertex there are equal numbers of incoming and outgoing edges (the "ice condition"). The six vertex configurations correspond to the six ways in which a water molecule in an ice crystal can orient itself so that its two protons (hydrogen atoms) are pointing towards two of the four adjacent oxygen atoms, and the ice condition reflects the fact that the protons from adjacent molecules will repel each other. The assumption that the ice condition is satisfied everywhere is tantamount to the assumption that the system is at temperature zero, so that in particular the Gibbs distribution is just the uniform distribution on the set of minimum-energy configurations.

Let us assume that our square grid is a finite rectangle, where sites along the boundary have edges that lead nowhere but nonetheless have a definite orientation. To turn ice-configurations on this rectangle into elements of a distributive lattice, we first pass to the dual model by rotating each directed edge 90 degrees clockwise about its midpoint. This gives a model on the dual square grid in which every internal square cell must be bounded by two clockwise edges and two counterclockwise edges. One may think of such an orientation as a "discrete conservative vector field" on the set of edges of the grid; here conservativity means that if one travels between two points in the grid, the number of forward-directed edges along which one travels, minus the number of backward-directed edges, is independent of the path one takes between the two points.

One can then introduce an integer-valued function on the vertices, called a *height function*, with the property that adjacent vertices have heights that differ by 1, such that the edge between vertex i and vertex j points from i to j if and only if j is the higher of the two vertices. (If one pursues the analogue with field theory, one might think of this as a potential function associated with the discrete vector field.) Height functions for the six-vertex model were first introduced by van Beijeren [80].

The set of height functions with prescribed boundary conditions can be shown to form a distributive lattice under the natural operations of taking the maximum or minimum of the heights of two height functions at all the various points in the grid [67]. Hence one can sample from the uniform distribution on the set of such height functions, and thereby obtain a random sample from the set of ice-configurations.

Note that this approach applies to models other than square ice; for instance, it also applies to the "twenty-vertex" model on a triangular grid, in which each vertex has three incoming edges and three outgoing edges. However, in all these applications it is important that the finite graph that one uses be planar. In particular, one cannot apply these ideas directly to the study of finite models with free periodic boundary conditions (or equivalently graphs on a torus), because such graphs are not in general simply connected.

Another class of models to which our methods apply are dimer models on bipartite planar graphs. A dimer configuration on such a graph is a subset of the edges such that every vertex of the graph belongs to exactly one of the chosen edges. This model corresponds to adsorption of diatomic molecules on a crystal surface, and the assumption that every vertex belongs to an edge corresponds to the assumption that the system is in a lowest-energy state, as will be the case at zero temperature.

Here, as in the case of ice-models, one can introduce a height function that encodes the combinatorial structure and makes it possible to view the states as elements of a distributive lattice. This approach can be traced back to Levitov [59] and Zheng and Sachdev [83] in the case of the square lattice, and to Blöte and Hilhorst [15] in the case of the hexagonal lattice. An independent development is due to Thurston [79], building on earlier work of Conway [24]. A generalization of this technique is described in [67].

Thurston's article describes the construction not in terms of dimer models (or equivalently perfect matchings of graphs) but rather in terms of tilings. That is, a dimer configuration on a square or hexagonal grid corresponds to a tiling of a plane region by dominoes (unions of two adjacent squares in the dual grid) or lozenges (unions of two adjacent equilateral triangles in the dual grid).

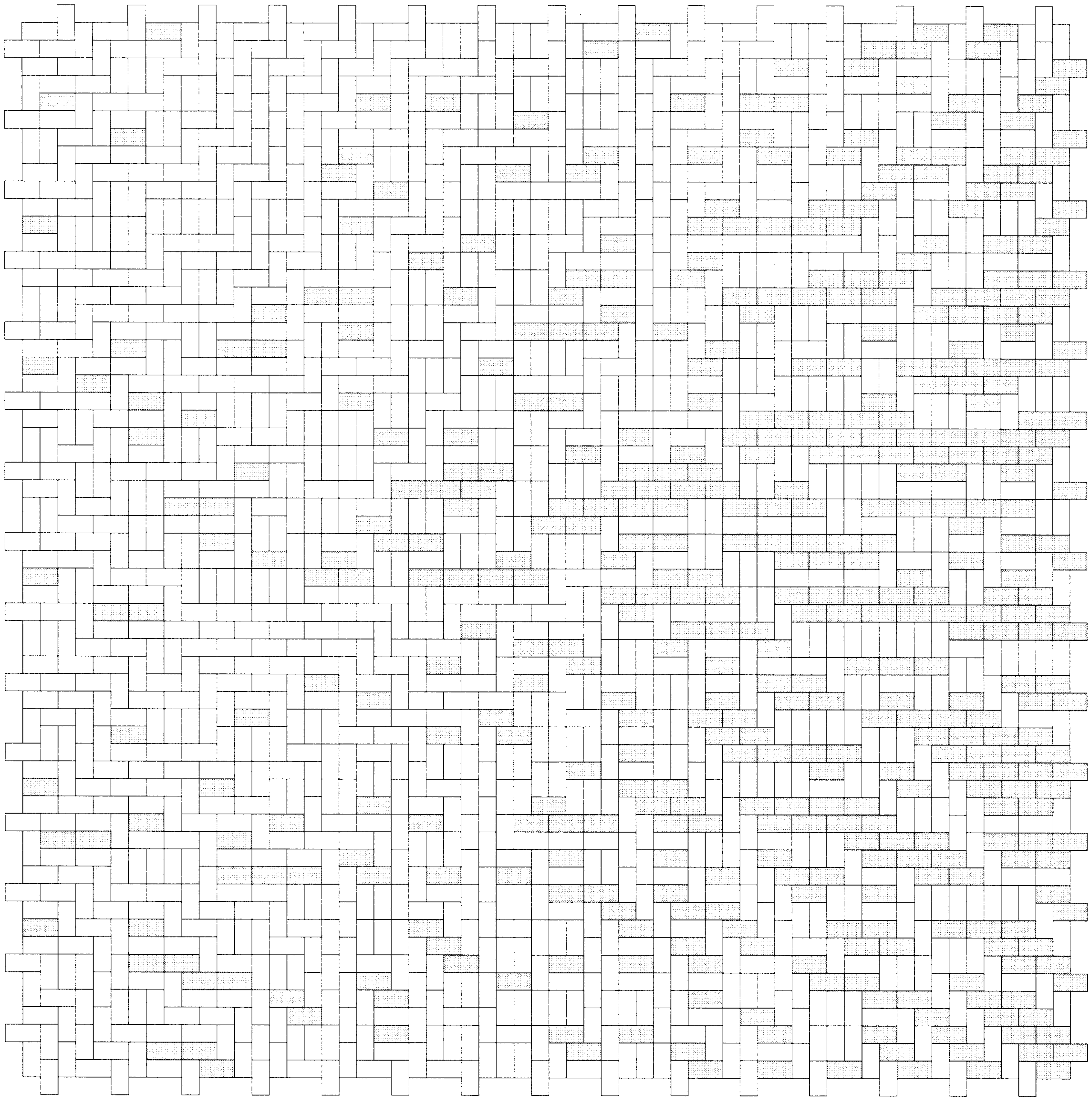

Figure 4-1: A random tiling of a finite region by dominoes.

Using the distributive lattice structure on the set of tilings, and applying coupling from the past, one can generate random tilings of finite regions. This permits us to study the effects that the imposition of boundary conditions can have even well away from the boundary. One such phenomenon is the "arctic circle effect" described in [49]; using our random-generation procedure we have been able to ascertain that such domain-wall effects are fairly general, albeit associated with boundary conditions that may be deemed non-physical.

For instance, Figure 4-1 shows a particular finite region and a tiling chosen uniformly at random from the set of all domino-tilings of that region. To highlight the non-homogeneity of the statistics of the tiling, we have shaded those horizontal dominoes whose left square is black, under the natural black/white checkerboard coloring of the squares. The heterogeneity of the statistics is a consequence of the "non-flatness" of the height function on the boundary; this phenomenon is given a more detailed study in [19]. The tiling was generated using software written by the Jim Propp's undergraduate research assistants, using the methods described in this article.

Computer programs for generating random configurations of this kind can used as exploratory tools for the general problem of understanding the role played by boundary conditions in these classes of models.

4.3 Combinatorial Applications

Order ideals and antichains

Let P be a finite partially ordered set, and call a subset I of P an *order ideal* if for all $x \in I$ and $y \leq x$, we have $y \in I$. The set of order ideals of P is denoted by $J(P)$; it is a distributive lattice under the operations of union and intersection, and it is a standard theorem (see [72]) that every finite distributive lattice is of the form $J(P)$ for some finite partially ordered set P.

To turn $J(P)$ into a spin-system, we let V be the set of elements of P, and we associate the order ideal I with the spin-configuration σ in which $\sigma(i)$ is $\uparrow$ or $\downarrow$ according to whether $i \in I$ or $i \notin I$. Let us give each spin-configuration that arises in this fashion equal probability $\pi(\sigma) = 1/|J(P)|$, and give the rest probability 0. Then it is easy to check that the ratio $\pi(\sigma_\uparrow)/\pi(\sigma_\downarrow)$ is $1/0$, $0/1$, or $1/1$, according to whether $\sigma \setminus \{i\}$ is not an order ideal, $\sigma \cup \{i\}$ is not an order ideal, or both sets are order ideals. Indeed, it is not much harder to see that π is attractive, so that by running the heat bath algorithm in a coupling-from-the-past framework, we can generate a uniform random element of $J(P)$.

We remind the reader that there is a one-to-one correspondence between order ideals of P and antichains of P (sets of pairwise-incomparable elements of P); specifically, for every order ideal I of P, the set of maximal elements of I forms an antichain, and every antichain determines a unique order ideal I. Hence, sampling uniformly from the elements of a finite distributive lattice is equivalent to sampling uniformly from the set of order ideals of a general finite poset, which is in turn equivalent to sampling uniformly from the set of antichains of a general finite poset.

We also point out that the heat bath procedure can often be done in parallel for many $i \in P$ at once. Suppose we color the elements of P so that no element covers another element of the same color. (If P is graded, then its Hasse diagram is bipartite and two colors suffice.) We use these vertex-colors to assign colors to the edges of the Hasse diagram G of $J(P)$. For definiteness, let us focus on one color, called "red". If I is an order ideal, and K is the set of all red elements of P whose adjunction to or removal from I yields an order ideal, then a "red move" is the operation of replacing I by the union of $I \setminus K$ with a random subset of K. If one alternates red moves with blue moves and so on, then one will converge to the uniform distribution on $J(P)$.

Cautionary note: While in many cases of interest this procedure will quickly find a random order ideal of a poset, it is not difficult to construct examples where the run time is very large. For instance, let P be the poset of $2n$ elements numbered $1, \ldots, 2n$, such that $x < y$ if and only if $x \leq n < y$ in the standard order on the integers. Then the Hasse diagram G of $J(P)$ will be two hypercubes joined at a single vertex, and the time for the upper and lower order ideals to coalesce will be exponential in n. Note however, that because of the bottleneck in this graph, the (uncoupled) random walk on this graph also takes exponential time to get close to random.

In the next two examples, there are other methods that can be applied to sample from the desired distribution, but it is still interesting to see how versatile the basic approach is.

Lattice paths. First, consider the set of all lattice-paths of length $a+b$ from the point $(a, 0)$ to the point $(0, b)$. There are $\binom{a+b}{a}$ such paths, and there are a number of elementary techniques that one can use in order to generate a path at random. However, let us define the number of *inversions* in such a path as the number of times that an upward step is followed (not necessarily immediately) by a leftward step, so that the path from $(a, 0)$ to (a, b) to $(0, b)$ has ab inversions, and let us decree that each lattice-path should have probability proportional to q to the power of the number of inversions, for some $q \geq 0$. It so happens in this case that one can work out exactly what the constant of proportionality is, because one can sum $q^{\#\text{ of inversions}}$ over all lattice-paths with two fixed endpoints (these are the coefficients of the so-called "Gaussian binomial coefficients" [72]), and as a result of this there exists an efficient bounded-time procedure for generating a random q-weighted lattice-path. However, one also has the option of making use of coupling from the past, as we now explain.

If we order the set of the unit squares inside the rectangle with corners $(0, 0)$, $(a, 0)$, $(0, b)$, (a, b) by decreeing that the square with lower-left corner (i', j') is less than or equal to the square with lower-left corner (i, j) if and only if $i' \leq i$ and $j' \leq j$, then we see that the unit squares that lie below and to the left of a lattice-path that joins $(a, 0)$ and $(0, b)$ form an order ideal, and that there is indeed a one-to-one-correspondence between the lattice-paths and the order ideals. Moreover, the number of inversions in a lattice path corresponds to the cardinality of the order ideal, so the order ideal I has probability proportional to $q^{|I|}$. It is not hard to show for any finite distributive lattice, a probability distribution of this form always makes it an attractive spin system. Therefore, the method applies. When $q = 1$, roughly $n^3 \log n$ steps on average are needed in order to generate an unbiased sample, where $a \approx b \approx n$.

Independent sets. A natural way to try to apply the heat bath approach to generate a random independent set in a graph G (that is, a subset no two of whose vertices are joined by an edge) is first to color the vertices so that no two adjacent vertices are the same color, and then, cycling through the color classes, replace the current independent set I by the union of $I \setminus K$ with some random subset of K, where K is the set of vertices of a particular color that are not joined by an edge to any vertex in I. It is simple to show that for any fixed color, this update operation preserves the uniform distribution on the set of independent sets, since it is nothing more than a random step in an edge-subgraph whose components are all degree-regular graphs (hypercubes, in fact). Since the composite mapping obtained by cycling through all the vertices gives an ergodic Markov chain, there is at most one stationary distribution, and the uniform distribution must be it.

Unfortunately, we do not know of any rigorous estimates for the rate at which the preceding algorithm gives convergence to the uniform distribution, nor do we know of a way to use coupling-ideas to get empirical estimates of the mixing time. However, in the case where G is bipartite, a pretty trick of Kim, Shor, and Winkler [56] permits us to apply monotone CFTP. Specifically, let us suppose that the vertices of G have been classified as white and black, so that every edge joins vertices of opposite color. If we write the independent set I as $I_{\text{white}} \cup I_{\text{black}}$, then the set of independent sets becomes a distributive lattice if one defines the meet of I and I' as $(I_{\text{white}} \cap I'_{\text{white}}) \cup (I_{\text{black}} \cup I'_{\text{black}})$ and their join as $(I_{\text{white}} \cup I'_{\text{white}}) \cup (I_{\text{black}} \cap I'_{\text{black}})$. Hence one can sample from the uniform distribution on the set of independent sets in any finite bipartite graph. (This Markov chain may be slowly mixing for some graphs; for instance, in the case of the complete bipartite graph on $n+n$ vertices, the Markov chain is isomorphic to the slowly-mixing Markov chain mentioned in the earlier cautionary note.)

Permutations. Recall that $\text{inv}(\pi)$ denotes the number of inversions of π, that is, the number of pairs (i, j) such that $i < j$ and $\pi(i) > \pi(j)$. For certain statistical applications, it is useful to sample permutations $\pi \in S_n$ on n items such that the probability of π is proportional to $q^{\text{inv}(\pi)}$; this is sometimes called "Mallows' phi model through Kendall's tau". For background see [52] and [27] and the references contained therein. We represent the permutation π by the n-tuple $[\pi(1), \ldots, \pi(n)]$.

Consider the following Markov chain which samples according to the aforementioned distribution. Pick a random pair of adjacent items. With probability $1/(q+1)$ put the two in ascending order, i.e. "sort them", and with probability $q/(q+1)$ put them in descending order, i.e. "un-sort" them. It is clear that this Markov chain is ergodic and preserves the desired probability distribution.

Define a partial order on S_n by $\pi < \sigma$ if and only if π can be obtained from σ by sorting adjacent elements (no un-sorting moves); this is called the weak Bruhat order [14]. The bottom element is the identity permutation $\hat{0} : i \mapsto i$ and the top element is the totally reversing permutation $\hat{1} : i \mapsto n+1-i$. The above Markov chain is a random walk on the Hasse diagram of this partial order. (See [72] for background on partially ordered sets.) This Markov chain, coupled with itself in the obvious way, does not preserve the partial order, even on S_3. However, it is still true that when the top and bottom states coalesce, all states have coalesced, as we will show below. In symbols, the claim is that $F_{t_1}^{t_2}(\hat{0}) = F_{t_1}^{t_2}(\hat{1})$ implies that $F_{t_1}^{t_2}(\cdot)$ is constant. Therefore the technique of coupling from the past can be applied to this Markov chain.

The reason that the top and bottom states determine whether or not all states get mapped to the same state is that suitable projections of the Markov chain *are* monotone. Given a permutation π, let $\theta_k(\pi)$ be an associated threshold function. That is, $\theta_k(\pi)$ is a sequence of 0's and 1's, with a 1 at location i if and only $\pi(i) > k$. Just as the Markov chain sorts or un-sorts adjacent sites in a permutation, it sorts or un-sorts adjacent sites in the 0-1 sequence. Indeed, these sequences of 0's and 1's correspond to lattice-paths of the sort considered in the first example.

A sequence s_1 of 0's and 1's *dominates* another such sequence s_2 if the partial sums of s_1 are at least as large as the partial sums of s_2, i.e., $\sum_{i=1}^{I} s_1(i) \geq \sum_{i=1}^{I} s_2(i)$ for all $0 \leq I \leq n$.

The Markov chain preserves dominance in sequences of 0's and 1's. This may be checked by case-analysis. For each k, we have that if $\theta_k(F_{t_1}^{t_2}(\hat{0})) = \theta_k(F_{t_1}^{t_2}(\hat{1}))$, then $\theta_k(F_{t_1}^{t_2}(\cdot))$ is constant. But note that if two permutations differ, then they must differ in some threshold function. Hence these threshold functions determine the permutation, and $F_{t_1}^{t_2}(\cdot)$ must be constant if it maps $\hat{0}$ and $\hat{1}$ to the same state.

Recently Felsner and Wernisch generalized this monotone Markov chain to sublattices of the weak Bruhat lattice. Given two permutations π_1 and π_2 with $\pi_1 \leq \pi_2$, the random walk is restricted to those permutations σ such that $\pi_1 \leq \sigma \leq \pi_2$. This Markov chain can be used to sample random linear extensions of a two-dimensional partially ordered set [31].

Even more recently, Kendall has shown how extend monotone coupling from the past to sample from a continuous state space known as the area-interaction point process, and reports that it works well in practice [53].

Chapter 5

Omnithermal Sampling

In the course of collecting examples of monotone Markov chains, the one that caught our attention as perhaps being the most interesting to people in another field was the heat bath Markov chain for sampling from the random cluster model. I went about implementing monotone coupling from the past as applied to this Markov chain principally to see how well it worked, and to serve as a demonstration that monotone CFTP is something that can be done. It turned out that (for q not too large) a relatively small number of "sweeps" through the lattice were necessary to achieve convergence. However, due to the time needed to compute connectivity queries that the Markov chain needed, it seemed unlikely that a physicist might be persuaded to use this exact sampling mechanism rather than put up with initialization bias from the related but non-monotone Swendsen-Wang or Wolff algorithms. However, as it turns out, the heat bath Markov chain can be extended to generate a sample state that encodes within it random samples of the random cluster model at all possible temperatures. Such "omnithermal samples" are much more informative than their single temperature counterparts that the Swendsen-Wang and Wolff algorithms produce. After all, many times one is interested in the shape of certain curves as a function of temperature, usually to characterize the singular behavior of the function around the critical temperature, and a single omnithermal sample yields an entire sampled curve, whereas a great many single-temperature samples would be required to get similar information. Given some quantity of computer time, *a priori* it is not obvious whether it is better to spend it generating a small number of very informative samples, or a larger number of less informative samples. This chapter serves as a preliminary report on experiments using perfectly random omnithermal samples.

5.1 Connectivity Algorithms

Since the vast majority of time spent simulating the single bond heat bath Markov chain for the random cluster model is spent determining connectivity, it is worth spending some time implementing good connectivity algorithms. Sweeney [74] gave an efficient dynamic algorithm for the case when the underlying graph is planar. Since the Markov chain incrementally updates the current subgraph, a dynamic algorithm can repeatedly update a data structure that is then used to answer the connectivity queries. Recently there has been much research in dynamic connectivity algorithms, some relying on the geometry associated with the graph, and some working for general graphs (see e.g. [43]).

The xrc program, which performed all of the simulations described in this chapter, contains within it three connectivity algorithms, each of which works for general graphs, and none of which are dynamic. One is used for generating single-temperature samples, the second for generating omnithermal samples, and the third, for analyzing omnithermal samples. The reader is refered to [26] for background on algorithms.

For the straightfoward single temperature single bond heat bath, the Markov chain needs to determine whether or not two vertices u and v (adjacent in the underlying graph) are connected in the current subgraph. For these queries, xrc does two interleaved breadth-first-searches, one starting from u and the other from v. If either u or v is in a small connected component, then the algorithm will quickly determine the answer one way or the other. (Note that if one did a search starting from just u, it might be that u is in a percolating component[1] but that v is not. In that case the procedure would take a very long time, and hence the reason for two interleaved searches.) If both u and v are in large components, then because of the geometry of the grid, chances are that these are the same component, and that there is a short path between u and v, which the procedure will quickly find. Thus we have a heuristic argument that the interleaved two-source breadth-first-search is usually fast, and this seems to be borne out in practice.

There is more than one omnithermal single bond heat bath Markov chain for the random cluster model, but both of them require the same connectivity information. We are given two vertices u and v (as before adjacent in the underlying graph), and what we wish to determine is the first value of p at which u and v become connected. One approach is to do a binary search on the possible values of p, using the above interleaved breadth first search. This approach works reasonably well, say with 16-bit values of p, but for increased precision, another algorithm becomes desireable: two-source interleaved Dijkstra's algorithm.

Once some omnithermal samples have been generated, it is necessary to analyze them. To do this, we sort the edges by their p value, and adjoin them one at a time to the current graph, initially empty. Each time an edge is adjoined, certain physical quantities such as the expected internal energy can be updated. For doing this, xrc uses the standard subset-union algorithm.

The implementation of the monotone coupling-from-the-past version of the single-bond heat bath algorithm, in both single-temperature and omnithermal versions (see subsection 4.1.1), appears to work well in practice, provided that q is not too large. For instance, when $q = 2$ on a 512×512 toroidal grid, at the critical temperature it is only necessary to start at around time -30 to get coalescence by time 0. (Here one step, also called a sweep, consists of doing a single-bond heat bath step for each edge of the graph.) Using the above connectivity algorithm, each sweep takes about twenty seconds on a Sparcstation. Omnithermal sweeps (see subsection 4.1.1) take longer, in part because determining connectivity is somewhat more involved, but principally because in the single-temperature case a large fraction of the heat bath steps do not require a connectivity query, whereas in the omnithermal case they all do. Since nearly all of the computer's time is spent computing

[1]A percolating component is an infinite component, so technically finite graphs do not have them. Nonetheless finite graphs will have very large components when their infinite counterparts have percolating components, and u might be in such a large component.

connectivity, the incentive for improving on current dynamic connectivity algorithms is clear.

5.2 Empirical Results

The preliminary omnithermal results are striking; in Figure 1-4 one can see quite clearly the critical point, and how the internal energy and spontaneous magnetization vary with temperature on the three-dimensional grid when $q = 2$. Other macroscopically-defined quantities may also be graphed as a monotone function of p. This makes it possible to obtain estimates of the critical exponents from just a single (omnithermal) sample, whereas other approaches require a large number of samples at a number of different values of p close to criticality.

It is generally thought that the internal energy, spontaneous magnetization, and other quantities follow a power law near the critical. Figures 5-1 and 5-2 show log-log plots of these quantities. In both log-log plots there are two curves, one for the region $p < p_c$, and one for the region $p > p_c$, though for the spontaneous magnetization only the $p > p_c$ curve is interesting. The slopes of these curves will be the critical exponents. There remains much work to do, but these figures should give an indication of what one can do with omnithermal samples.

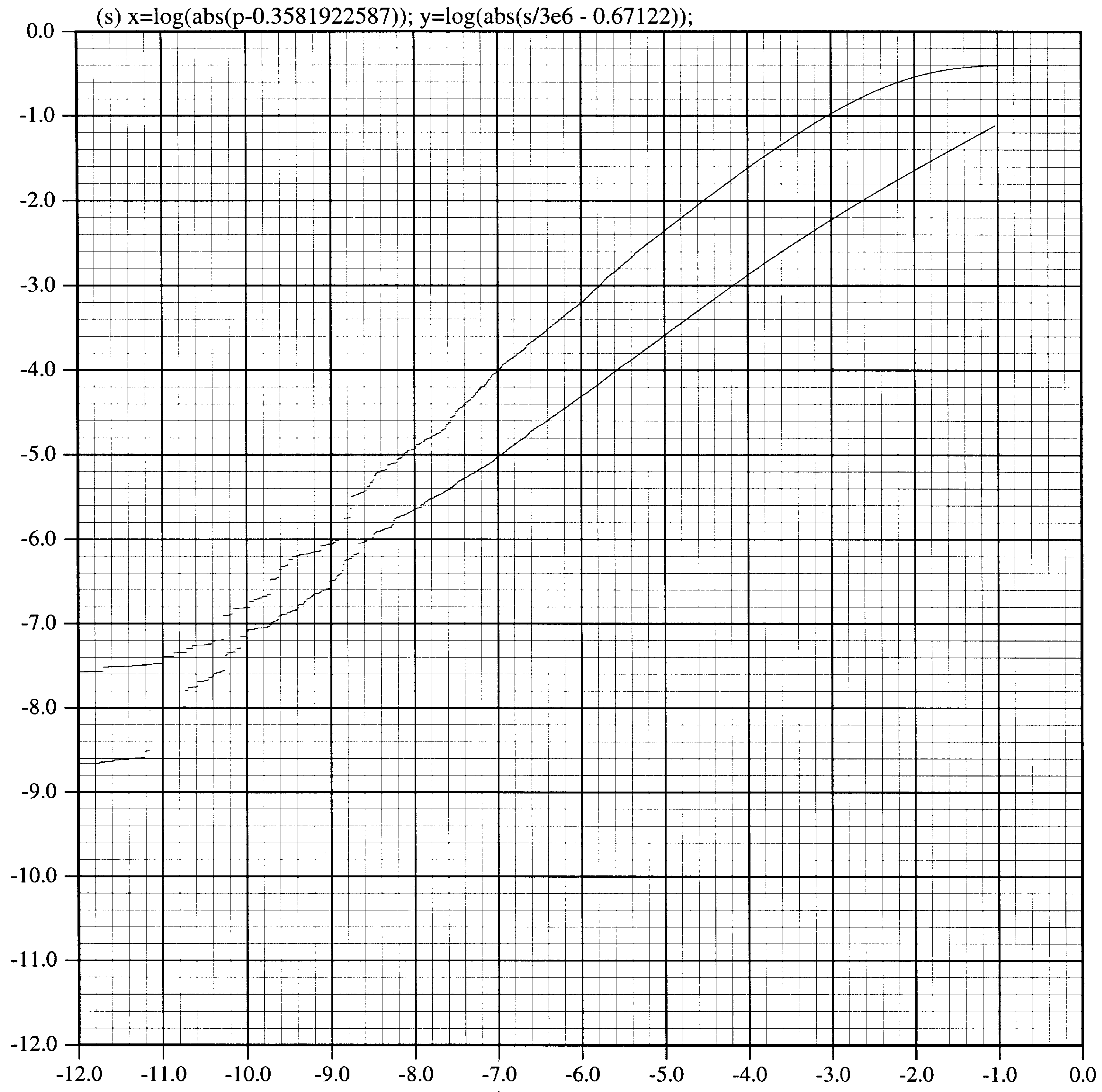

Figure 5-1: A log-log plot of the internal energy minus the critical energy versus the difference between p and p_c.

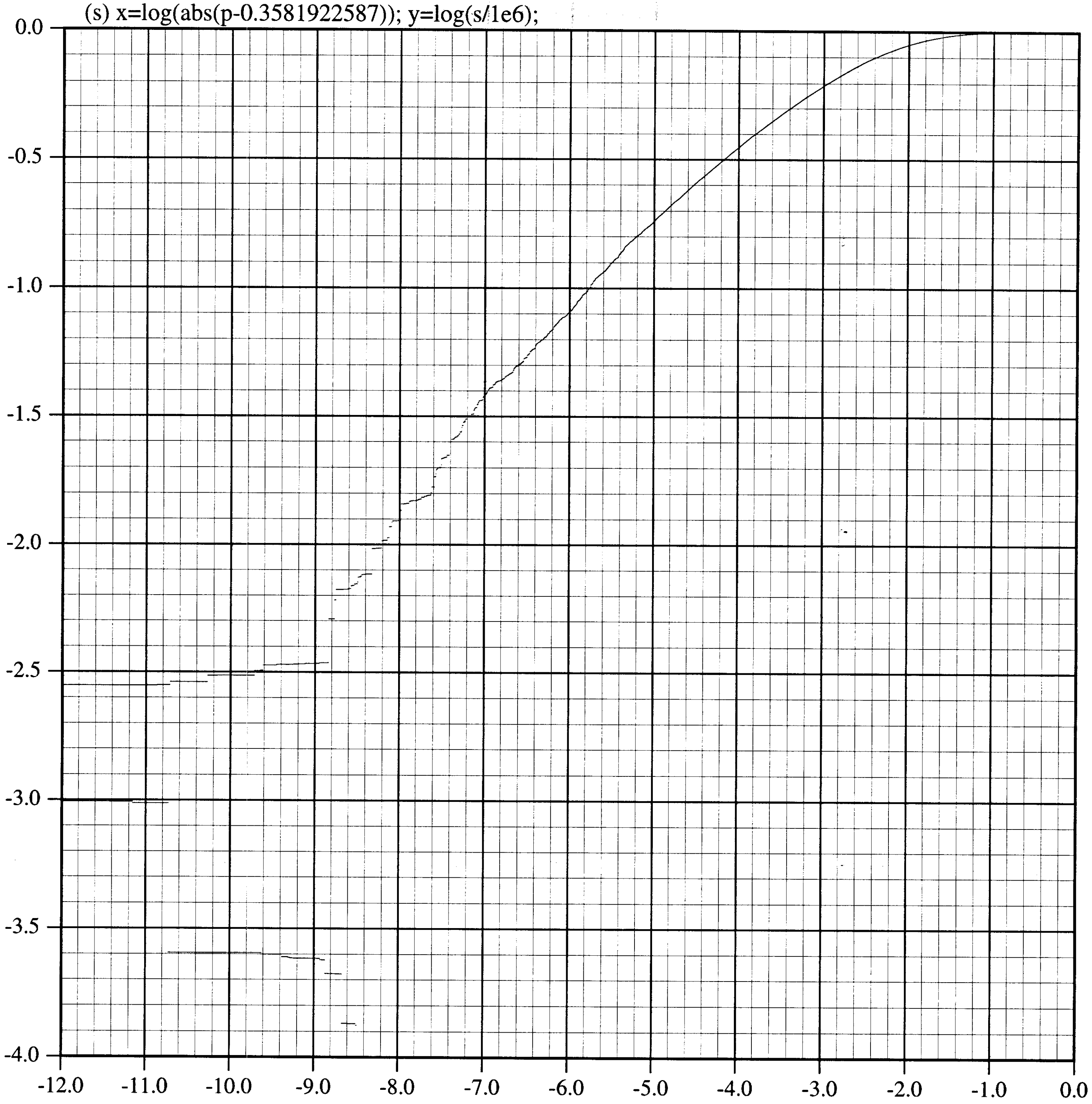

Figure 5-2: A log-log plot of the spontaneous magnetization versus the difference between p and p_c.

Chapter 6

Random Spanning Trees via Coupling From the Past

In this chapter we see how to obtain random spanning trees (arborescences) from a directed graph within $O(T_c)$ time. Most graphs have exponentially many arborescences; the complete graph has n^{n-1} of them. Visiting all the states in the space of trees is infeasible, but we need only wait a constant multiple of the time needed for the random walk on the underlying graph to visit all the vertices. First we describe how to sample arborescences with prescribed root efficiently, and then we extend the algorithm to sampling arborescences with unconstrained root.

Consider the following Markov chain on directed spanning trees, with edges directed to the root. Take the root, move it one step according to the underlying Markov chain on the graph. The new vertex becomes the new root. Make an arc from the old root to the new one, and delete the out-going edge from the new root. See Figure 6-1. It is easy to check that the stationary distribution of this Markov chain is such that the probability of a directed tree is proportional to the product of the edge transition probabilities of its arcs.

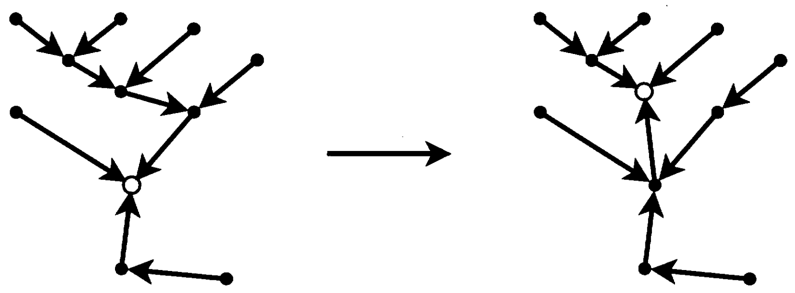

Figure 6-1: Example move of the Markov chain on arborescences. The root, shown in white, moves according to the underlying Markov chain.

Broder and Aldous used this Markov chain to sample random spanning trees of undirected graphs. They used the fact that the simple random walk on an undirected graph is a reversible Markov chain to make an algorithm that outputs a random tree in finite time. Kandel et al. also use this Markov chain in their sampling of spanning arborescences of Eulerian directed graphs. We will apply the method of coupling from the past to the tree Markov chain to sample from the spanning arborescences of a general directed graph.

Consider the following Markov chain M_r on the set of spanning trees rooted at r: Given a spanning tree rooted at r, perform the random walk as above until the root returns to r. Call the path the walk takes from r to r an excursion. The resulting spanning tree is the next state of the Markov chain. The following two easy lemmas show that this Markov chain's steady state distribution is random on the set of spanning trees rooted at r.

Lemma 14 *M_r is irreducible and aperiodic if the underlying graph is strongly connected.*

Proof: Since the graph is strongly connected, there is a trajectory $r = u_0, u_1, \dots, u_l = r$ that visits every vertex. Consider a tree T rooted at r. For i from $l-1$ down to 1, perform an excursion(s) that goes to u_i via the trajectory, then returns to r via the path in T. For each vertex v other than r, the last time v was reached, the succeeding vertex is its parent in the tree. Regardless of the initial starting tree, after a fixed number of excursions the final tree is T. Thus M_r can be neither reducible nor periodic. $\square$

Lemma 15 *M_r preserves the right distribution on trees rooted at r.*

Proof: It is straightforward to see that M preserves the distribution on trees. Pick a random tree, and do the walk N steps. Each tree in the walk is random. If the ith tree is rooted at r, then it is a random tree rooted at r. Consider the subsequence of trees rooted at r. Each tree occurs in the subsequence with the proper frequency, and each tree (other than first) is obtained from the previous tree by running M_r one step. $\square$

We can use the Markov chain M_r to in making a `RandomMap()` procedure that satisfies the requirements of the CFTP procedure: (1) `RandomMap()` preserves the steady state distribution on trees, (2) maps returned by `RandomMap()` may be easily composed, and (3) it is easy to determine whether or not a composition of random tree maps is collapsing.

We will represent spanning trees with arrays of size n, with the ith array entry giving the parent of vertex i, and the array entry of the root being blank. We will also represent maps from the set of trees rooted at r to the set of trees rooted at r with an array of size n. Each array entry is either blank, or else contains a vertex label. (We won't be able to represent all such maps in this way, but this does not matter.) Given a map M (represented as an array) and a tree T rooted at r, the new tree can be obtained by performing the following operation for each array index i: if $M[i]$ is not blank, copy it into $T[i]$. The way we construct the map array will guarantee that the result will always be a tree.

Given an excursion from the root to itself, if we visit vertex x, set $M[x]$ to be the next vertex we visit. If we visit x multiple times, it is the next vertex of the last time the excursion visits x. If the excursion does not visit x, $M[x]$ is blank. Finally, set $M[r]$ to be blank. Applying the map defined by M to a tree has the effect of running the Markov chain M_r one step. Thus condition (1) is satisfied.

It is straightforward to compose two spanning tree maps M_1 and M_2. If the map M_2 is applied before M_1, the composition map is obtained by setting $(M_1 \circ M_2)[i]$ to $M_1[i]$, unless $M_1[i]$ is blank, in which case we set $(M_1 \circ M_2)[i]$ to $M_2[i]$. Thus we can compose tree maps efficiently. We can also test if a map is collapsing — it is collapsing if the only blank entry is at the root. Hence all the requirements for CFTP are satisfied:

Theorem 16 *The procedure shown in Figure 6-2 returns a random arborescence with root r. The expected number of times we observe the Markov chain is $\leq 3T_c$, and the memory is $O(n)$.*

Proof: The procedure repeatedly samples excursions which define random tree maps as described above, and composes them, prepending new maps to the current map as prescribed by CFTP, and returns the resulting tree. The expected runtime is at most three cover times: we wait until we see the root, then visit all the vertices, and then return to the root. $O(n)$ memory is used to store the directed tree and remember which vertices were discovered during the present excursion. □

Note that the $3T_c$ time bound is rather pessimistic. If many arborescences with the given root are desired, then the average time per directed tree is one cover-and-return time.

Corollary 17 *We may sample random arborescences within 18 cover times.*

Proof: The root of a random arborescence is distributed according to π, the stationary distribution of the underlying Markov chain. So we may pick a random root using the unbiased state-sampler treated in Chapter 2, and then pick a random tree given that root. We need only observe the Markov chain for 15+3 cover times on average. The computational effort is also $O(T_c)$, and the memory is $O(n)$. □

Suppose one is given a weighted directed graph and a random directed spanning tree is desired, with probability proportional to the product of the edge weights. If all the weighted out-degrees are the same, then we may normalize them to be one, and directly apply the above algorithm on the associated Markov chain. Even if the out-degrees are different, we may still sample a random arborescence with prescribed root. The only potential difficulty lies in picking the root of a random arborescence. But this problem is readily resolved by considering the continuous time Markov chain associated with the graph. It is straightforward to generalize the unbiased sampler, specifically the `RandomMap()` procedure, to work in continuous time. The waiting time for a transition is an exponential random variable. If we define T_c to be the expected number of transitions before the graph is covered, then the runtime is $O(T_c)$.

```
wait until we visit $r$
for $i \leftarrow 1$ to $n$
   $status[i] \leftarrow$ UNSEEN
$Tree[r] \leftarrow$ nil
$status[r] \leftarrow$ DEFINED
$num\_assigned \leftarrow 1$
while $num\_assigned < n$
   $ptr \leftarrow 0$
   $s \leftarrow r$
   repeat
      $t \leftarrow$ NextState()
      if $status[s] =$ UNSEEN then
         $status[s] \leftarrow$ SEEN
         $stack[ptr++] \leftarrow s$
      if $status[s] =$ SEEN then
         $Tree[s] \leftarrow t$
      $s \leftarrow t$
   until $s = r$
   $num\_assigned \leftarrow num\_assigned + ptr$
   while $ptr > 0$
      $status[stack[--ptr]] \leftarrow$ DEFINED
return $Tree$
```

Figure 6-2: Pseudocode for generating a random arborescence with prescribed root within 3 cover times. The parent of node i is stored in $Tree[i]$. If node i has not been seen, then $status[i] =$ UNSEEN. The nodes seen for the first time in the present excursion from the root are stored on a stack. If i is such a node, then $status[i] =$ SEEN and $Tree[i]$ may be changed if i is seen again in the present excursion. When the root is encountered the present excursion ends; $status[i]$ is set equal to DEFINED, making $Tree[i]$ immutable for the nodes i seen so far and thereby effectively prepending the excursion just finished to the preceding excursions.

Chapter 7

Random Spanning Trees via Cycle Popping

7.1 Cycle Popping Algorithms

In this chapter we describe a process called cycle poppingn that is quite effective at returning random spanning trees of a directed weighted graph. As with CFTP, we give a couple of variations, some being simpler, others being more general. One variation, `RandomTreeWithRoot()`, will return a tree with a specified root. When the root is not specified, then sometimes it is possible to randomly select the root so that the problem is reduced to a single call to `RandomTreeWithRoot()`. This is the case for undirected or Eulerian graphs. For general graphs, one can use `RandomTree()`.

Recall from section 1.4 that for a given graph G we can define two different Markov chains, $\bar{G}$ and $\tilde{G}$, where in $\tilde{G}$ we first adjoin self loops to make the graph out-degree regular. `RandomTreeWithRoot()` uses $\bar{G}$ since $\Upsilon_r(\bar{G}) = \Upsilon_r(G)$. `RandomTree()` uses $\tilde{G}$ since $\Upsilon(\tilde{G}) = \Upsilon(G)$. Both procedures use a subroutine `RandomSuccessor(`u`)` that returns a random successor vertex using the appropriate Markov chain. As before, Markov chain parameters written with overbars refer to $\bar{G}$, and parameters written with tildes refer to $\tilde{G}$.

`RandomTreeWithRoot()` (see Figure 7.1) maintains a "current tree", which initially consists of just the root. While there remain vertices not in the tree, the algorithm does a random walk from one such vertex, erasing cycles as they are created, until the walk encounters the current tree. Then the cycle-erased trajectory gets added to the current tree. It has been known for some time that the path from a vertex to the root of a random spanning tree is a loop-erased random walk (see e.g. [66] and [18]), but this is the first time that anyone has used this fact to make a provably efficient algorithm. See [58] for background on loop-erased random walks.

Theorem 18 `RandomTreeWithRoot(`r`)` *returns a random spanning tree rooted at* r.

The proofs of this and subsequent theorems in the introduction are in section 7.2.

Suppose that what we want is a random spanning tree without prescribing the root. If we can easily pick a random vertex from the steady state distribution $\tilde{\pi}$ of the random walk on $\tilde{G}$, then pick a $\tilde{\pi}$-random root, and call `RandomTreeWithRoot()` with this root. The result

```
RandomTreeWithRoot($r$)
  for $i \leftarrow 1$ to $n$
    $InTree[i] \leftarrow$ false
  $Tree[r] \leftarrow$ nil
  $InTree[r] \leftarrow$ true
  for $i \leftarrow 1$ to $n$
    $u \leftarrow i$
    while not $InTree[u]$
      $Tree[u] \leftarrow$ RandomSuccessor($u$)
      $u \leftarrow Tree[u]$
    $u \leftarrow i$
    while not $InTree[u]$
      $InTree[u] \leftarrow$ true
      $u \leftarrow Tree[u]$
  return $Tree$
```

Figure 7-1: Algorithm for obtaining random spanning tree with prescribed root r.

is a random spanning tree. Aldous calls this theorem "the most often rediscovered result in probability theory" [5]; [17] includes a nice proof.

For undirected graphs and Eulerian graphs, $\bar{\pi}$ is just the uniform distribution on vertices. In the case of undirected graphs, since any vertex r may be used to generate a free spanning tree, it turns out to be more efficient to pick r to be a random endpoint of a random edge, sample from Υ_r, and then pick a uniformly random vertex to be the root.

Theorem 19 *The number of times that* `RandomTreeWithRoot(r)` *calls* `RandomSuccessor()` *is given by the mean commute time between r and a $\bar{\pi}$-random vertex. (The running time is linear in the number of calls to* `RandomSuccessor()`*.)*

With E_iT_j denoting the expected number of steps for a random walk started at i to reach j, the mean commute time between i and j is $E_iT_j + E_jT_i$, and is always dominated by twice the cover time.

The *mean hitting time* is the expected time it takes to go from a π-random vertex to another π-random vertex:

$$\tau = \sum_{i,j} \pi(i)\pi(j)E_iT_j.$$

(A nice fact, which the proofs will not need, is that for each start vertex i, $\tau = \sum_j \pi(j)E_iT_j$ [5].) If G is stochastic, and we have a π-random vertex r as root, `RandomTreeWithRoot()` makes an average of 2τ calls to `RandomSuccessor()`. For undirected graphs, a random endpoint of a random edge is $\bar{\pi}$-random, so the variation described above runs in $2\bar{\tau}$ time. In these cases it is perhaps surprising that the running time should be so small, since the expected time for just the first vertex to reach the root is τ. The expected additional work needed to connect all the remaining vertices to the root is also τ.

For general graphs RandomTree() (see Figure 7.2) may be used to sample a random spanning tree within $O(\tilde{\tau})$ time. Note that since the root of a random tree is distributed according to $\tilde{\pi}$, the second algorithm automatically yields a random sampling procedure for generic Markov chains: return the root of a random spanning tree.

```
RandomTree()
   $\varepsilon \leftarrow 1$
   repeat
      $\varepsilon \leftarrow \varepsilon/2$
      $tree \leftarrow$ Attempt()($\varepsilon$)
   until $tree \neq$ Failure
   return $tree$

Attempt()($\varepsilon$)
   for $i \leftarrow 1$ to $n$
      $InTree[i] \leftarrow$ false
   $num\_roots \leftarrow 0$
   for $i \leftarrow 1$ to $n$
      $u \leftarrow i$
      while not $InTree[u]$
         if Chance($\varepsilon$) then
            $Tree[u] \leftarrow$ nil
            $InTree[u] \leftarrow$ true
            $num\_roots \leftarrow num\_roots + 1$
            if $num\_roots = 2$ then
               return Failure
         else
            $Tree[u] \leftarrow$ RandomSuccessor($u$)
            $u \leftarrow Tree[u]$
      $u \leftarrow i$
      while not $InTree[u]$
         $InTree[u] \leftarrow$ true
         $u \leftarrow Tree[u]$
   return $Tree$
```

Figure 7-2: Algorithm for obtaining random spanning trees. Chance(ε) returns true with probability ε.

The second algorithm is essentially the same as the first algorithm, except that it adjoins an extra vertex labeled "death" to the graph. The probability of a normal vertex moving to death is ε, and the other transition probabilities are scaled down by $1-\varepsilon$. Since the death node is a sink, it makes a natural root from which to grow a spanning tree. The death node is then deleted from the spanning tree, resulting in a rooted forest in the original graph. If the forest has one tree, then it is a random tree. Otherwise ε is decreased and another try is made.

Theorem 20 *If* `Attempt()` *returns a spanning tree, then it is a random spanning tree. Furthermore,* `RandomTree()` *calls* `RandomSuccessor()` *on average $\leq 22\tau$ times, so the expected running time of* `RandomTree()` *is $O(\tau)$.*

7.2 Analysis of Cycle Popping

We will describe a randomized process that results in the random generation of a tree with a given root r. The procedure `RandomTreeWithRoot()` simulates this process. Then we reduce the problem of finding a random tree to that of finding a random tree with a given root, and analyze the running time.

Associate with vertex r an empty stack, and associate with each vertex $u \neq r$ an infinite stack of random vertices $S_u = S_{u,1}, S_{u,2}, S_{u,3}, \ldots$ such that

$$\Pr[S_{u,i} = v] = \Pr[\texttt{RandomSuccessor}(u) = v],$$

and such that all the items in all the stacks are mutually independent. The process will pop items off the stacks. To pop an item off u's current stack $S_{u,i}, S_{u,i+1}, S_{u,i+2}, \ldots$, replace it with $S_{u,i+1}, S_{u,i+2}, \ldots$.

The tops of the stacks define a directed graph G, which contains edge (u, v) if and only if u's stack is nonempty and its top (first) item is v. If there is a directed cycle in G, then by "popping the cycle" we mean that we pop the top item of the stack of each vertex in the cycle. The process is summarized in Figure 7.2. If this process terminates, the result will

```
while $G$ has a cycle
    Pop any such cycle off the stacks
return tree left on stacks
```

Figure 7-3: Cycle popping procedure.

be a directed spanning tree with root r. We will see later that this process terminates with probability 1 iff there exists a spanning tree with root r and nonzero weight. But first let us consider what effect the choices of which cycle to pop might have.

For convenience, suppose there are an infinite number of colors, and that stack entry $S_{u,i}$ has color i. Then the directed graph G defined by the stacks is vertex-colored, and the cycles that get popped are colored. A cycle may be popped many times, but a colored cycle can only be popped once. If eventually there are no more cycles, the result is a colored tree.

Theorem 21 *The choices of which cycle to pop next are irrelevant: For a given set of stacks, either 1) the algorithm never terminates for any set of choices, or 2) the algorithm returns some fixed colored tree independent of the set of choices.*

Proof: Consider a colored cycle C that can be popped, i.e. there is a sequence of colored cycles $C_1, C_2, C_3, \ldots, C_k = C$ that may be popped one after the other until C is popped. But suppose that the first colored cycle that the algorithm pops is not C_1, but instead $\tilde{C}$. Is it still possible for C to get popped? If $\tilde{C}$ shares no vertices with $C_1, \ldots, C_k$, then the answer is clearly yes. Otherwise, let C_i be the first of these cycles that shares a vertex with $\tilde{C}$. If C_i and $\tilde{C}$ are not equal as cycles, then they share some vertex w which has different successor vertices in the two cycles. But since none of $C_1, \ldots, C_{i-1}$ contain w, w has the same color in C_i and $\tilde{C}$, so it must have the same successor vertex in the two cycles. Since $\tilde{C}$ and C_i are equal as cycles, and $\tilde{C}$ shares no vertices with $C_1, \ldots, C_{i-1}$, $\tilde{C}$ and C_i are equal as colored cycles. Hence we may pop colored cycles $\tilde{C} = C_i, C_1, C_2, \ldots, C_{i-1}, C_{i+1}, \ldots, C_k = C$.

If there are infinitely many colored cycles which can be popped, then there always will be infinitely many colored cycles which can be popped, and the algorithm never terminates. If there are a finite number of cycles which can be popped, then every one of them is eventually popped. The number of these cycles containing vertex u determines u's color in the resulting tree. $\square$

To summarize, the stacks uniquely define a tree together with a partially ordered set of cycles layered on top of it. The algorithm peels off these cycles to find the tree.

An implementation of the cycle popping algorithm might start at some vertex, and do a walk on the stacks so that the next vertex is given by the top of the current vertex's stack. Whenever a vertex is re-encountered, then a cycle has been found, and it may be popped. If the current tree (initially just the root r) is encountered, then if we redo the walk from the start vertex with the updated stacks, no vertex encountered is part of a cycle. These vertices may then be added to the current tree, and the implemenation may then start again at another vertex. `RandomTreeWithRoot()` is just this implementation. `RandomSuccessor(`u`)` reads the top of u's stack and deletes this item; in case this item wasn't supposed to be popped, it gets stored in the *Tree* array. The *InTree* array gives the vertices of the current tree.

If there is a tree with root r and nonzero weight, then a random walk started at any vertex eventually reaches r with probability 1. Thus the algorithm halts with probability 1 if such a tree exists.

Proof of Theorem 18: Consider the probability that the stacks define a tree T rooted at r and a set $\mathcal{C}$ of colored cycles. By the i.i.d. nature of the stack entries, this probability factors into a term depending on $\mathcal{C}$ alone and a term depending on T alone — the product of the cycle weights, and the weight of T. Even if we condition on a particular set of cycles being popped, the resulting tree is distributed according to Υ_r. $\square$

The proof of Theorem 18 also shows that if we sum the weights of sets of colored cycles that exclude vertex r, the result is the reciprocal of the weighted sum of trees rooted at r.

Proof of Theorem 19: How many times will `RandomSuccessor(`u`)` get called on average? Since the order of cycle popping is irrelevant, we may assume that the first trajectory starts at u. It is a standard result (see [5]) that the expected number of times the random walk started at u returns to u before reaching r is given by $\pi(u)[E_u T_r + E_r T_u]$, where the "return" at time 0 is included, and $E_i T_j$ is the expected number of steps to reach j starting from i. Thus the number of calls to `RandomSuccessor()` is

$$\sum_u \pi(u)(E_u T_r + E_r T_u)$$

$\square$

If the root r is π-random, then the expected number of calls to RandomSuccessor() is

$$\sum_{u,r} \pi(u)\pi(r)(E_u T_r + E_r T_u) = 2\tau.$$

Since the number of calls to RandomSuccessor() is at least $n-1$, we get for free

$$\tau \geq \frac{n-1}{2}.$$

This inequality isn't hard to obtain by other methods, but this way we get a nice combinatorial interpretation of the numerator.

Proof of Theorem 20: Consider the original graph modified to include a "death node", where every original node has an ε chance of moving to the death node, which is a sink. Sample a random spanning tree rooted at the death node. If the death node has one child, then the subtree is a random spanning tree of the original graph. The Attempt() procedure aborts if the death node will have multiple children, and otherwise it returns a random spanning tree.

The expected number of steps before the second death is $2/\varepsilon$, which upper bounds the expected running time of Attempt(). Suppose that we start at a π-random location and do the random walk until death. The node at which the death occurs is a π-random node, and it is the death node's first child. Suppose that at this point all future deaths are suppressed. By Theorem 19, the expected number of additional calls to RandomSuccessor() before the tree is built is at most 2τ. This bound has two consequences. First, the expected number of steps that a call to Attempt() will take is bounded above by $1/\varepsilon + 2\tau$. More importantly, the expected number of suppressed deaths is bounded by $2\tau\varepsilon$. Thus the probability that a second death will occur is bounded by $2\tau\varepsilon$. But the probability of aborting is independent of which vertex Attempt() starts at. Hence the probability of aborting is at most $\min(1, 2\tau\varepsilon)$.

The expected amount of work done before $1/\varepsilon$ becomes bigger than τ is bounded by $O(\tau)$. Afterwards the probability that a call to Attempt() results in Failure decays exponentially. The probability that Attempt() gets called at least i additional times is $2^{-\Omega(i^2)}$, while the expected amount of work done the ith time is $\tau 2^{O(i)}$. Thus the total amount of work done is $O(\tau)$. (□)

If constant factors do not concern the reader, the proof is done. The constant factor of 22 is derived below for the more diligent reader.

Let ε_j be the value of ε the jth time RandomTree() calls Attempt() procedure. The expected number of times T that RandomTree() calls RandomSuccessor() is bounded by

$$T \leq \sum_{i=1}^{\infty} \min(2/\varepsilon_i, 1/\varepsilon_i + 2\tau) \prod_{j=1}^{i-1} \min(1, 2\tau\varepsilon_j).$$

Suppose $\varepsilon_j = s^{-j}$ (we will set s below, the algorithm uses $s = 2$). Let k be the smallest j with $2\tau\varepsilon_j \leq 1$, and let $\kappa = 2\tau\varepsilon_k$; $1/s < \kappa \leq 1$. Breaking the sum apart and using $\varepsilon_j = \kappa/(2\tau)s^{k-j}$

we get

$$
\begin{aligned}
T &\le \sum_{i=1}^{k-1} \frac{4\tau}{\kappa} s^{i-k} + \sum_{i=k}^{\infty} \left(\frac{2\tau}{\kappa} s^{i-k} + 2\tau \right) \prod_{j=k}^{i-1} \kappa / s^{j-k} \\
\frac{T}{2\tau} &\le \sum_{i=1}^{k-1} \frac{2}{\kappa} \frac{1}{s^i} + \sum_{i=0}^{\infty} \left(\frac{1}{\kappa} s^i + 1 \right) \prod_{j=0}^{i-1} \kappa / s^j \\
&< \frac{2}{\kappa} \frac{1}{s-1} + \sum_{i=0}^{\infty} \left(\frac{s^i}{\kappa} + 1 \right) s^i s^{-i(i-1)/2} \kappa^i \\
&= \frac{2/\kappa}{s-1} + \sum_{i=0}^{\infty} s^{-i(i-3)/2} \kappa^{i-1} + \sum_{i=0}^{\infty} s^{-i(i-1)/2} \kappa^i
\end{aligned}
$$

Now since this expression is concave up in κ, we may evaluate it at $\kappa = 1$ and $\kappa = 1/s$, and take the maximum as an upper bound. It should not be surprising that evaluating at these two values of κ yields the same answer. With $s = 2$ we get a bound of $T < 21.85\tau$.

□

Suppose that the initial ε is chosen to be $1/s$ raised to a random power between 0 and 1. Then κ is $1/s$ raised to a random power between 0 and 1, and we have

$$
E[\kappa^i] = \begin{cases} \frac{1-1/s^i}{i \ln s} & i \ne 0 \\ 1 & i = 0 \end{cases}.
$$

Then when $s = 2.3$ we get $T < 21\tau$.

Chapter 8

Trees and Matchings

8.1 Generalizing Temperley's Bijection

Temperley [76] observed that asymptotically the $m \times n$ rectangular grid has about as many spanning trees as the $2m \times 2n$ rectangular grid has perfect matchings. Soon afterwards he found a bijection between trees of the $m \times n$ grid and perfect matchings in the $(2m+1) \times (2n+1)$ rectangular grid with a corner removed [77]. Propp, and, independently, Burton and Pemantle [18], generalized this bijection to map spanning trees of general (undirected unweighted) plane graphs to perfect matchings of a related graph. Here we consider the directed weighted case. This final generalization turns out to be just what is needed to generate random matchings of hexagonal and square-octagon lattices, even though these lattices are themselves unweighted and undirected. Analyzing the directed weighted case also yields some interesting identities.

Let G be a plane graph with a distinguished face (the "outer face") and a distinguished vertex. We make a new weighted graph H based on G, and to avoid confusion, we will say that H has nodes and links whereas G has vertices and edges. Figure 8-1 provides an illustration of this construction. We will have some liberty in choosing the link weights, but otherwise G and its embedding will determine H.

The nodes of H are the vertices, edges, and faces of G. Each edge of G is incident to two vertices of G (its endpoints) and two faces of G (the faces which it borders). Every edge node of H is linked to the two vertex nodes and two face nodes to which it is incident in G. The links between an edge node and a vertex node are always given weight 1, but the links between an edge node and a face node may be given any nonnegative weight; weight 0 is equivalent to omitting the link.

Let the directed weighted graph F have as its vertices the face nodes of H. Given two face nodes u and v, there is a directed edge from u to v with weight w in F if and only H has an edge node e incident to u and v, and the link between e and u has weight w.

Another graph V can be defined from H analogously, except using vertex nodes rather than face nodes. This graph V is just G with edges made bidirectional and given weight 1.

Intuitively, H contains within it both G, and the dual of G with edges bidirectional and weighted. Now H won't have any matchings because the number of vertex nodes and face nodes is two more than the number of edge nodes, but if we let H' be H with the distinguished vertex node and the distinguished face node omitted, then H' will in general have matchings. Let the weight of a matching be the product of the weights of edges. The generalized Temperley bijection takes a matching of H' to a spanning tree of G rooted at the distinguished vertex, and a spanning tree of G's dual, rooted at the distinguished face, such that the two spanning trees share no edge nodes, but together contain all the edge nodes. Either spanning tree is enough to determine the matching of H', so instead of picking a uniformly random spanning tree from G's dual, we may choose a spanning tree of G's dual so that the probability is proportional to the product of the link weights in F. Now it's clear that the weight of a matching in H' is the product of the weights of the two spanning trees, and the weight of the spanning tree in V is 1, so the weight of a matching is the weight of the spanning tree in F. If we pick a random spanning tree in F, the result is a random matching in H'.

8.2 The Hexagonal Lattice

In this section will illustrate the above technique in Figure 8-1 by giving a bijection between spanning trees of a directed graph and matchings in the hexagonal lattice. Here G is itself a hexagonal lattice, and H' is a hexagonal lattice with about three times as many hexagons. Jim Propp and I, and independently Rick Kenyon [55], found the hexagonal lattice application of the above technique.

8.3 The Square-Octagon Lattice

For the square-octagon lattice, an excerpt is shown in Figure 8-2, if one tries to generalize the tree-matching bijection by assigning V's, E's, and F's to the vertices in a compatible fashion, then with much work you can show that there is in fact, no way to do it. Nonetheless, it is possible to generalize the bijection to this lattice. To do it we need to do two transformations on the lattice. The first transformation is called "urban renewal", a term coined by Jim Propp who learned of the method from Greg Kuperberg. In the second transformation, we adjust the edge weights. At that point, if a square-octagon region has suitable boundary conditions, it will possible to assign V's, E's, and F's to the vertices in a way that allows us to use the generalized Temperley bijection described in section 8.1. Before actually doing this, we outline some the results that will follow.

First of all, using the random tree generator of Chapter 7, and the bijection between spanning trees of a weighted graph and matchings of the square-octagon regions, random such matchings can be sampled in linear time. The expected number of steps to create the random spanning tree is bounded by 1.27025 times the number of vertices (this constant is R, defined below), and the remaining steps are readily done deterministically in linear time.

Secondly, this tree formulation gives rise to a closed-form formula for the number of matchings of a certain class of regions. Taking large regions, and dividing the log of the number of matchings by the number of octagons gives that twice the entropy per octagon is

$$\ln(2) + \int_0^1 \int_0^1 \ln(5/2 + \cos(\pi x) + \cos(\pi y)) dx\, dy,$$

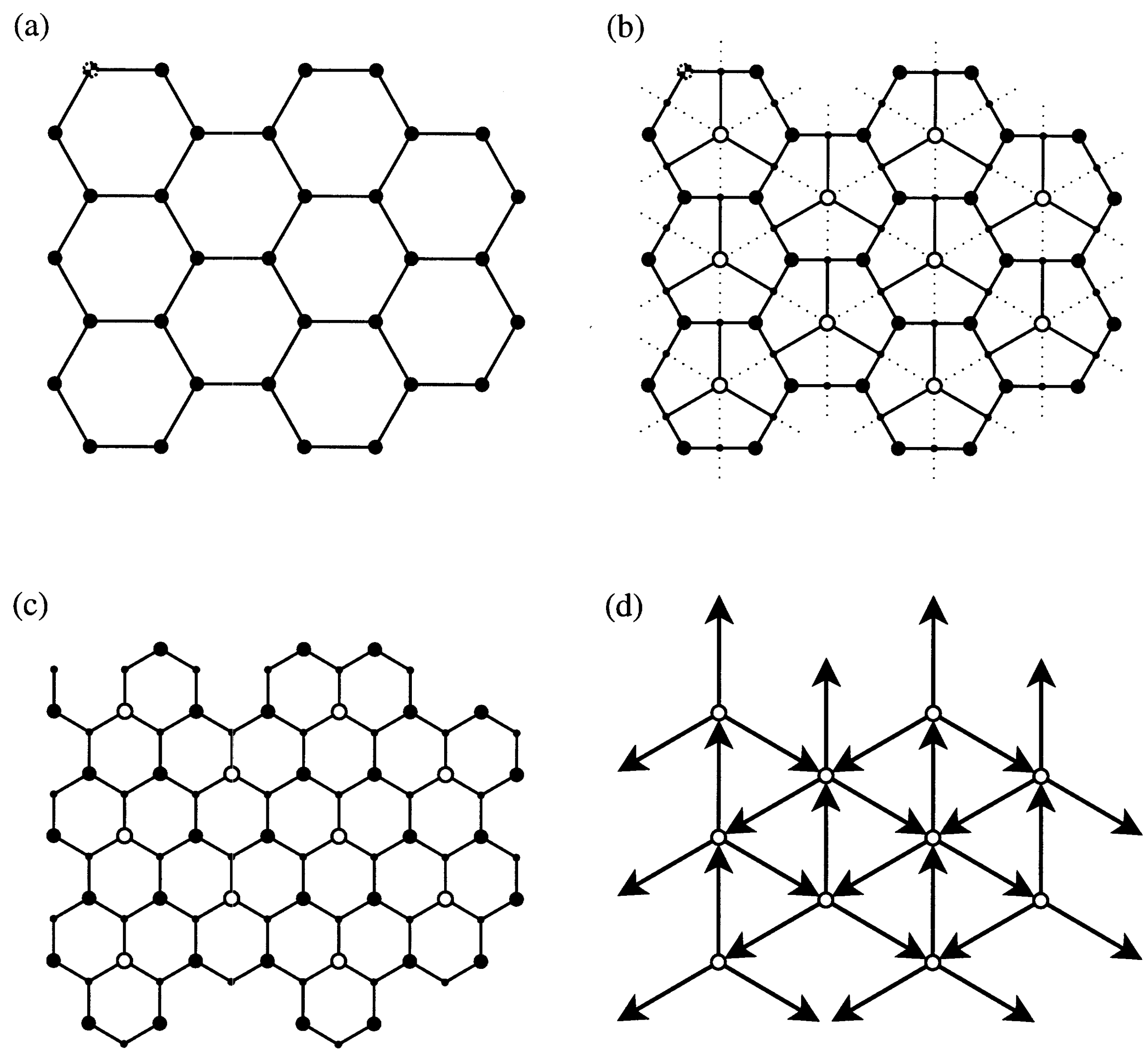

Figure 8-1: Panel (a) contains the plane graph G, with vertices shown as small disks. The outer face and upper left-most vertex are distinguished. Panel (b) shows the graph H derived from G. Vertex nodes are shown as small disks, face nodes are shown as circles (except the outer face node, which is not shown), and edge nodes are shown as even smaller disks. The links between edge nodes and vertex nodes are prescribed to have weight 1. For the links between edge nodes and face nodes, we choose some of them to have weight 1 (the ones drawn with solid lines) and others to have weight 0 (the ones drawn with dashed lines). Panel (c) shows the graph H', which is H with the nodes corresponding to the outer face and distinguished vertex deleted. Links with weight 0 are not shown here, and the edge nodes have been displaced slightly from there positions in panel (b). Panel (d) shows the graph F, also derived from H. The links in F are directed, and the links of weight 0 have been omitted. Directed spanning trees of F rooted at the outer face are in one-to-one correspondence with matchings of H'.

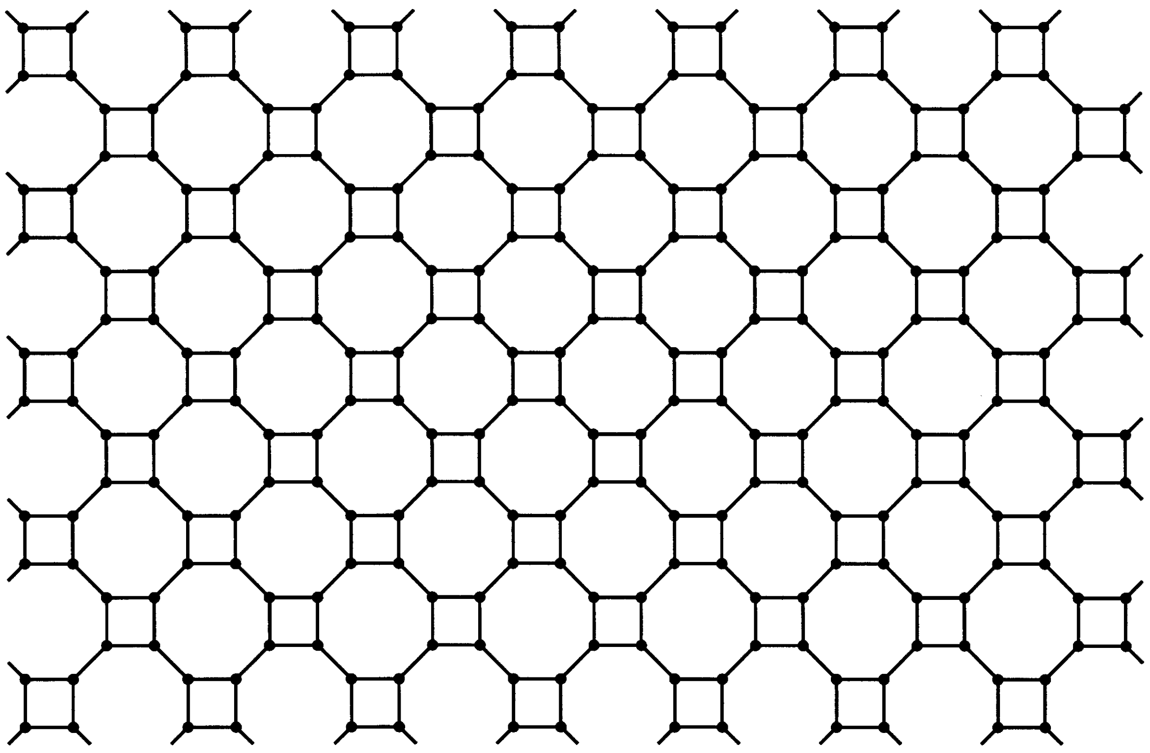

Figure 8-2: A portion of the square-octagon lattice.

or about 1.507982602. This is the maximal entropy, as the tree method only works for regions of maximal entropy. (This value for the entropy also follows from formulas given in [30].)

Thirdly, the connection with trees yields the probability that a given vertex is matched with its neighbor that borders the same two octagons: 0.11892523996.... Analytically, this probability is $1/2 - (3/10)R$ where R is given by

$$R = \int_0^1 \int_0^1 \frac{1}{1 - 2/5\cos(\pi x) - 2/5\cos(\pi y)} dx\, dy$$

or the more computationally efficient

$$R = \sum_{k=0}^{\infty} \binom{2k}{k}^2 (1/5)^{2k}$$

or the closed form

$$R = (2/\pi)K(4/5)$$

with $K(k)$ denoting Legendre's complete elliptic integral of the first kind. This last formula for R was derived by Henry Cohn from the second formula. Richard Stanley pointed out that the transcendence of R is proved in [68].

The reader is also refered to [54], where Kenyon develops a procedure for computing entropy and placement probabilities of perfect matchings in bipartite planar lattices without using trees.

8.3.1 Urban renewal

The mapping between square-octagon matchings and spanning trees makes use of a trick called "urban renewal". Tricks such as urban renewal have been used by researchers in the statistical mechanics literature for decades, but since understanding it is essential for what follows, a description is included here. This cute name comes from viewing the square-octagon lattice as a set of cities (the squares) that communicate with one another via the edges that separate octagons. Now the graph of cities is itself bipartite, so we may say that every city is either rich or poor, with every poor city having four rich neighbors and vice versa. The process of urban renewal on a poor city merges each of its four vertices with its four neighboring vertices, and then multiplies each of the edge weights of the poor city by 1/2, as shown in Figure 8-3.

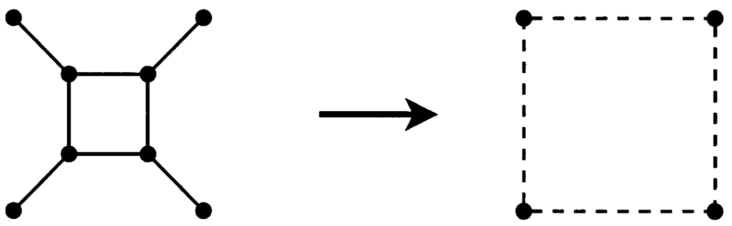

Figure 8-3: Urban renewal. The poor city is the inner square in the left graph, and is connected to the rest of the graph (not shown) via only four vertices at the corners, some of which may actually be absent. The city and its connections are replaced with weight 1/2 edges, shown as dashed lines.

Prior to urban renewal, every matching will match either 0, 2, or 4 of the poor city's vertices with the rest of the graph, and if 2 vertices, then these vertices are adjacent. If zero vertices, then since the city has two possible matchings, a pair of matchings in the before graph get mapped to one matching of the same weight in the after graph. If two vertices, then the matching in the before graph gets mapped to a matching in the after graph that uses the weight 1/2 edge connecting the corresponding two vertices in the after graph. Those matchings which use four vertices get mapped to a pair of matchings in the after graph, each using two weight 1/2 edges. Thus urban renewal on a poor city will reduce the weighted sum of matchings by 1/2, and given one random bit, a random matching in the before graph is readily transformed into a random matching in the after graph, and vice versa.

Along the boundaries, some of the poor cities won't have four neighbors, but urban renewal can still be done. One way to see this is to adjoin a pair of connected vertices to the graph for each missing poor city's neighbor, and connect one of these vertices to the poor city. This operation won't affect the number of matchings or placement probabilities, and after urban renewal, the pair may be deleted again, again without affecting the matchings — so if some of the poor city's vertices don't have neighbors, these vertices are deleted by urban renewal. Doing urban renewal on each of the poor cities in the square-octagon lattice will yield the more familiar Cartesian lattice.

8.3.2 Weighted directed spanning trees

The following derivations for the formulae for entropy and the placement probabilities closely follow the derivation for entropy of regular domino tilings. Consider the square-octagon region shown in Figure 8-4. It has 3 octagonal bumps on the left, and four on top, so by convention let's call it a region of order 3, 4. (In a region of order L, M, there are $2LM$ octagons.) An octagonal column and octagonal row meet at a unique square, these will be the rich cities. The rich cities have been labeled by their coordinates to enhance clarity. The other $(L+1)(M+1)$ squares will be the poor cities, and we will do urban renewal on them as shown in Figure 8-4. We will compute the weighted number of matchings of the resulting graph, and multiply by $2^{(L+1)(M+1)}$.

Now for any vertex, we may reweight all the edges incident to the vertex without affecting the probability distribution on matchings. Reweight them as follows: For the square (rich city) with coordinates i, j, multiply the weights of the edges incident to the top left and lower right corners by 2^{-i-j}, the other two corners by 2^{i+j}.

2^{-i-j}	2^{i+j}
2^{i+j}	2^{-i-j}

Edges that are internal to the rich cities remain weighted by 1. The long edges come in pairs. The lower or right edge of the pair gets its weight doubled, to become 1, while the upper or left edges of the pair get their weight halved to become 1/4.

The next thing we need to do is interpret this graph as a plane graph and its dual (see Figure 8-5). The upper left vertices of the small squares represent vertices, the lower right vertices represent faces, and the other two vertices represent edges. The result is the graph shown in Figure 8-5, which has $LM + 1$ vertices — LM of them on a grid, and one "outer vertex" (not in the original graph) that all the open edges connect to. A random spanning tree on the vertices of this graph rooted at the outer vertex, determines a dual tree on the faces of this graph, rooted at the upper left face, which together determine a matching of the graph in Figure 8-4. The weight of the matching equals the weight of the primal tree, since the reweighting left every dual tree with weight one. So now we need to compute the weighted sum of directed spanning trees of the graph F in Figure 8-5, where the weight of a tree is $(1/4)^{\#\text{ tree edges directed up or left}}$.

Remark These trees may be sampled quickly via a biased random walk and one of the tree algorithms given in Chapter 7. At this point we note that in the case of sampling spanning trees of the plane with these edge weights, the straightforward Aldous/Broder style process does not work as it does in the unweighted case, as this random walk is non-recurrent. In contrast, the new tree algorithm can in principle be extended to the plane. Start at some vertex, with probability one the random walk will eventually never return to that vertex, and drift away to infinity, leaving a path behind. There are four possibilities for the first edge in the path, and their probabilities will be computed below. In principle one could write a procedure, which given a set of vertices and information on whether each vertex is in the infinite path, and if so, what the next vertex is. Such a procedure would take a lot of work to construct, and I don't know how to go about doing it. But given this procedure as a black box, and a finite region of the plane, one could determine those vertices of the region in the infinite path. Then starting from each of the other vertices, do a random walk until

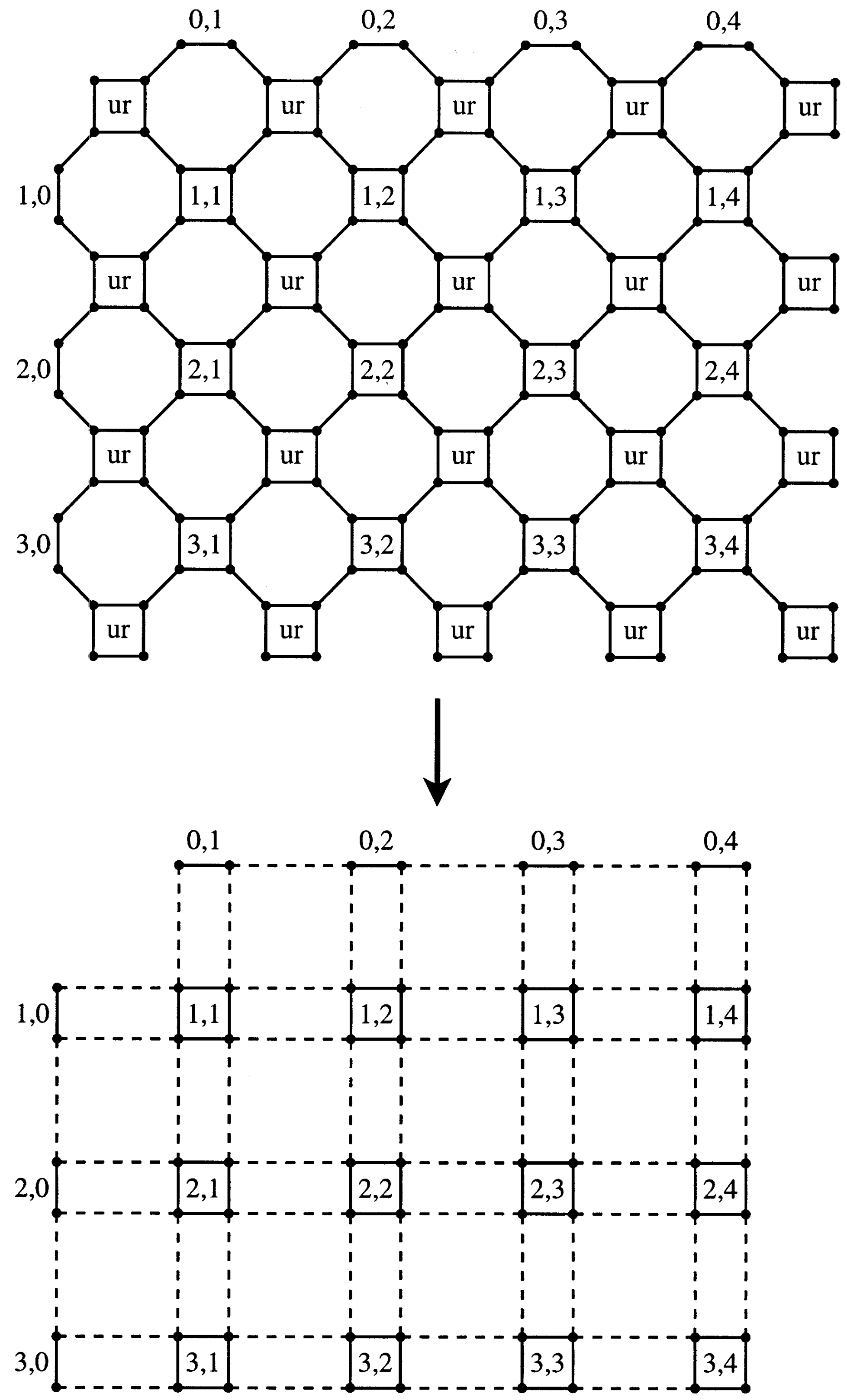

Figure 8-4: Region of order 3, 4 before and after urban renewal. The poor cities on which urban renewal is done are labeled with "ur", the rich cities are labeled with their coordinates. Dashed edges have weight 1/2.

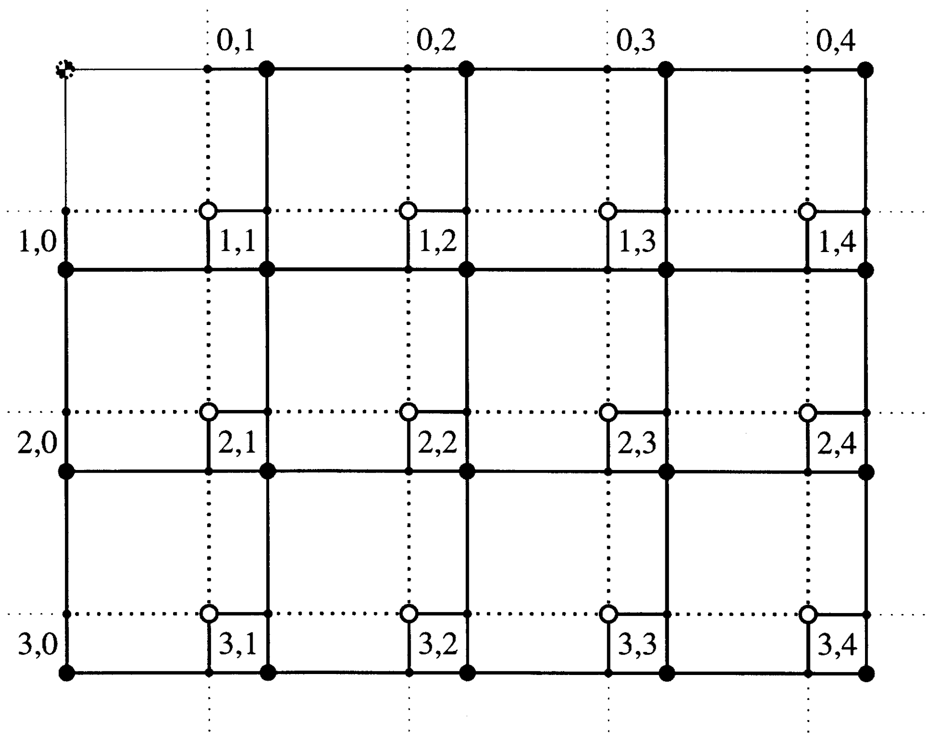

Figure 8-5: The region from Figure 8-4 after reweighting — dashed edges have weight 1/4. A distinguished vertex and distinguished outer face have been adjoined to give a graph H which nicely decomposes into a graph G and a weighted directed graph F.

the infinite path is encountered. (This encounter may happen outside the finite region, but it will happen with probability 1.) At that point we will have a partial view of a random weighted spanning tree of the plane. Naturally we won't be able to view the entire spanning tree of the plane, but we would be able to view any finite portion of the tree.

8.3.3 Entropy

To count the weighted sum of spanning trees of the above region, we can appeal to the matrix tree theorem. Make a $(LM+1) \times (LM+1)$ matrix A with entry $A_{x,y}$ equal to the negative of the weight from node x to node y, and $A_{x,x}$ set to make the row sums zero. Delete the row and column of the root vertex, which is conveniently the outer face node, and compute the determinant.

We compute the determinant using the same method that worked for ordinary domino tilings, i.e. we use Fourier analysis. Compute all the eigenvalues, and multiply them. With $1 \leq k_1 \leq L$ and $1 \leq k_2 \leq M$, the (k_1, k_2)th eigenvector E_{k_1,k_2} is given by

$$E_{k_1,k_2}(i,j) = 2^{i+j} \sin(i\pi k_1/(L+1)) \sin(j\pi k_2/(M+1))$$

and its eigenvalue is $5/2 - \cos(\pi k_1/(L+1)) - \cos(\pi k_2/(M+1))$.

It can be checked that these eigenvectors are linearly independent. Since there are only LM eigenvalues, multiplying them gives the weighted sum of trees.

Therefore, the number of matchings of a region of order L, M shown above is given by

$$2^{(L+1)(M+1)} \times \prod_{k_1=1}^{L} \prod_{k_2=1}^{M} (5/2 - \cos(\pi k_1/(L+1)) - \cos(\pi k_2/(M+1))$$

which may be rewritten as

$$2^{(L+1)(M+1)} \times \prod_{k_1=1}^{L} \prod_{k_2=1}^{M} (5/2 + \cos(\pi k_1/(L+1)) + \cos(\pi k_2/(M+1))).$$

(For our working example with $L = 3$ and $M = 4$, the answer is 28295974656, which may be double-checked using a program called `vaxmacs`.) Taking the logarithm, dividing by LM, and passing to the limit of large L and M gives the above formula for the entropy.

8.3.4 Placement probabilities

Consider a vertex in the spanning tree of the face graph of Figure 8-5. If its successor is to the right or down, then in the square-octagon graph it is paired with another vertex of its city, otherwise it is paired with a vertex of a different city. Chapter 7 gives an algorithm for sampling random spanning trees, and by analyzing it we can compute the probability of these two events. We can sample the path from a vertex to root of a random tree by doing a loop-erased random walk from that vertex until it reaches the root. The moves are right or down with probability 4/10 and up or left with probability 1/10, since in the face graph the links going to the left or up have 1/4 the weight of the other links. For large graphs, the random walk drifts to the right and down, so we consider this biased random walk on $\mathbb{Z}^2$. Starting at the origin, with probability 1 the origin is visited finitely many times. Consider the last visit, we seek the probability that the walk left the origin by going up or to the left.

Let p^+ be the probability of returning to 0 given that we just left 0 going up or left, and p^- is the probability of return given that we just left going right or down.

Any path of length $2l > 0$ by which the random walk returns to the origin has probability $(4/100)^l$. If we reflect the path so that the up and right moves switch, and the left and down down moves switch, we get a probability-preserving bijection between the paths that leave going by going up and the paths that leave by going down. Adding up these probabilities gives the equation

$$\frac{1}{10}p^+ = \frac{4}{10}p^-.$$

The probability that the random walk returns to the origin k times, and then leaves going up-left and never returns, is given by $(p^+/5 + 4p^-/5)^k \times (1 - p^+)/5$. Summing over k, we find the probability of last exit going up or left is

$$\frac{(1-p^+)/5}{1-2p^+/5} = \frac{1}{2} - \frac{3/10}{1-2p^+/5} = \frac{1}{2} - \frac{3}{10}R,$$

where R is the expected number of visits to the origin, counting the visit at time 0:

$$R = \sum_{k=0}^{\infty} (2p^+/5)^k = \frac{1}{1 - 2p^+/5}.$$

The above infinite sum formula for R is easy to derive. The integral formula follows from some Fourier analysis on large finite grids, but since this formula doesn't appear appear to be exceedingly useful, its derivation is omitted.

8.4 Recontiling Markov chain

Sections 8.2 and 8.3 generalize the tree-based algorithm for generating random matchings of regions with certain boundary conditions to work for the hexagonal lattice and the square-octagon lattice with certain boundary conditions. The tree algorithm in Chapter 7 makes it possible to generate the necessary trees quickly. But for regions with other boundary conditions, the tree approach does not seem to work, at least not directly. In this section we give another monotone Markov chain for generating random matchings of a bipartite plane graph. It relies on our ability to find matchings of certain regions very quickly, and monotonicity allows us to use monotone coupling from the past (Chapter 3) to generate perfectly random samples.

Let A and B be regions of $\mathbb{Z}^2$. We wish to randomly tile A, and we are able to randomly tile B. Regard the tilings as matchings of points on a grid. (Then heights are defined on the squares.) If there is a simple cycle in A such that every other edge is in the matching, then by "alternate the cycle" we mean "replace all edges of the matching contained within the cycle with edges of the cycle not contained in the matching." Consider the following Markov chain on the tilings of A. Randomly tile B, and then look at the union of the two matchings. Alternate all cycles in this union.

Theorem 22 *This Markov chain satisfies detailed balance, and hence preserves the uniform distribution.*

Proof: Consider two tilings T_1 and T_2 of region A. Let S be the set of tilings of region B that take T_1 to T_2 under the above Markov chain, and let S' be the tilings that take T_2 to T_1. Suppose that in addition to alternating the cycles in T_1, we alternate them in the tiling R of region B as well. Each modified tiling R' is in S', and this mapping is reversible, so $|S'| \geq |S|$. By symmetry $|S'| = |S|$. Since the tilings of B are chosen uniformly at random independent of the tiling of A, the probability of going from T_1 to T_2 equals the probability of going from T_2 to T_1. $\square$

Here we assume the reader is familiar with the notion of height functions for mathcings of planar bipartite graphs — see e.g. [79].

Lemma 23 *Alternating the edges in a cycle either raises the heights of all interior squares by 4, or lowers them all by 4.*

Theorem 24 *This Markov chain is monotone.*

Proof: Consider a matching R of region B, and two matchings U and L of region A with $U \geq L$. Let $R(X)$ denote the matching of A obtained by updating matching X of A with matching R of B. The proof is by induction on the number of cycles in $(R \cup U)$ plus the number of cycles in $(R \cup L)$. If this number is 0, then $R(U) = U \geq L = R(L)$. Otherwise we may suppose that $(R \cup U)$ has a cycle (the other case is symmetric). Consider an innermost such cycle, i.e. a cycle which itself does not contain other cycles in its interior besides 2-cycles. If alternating this cycle raises the heights, then alternate it to get $U' \geq L$. By induction we have $R(U) = R(U') \geq R(L)$. Otherwise, alternating the cycle lowers the height of U. Shade the squares within the cycle whose height in U equals their height in L, and consider a boundary edge e of this shaded region. If e is an edge of the cycle, then either 1) it belongs to R or 2) it belongs to U. If case 2), then the height in U crossing the edge goes down by 3, and so it must go down by 3 in L as well since $L \leq U$, and hence e belongs to L. Either way e belongs to $(R \cup L)$. If e is interior to the cycle, then either 1) the height in U goes up 1 crossing the edge, but the height in L goes down by 3, or 2) the height in U goes up by 3 crossing the edge, but the height in L goes down by 1. In case 1) e belongs L, while in case 2), e belongs to U, but since edge e is interior to an innermost cycle, e must also belong to R. Either way e belongs to $(R \cup L)$. In any case, the boundary of the shaded region consists of one or more alternating cycles in $(R \cup L)$, which can't be pushed up, so they will get pushed down. Alternate these cycles in L, and also the one in U. We have $U' \geq L'$, and by induction $R(U) = R(U') \geq R(L') = R(L)$. □

Note that we can take B to be a superset of A, or we may instead use a sequence of different B's which cover region A. The Markov chain given in section 4.2.3 effectively uses a sequence of squares for the B's. Even if using a square grid for B is not optimal for tilted regions A, there is still considerable flexibility, and it seems likely that one can improve on the previous Markov chain.

Glossary

aperiodic: in reference to Markov chains, one for which the gcd of all cycles with positive probability is one

approximate sampling: the process of generating sample points from some probability distrubtion π' that is in some sense close to a desired probability distribution π. See also "exact sampling".

bipartite graph: a graph whose vertices may be partitioned into two sets, such that every edge connects two vertices from opposite sets

bond-correlated percolation model: see "random cluster model"

CFTP: see coupling from the past

coalescing random walk model: see section 2.3

coupling from the past: a general protocol for obtaining exact samples from a Markov chain (see section 2.1). Specializations include monotone-CFTP (Chapter 3), tree-CFTP (Chapter 6), voter-CFTP (section 1.2), cover-CFTP (section 2.2), and coalescing-CFTP (section 2.3).

critical exponents: see section 4.2

cycle popping: a simple technique for generating random spanning trees — see Chapter 7

de Bruijn sequence: A de Bruijn sequence of order n is a cyclic string of 2^n 0's and 1's such that each possible string of n 0's and 1's occurs in the de Bruijn sequence exactly once.

degree: the number of edges incident to a vertex in a graph. For directed graphs the in-degree is the number of edges directed in to a vertex, while the out-degree is the number of edges directed away from a vertex. In a weighted graph, rather than counting edges, their weights are added.

detailed balance: If π is a probability distribution, and a Markov chain has transition probabilities given by $p_{i,j}$, then the Markov chain has detailed balance if for all i and j, $\pi(i)p_{i,j} = \pi(j)p_{j,i}$. If a Markov chain satisfies detailed balance, then it preserves the probability distribution π.

distributive lattice: A lattice for which $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$ and $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$. Every finite distributive lattice is isomorphic to the lattice of order ideals of some partially ordered set.

edge: a pair of vertices

ergodic: in reference to Markov chains, one that is irreducible and aperiodic. Note: sometimes "ergodic" is used to mean irreducible without regard to periodicity.

Eulerian graph: A directed graph for which the in-degree of each vertex equals its out-degree.

exact sampling: the process of generating sample points from some desired probability distribution π. See also "approximate sampling".

Gibbs distribution: See also "partition function". The probability distribution for which $\Pr[\text{state is } s] = Z^{-1}e^{E_s/kT}$, where E_s is the energy of state s, k is Boltzmann's constant, T is the temperature, and Z is the partition function.

graph: A set of points, called vertices, together with a set of pairs of vertices, called edges. If the pairs are unordered, the graph is undirected, otherwise it is directed. If the edges have weights associated with them, the graph is weighted.

heat bath: A randomizing operation based on a partitioning of the state space. Given a state, the randomizing operation determines its part of the state space, and then picks a random state in that part. A heat bath Markov chain uses many different partitions.

irreducible: in reference to Markov chains, one for which there exists a vertex x such that every other vertex y has a path of positive probability leading to x

Ising model: A statistical mechanical model originally introduced by Ising to model ferromagnetic systems. Given a weighted undirected graph, an Ising state is associates with each vertex (also known as a site) a spin up state $\uparrow$ or a spin down state $\downarrow$. The energy of an Ising state is the sum over all edges which connect sites with opposite spin, the weight of the edge.

Las Vegas: an algorithm which uses randomness for efficiency, but always outputs the correct answer

lattice: a partially ordered set such that every pair of elements x and y have a least upper bound $x \vee y$ and a greatest lower bound $x \wedge y$

lattice: an infinite graph, which may be embedded in the plane, has translational symmetry, and such that there are only finitely many equivalence classes of vertices modulo the translations

matching: A partition of the vertices of a graph into pairs, such that each pair is given by the endpoints of an edge of the graph.

Markov chain: A state space together with a state and a randomizing operation. As time goes forward, the state is updated according to the randomizing operation.

monotone: In reference to Markov chains, one whose state space has a partial order $\leq$, such that if $x \leq y$ and both states are updated via the same randomizing operation, then the updated x is $\leq$ the updated y.

Monte Carlo: an algorithm which uses randomness for efficiency, and which might possibly output an incorrect answer

omnithermal sample: An object which maps any temperature to a random sample at that temperature.

order ideal: A subset A of some partially ordered set X, such that $x \leq y$ and $y \in A$ implies $x \in A$. The order ideals of a poset, together with the partial order of subset-inclusion, form a distributive lattice.

partial order: a binary relation $\leq$ on some set, such that 1) $x \leq y$ and $y \leq z$ implies $x \leq z$, and 2) $x \leq y$ and $y \leq x$ implies $x = y$

partition function: usually denoted with Z, $Z = \sum_{\text{states } s} e^{E_s/kT}$, where T is the temperature of a system, E_s is the energy of state s of the system, and k is Boltzmann's constant.

phase transition: see section 4.2

poset: A partially ordered set.

Potts model: A generalization of the Ising model, introduced by Potts. Each site (vertex) in a weighted undirected graph may assume one of q different spin values. The energy of a Potts state is the sum over all edges connecting sites with different spins, the weight of the edge.

random cluster model: A statistical mechanical model introduced by Fortuin and Kasteleyn. Given a weighted undirected graph, a random cluster state is a subgraph (same vertex set, subset of the edges). The probability of a particular random cluster state is proportional to $p^{\#\text{ edges}}(1-p)^{\#\text{ non-edges}}q^{\#\text{ connected components}}$. The random cluster model is closely related to the Ising (and Potts) models, and percolation.

recontile: an operation that brings two perfect matchings in closer agreement with one another, by reconciling their differences. See section 8.4.

region: in reference to perfect matchings, a finite subset of $\mathbb{Z} \times \mathbb{Z}$

sampling: generating random variables from some probability space. See also "approximate sampling" and "exaaact sampling".

spanning tree: A connected acyclic subgraph of a graph. When the graph is directed, we assume that the spanning tree is in-directed, i.e. there is a root vertex, such that every edge of the spanning tree is directed towards the root.

stationary distribution: in reference to Markov chains, a probability distribution π such that if a state is chosen according to π and the chain is run one step, the resulting state is also distributed according to π. If the Markov chain is irreducible, then there is only one stationary distribution π.

steady state distribution: see "stationary distribution"

voter model: see section 2.3

Bibliography

[1] David Aldous. On simulating a Markov chain stationary distribution when transition probabilities are unknown, 1994. Preprint.

[2] David Aldous and Persi Diaconis. Strong uniform times and finite random walks. *Advances in Applied Mathematics*, 8(1):69–97, 1987.

[3] David Aldous, László Lovász, and Peter Winkler. Fast mixing in a Markov chain, 1995. In preparation.

[4] David J. Aldous. A random walk construction of uniform spanning trees and uniform labelled trees. *SIAM Journal of Discrete Mathematics*, 3(4):450–465, 1990.

[5] David J. Aldous and James A. Fill. *Reversible Markov Chains and Random Walks on Graphs.* Book in preparation.

[6] Noga Alon and Joel H. Spencer. *The Probabilistic Method.* John Wiley & Sons, Inc., 1992. With appendix by Paul Erdős.

[7] S. F. Altschul and B. W. Erickson. Significance of nucleotide sequence alignments: A method for random sequence permutation that preserves dinucleotide and codon usage. *Molecular Biology and Evolution*, 2:526–538, 1985.

[8] Søren Asmussen, Peter W. Glynn, and Hermann Thorisson. Stationary detection in the initial transient problem. *ACM Transactions on Modeling and Computer Simulation*, 2(2):130–157, 1992.

[9] Rodney J. Baxter. *Exactly Solved Models in Statistical Mechanics.* Academic Press, 1982.

[10] Laurel Beckett and Persi Diaconis. Spectral analysis for discrete longitudinal data. *Advances in Mathematics*, 103(1):107–128, 1994.

[11] Julian Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society* B, 48(3):259-302, 1986.

[12] Norman Biggs. *Algebraic Graph Theory.* Cambridge University Press, second edition, 1993.

[13] J. J. Binney, N. J. Dowrick, A. J. Fisher, and M. E. J. Newman. *The Theory of Critical Phenomena: An Introduction to the Renormalization Group.* Oxford University Press, 1992.

[14] Anders Björner. Orderings of Coxeter groups. In *Combinatorics and Algebra*, pages 175–195. American Mathematical Society, 1984. Contemporary Mathematics, #34.

[15] H. W. J. Blöte and H. J. Hilhorst. Roughening transitions and the zero-temperature triangular Ising antiferromagnet. *Journal of Physics* A, 15(11):L631–L637, 1982.

[16] Béla Bollobás. *Graph Theory: An Introductory Course.* Springer-Verlag, 1979. Graduate texts in mathematics, #63.

[17] Andrei Broder. Generating random spanning trees. In *Foundations of Computer Science*, pages 442–447, 1989.

[18] Robert Burton and Robin Pemantle. Local characteristics, entropy and limit theorems for spanning trees and domino tilings via transfer-impedances. *The Annals of Probability*, 21(3):1329–1371, 1993.

[19] Henry Cohn, Noam Elkies, and James Propp. Local statistics for random domino tilings of the Aztec diamond. *Duke Mathematical Journal*, 1996. To appear.

[20] Charles J. Colbourn. *The Combinatorics of Network Reliability.* Oxford University Press, 1987.

[21] Charles J. Colbourn, Robert P. J. Day, and Louis D. Nel. Unranking and ranking spanning trees of a graph. *Journal of Algorithms*, 10:271–286, 1989.

[22] Charles J. Colbourn, Bradley M. Debroni, and Wendy J. Myrvold. Estimating the coefficients of the reliability polynomial. *Congressus Numerantium*, 62:217–223, 1988.

[23] Charles J. Colbourn, Wendy J. Myrvold, and Eugene Neufeld. Two algorithms for unranking arborescences. *Journal of Algorithms.* To appear.

[24] John Conway and Jeffrey Lagarias. Tiling with polyominoes and combinatorial group theory. *Journal of Combinatorial Theory, series A*, 53:183–208, 1990.

[25] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9:251–280, 1990.

[26] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* The MIT Press and McGraw-Hill Book Company, 1990.

[27] Persi Diaconis. *Group Representations in Probability and Statistics.* Institute of Mathematical Statistics, 1988.

[28] Persi Diaconis and Laurent Saloff-Coste. What do we know about the Metropolis algorithm? In *Symposium on the Theory of Computing*, pages 112–129, 1995.

[29] Persi Diaconis and Daniel Stroock. Geometric bounds for eigenvalues of Markov chains. *The Annals of Applied Probability*, 1(1):36–61, 1991.

[30] Chungpeng Fan and F. Y. Wu. General lattice model of phase transitions. *Physical Review B*, 2(3):723–733, 1970.

[31] Stefan Felsner and Lorenz Wernisch. Markov chains for linear extensions, the two-dimensional case, 1996. Manuscript.

[32] W. Fernandez de la Vega and A. Guénoche. Construction de mots circulaires aléatoires uniformément distribués. *Mathématiques et Sciences Humaines*, 58:25–29, 1977.

[33] James A. Fill, 1995. Personal communication.

[34] Walter M. Fitch. Random sequences. *Journal of Molecular Biology*, 163:171–176, 1983.

[35] C. M. Fortuin and P. W. Kasteleyn. On the random cluster model. I. Introduction and relation to other models. *Physica*, 57(4):536–564, 1972.

[36] C. M. Fortuin, P. W. Kasteleyn, and J. Ginibre. Correlation inequalites on some partially ordered sets. *Communications in Mathematical Physics*, 22:89–103, 1971.

[37] Arnoldo Frigessi, Chii-Ruey Hwang, Shuenn-Jyi Sheu, and Patrizia di Stefano. Convergence rates of the Gibbs sampler, the Metropolis algorithm and other single-site updating dynamics. *Journal of the Royal Statistical Society* B, 55(1):205–219, 1993.

[38] Stuart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6), 1984.

[39] David Griffeath. *Additive and Cancellative Interacting Particle Systems.* Springer-Verlag, 1979. Lecture Notes in Mathematics, #724.

[40] Geoffrey Grimmett. The stochastic random-cluster process, and the uniqueness of random-cluster measures. *The Annals of Probability.* To appear.

[41] Geoffrey Grimmett. *Percolation.* Springer-Verlag, 1989.

[42] A. Guénoche. Random spanning tree. *Journal of Algorithms*, 4:214–220, 1983. In French.

[43] Monika Rauch Henzinger and Valerie King. Randomized dynamic algorithms with polylogarithmic time per operation. In *Symposium on the Theory of Computing*, pages 519–527, 1995.

[44] Richard Holley. Remarks on the FKG inequalities. *Communications in Mathematical Physics*, 36:227–231, 1974.

[45] Richard Holley and Thomas Liggett. Ergodic theorems for weakly interacting systems and the voter model. *Annals of Probability*, 3:643–663, 1975.

[46] Salvatore Ingrassia. On the rate of convergence of the Metropolis algorithm and Gibbs sampler by geometric bounds. *The Annals of Applied Probability*, 4(2):347–389, 1994.

[47] Mark Jerrum and Alistair Sinclair. Approximating the permanent. *SIAM Journal on Computing*, 18(6):1149–1178, 1989.

[48] Mark Jerrum and Alistair Sinclair. Polynomial-time approximation algorithms for the Ising model. *SIAM Journal on Computing*, 22(5):1087–1116, 1993.

[49] William Jockusch, James Propp, and Peter Shor. Random domino tilings and the arctic circle theorem, 1995. Preprint.

[50] Valen E. Johnson. Testing for convergence of Markov chain Monte Carlo algorithms using parallel sample paths, 1995. Preprint.

[51] D. Kandel, Y. Matias, R. Unger, and P. Winkler. Shuffling biological sequences, 1996. To appear in a special issue on computational molecular biology.

[52] Maurice G. Kendall. *Rank Correlation Methods.* Hafner Publishing Company, third edition, 1962.

[53] Wilfrid S. Kendall. Perfect simulation for the area-interaction point process. In *Proceedings of the Symposium on Probability Towards the Year 2000*, 1996. To appear.

[54] Richard Kenyon. Local statistics of lattice dimers, 1996. Manuscript.

[55] Richard W. Kenyon, 1996. Personal communication.

[56] Jeong Han Kim, Peter Shor, and Peter Winkler. Random independent sets. Article in preparation.

[57] V. G. Kulkarni. Generating random combinatorial objects. *Journal of Algorithms*, 11(2):185–207, 1990.

[58] Gregory F. Lawler. *Intersections of Random Walks.* Birkhäuser, 1991.

[59] L. S. Levitov. Equivalence of the dimer resonating-valence-bond problem to the quantum roughening problem. *Physical Review Letters*, 64(1):92–94, 1990.

[60] Elliott Lieb. Residual entropy of square ice. *Physical Review*, 162:162–172, 1967.

[61] Thomas Liggett. *Interacting Particle Systems.* Springer-Verlag, 1985.

[62] László Lovász and Miklós Simonovits. On the randomized complexity of volume and diameter. In *Foundations of Computer Science*, pages 482–491, 1992.

[63] László Lovász and Peter Winkler. Exact mixing in an unknown Markov chain. *Electronic Journal of Combinatorics*, 2, 1995. Paper #R15.

[64] F. Martinelli, E. Olivieri, and R. H. Schonmann. For 2-d lattice spin systems weak mixing implies strong mixing. *Communications in Mathematical Physics*, 165(1):33–47, 1994.

[65] Louis D. Nel and Charles J. Colbourn. Combining Monte Carlo estimates and bounds for network reliability. *Networks*, 20:277–298, 1990.

[66] Robin Pemantle. Choosing a spanning tree for the integer lattice uniformly. *The Annals of Probability*, 19(4):1559–1574, 1991.

[67] James Propp. Lattice structure for orientations of graphs, 1993. Preprint.

[68] Theodor Schneider. *Einfuehrung in die transzendenten Zahlen.* Springer, 1957.

[69] Roberto H. Schonmann. Slow droplet-driven relaxation of stochastic Ising models in the vicinity of the phase coexistence region. *Communications in Mathematical Physics*, 161(1):1–49, 1994.

[70] Alistair Sinclair. *Algorithms for Random Generation and Counting: A Markov Chain Approach.* Birkhäuser, 1993.

[71] Alan D. Sokal. Monte Carlo methods in statistical mechanics: Foundations and new algorithms, 1989. Lecture notes from Cours de Troisième Cycle de la Physique en Suisse Romande.

[72] Richard P. Stanley. *Enumerative Combinatorics*, volume 1. Wadsworth, Inc., 1986.

[73] Daniel W. Stroock and Boguslaw Zegarlinski. The logarithmic Sobolev inequality for discrete spin systems on a lattice. *Communications in Mathematical Physics*, 149(1):175–193, 1992.

[74] Mark Sweeny. Monte Carlo study of weighted percolation clusters relevant to the Potts models. *Physical Review* B, 27(7):4445–4455, 1983.

[75] Robert H. Swendsen and Jian-Sheng Wang. Nonuniversal critical dynamics in Monte Carlo simulations. *Physical Review Letters*, 58(2):86–88, 1987.

[76] H. N. V. Temperley. In *Combinatorics: Being the Proceedings of the Conference on Combinatorial Mathematics held at the Mathematical Institute, Oxford*, pages 356–357, 1972.

[77] H. N. V. Temperley. In *Combinatorics: Proceedings of the British Combinatorial Conference 1973*, pages 202–204, 1974. London Mathematical Society Lecture Notes Series #13.

[78] Lawrence E. Thomas. Bound on the mass gap for finite volume stochastic Ising models at low temperature. *Communications in Mathematical Physics*, 126(1):1–11, 1989.

[79] William Thurston. Conway's tiling groups. *American Mathematical Monthly*, 97:757–773, 1990.

[80] Henk van Beijeren. Exactly solvable model for the roughening transition of a crystal surface. *Physical Review Letters*, 38(18):993–996, 1977.

[81] Ulli Wolff. Collective Monte Carlo updating for spin systems. *Physical Review Letters*, 62(4):361–364, 1989.

[82] F. Y. Wu. The Potts model. *Reviews of Modern Physics*, 54(1):235–268, 1982.

[83] Wei Zheng and Subir Sachdev. Sine-Gordon theory of the non-Néel phase of two-dimensional quantum antiferromagnets. *Physical Review* B, 40:2704–2707, 1989.