1

Adrian Barbu, Song-Chun Zhu

Monte Carlo Methods

July 27, 2020

Springer

*Adrian Barbu is a professor of Statistics at Florida State University, Tallahassee, FL 32306.
Song-Chun Zhu is a professor of Computer Science and Statistics at University of California, Los Angeles, CA 90095. Emails: abarbu@stat.fsu.edu; sczhu@stat.ucla.edu

Preface

Real world systems studied in sciences (e.g. physics, chemistry, and biology) and engineering (e.g. vision, graphics, machine learning, and robotics) involve complex interactions between large numbers of components. The representations for such systems are probabilistic models defined on graphs in high-dimensional spaces, to which analytic solutions are often unavailable. As a result, Monte Carlo methods have been used as a common tool for simulation, estimation, inference and learning in science and engineering. It is no surprise that the Metropolis algorithm topped the list of 10 most frequently used algorithms for practice of sciences in the 20th century (Dongarra and Sullivan, 2000). With ever-growing computing capacities, researchers are tackling more complex problems and adopting more advanced models. Monte Carlo methods will continue to play important roles in the development of sciences and engineering in the 21st century. The recent use of Hamiltonian Monte Carlo and Langevin Monte Carlo in developing stochastic gradient descent algorithm in deep learning is another example of this trend.

In history, several communities have contributed to the evolution of Monte Carlo methods.

- *Physics and chemistry*: the early Metropolis algorithm (Metropolis, Rosenbluth, Rosenbluth, Teller and Teller, 1953), simulated annealing (Kirkpatrick, Gelatt, and Vecchi, 1983), cluster sampling (Swendsen and Wang, 1987; Edwards and Sokal 1988), and recent work on disconnectivity graphs (Becker and Karpus, 1997) for visualizing the landscape of spin-glass models.
- *Probability and statistics*: stochastic gradient (Robin and Monro, 1951, Younes 1988), Hastings dynamics (Hastings, 1970), data augmentation (Tanner and Wong, 1987), reversible jumps (Green, 1995), dynamic weighting (Wong and Liang, 1997) for studying bio-informatics, and numerous analyses for bounding the convergence of Markov chain Monte Carlo (Diaconis 1988, Diaconis and Stroock, 1991, and Liu, 1991).
- *Theoretical computer science*: the convergence rate of clustering sampling (Jerrum and Sinclair, 1989, Cooper and Frieze, 1999)
- *Computer vision and pattern theory*: Gibbs sampler (Geman and Geman, 1984) for image processing, Jump-diffusion (Miller and Grenander, 1994) for segmentation, the condensation or particle filtering algorithm for object tracking (Isard and Blake, 1996), recent work on Data-driven Markov chain Monte Carlo (Tu and Zhu, 2002), and generalized Swendsen-Wang cuts (Barbu and Zhu, 2005) for image segmentation and parsing.

Considering that these diverse areas speak different languages, inter-disciplinary communication has been rare. This poses a considerable challenge for practitioners in computer science and engineering who desire to use Monte Carlo methods.

In one respect, effective Monte Carlo algorithms must explore the underlying structures of a problem domain, and they are thus domain or problem specific and hardly accessible to outsiders. For example, many important works in physics, like (Swendsen and Wang, 1987), are only 2-3 pages long and include no background or introduction, causing them to appear utterly mysterious to computer scientists and engineers.

On the other hand, general or domain agnostic Monte Carlo algorithms invented by statisticians are well-explained, but are usually found not very effective when they are implemented in generic ways by engineers without utilizing the structures of the underlying models and representations. As a result, there is a widespread misconception among scientists and graduate students that these methods are too slow and usually do not work. This is unfair to the Monte Carlo methods and unfortunate for the innocent students.

This textbook is written for researchers and graduate students in statistics, computer science, and engineering, based on materials and draft versions taught by the senior author in the Department of Statistics and Computer Science at the University of California, Los Angeles in the past 10 years. It covers a broad range of topics in Monte Carlo computing with both theoretical foundations and intuitive concepts developed in the four communities above, while omitting small tricks which are less applicable or do not work in practice. It illustrates the art of Monte Carlo design using classical problems in computer vision, graphics and machine learning, and thus can be used as a reference book by researchers in computer vision and pattern recognition, machine learning, graphics, robotics, and artificial intelligence.

The authors 'd like to thank many current and former Ph.D students at UCLA for their contributions to this book. Mitchell Hill contributed Chapters 9, 10 and 11 based on his thesis work which has enriched the contents of the book. Zachary Stokes worked on polishing many details in the manuscript. Maria Pavlovskaia, Kewei Tu, Zhuowen Tu, Jacob Porway, Tianfu Wu, Craig Yu, Ruiqi Gao, and Erik Nijkamp have contributed materials and figures as examples. Two UCLA colleagues Professors Yingnian Wu and Qing Zhou have given invaluable comments for improving the book.

The authors would also like to acknowledge the support of DARPA, ONR MURI grants, and NSF during the development of this book.

Tallahassee FL, Los Angeles CA — *Adrian Barbu*
September, 2018 — *Song-Chun Zhu*

Biosketches of the Authors

Adrian Barbu received his PhD in Mathematics from the Ohio State University in 2000 and his PhD in Computer Science from the University of California, Los Angeles in 2005 (advised by Dr. Song-Chun Zhu). From 2005 to 2007 he worked in Siemens Corporate Research, first as a research scientist and later as a project manager, with the focus on medical imaging problems. He received the 2011 Thomas A. Edison Patent Award with his co-authors from Siemens for their work on Marginal Space Learning. In 2007 he joined the Statistics department at Florida State University, first as an assistant professor, then associate professor, and since 2019 as a professor. He has published more than 70 papers in computer vision, machine learning and medical imaging and he has more than 25 patents related to medical imaging and image denoising.

Song-Chun Zhu received his PhD degree in Computer Science from Harvard University in 1996. He is currently a professor of Statistics and Computer Science, and director of the Center for Vision, Learning, Cognition and Autonomy, at the University of California, Los Angeles. His main research interest has been in pursuing a unified statistical and computational framework for vision and intelligence, which includes the Spatial, Temporal and Causal And-Or graph (STC-AOG) as a unified representation and numerous Monte Carlo methods for inference and learning. He has published over 300 papers in the areas of computer vision, statistical learning, cognition, AI, and robot autonomy. He has received a number of honors, including the David Marr Prize in 2003 for image parsing, and twice Marr Prize honorary nominations in 1999 for texture modeling and in 2007 for object modeling. In 2001, he received the NSF Career award, ONR Young Investigator award and Sloan Fellowship. In 2008 he received the J.K. Aggarwal Prize from the Intl. Association of Pattern Recognition for "contributions to a unified foundation for visual pattern conceptualization, modeling, learning, and inference". In 2013 he received the Helmholtz Test-of-Time Prize for a paper on image segmentation. He has been a fellow of IEEE Computer Society since 2011, and the principal investigator leading several ONR MURI and DARPA teams working on scene and event understanding and cognitive robots under a unified mathematical framework.

Contents

Chapter 1
Introduction to Monte Carlo Methods

The Casino de Monte Carlo in Monaco

"Life is not always a matter of holding good cards, but sometimes, playing a poor hand well." - Jack London

Introduction

Monte Carlo, named after a casino in Monaco, simulates complex probabilistic events using simple random events, such as the tossing of a pair of dice to simulate the casino's overall business model. In Monte Carlo computing, a pseudo-random number generator is repeatedly called which returns a real number in $[0, 1]$, and the results are used to generate a distribution of samples that is a fair representation of the target probability distribution under study. This chapter introduces the primary concepts of Monte Carlo, including the two major categories (sequential and Markov chain) and the five objectives (simulation, estimation, optimization, learning, and visualization). Examples of each task are given, and applications including approximate counting, ray tracing, and particle filtering are studied.

1.1 Motivation and Objectives

In general, Monte Carlo methods are divided into two categories:

- Sequential Monte Carlo, which preserves and propagates a population of examples by sequential sampling and importance reweighing, often in a low dimensional state space.
- Markov chain Monte Carlo, which simulates a Markov chain to explore the state space with a stationary probability designed to converge to a given target probability.

In engineering applications, e.g. computer vision, graphics and machine learning, the target functions are defined on graph representations, and researchers are faced with a choice between three types of modeling and computing paradigms that make trade-offs between model accuracy and computational complexity.

- Approximate model with exact computing: One simplifies the representation by breaking the loopy connections or removing certain energy terms. Once the underlying graph becomes a tree or a chain, algorithms like dynamic programming are applicable to find the exact solution to the approximated problem. Also in this class are problems for which a convex approximation of the energy is found and a convex optimization algorithm is used to find the global energy optimum. Examples include L_1-penalized regression (lasso) [7] and classification, where the non-convex L_0 penalty on the number of nonzero model weights is replaced with the convex L_1 penalty.
- Exact model with local computing: One stays with the original representation and target function, but uses an approximate algorithm, e.g. gradient descent, to find a local solution and thus relies on heuristics to guide the initial states.
- Exact model with asymptotic global computing: This class contains the Monte Carlo methods, which simulate large enough samples over time, and converge to the globally optimal solution with high probability.

Monte Carlo methods have been used in many different tasks, which we shall elaborate with examples in the next section.

1. Simulating a system and its probability distribution $\pi(x)$

$$x \sim \pi(x); \tag{1.1}$$

2. Estimating a quantity through Monte Carlo integration

$$c = E_\pi(f(x)] = \int \pi(x) f(x) \mathrm{d}x; \tag{1.2}$$

3. Optimizing a target function to find its modes (maxima or minima)

$$x^* = \arg\max \pi(x); \tag{1.3}$$

4. Learning parameters from a training set to optimize some loss functions, such as the maximum likelihood estimation from a set of examples $\{x_i, i = 1, 2, ..., M\}$

$$\Theta^* = \arg\max \sum_{i=1}^{M} \log p(x_i; \Theta); \text{and} \tag{1.4}$$

5. Visualizing the energy landscape of a target function and thus quantifying the difficulty of one of the tasks above and the efficiency of various algorithms. For example, biologists are interested in the energy landscape of protein folding. Different proteins have different landscapes and local minima of the energy landscape could be related to certain diseases (e.g. Alzheimer's disease). In computer vision the energy landscape of learning algorithms such as Convolutional Neural Networks (CNN) are interesting to study to understand why they seem to give good results independent of the initialization (are all the local minima equivalent up to a permutation of the filters?), or for other learning algorithms to understand what are the difficulties of learning the correct model and how the energy landscape changes with the number of observations.

As one can see, Monte Carlo methods can be used for many complex problems.

1.2 Tasks in Monte Carlo Computing

Real world systems studied in sciences (e.g. physics, chemistry, and biology) and engineering (e.g. vision, graphics, machine learning and robotics) involve complex interactions between large numbers of components. Such systems are often represented as graphs where the vertices represent components and the edges the interactions. The behavior of the system is governed by a probabilistic model defined on the graph.

For example, in statistical physics, ferro-magnetic materials are represented by the classical Ising and Potts models [6]. These models are also used in computer vision to represent the dependency between adjacent pixels in terms of Gibbs distributions and Markov random fields.

In general we are given a number of observations $\{\mathbf{x}_1, ..., \mathbf{x}_n\} \sim f(\mathbf{x})$ that represent samples from the "true" probabilistic model $f(\mathbf{x})$. In reality, $f(\mathbf{x})$ is usually unknown and can only be approximated by the empirical samples $\{\mathbf{x}_1, ..., \mathbf{x}_n\}$.

1.2.1 Task 1: Sampling and simulation

Many times we are interested in learning the unknown "true" model $f(\mathbf{x})$, which means approximating it with a parametric model $P(\mathbf{x};\theta)$. In many occasions, learning the model or even finding how well the learned model $P(\mathbf{x};\theta)$ compares to the true model means obtaining samples $\mathbf{x} \sim P(\mathbf{x};\theta)$ from it and computing some sufficient statistics on these samples. Thus, sampling is one of the fundamental tasks of Monte Carlo computing.

For example, we denote a 2D image lattice by

$$\Lambda = \{(i, j) : 1 \leq i, j \leq N\}. \tag{1.5}$$

Each pixel is a vertex with image intensity $\mathbf{I}_{(i,j)} \in \{0, ..., 255\}$. An image, denoted by $\mathbf{I}_\Lambda$ is a microscopic state of the underlying system governed by a probability $\pi(\mathbf{I}_\Lambda; \Theta)$. In other words, when the system reaches a dynamic equilibrium, its state follows a Gibbs distribution

$$\mathbf{I}_\Lambda \sim \pi(\mathbf{I}_\Lambda; \Theta) \tag{1.6}$$

where Θ is a vector of K parameters, and the Gibbs distribution can be written in the following form,

$$\pi(\mathbf{I}_\Lambda; \Theta) = \frac{1}{Z} \exp\{- < \Theta, H(\mathbf{I}_\Lambda) >\}. \tag{1.7}$$

In the above formula, Z is a normalizing constant, $H(\mathbf{I}_\Lambda)$ is a vector of K sufficient statistics of image $\mathbf{I}_\Lambda$ and the inner product is called the potential function $U(\mathbf{I}) = < \Theta, H(\mathbf{I}_\Lambda) >$.

When the lattice is sufficiently large, the probability mass of $\pi(\mathbf{I}_\Lambda; \theta)$ will focus on a subspace, called the *micro-canonical ensemble* in statistical physics [4]

$$\Omega_\Lambda(\mathbf{h}) = \{\mathbf{I}_\Lambda : H(\mathbf{I}_\Lambda) = \mathbf{h}\}. \tag{1.8}$$

Here, $\mathbf{h} = (h_1, ..., h_k)$ is a constant vector called the macroscopic state of the system.

Therefore, drawing fair samples from the distribution $\Omega_\Lambda(\mathbf{h}) \sim \pi(\mathbf{I}_\Lambda; \Theta)$ is equivalent to sampling from the ensemble $\Omega_\Lambda(\mathbf{h}) \in \Omega_\Lambda(\mathbf{h})$. In plain language, the sampling process aims to simulate the "typical" microscopic state of the system. In computer vision, this is often called *synthesis* – a way to verify the sufficiency of the underlying model.

Example 1.1. **Simulating Gaussian noise images.** In a large lattice, we define a "Gaussian noise" pattern as an ensemble of images with fixed mean and variance.

$$\text{Gaussian noise} = \Omega_\Lambda(\mu, \sigma^2) = \{\mathbf{I}_\Lambda : \frac{1}{N^2} \sum_{(i,j)\in\Lambda} I(i, j) = \mu, \frac{1}{N^2} \sum_{(i,j)\in\Lambda} (I(i, j) - \mu)^2 = \sigma^2\}.$$

In this case, the model has $K = 2$ sufficient statistics. Figure 1.1 displays a typical noise image as a sample from this ensemble or distribution.

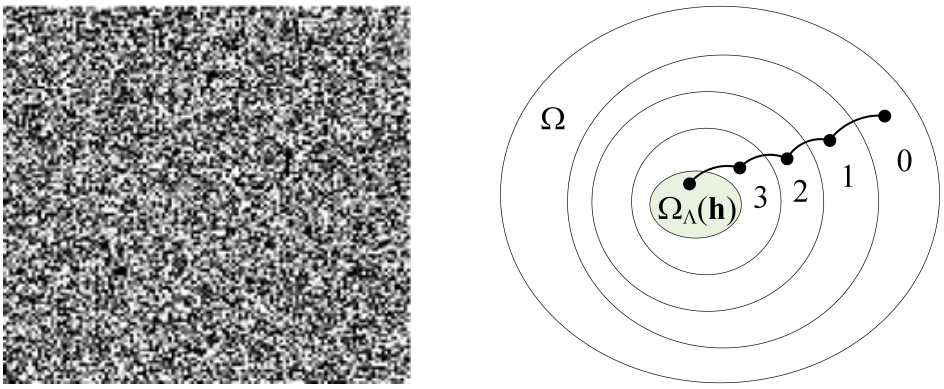
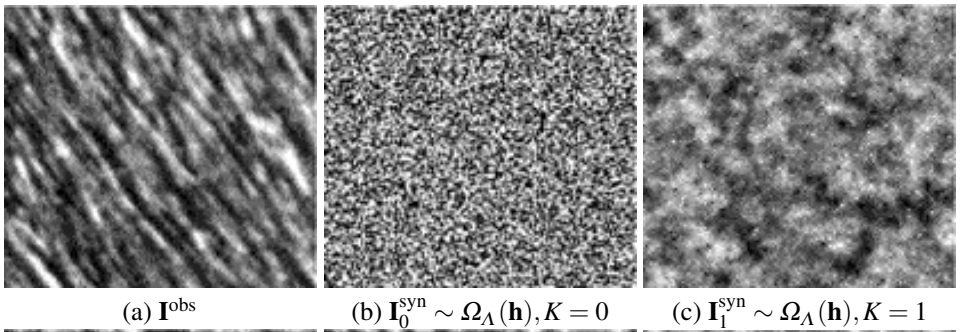


Fig. 1.1: Left: A typical image sampled from a Gaussian model. Right: a set of nested ensemble spaces $\Omega_\Lambda(\mathbf{h})$ with increasing number of constraints from $K = 0, 1, 2, 3$.

Note

Why is that the highest probability image $\mathbf{I}_\Lambda$ is not a typical image from $\Omega_\Lambda(\mu, \sigma^2)$?



(a) $\mathbf{I}^{\text{obs}}$ (b) $\mathbf{I}_0^{\text{syn}} \sim \Omega_\Lambda(\mathbf{h}), K = 0$ (c) $\mathbf{I}_1^{\text{syn}} \sim \Omega_\Lambda(\mathbf{h}), K = 1$

(d) $\mathbf{I}_3^{\text{syn}} \sim \Omega_\Lambda(\mathbf{h}), K = 3$ (e) $\mathbf{I}_4^{\text{syn}} \sim \Omega_\Lambda(\mathbf{h}), K = 4$ (f) $\mathbf{I}_7^{\text{syn}} \sim \Omega_\Lambda(\mathbf{h}), K = 7$

Fig. 1.2: Simulating texture patterns from 5 different equivalence classes. © [1997] MIT Press. Reprinted, with permission, from ref. [10].

Example 1.2. **Simulating texture patterns**. As we will discuss in later Section 5.5, each texture pattern is defined as an equivalence class,

$$\text{texture} = \Omega_\Lambda(\mathbf{h}) = \{\mathbf{I}_\Lambda : H(\mathbf{I}_\Lambda) = \mathbf{h} = (h_1, ..., h_K)\}. \tag{1.9}$$

In this example, the sufficient statistics $H_k(\mathbf{I}_\Lambda), k = 1, 2, ..., K$ are the histograms of Gabor filters. That is, any two texture images will be perceptually equivalent if they share the same set of histogram of Gabor filters. More detailed discussions are referred to chapter 5.5 and references [9, 10].

Figure 1.2 displays an example of texture modeling and simulation, and demonstrates the power of the Markov chain Monte Carlo (MCMC) methods. Since the 1960s, a famous psycho-physicist Julesz studied texture perception, raised a classical problem which was later called the *Julesz quest*:

> "What is the set of features and statistics such that two texture images sharing the same features and statistics cannot be told apart by pre-attentive processing?"

While the psychological interest is to find the sufficient statistics $\mathbf{h}$ from an image image $\mathbf{I}_\Lambda$, the Julesz quest posed a significant challenge technically: how do we generate fair samples for a given statistics $\mathbf{h}$? This was answered in the late 1990s by Zhu, Wu and Mumford using the Markov chain Monte Carlo method (MCMC) [10]. Figure 1.2.(a) is an observed texture image $\mathbf{I}^{\text{obs}}$, from which one can extract any sufficient statistics, $\mathbf{h}$, under consideration. To verify the statistics, one needs to draw typical samples drawn from the ensembles, or equivalently some Gibbs distributions, which satisfy the K feature statistics. Figure 1.2.(b-f) are examples for $K = 0, 1, 3, 4, 7$ respectively. Each statistic is a histogram of Gabor filtered responses pooled over all pixels, and is selected sequentially in a learning process [10]. As is demonstrated, with $K = 7$ selected statistics, the generated texture images $\mathbf{I}^{\text{syn}}_7$ is perceptually equivalent to the observed image $\mathbf{I}^{\text{obs}}$, i.e.

$$h_k(\mathbf{I}^{\text{syn}}_7) = h_k(\mathbf{I}^{\text{obs}}), \quad k = 1, 2, ..., 7. \tag{1.10}$$

Therefore, the MCMC method plays a key role in solving the Julesz quest.

1.2.2 Task 2: Estimating quantities by Monte Carlo simulation

In scientific computing, one common problem is to compute the integral of a function in a very high dimensional space Ω,

$$c = \int_\Omega \pi(\mathbf{x}) f(\mathbf{x}) d\mathbf{x}. \tag{1.11}$$

This is often estimated through Monte Carlo integration. By drawing M samples from $\pi(\mathbf{x})$,

$$x_1, x_2 ..., x_M \sim \pi(\mathbf{x}),$$

one can estimate c by the sample mean

$$\bar{c} = \frac{1}{M} \sum_{i=1}^{M} f(\mathbf{x}_i). \tag{1.12}$$

This is often done by the sequential Monte Carlo (SMC) method. We briefly discuss three examples of SMC.

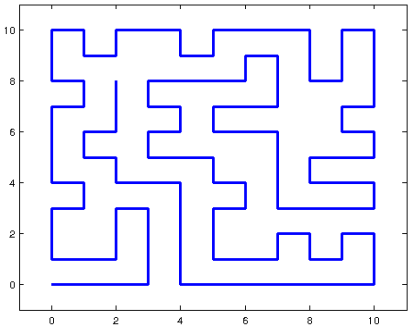


Fig. 1.3: A self-avoiding walk of length 115.

Example 1.3. **Approximate counting.** In chemistry, an interesting problem is to calculate the number of polymers in a unit area. This is abstracted into a Self-Avoiding-Walks (SAW) problem in Monte Carlo computing. In an $N \times N$

lattice, a SAW $\mathbf{r}$, is a path which does not go through any site twice. An example of a SAW is given in Figure 1.3. Denote the set of SAWs by

$$\Omega_{N^2} = \{\mathbf{r} : \mathrm{SAW}(\mathbf{r}) = 1\}. \tag{1.13}$$

where $SAW()$ is a logic indicator. As we will discuss in Chapter 2, the cardinality of Ω_{N^2} can be estimated by Monte Carlo integration,

$$|\Omega_{N^2}| = \sum_{\mathbf{r}\in\Omega_{N^2}} 1 = \sum_{\mathbf{r}\in\Omega_{N^2}} \frac{1}{p(r)} p(r) = E_p[\frac{1}{p(r)}] \approx \frac{1}{M}\sum_{i=1}^{M} \frac{1}{p(\mathbf{r}_i)}. \tag{1.14}$$

In the above formulas, the SAW paths are sampled from a reference model $p(\mathbf{r}_i)$ through random walks that grow the chain sequentially. For example, when $N = 10$, the estimated number of SAW paths starting from the lower-left corner $(0,0)$ to the upper-right corner $(10,10)$ is $(1.6 \pm 0.3) \times 10^{24}$. The true number is 1.56875×10^{24}.

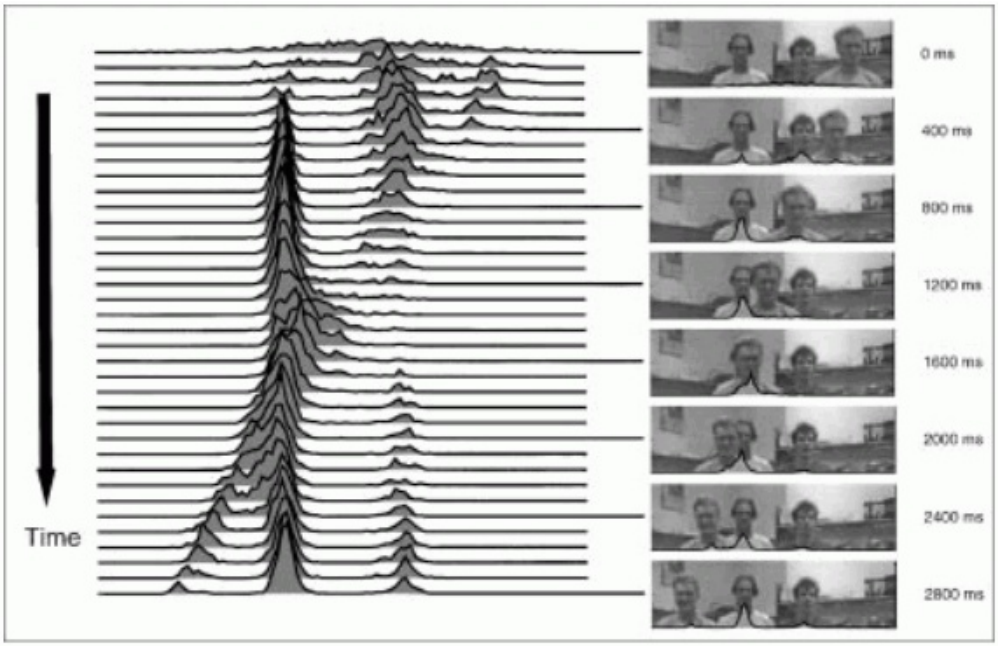


Fig. 1.4: Tracking objects by Sequential Monte Carlo. © [1998] Springer. Reprinted, with permission, from ref. [3].

Example 1.4. **Particle filtering.** In computer vision, a well-known task is tracking objects in a video sequence. Figure 1.4 is a simplified example, where the object's (i.e. humans here) position is represented by the horizontal axis x, and each row is a video frame $\mathbf{I}(t)$ at time t. Given an input video $\mathbf{I}[0,t]$, the objective of online tracking is to approximately represent the posterior probability of position by a set of samples,

$$S(t) = \{(x_i(t), \omega_i(t)) : i = 1,2,...,M\} \approx \pi(x(t)\,|\,\mathbf{I}[0,t]), \tag{1.15}$$

where $\omega_i(t)$ is the weight for $x_i(t)$. $S(t)$ encodes a non-parametric distribution as is illustrated by each row in Figure 1.4, and is propagated in time through the following recursive integration,

$$\pi(x(t+1)\,|\,\mathbf{I}[0,t+1]) = \int g(\mathbf{I}(t+1)\,|\,x(t+1)) p(x(t+1)\,|\,x(t)) \cdot \pi(x(t)\,|\,\mathbf{I}[0,t]) dx(t). \tag{1.16}$$

In this integration, $p(x(t+1)\,|\,x(t))$ is the dynamic model for object movement, and $g(\mathbf{I}(t+1)\,|\,x(t+1))$ is the image likelihood model measuring the fitness of position $x(t+1)$ to the observation. Each sample in the set $S(t)$ is called a particle. By representing the whole posterior probability, the sample set $S(t)$ preserves flexibility for achieving robustness in object tracking.

Example 1.5. **Monte Carlo Ray tracing.** In computer graphics, Monte Carlo integration is used to implement the ray-tracing algorithm for image rendering. Given a three-dimensional physical scene with geometry, reflection, and illumination, the photons emitted from light sources will bounce between object surfaces, or go through transparent objects before they hit the imaging plane. The ray tracing approach calculates the color and intensity of each pixel on the imaging plane by summation (integration) over all light sources for which a ray can be drawn through the pixel and various objects back to the source. This calculation is computationally intensive and can be approximated by Monte Carlo integration, as we will elaborate in Chapter 2.

1.2.3 Task 3: Optimization and Bayesian inference

A basic assumption in computational vision, since Helmholtz (1860), is that biological and machine vision compute the most probable interpretation(s) from an input image. Denoting the interpretation by W, for the perceived world, one can pose it as an optimization problem that maximizes a Bayesian posterior probability,

$$W^* = \arg\max \pi(W \,|\, \mathbf{I}) = \arg\max p(\mathbf{I} \,|\, W) p(W), \tag{1.17}$$

where $p(W)$ is the prior model for how the real world scene may be organized, and $p(\mathbf{I} \,|\, W)$ is the likelihood for generating image $\mathbf{I}$ from a given scene W.

Sometimes, images have multiple plausible interpretations, and thus in a more general setting, one needs to keep multiple distinct interpretations to approximately represent the posterior

$$\{(W_i, \omega_i) : i = 1, 2, ..., M\} \approx p(W \,|\, \mathbf{I}). \tag{1.18}$$

Markov chain Monte Carlo can be used to obtain samples from the posterior $p(W \,|\, \mathbf{I})$; however, sampling the posterior is not the same thing as maximizing it. The posterior can also be maximized by simulated annealing, which means sampling $p(W \,|\, \mathbf{I})^{1/T}$ where T is a temperature parameter that changes during the procedure. At the beginning of the annealing procedure, the temperature is high, which means $p(W \,|\, \mathbf{I})^{1/T}$ is close to uniform and the MCMC can freely explore the solution space. During the annealing, the temperature is slowly decreased according to an annealing schedule. As the temperature decreases, the probability $p(W \,|\, \mathbf{I})^{1/T}$ becomes more and more concentrated around the maximum locations and the MCMC explores these locations more carefully. When the temperature is very small, the MCMC should approach a maximum of the posterior $p(W \,|\, \mathbf{I})$.

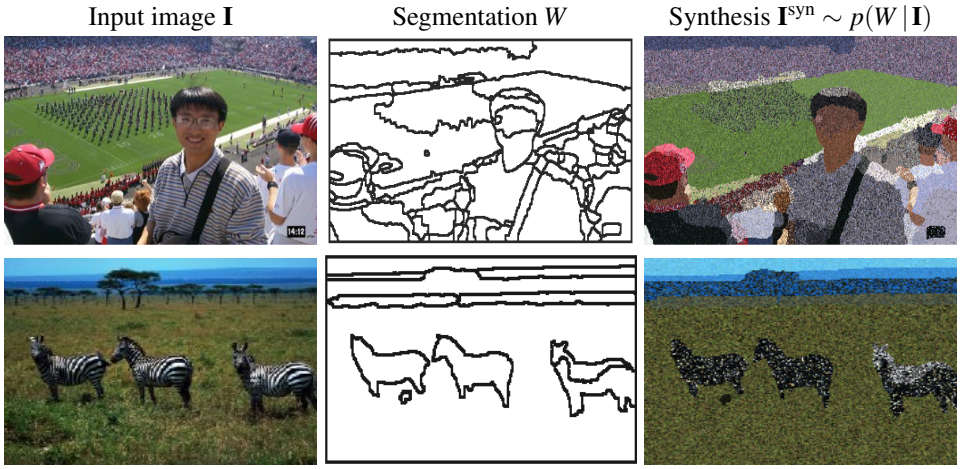


Fig. 1.5: Image segmentation by Data-driven Markov chain Monte Carlo. © [2002] IEEE. Reprinted, with permission, from ref. [8].

Example 1.6. **Image segmentation and parsing.** In computer vision, image segmentation and parsing is a core problem. In such tasks, since the underlying scene complexity is unknown, the number of variables in W is not fixed.

Therefore the prior model $\pi(W)$ is distributed over a heterogeneous solution space, which is a union of subspaces of varying dimensions. When the objects in a scene are compositional, the solution W is a parse graph, and the structures of the solution space then become more complicated. Finding optimal solutions in such complex spaces can be executed by Monte Carlo methods, which simulate Markov chains to traverse the solution spaces by mixing a number of dynamics: death and birth, split and merge, model switching, and boundary diffusion. To improve computational efficiency, the Markov chains are guided by marginal distributions computed using data-driven approaches. We will elaborate the details in Chapter 8.

Figure 1.5 illustrates two instances computed by the Data-driven Markov Chain Monte Carlo method [8]. The left column shows the two input images, and the segmentation results are in the middle with each region being fitted to some likelihood model. To verify the world W^* computed by the computer algorithm, we sample typical images from the likelihood $\mathbf{I}^{\text{syn}} \sim p(W \mid \mathbf{I})$. In the top example, the likelihood does not include face models, and thus the human face is not constructed.

1.2.4 Task 4: Learning and model estimation

In statistical and machine learning, one needs to compute parameters that optimize some loss functions, which are often highly non-convex, especially when hidden variables are involved. In the following, we briefly discuss two examples.

Example 1.7. **Learning Gibbs distributions.** Consider the Gibbs model that we mentioned in Section 1.2.1. We omit the lattice sign, Λ, for clarity,

$$p(\mathbf{I};\Theta) = \frac{1}{Z}\exp\{- < \Theta, H(\mathbf{I}) >\}. \tag{1.19}$$

Given an set of examples $\{\mathbf{I}_i^{\text{obs}}, i = 1,2,...,M\}$, the objective of learning is to estimate the parameters by maximizing the likelihood of the data,

$$\Theta^* = \arg\max \ell(\Theta), \text{ with } \ell(\Theta) = \sum_{i=1}^{M} \log p(\mathbf{I}_i^{\text{obs}};\Theta). \tag{1.20}$$

The loss function $\ell(\Theta)$ is convex with respect to Θ. Setting $\frac{\partial \ell}{\partial \Theta} = 0$, we derive the following constraint equations,

$$\int H(\mathbf{I})p(\mathbf{I};\Theta)d\mathbf{I} = \mathbf{h} = \frac{1}{M}\sum_{i=1^M} H(\mathbf{I}_i^{\text{obs}}). \tag{1.21}$$

Θ usually has to be solved for by stochastic gradient. Let t denote the time step, then one samples a set of typical examples $\{\mathbf{I}_i^{\text{syn}}, i = 1,2,...,M\}$ from the current model $p(\mathbf{I};\Theta(t))$ using Markov chain Monte Carlo as in example 1.2, and uses the sample mean $\hat{\mathbf{h}}(t) = \frac{1}{M}\sum_{i=1}^{M} H(\mathbf{I}_i^{\text{syn}})$ to estimate the expectation (i.e. Monte Carlo integration). The parameter is updated by gradient ascent,

$$\frac{d\Theta}{dt} = \eta(\mathbf{h} - \hat{\mathbf{h}}(t)), \tag{1.22}$$

where η is a step size.

The intuition is that the parameters Θ are updated so that the distribution on the observed data and the distribution obtained from the model cannot be told apart according to some sufficient statistics represented by $H(\mathbf{I})$.

Example 1.8. **Restricted Bolzmann Machines.** In deep learning, a Restricted Bolzmann machine (RBM) is a neural network with binary inputs and outputs. It has a matrix of weights (i.e. parameters) $W = (W_{ij})$ connecting a vector of visible units (inputs) $\mathbf{v}$ with a vector of hidden units (outputs) $\mathbf{h}$. Note that this notation has different meaning from the $\mathbf{h}$ in the previous example. It also has vectors $\mathbf{a}, \mathbf{b}$ of biases for the visible units and hidden units respectively. The probability of a RBM is a Gibbs distribution

$$p(\mathbf{v},\mathbf{h};\Theta) = \frac{1}{Z}\exp(-E(\mathbf{v},\mathbf{h}))$$

based on the RBM energy function

$$E(\mathbf{v},\mathbf{h};\Theta) = -\mathbf{a}^T\mathbf{v} - \mathbf{b}^T\mathbf{h} - \mathbf{v}^T W\mathbf{h}.$$

Training the RBM with a set of examples $\mathbf{v}_1,...,\mathbf{v}_n$ usually means maximizing the log likelihood:

$$\Theta^* = (W,\mathbf{a},\mathbf{b})^* = \text{argmax}\sum_{i=1}^{n}\log\int p(\mathbf{v}_i,\mathbf{h};\Theta)d\mathbf{h}$$

This optimization is done using Monte Carlo methods in the same way as the previous example. A variant method used in [2] is the so called contrastive divergence.

1.2.5 Task 5: Visualizing the landscape

In previous tasks, the Monte Carlo methods are used to draw fair examples from a target distribution (task 1), and then these samples are used to estimate quantities by Monte Carlo integration (task 2), and to optimize some posterior probability in the state space (task 3) or loss function in the model space (task 4). The most ambitious task that uses Monte Carlo methods is to visualize the whole energy landscape. This energy function can be the negative log-posterior probability $-\log p(W\,|\,\mathbf{I})$ on Ω_X for inference tasks, or the loss function $L(\Theta|Data)$ in the parameter space for learning tasks.

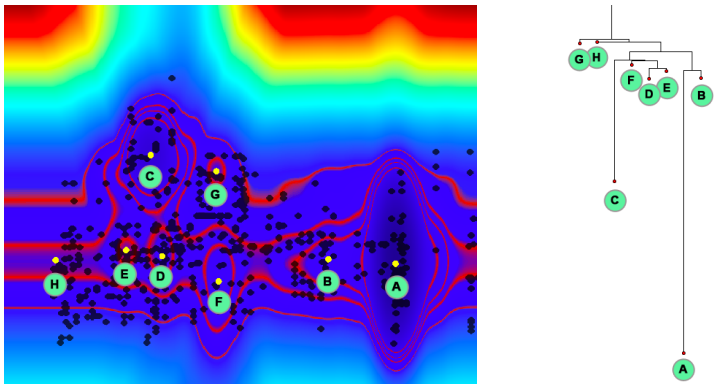


Fig. 1.6: Visualizing the landscape. (left) An an energy function in 2D space. (right) the tree representation. Dark color means lower energy. © [2014] Maria Pavlovskaia. Reprinted, with permission, from ref. [5].

In real world applications, these functions are highly non-convex with complex, often horrific, landscapes which are characterized by an exponential number of local minima in high-dimensional spaces. Figure 1.6 illustrates a simplified two-dimensional energy function in a K-means clustering and learning problem. This energy function has multiple local minima of varying depths and widths denoted by letters $A,B,...H$. The red curves are level sets consisting of points with the same energy levels.

The objective of task 5 is to draw effective samples from the whole space using effective Markov chain Monte Carlo methods, and then to map all local minima in their energy basins while locating the saddle point connecting adjacent basins. The result is represented by a tree structure, which physicist call *disconnectivity graphs* [1] when they map the landscapes of Spin-glass models. In this graph, each leaf node represents a local minimum and its depth represents the energy level. The energy level at which two adjacent leaf nodes meet is decided by their saddle point.

In the following, we show a learning example where the landscape is in the model space, not the state space, and thus is more difficult to compute.

Example 1.9. **Landscape of data clustering.** K-mean clustering is a classical problem in statistical and machine learning. Given a finite number of points whose color indicates the true labels, the learning problem is to find the parameters Θ that best fit the data. Here, Θ includes the means, variances, and weights of $K = 3$ Gaussian models. The energy

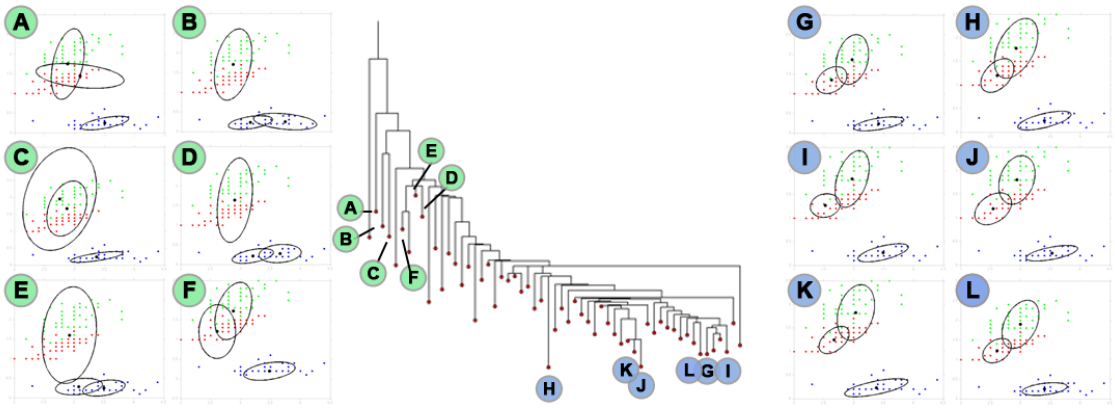


Fig. 1.7: Visualizing the landscape of a clustering problem. © [2014] Maria Pavlovskaia. Reprinted, with permission, from ref. [5].

function $\ell(\Theta)$ is a posterior probability with likelihood and a prior for Θ. In the literature, the popular algorithms are K-means and EM algorithms which find only local minima. By exploring the space where each point is a model Θ, one can visualize the landscape in Figure 1.7. The input data is from the Iris dataset in machine learning. Twelve of the local minima $A, B, ...L$ are illustrated on the two sides, where each Gaussian is an ellipse.

With this landscape, one can further visualize the behavior of various algorithms, and quantize the intrinsic difficulties of the target function, either for inference or learning. One can also use it to study the key factors that affect the complexity of the landscape.

Example 1.10. **SWC for Gaussian Mixture Models.** Let $\{\mathbf{x}_i \in \mathbb{R}^d, i = 1, ..., n\}$ be data points assumed to originate from a mixture of k multivariate Gaussians with unknown mixture weights α_i, means $\mu_i \in \mathbb{R}^d$ and covariance matrices Σ_i, for $i = 1, \ldots, K$. Let Θ contain all the unknown mixture parameters $\alpha_i, \mu_i, \Sigma_i, i = 1, \ldots, K$.

The log likelihood (energy) of the Gaussian mixture model is:

$$\log P(\Theta) = \sum_{i=1}^{n} \log \sum_{j=1}^{K} \alpha_j G(\mathbf{x}_i; \mu_j, \Sigma_j) - \log Z(\Theta), \tag{1.23}$$

where $G(\mathbf{x}_i; \mu_j, \Sigma_j) = \frac{1}{\sqrt{\det(2\pi\Sigma_j)}} \exp\left[-\frac{1}{2}(\mathbf{x}_i - \mu_j)^T \Sigma_j^{-1} (\mathbf{x}_i - \mu_j)\right]$ is a Gaussian and $Z(\theta)$ is the normalization constant.

If the labels assigning points to clusters are known, say $L = (l_1, \ldots, l_n)$ then the log likelihood is

$$\log P(L, \Theta) = \sum_{j=1}^{K} \sum_{i \in L_j} \log G(\mathbf{x}_i; \mu_j, \Sigma_j)$$

where $L_j = \{i, l_i = j\}$.

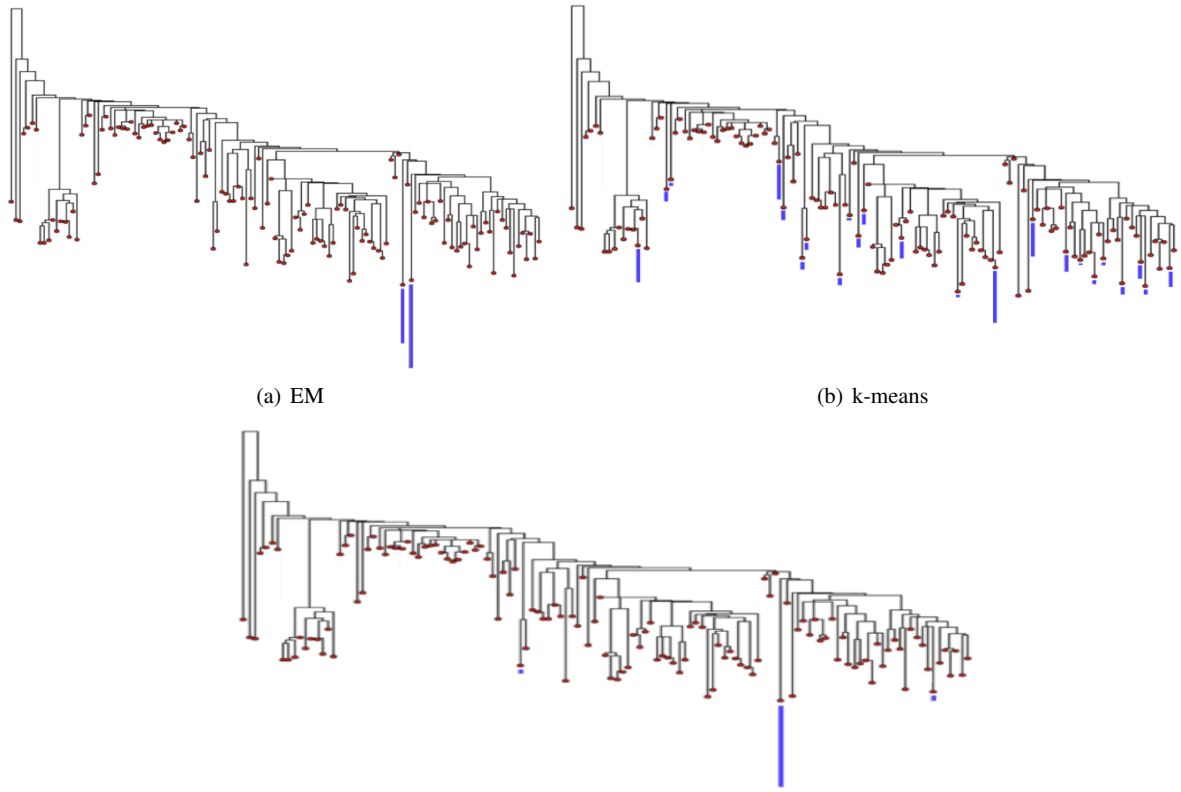
Sampling $P(\Theta)$ can be done by sampling $P(L, \Theta)$ and taking the marginal $P(\theta)$. Sampling $P(L, \Theta)$ can be done by alternating sampling $P(L|\Theta)$ and $P(\Theta|L)$.

For sampling $P(L|\Theta)$ we can use the SWC algorithm. We construct the SWC graph as the k-NN graph and use a constant probability q for all the edge weights.

Sampling $P(\Theta|L)$ is more involved. First, we should observe that $P(\Theta|L) = \prod_{j=1}^{K} \prod_{i \in L_j} G(\mathbf{x}_i; \mu_j, \Sigma_j)$ splits in independent parts: $P(\Theta|L) = \prod_{j=1}^{K} P(\Theta_j|L_j)$, where $\theta_j = (\alpha_j, \mu_j, \Sigma_j)$. Thus we can sample $P(\Theta_j|L_j)$ independently for each j by sampling $P(\mu_j|L_j, \Sigma_j)$ and $P(\Sigma_j|\mu_j, L_j)$. Now

$$P(\mu_j|\Sigma_j, L_j) = \prod_{i \in L_j} G(\mathbf{x}_i; \mu_j, \Sigma_j) \propto G(\mu_j, \frac{1}{n_j} \sum_{i \in L_j} \mathbf{x}_i, \frac{1}{n_j} \Sigma_j)$$

is just a Gaussian, where $n_j = |L_j|$. Also,



(a) EM (b) k-means

(c) SW-cut

Fig. 1.8: The performance of the EM, k-means, and SW-cut algorithm on the ELM. © [2014] Maria Pavlovskaia. Reprinted, with permission, from ref. [5].

$$P(\Sigma_j|\mu_j, L_j) = \det(\Sigma_j)^{-n_j/2} \exp(-\frac{1}{2}\sum_{i\in L_j}(\mu_j - \mathbf{x}_i)^T \Sigma_j^{-1}(\mu_j - \mathbf{x}_i)) = \det(\Sigma_j)^{-n_j/2} \exp(-\frac{1}{2}\mathrm{tr}(\hat{\Sigma}\Sigma_j^{-1}))$$

where $\hat{\Sigma} = \sum_{i\in L_j}(\mu_j - \mathbf{x}_i)(\mu_j - \mathbf{x}_i)^T$ and we used that $\mathrm{tr}(AB) = \mathrm{tr}(BA)$ with $A = (\mu_j - \mathbf{x}_i)$ and $B = (\mu_j - \mathbf{x}_i)^T\Sigma^{-1}$. Since $\hat{\Sigma}$ is symmetric and positive definite, there exists symmetric positive definite S such that $\hat{\Sigma} = S^2$. Then writing $B = S\Sigma_j^{-1}S$ we get

$$P(\Sigma_j|\mu_j, L_j) = \det(\Sigma)^{-n_j/2} \exp(-\frac{1}{2}\mathrm{tr}(S\Sigma^{-1}S)) = \det(S)^{-n_j/2} \det(B)^{n_j/2} \exp(-\frac{1}{2}\mathrm{tr}(B)).$$

Now writing $B = UDU^T$ where $D = diag(\lambda_1, ..., \lambda_d)$ is diagonal we obtain

$$P(\Sigma_j|\mu_j, L_j) \propto \det(D)^{n_j/2} \exp(-\frac{1}{2}\mathrm{tr}(D)) = \prod_{i=1}^{d} \lambda_i^{n/2} e^{-\lambda_i/2}$$

so to sample Σ_j we first sample the eigenvalues λ_i independently from the Gamma distribution $\Gamma(1 + \frac{n_j}{2}, 2)$ to obtain $D = diag(\lambda_1, ..., \lambda_d)$, then take any rotation matrix U to obtain $B = UDU^T$ and $\Sigma_j = SUDU^TS$.

In Figure 1.8 are shown energy landscapes for a 1D Gaussian mixture model with four mixture components and low separability. One can see that k-means get stuck in many local minima and SWC always finds the global minimum.

References

1. Oren M Becker and Martin Karplus. The topology of multidimensional potential energy surfaces: Theory and application to peptide structure and kinetics. *The Journal of chemical physics*, 106(4):1495–1517, 1997.
2. Geoffrey Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.
3. Michael Isard and Andrew Blake. Condensation: conditional density propagation for visual tracking. *International journal of computer vision*, 29(1):5–28, 1998.
4. John T Lewis, Charles-Edouard Pfister, and Wayne G Sullivan. Entropy, concentration of probability and conditional limit theorems. *Markov Process. Relat. Fields*, 1(GR-PF-ARTICLE-1995-004):319–386, 1995.
5. Maria Pavlovskaia. *Mapping Highly Nonconvex Energy Landscapes in Clustering, Grammatical and Curriculum Learning*. PhD thesis, Doctoral Dissertation, UCLA, 2014.
6. Renfrey Burnard Potts. Some generalized order-disorder transformations. In *Proceedings of the Cambridge Philosophical Society*, volume 48, pages 106–109, 1952.
7. Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.
8. Zhuowen Tu and Song-Chun Zhu. Image segmentation by data-driven markov chain monte carlo. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(5):657–673, 2002.
9. Ying Nian Wu, Song Chun Zhu, and Xiuwen Liu. Equivalence of julesz and gibbs texture ensembles. In *ICCV*, volume 2, pages 1025–1032, 1999.
10. Song Chun Zhu, Ying Nian Wu, and David Mumford. Minimax entropy principle and its application to texture modeling. *Neural Computation*, 9(8):1627–1660, 1997.

Chapter 2
Sequential Monte Carlo

"Give me a place to stand and with a lever I shall move the earth" - Archimedes

Introduction

Sequential Monte Carlo (SMC) is used when the distribution of interest is one-dimensional or multi-dimensional and factorizable. If $f(x)$ denotes the true probability distribution function controlling a process and $\pi(x)$ denotes a target probability distribution based on a model, then the goal is to find a model to make the target density function $\pi(x)$ converge to $f(x)$. In order to find this model, a known, trial probability density $g(x)$ may be used. In this chapter several concepts related to the selection of $g(x)$ for SMC are covered including sample weighting and importance sampling. Applications covered include self-avoiding walks, Parzen windows, ray tracing, particle filtering, and glossy highlights. The chapter ends with a discussion of Monte Carlo Tree Search.

2.1 Sampling a 1-dimensional density

Suppose $f(x) : \mathbb{R} \to \mathbb{R}$ is a one dimensional probability density function (pdf). Then the cumulative density function (cdf) $F(x) : \mathbb{R} \to [0,1]$ is defined as

$$F(x) \stackrel{\text{def}}{=} \int_{-\infty}^{x} f(x)\mathrm{d}x$$

We can obtain samples from the pdf $f(x)$ by taking uniform samples, u, back through the cdf $F(x)$ as $x = F^{-1}(u)$. More rigorously we have

Lemma 2.1 *Suppose $U \sim \text{Unif}[0,1]$ and F is the cdf of a one dimensional pdf f. Then, $X = F^{-1}(U)$ has the distribution f. Here we define $F^{-1}(u) = \inf\{x : F(x) \geq u\}$.*

Proof.

$$P(X \leqslant x) = P(F^{-1}(u) \leqslant x) = P(U \leqslant F(x)) = F(x) = \int_{-\infty}^{x} f(x)\mathrm{d}x. \square$$

By definition, we know that $\frac{\mathrm{d}u}{\mathrm{d}x} = \frac{\mathrm{d}F(x)}{\mathrm{d}x} = f(x)$, and thus $P\big(x \in (x_0, x_0 + \mathrm{d}x)\big) = P\big(u \in (u_o, u_0 + \mathrm{d}u)\big) = f(x) \cdot \mathrm{d}x$.

In higher dimensional space, as long as all the data sequence can be quantized/ordered, then $f(x)$ can be reduced to a one dimensional problem. However, we typically do not use this method when $d \geq 3$, since the computational complexity increases exponentially.

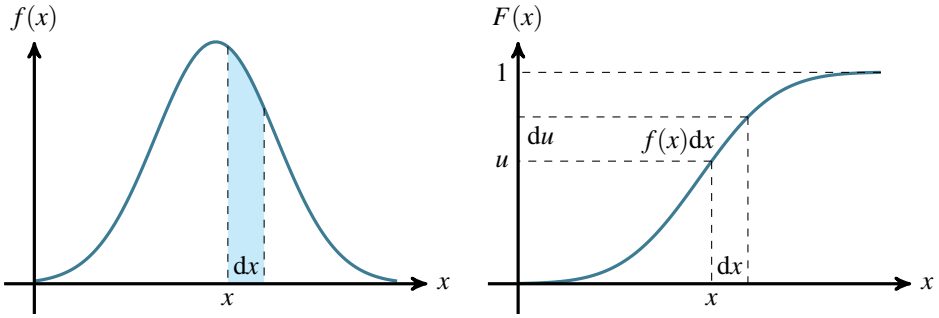


Fig. 2.1: Left: A pdf, $f(x)$. Right: Its corresponding cdf, $F(x)$. The shaded region on the left has area $f(x)\mathrm{d}x = \mathrm{d}u$.

2.2 Importance Sampling and Weighted Samples

Suppose that we want to estimate a quantity

$$C = \int_{\Omega} \pi(x) \cdot h(x)\mathrm{d}x = E_{\pi}[h(x)] \tag{2.1}$$

where $\pi(x)$ is a probability density function.

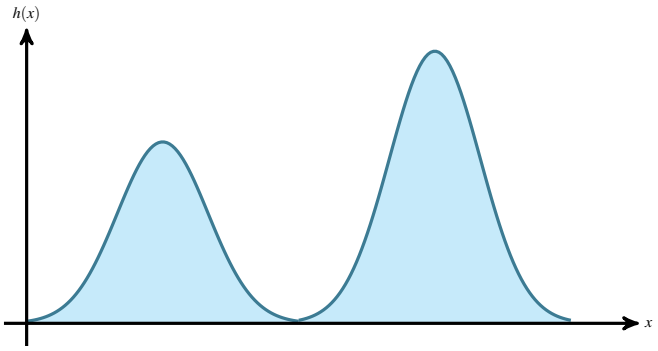


Fig. 2.2: A multi-modal pdf $h(x)$.

If we can draw samples from $\pi(x)$, $D = \{x_1, x_2, \cdots, x_n\} \sim \pi(x)$, then C can be easily estimated by

$$\hat{C} = \frac{1}{n}\sum_{i=1}^{n} h(x_i)$$

Since the information of $\pi(x)$ is inherent in D, we do not need to write it in the formula. However, if it is hard to draw samples directly from $\pi(x)$, we could easily draw samples, $D' = \{x'_1, x'_2, \cdots, x'_n\}$, from a simpler trial distribution, $g(x)$. Then eq.(2.1) can be expressed as

$$C = \int_{\Omega} \pi(x) \cdot h(x)\mathrm{d}x = \int_{\Omega} g(x) \cdot \left[\frac{\pi(x)}{g(x)} \cdot h(x)\right]\mathrm{d}x. \tag{2.2}$$

Suppose that the ratio $\frac{\pi(x)}{g(x)} \cdot h(x)$ is computable. Then we can estimate C as

$$\hat{C} = \frac{1}{n}\sum_{i=1}^{n}\frac{\pi(x_i')}{g(x_i')}\cdot h(x_i') = \frac{1}{n}\sum_{i=1}^{n}\omega(x_i')h(x_i'), \tag{2.3}$$

where $\omega(x_i')$ is the weight of sample i.

> **Note**
>
> In eq. (2.3), the weights $\{\omega(x_i'), i = 1,2,\ldots,m\}$ depend on $g(x_i')$ in the denominator. Thus we cannot let $g(x) = 0$ whenever $\pi(x) \neq 0$.

Suppose that $\pi(x) = \frac{1}{Z}\exp(-E(x)/T)$, where Z is the normalizing constant but it is not computable. So $\pi(x)$ is represented by a weighted sample, $\{(x^{(i)}, \omega^{(i)}), i = 1,\cdots,m\}$.

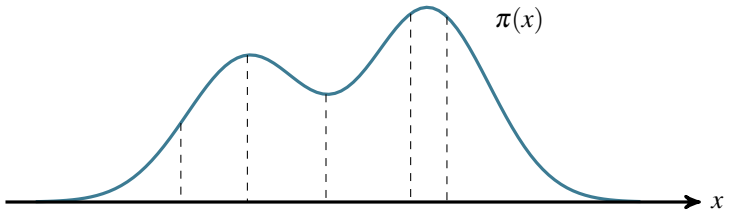


Fig. 2.3: A pdf $\pi(x)$ is approximated by a weighted sample.

A special case: if

$$g(x) = \text{Unif}[a,b] = \begin{cases} \frac{1}{b-a} & x \in [a,b] \\ 0 & \text{otherwise,} \end{cases}$$

then

$$\hat{C} = \frac{1}{n}\sum_{i=1}^{n}\frac{\pi(x_i')}{g(x_i')}\cdot h(x_i') = \frac{b-a}{n}\sum_{i=1}^{n}\pi(x_i')\cdot h(x_i')$$

In general, we will have the following three scenarios:

1) We draw from the uniform distribution, giving each sample a specific weight.
2) We draw from an easier distribution, $g(x)$, which is an approximation of $\pi(x)$, giving each sample a specific frequency and weight.
3) We draw directly from $\pi(x)$, giving each sample a specific frequency but equal weight.

It is easy to justify that 1) $\ll$ 2) $\ll$ 3), where "$a \ll b$" denotes a is much worse than b. Intuitively, the best case is $g(x) = \pi(x)$.

Since we require that

$$\lim_{n\to\infty}(\hat{C} - C) = 0,$$

and all three scenarios satisfy this, the only difference between them lies in the number of samples required for the estimator to converge or for its variance to vanish. This second point is given by,

$$\lim_{n\to\infty}||\hat{C} - C||^2 = 0.$$

The approximate distribution, $g(x)$, serves the purpose of a lever that can be used to handle $\pi(x)$. The Greek mathematician Archimedes (212 BC) is known for his famous remark:

"Give me a place to stand and with a lever I shall move the earth".

Inspired by him, we could refer to $g(x)$ as an *Archimedes lever* for $\pi(x)$.

Example 2.1. For the second scenario above, here is an example of an Archimedes lever.

$$\pi(x) = \frac{1}{Z}\exp\{-\sum_{i=1}^{K}\beta_i h_i(x)\}, \qquad g(x) = \frac{1}{Z'}\exp\{-\sum_{i=1}^{K'<K}\beta_i h_i(x)\}$$

Example 2.2. In the Gaussian case, we can use the scheme illustrated in the following example

$$\pi(x) = \frac{1}{Z}\exp\{-(ax^2+bx+c)\}, \qquad g(x) = \frac{1}{Z'}\exp\{-ax^2\}$$

Usually, we use a set of weighted samples from $g(x)$, $\{(x_i,\omega_i), i = 1,2,\ldots,m\}$, which is called the "empirical distribution" to represent $\pi(x)$. When $\mathbf{x} = (x_1,x_2,\ldots,x_n) \in \mathbb{R}^n$ is high-dimensional, we can simplify it in two ways.

$$\begin{aligned} g(\mathbf{x}) &= g(x_1,x_2,\ldots,x_n) \\ &\cong g(x_1)\cdot g(x_2)\cdot\cdots\cdot g(x_n) \quad \text{(by factorization and independence)} \qquad (2.4) \\ &\cong g(x_1)\cdot g(x_2)\cdot g(x_3|x_2)\cdot g(x_4|x_1,x_2)\cdot\cdots \quad \text{(by factorization)} \qquad (2.5) \end{aligned}$$

In (2.4), we are assuming that the x_i are independent, and we can sample each dimension individually; however, in truth they are always dependent. To correct this, we need to simplify the problem instead to (2.5) using conditional dependencies.

Since $\hat{C} = \frac{1}{m}\sum_{i=1}^{m} w(x_i)h(x_i)$, we know that $\mathrm{var}_m(\hat{C}) = \frac{1}{m}\mathrm{var}_1(\hat{C})$. This indicates that when we have sufficiently many samples, the total variance will go to zero, and the rate of convergence is $\frac{1}{m}$, regardless of the dimension n! The three plots in Figure 2.4 illustrate this idea. The convergence of the left distribution is quick while the convergence rate of the middle one can be slow. The right distribution might have problems with weights blowing up to ∞.

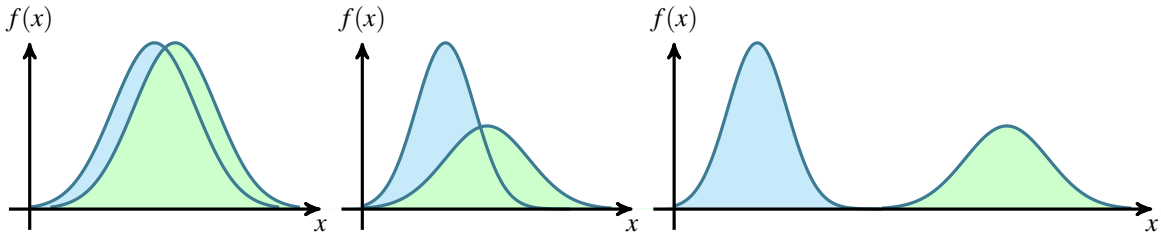


Fig. 2.4: Left: When $g(x)$ and $\pi(x)$ are close, convergence is fast. Middle: When $g(x)$ and $\pi(x)$ are far, convergence is much slower. Right: In general, $g(x)$ should be nonzero wherever $\pi(x)$ is nonzero, which might not happen in this case.

The heuristics for measuring the effectiveness of samples from $g(x)$ is to measure the variance of the weights. A useful rule of thumb is to use the *effective sample size* (ESS) to measure how different the trial distribution is from the target distribution. Suppose m independent samples are generated from $g(x)$. Then, the ESS is defined as

$$\mathrm{ESS}(m) = \frac{m}{1+\mathrm{var}[\omega(x)]} \qquad (2.6)$$

In the ideal case of $g(x) = \pi(x)$, then $\omega(x) = 1$, $\mathrm{var}_g[\omega(x)] = 0$, thus the whole samples are effective. Since the target distribution, π, is known only up to a normalizing constant in many problems, the variance of the *normalized* weight needs to be estimated by the *coefficient of variation* of the unnormalized weight:

$$\mathrm{cv}^2(\omega) = \frac{1}{m-1}\sum_{i=1}^{m}\frac{(\omega_i-\overline{\omega})^2}{\overline{\omega^2}} \qquad (2.7)$$

Generalization: *Stratified sampling* – a method to reduce $\mathrm{var}(\hat{C})$. Suppose the space Ω is the union of a number of disjoint subspaces $\Omega = \cup_{j=1}^{M}\Omega_j$. In each subspace, Ω_j, we can define different $g_j(x)$ as trial distributions. Thus, we will have

$$C = \sum_{j=1}^{M} \int_{\Omega_j} g_i(x) \cdot \frac{\pi(x)}{g_i(x)} \cdot h(x) \mathrm{d}x \tag{2.8}$$

In this calculation we can ignore the overlap of $g_i(x)$'s in higher dimensional space.

2.3 Sequential Importance Sampling (SIS)

In high-dimensional spaces, it is usually very hard to find an effective $g(x)$. Suppose we can decompose $\mathbf{x}$ as $\mathbf{x} = (x_1, \ldots, x_n)$ by chain rule. Then, our trial density can be constructed as

$$g(\mathbf{x}) = g_1(x_1) \cdot g_2(x_2|x_1) \cdots g_n(x_n|x_1, \ldots, x_{n-1}). \tag{2.9}$$

Usually this is impractical, but in some cases it can be done if the $\pi(x)$ is similarly factorized. Corresponding to the decomposition of $\mathbf{x}$, we can rewrite the target density as

$$\pi(\mathbf{x}) = \pi_1(x_1) \cdot \pi_2(x_2|x_1) \cdots \pi_n(x_n|x_1, \ldots, x_{n-1}). \tag{2.10}$$

and the importance weight as

$$\omega(\mathbf{x}) = \frac{g(\mathbf{x}) = g_1(x_1) \cdot g_2(x_2|x_1) \cdots g_n(x_n|x_1, \ldots, x_{n-1})}{\pi(\mathbf{x}) = \pi_1(x_1) \cdot \pi_2(x_2|x_1) \cdots \pi_n(x_n|x_1, \ldots, x_{n-1})} \tag{2.11}$$

In the following, we will discuss two examples.

1) Self-Avoiding Walks for representing polymer growth
2) Non-linear/particle filtering for object tracking

2.3.1 Application: the number of self-avoiding walks

The self-avoiding random walk (SAW) problem in a two- or three-dimensional lattice space is the problem of counting how many self-avoiding walks exist in a given domain.

We can use a hard-core model to describe this problem. A chain of atoms $\mathbf{x} = (x_1, x_2, \cdots, x_N)$ is connected by covalent bonds. For clarity, we assume each molecule to be a point in 2D/3D space/lattice, and a bond has length = 1, so the potential is referred to as hard-core. In the 2D or 3D space, the chain is not allowed to intersect itself.

In this section we will focus on the 2D domain $\{0, 1, ..., n\} \times \{0, 1, ..., n\}$. Suppose we always start from position $(0, 0)$, i.e. lower-left corner, as illustrated in Figure 2.5. Representing moves left/right/up/down by the numbers 1,2,3,4

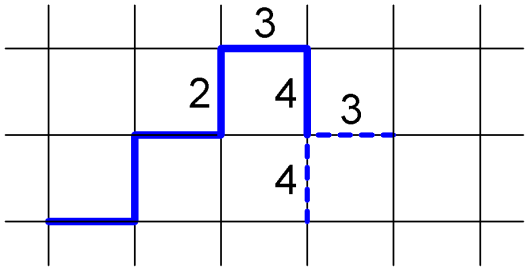


Fig. 2.5: An example of a self-avoiding walk.

respectively, the Gibbs/Boltzmann distribution for the chain of a SAW is

$$\pi(x) = \text{unif}[\Omega], \qquad \Omega = \{x : \text{SAW}(x) = 1\}, \quad x \in \{1,2,3,4\}^n.$$

We are interested in the total number of SAWs. To estimate this value, we use Monte Carlo integration. We design a trial probability $g(x)$ for a SAW, x, that is easier to sample. We then sample M SAWs from $g(x)$ and estimate the total count by

$$||\Omega|| = \theta = \sum_{x\in\Omega} 1 = \sum_{x\in\Omega} \frac{1}{g(x)} g(x) \cong \frac{1}{M} \sum_{i=1}^{M} \frac{1}{g(x_i)}, \tag{2.12}$$

where $\frac{1}{g(x_i)}$ serves as the weight, $\omega(x_i)$, of x_i.

The trial probability covers all possible paths, so we can use it to compute the size of many subsets of the set of SAWs, such as the set of SAWs that start in one corner and end in another, or the set of SAWs of length n. We do not need to worry about the normalization constant under this new subset.

Therefore, the problem lies in how to design $g(x)$, and there are several ways to do that. We explore three different formulations for $g(x)$ in a 2D grid with $n = 10$ to generate $M = 10^7$ samples.

a) Design 1. As an initial method, we use

$$g_1(x) = \prod_{j=1}^{m} \frac{1}{k_j}$$

where m is the total length of the path, and k_j is the number of possible choices for the j-th move, and at step j we sample uniformly from the k_j choices. Using $M = 10^7$ samples, the estimated number of SAWs is $K_1 = 3.3844 \cdot 10^{25}$. The length distribution of sampled walks is visualized in Figure 2.6. Since we do not constrain the length of the walk, the distribution obtained resembles a Gaussian.

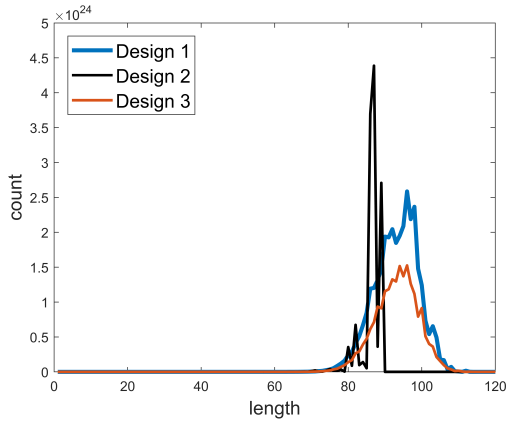


Fig. 2.6: Left: Distribution of SAW lengths for the three designs.

b) Design 2. As an alternative design for the trial distribution, we introduce an early termination probability $\varepsilon = 0.1$ at each step and get

$$g_2(x) = (1-\varepsilon)^m \prod_{j=1}^{m} \frac{1}{k_j}.$$

Clearly, in this case we expect to get shorter walks than with design 1. The length distribution of sampled walks is also shown in Figure 2.6, and the estimated number of SAWs is $K_2 = 6.3852 \cdot 10^{25}$.

c) Design 3. For the third design, we adjust design 1 to favor longer walks. For any walk longer than 50, we generate 5 more children that branch from the walk and reweight each of the children by $w_0 = w/5$. The length distribution of sampled walks is again shown in Figure 2.6, and the estimated number of SAWs is $K_3 = 7.3327 \cdot 10^{25}$. Examples of longest SAWs from the three designs are shown in Figure 2.7.

The log-log plot for the estimated number of SAWs, K, against the sample size M is shown in Figure 2.8. It is clear that design 3 converges the fastest and design 2 converges the slowest.

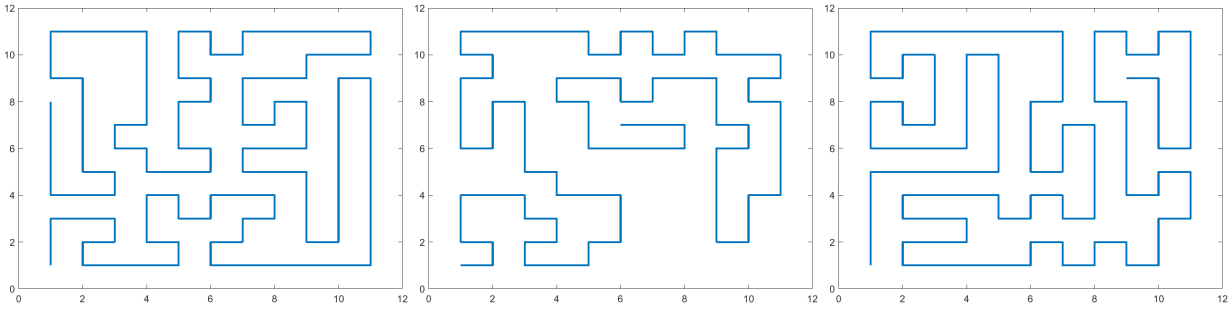


Fig. 2.7: Longest SAW for Design 1 (length 114, left), Design 2 (length 90, middle) and Design 3 (length 115, right).

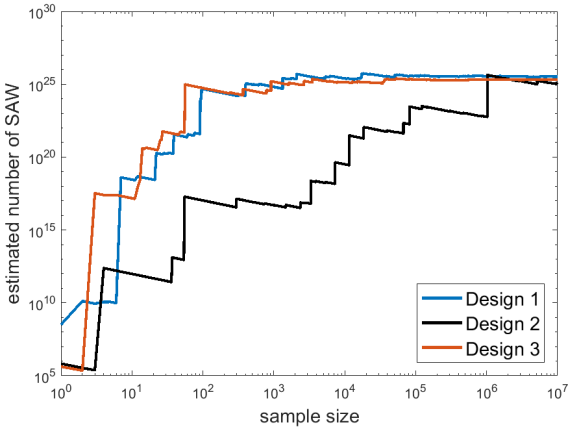


Fig. 2.8: Convergence rate comparison of the three designs.

Other ways to design the trial probability $g(x)$ include:

- Stop at any time (design 2)
- Fix the length N
- Enrich. Encourage longer samples – begin with a certain length (design 3)
- Global guidance? (e.g. $(0,0) \to (n,n)$)

We may also be interested in counting the SAWs that stretch from $(0,0)$ to (n,n). To get samples that reach (n,n), we resample until one is obtained and then reweight it by $w_0 = w/u$, where u is number of attempts. This means the more attempts we make, the more down-weighted this sample is. Generating 10^6 samples, we estimate the total number of SAWs from $(0,0)$ to (n,n) to be about $1.7403 \cdot 10^{24}$ (which is very close to the true value $1.5687 \cdot 10^{24}$).

2.3.2 Application: particle filtering for tracking objects in a video

Assume we have an object tracking problem where the state at time t is denoted by $\mathbf{x}_t$ and the observed image features are denoted by $\mathbf{z}_t$. Denote the state history by $\mathcal{X}_t = \{\mathbf{x}_1, ..., \mathbf{x}_t\}$ and the feature history by $\mathcal{Z}_t = \{\mathbf{z}_1, ..., \mathbf{z}_t\}$.

We assume that the object dynamics follow a Markov chain, i.e. the current state depends only on the preceding state, independent of the state history, as illustrated in Figure 2.9.

$$p(\mathbf{x}_{t+1}|\mathcal{X}_t) = p(\mathbf{x}_{t+1}|\mathbf{x}_t)$$

Michael Isard

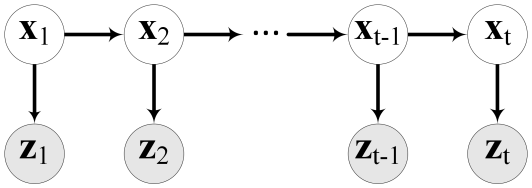


Fig. 2.9: The observations $\mathbf{z}_t$ are independent with each other and with respect to the dynamical process $\mathbf{x}_t$. The dynamical process $\mathbf{x}_t$ is dependent only on the previous state $\mathbf{x}_{t-1}$.


Andrew Blake

The observations $\mathbf{z}_t$ are assumed independent with each other and with respect to the dynamical process, as shown in Figure 2.9. We need to estimate $p(\mathbf{x}_{t+1}|\mathcal{Z}_{t+1})$, i.e. the distribution of the state x_{t+1} given the data received so far. We have

$$\begin{aligned} p(\mathbf{x}_{t+1}|\mathcal{Z}_{t+1}) &= p(\mathbf{x}_{t+1}|\mathbf{z}_{t+1},\mathcal{Z}_t) = \frac{p(\mathbf{x}_{t+1},\mathbf{z}_{t+1}|\mathcal{Z}_t)}{p(\mathbf{z}_{t+1}|\mathcal{Z}_t)} \\ &\propto p(\mathbf{z}_{t+1}|\mathbf{x}_{t+1},\mathcal{Z}_t)p(\mathbf{x}_{t+1}|\mathcal{Z}_t) = p(\mathbf{z}_{t+1}|\mathbf{x}_{t+1})p(\mathbf{x}_{t+1}|\mathcal{Z}_t), \end{aligned}$$

because $\mathbf{z}_{t+1}$ is independent of $\mathcal{Z}_t$. We can compute

$$p(\mathbf{x}_{t+1}|\mathcal{Z}_t) = \int p(\mathbf{x}_{t+1},\mathbf{x}_t|\mathcal{Z}_t)d\mathbf{x}_t = \int p(\mathbf{x}_{t+1}|\mathbf{x}_t)p(\mathbf{x}_t|\mathcal{Z}_t)d\mathbf{x}_t,$$

so we conclude that

$$p(\mathbf{x}_{t+1}|\mathcal{Z}_{t+1}) \propto \int p(\mathbf{z}_{t+1}|\mathbf{x}_{t+1})p(\mathbf{x}_{t+1}|\mathbf{x}_t)p(\mathbf{x}_t|\mathcal{Z}_t)d\mathbf{x}_t$$

The probability $p(\mathbf{z}_{t+1}|\mathbf{x}_{t+1})$ can be considered the bottom-up probability of detection, while the product $p(\mathbf{x}_{t+1}|\mathbf{x}_t)p(\mathbf{x}_t|\mathcal{Z}_t)$ is the prediction based on the dynamic model.

The Conditional Density Propogation (Condensation) algorithm [4] represents the distribution of $p(\mathbf{x}_t|\mathcal{Z}_t)$ using importance sampling as a weighted sample set $\{\mathbf{s}_t^{(n)}, n = 1,...,N\}$ with weights $\pi_t^{(n)}$. One step of the algorithm is illustrated in Figure 2.10 and described in Figure 2.11.

2.3.2.1 Application: Curve Tracking

A curve at time t is represented as $\mathbf{r}(s,t)$ parameterized as a B-spline:

$$\mathbf{r}(s,t) = (B(s)Q^x(t), B(s)Q^y(t)), s \in [0,L],$$

where $B(s) = (B_1(s),...,B_{N_B}(s))^T$ is a vector of B-spline basis functions. The vector $X_t = (Q^x,Q^y)^T$ contains the coordinates of the spline control points.

The dynamical model is first order autoregressive:

$$\mathbf{x}_t - \bar{\mathbf{x}} = A(\mathbf{x}_{t-1} - \bar{\mathbf{x}}) + B\mathbf{w}_t,$$

where $\mathbf{w}_t$ are independent vectors of i.i.d $\mathcal{N}(0,1)$ and $\mathbf{x}_t = \begin{pmatrix} X_{t-1} \\ X_t \end{pmatrix}$.

The dynamical model can also be expressed as:

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}) \propto \exp(-\frac{1}{2}\|(\mathbf{x}_t - \bar{\mathbf{x}}) - A(\mathbf{x}_{t-1} - \bar{\mathbf{x}})\|^2).$$

The observation model for 2D curves can be for example:

$$p(\mathbf{z}|\mathbf{x}) \propto \exp\left(-\sum_{m=1}^{M} \frac{1}{2rM} f\left(\mathbf{z}_i(\frac{m}{M}) - \mathbf{r}(\frac{m}{M});\mu\right)\right),$$

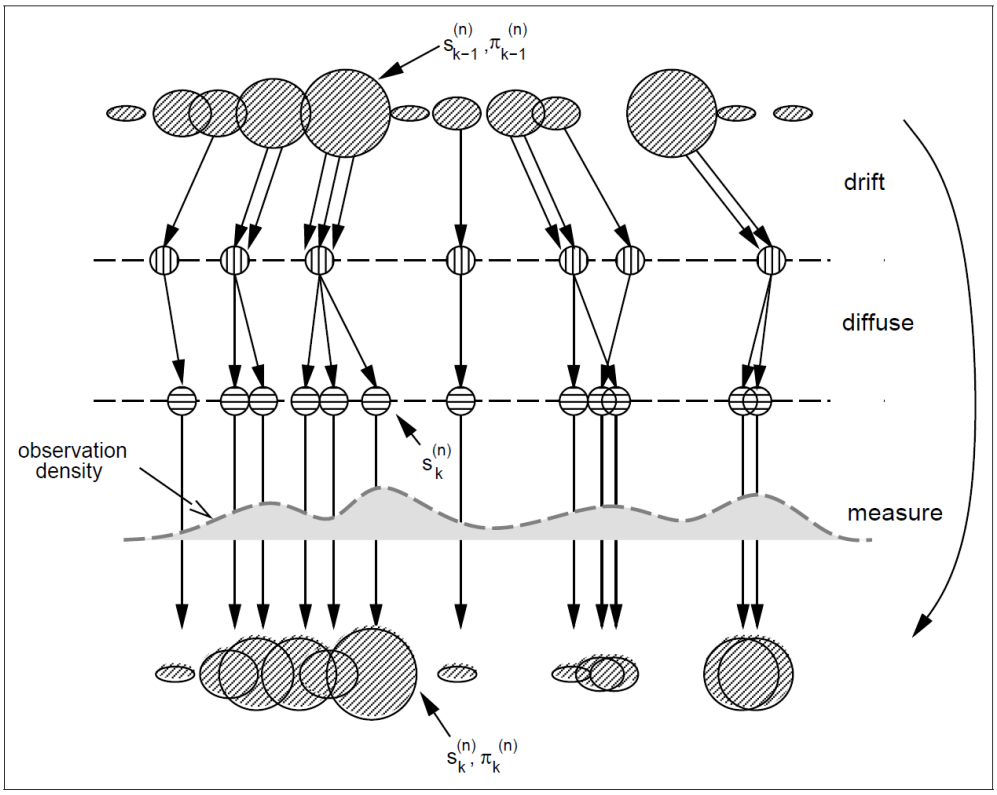


Fig. 2.10: One time step of the Condensation algorithm [4]. © [1998] Springer. Reprinted, with permission, from ref. [4].

Input: Sample set $\{(\mathbf{s}_{t-1}^{(n)}, \pi_{t-1}^{(n)}), n = 1, ..., N\}$
Compute the cumulative distribution values $c_{t-1}^{(k)} = \sum_{i=1}^{k} \pi_{t-1}^{(i)}$.
for $n = 1, \ldots, N$ **do**
  Drift: Sample $\mathbf{s}_t^{\prime(n)}$ from the cumulative distribution $c_{t-1}^{(k)}, k = 1, ..., N$.
  Diffuse: Sample $\mathbf{s}_t^{(n)}$ from the dynamical model $\mathbf{s}_t^{(n)} \sim p(\mathbf{x}_t | \mathbf{x}_{t-1} = \mathbf{s}_t^{\prime(n)})$.
  Measure and reweigh the sample $\mathbf{s}_t^{(n)}$ as $\pi_t^{(n)} = p(\mathbf{z}_t | \mathbf{x}_t = s_t^{(n)})$.
end for
Normalize $\pi_t^{(n)}$ such that $\sum_{n=1}^{N} \pi_t^{(n)} = 1$.
Output: Sample set $\{(\mathbf{s}_t^{(n)}, \pi_t^{(n)}), n = 1, ..., N\}$

Fig. 2.11: One time step of the Condensation algorithm [4].

where r, μ are constants, M is the number of points for curve discretization, $f(x;\mu) = \min(x^2, \mu^2)$, and $\mathbf{z}_1(s)$ is the closest image feature to $\mathbf{r}(s)$:

$$\mathbf{z}_i(s) = \mathbf{z}(s') \text{ where } s' = \underset{s' \in g^{-1}(s)}{\operatorname{argmin}} |\mathbf{r}(s) - \mathbf{z}(s')|$$

An example of a tracking result obtained using this model is shown in Figure 2.12.

2.3.3 Summary of the SMC framework

In Sequential Monte Carlo, the term "sequential" has two meanings:

1. Unfold a joint state $\mathbf{x} = (x_1, x_2, \cdots, x_d)$ into components, as in the Self-Avoiding Walks from Section 2.3.1.
2. Update X_t over time as in the particle filter from Section 2.3.2.

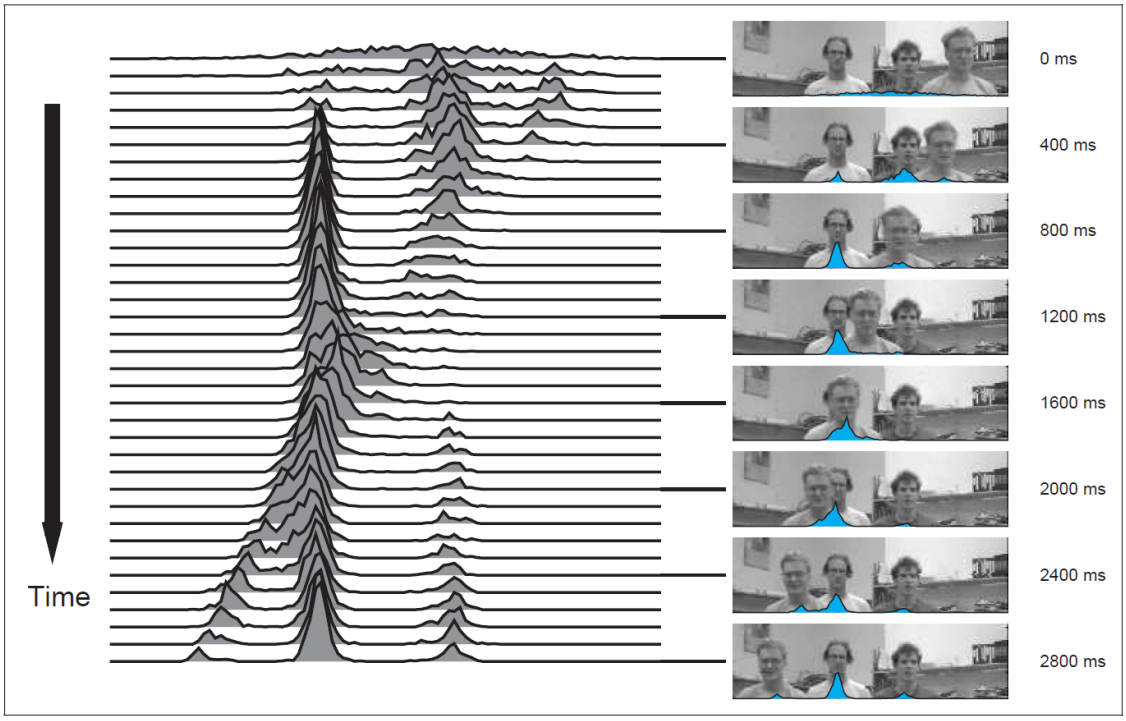


Fig. 2.12: 1D projection of the state density across multiple frames of a video, from [4]. © [1998] Springer. Reprinted, with permission, from ref. [4].

The following issues arise in the design of SMC/SIS:

1. Choice of the trial probability. For example, in particle filtering

$$p(\mathbf{x}_{t+1}|\mathcal{Z}_{t+1}) = \int_{\mathbf{x}_t} p(\mathbf{z}_{t+1}|\mathbf{x}_{t+1})p(\mathbf{x}_{t+1}|\mathbf{x}_t)p(\mathbf{x}_t|\mathcal{Z}_t)d\mathbf{x}_t$$

one may generate particles using

a) A data-driven approach by sampling from $p(\mathbf{z}_{t+1}|\mathbf{x}_{t+1})$ (tracking by detection). This is important when the target is lost.
b) A dynamics-driven approach by sampling from $p(\mathbf{x}_{t+1}|\mathbf{x}_t)p(\mathbf{x}_t|\mathcal{Z}_t)$ and reweighing according to the evidence $p(\mathbf{z}_{t+1}|\mathbf{x}_{t+1})$.

A better alternative is to generate particles using both the data-driven and the dynamics driven approaches as different channels, which can complete each other depending on the data quality at each time step.

2. How to rejuvenate the sample – punning, enriching, resampling/reweighing.

Example 2.3. In the Self-Avoiding Walk, suppose we have a partial sample $x^{(j)}$ of length n, with n large enough and with trial probability $g_1(x^{(j)}) = \frac{1}{k_1}\frac{1}{k_2}\dots\frac{1}{k_n}$, which is very small, contributing $w^{(j)} = k_1 \dots k_n$ to the final summation, which is very large. One idea is to make k copies of $x^{(j)}$, with each copy having weight $\frac{1}{k}w^{(j)}$. This is equivalent to changing the proposal probability $g(x)$ so that $g(x^{(j)})$ is k times larger.

Example 2.4. Similarly, in particle filtering, we could resample the weighted sample set $\hat{S} = \{(\hat{x}^{(j)}, w^{(j)}), j = 1, \dots, m\}$ with an equal-weight set $S = \{x^{(j)}, j = 1, \dots, m\}$ containing repeated copies of the strong samples, so that in the next step the strong samples will have many offspring.

In both examples performance is significantly improved by this resampling scheme.
Criterion for resampling. In SMC we can monitor the sample $S = \{x^{(j)}, w^{(j)}), j = 1, \dots, m\}$ by the variance of the weight vector $\mathbf{w} = (w^{(1)}, \dots, w^{(m)})$, or the coefficient of variation

$$CV(\mathbf{w}) = \sqrt{\frac{\sum_{j=1}^{m}(w^{(j)} - \overline{w})^2}{(m-1)\overline{w}^2}}.$$

When $CV(\mathbf{w})$ is too large, $CV(\mathbf{w}) > c_0$, a resampling step is necessary.

Reweighing. When resampling $S = \{(x^{(j)}, w^{(j)}), j = 1, ..., m\}$, we may not always have to use the weight vector $\mathbf{w} = (w^{(1)}, \ldots, w^{(m)})$ to generate weights proportionally. Instead, we could choose an arbitrary vector with nonzero entries $\mathbf{a} = (a^{(1)}, \ldots, a^{(m)}), a_i > 0$ and reweight the samples as $w^{*(j)} = w^{(j)}/a^{(j)}$. The entries of $\mathbf{a}$ should be designed to penalize redundant samples and to encourage distinctive samples.

2.4 Application: Ray Tracing by SMC

Another application of SMC is ray tracing [10], which is used for calculating the radiance of a surface given the description of the light sources acting on it.

Given the incident radiance function, $L_i(\mathbf{x}, \omega_i)$, at point $\mathbf{x}$, the reflected radiance follows the *reflectance equation*:

$$L_r(\mathbf{x}, \omega_r) = \int_{S^2} f_r(\mathbf{x}, \omega_i \leftrightarrow \omega_r) L_i(x, \omega_i) |\cos(\theta_i)| d\sigma(\omega_i), \tag{2.13}$$

where f_r is the bidirectional reflectance distribution function (BRDF), S^2 is the unit sphere in 3D, σ is the solid angle measure, θ_i is the angle between ω_i and the surface normal at $\mathbf{x}$.

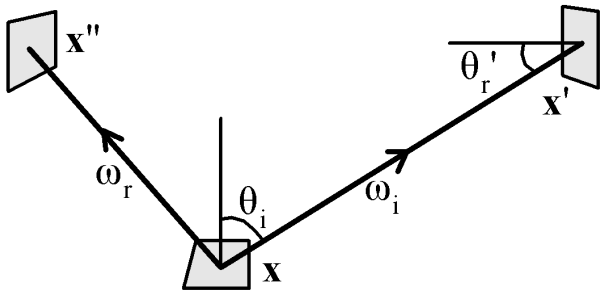


Fig. 2.13: Illustration of the reflectance equation. © [1995] ACM. Reprinted, with permission, from ref. [10].

If we want to use only points in the scene, we could also write the reflectance equation as follows:

$$L_r(\mathbf{x} \to \mathbf{x}") = \int_{\mathscr{M}} f_r(\mathbf{x}' \leftrightarrow \mathbf{x} \leftrightarrow \mathbf{x}'') L_i(\mathbf{x}' \to \mathbf{x}) G(\mathbf{x} \leftrightarrow \mathbf{x}') dA(\mathbf{x}'), \tag{2.14}$$

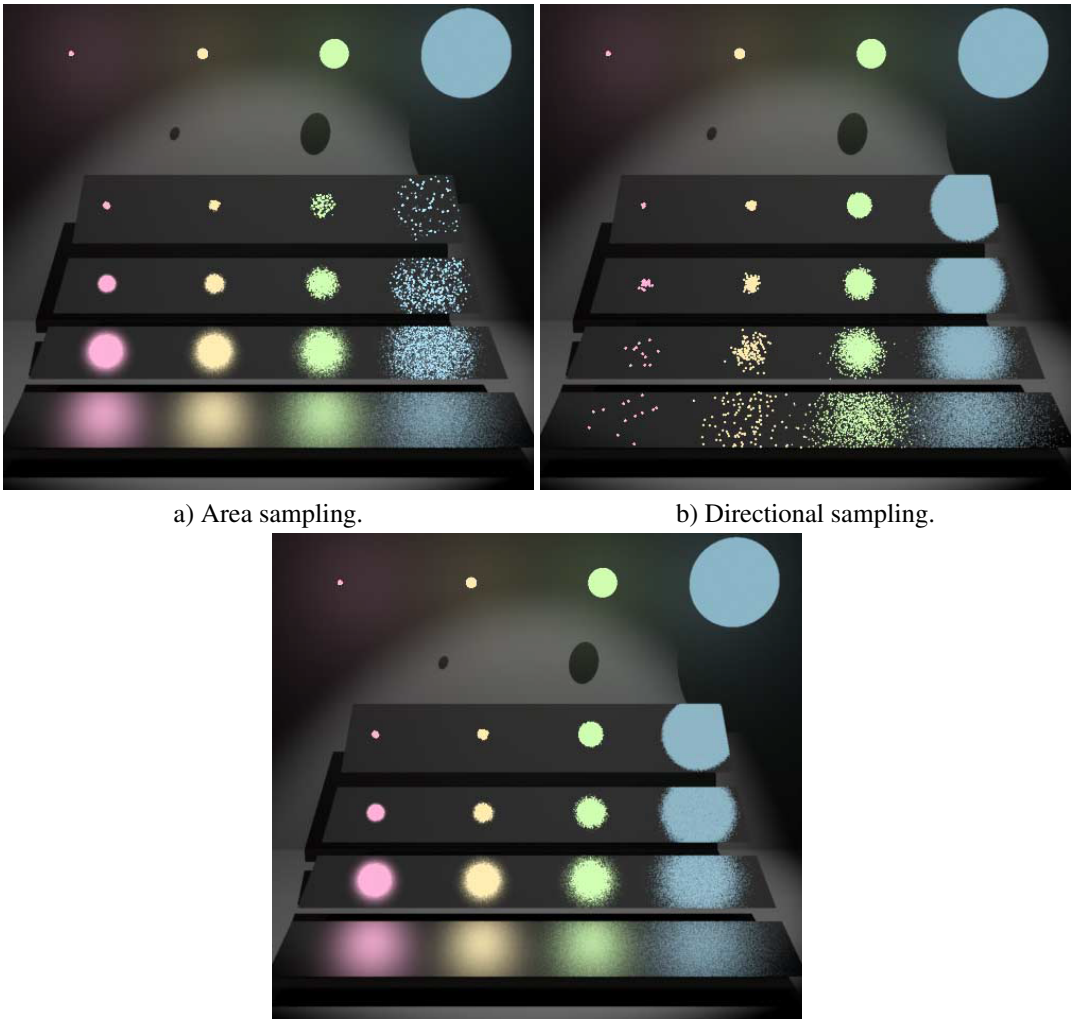
where $G(\mathbf{x} \leftrightarrow \mathbf{x}') = V(\mathbf{x} \leftrightarrow \mathbf{x}') \dfrac{\cos(\theta_r')\cos(\theta_i)}{\|\mathbf{x} - \mathbf{x}'\|2}$ and A is the measure of surface area, θ_r' and θ_i are the angles between $\mathbf{x} \leftrightarrow \mathbf{x}'$ and the surface normals at $\mathbf{x}$ and $\mathbf{x}'$, as illustrated in Figure 2.13. The function $V(\mathbf{x} \leftrightarrow \mathbf{x}')$ is 1 if $\mathbf{x}$ and $\mathbf{x}'$ are mutually visible and 0 else.

We arrive at the global illumination problem of finding the equilibrium radiance distribution L that satisfies:

$$L(\mathbf{x} \to \mathbf{x}") = L_e(\mathbf{x} \to \mathbf{x}") + \int_{\mathscr{M}} f_r(\mathbf{x}' \leftrightarrow \mathbf{x} \leftrightarrow \mathbf{x}'') L(\mathbf{x}' \to \mathbf{x}) G(\mathbf{x} \leftrightarrow \mathbf{x}') dA(\mathbf{x}'),$$

where the emitted radiance distribution L_e is given. This is the three point rendering equation [5]. It can be written concisely as $L = L_e + \mathcal{T}L$ where $\mathcal{T}$ is the light transport operator. Under weak assumptions, the solution is the Neumann series:

$$L = \sum_{i=1}^{\infty} \mathcal{T}^i L_e.$$



a) Area sampling. b) Directional sampling.

c) Weighted combination of the samples from (a) and (b)

Fig. 2.14: Sampling of glossy highlights from area light sources. © [1995] ACM. Reprinted, with permission, from ref. [10].

2.4.1 Example: glossy highlights

Consider the ray tracing problem of rendering the highlight produced by an area light source S on a nearby glossy surface, as illustrated in Figure 2.14. There are two obvious strategies to use Monte Carlo methods to approximate the reflected radiance, using eq. (2.13) and (2.14) respectively. For both strategies we use importance sampling, where the samples $x_1, ..., x_n$ are obtained from a distribution $p(x)$. Then the integral is approximated as:

$$\int_\Omega f(x)d\mu(x) \approx \frac{1}{n}\sum_{i=1}^{n}\frac{f(x_i)}{p(x_i)}.$$

With *area sampling* we randomly choose points on the surface to approximate (2.14). The points could, for example, be chosen uniformly on S with respect to the surface area or the emitted power.

With *directional sampling* we approximate (2.13) using random samples of the direction ω_i. Usually $p(\omega_i)d\sigma(\omega_i)$ is chosen proportional to $f_r(\mathbf{x}\omega_i \leftrightarrow \omega_r)$ or to $f_r(\mathbf{x}\omega_i \leftrightarrow \omega_r)|\cos(\theta_i)|$.

Figure 2.14 shows examples of renderings using different sampling methods. The scene contains four spherical light sources of different radii and color, and a spotlight overhead. All spherical light sources emit the same total power. There are also four shiny rectangular plates of varying surface roughness, tilted so that the reflected light sources are visible. Given a viewing ray that strikes a glossy surface, images (a), (b), (c) use different techniques for the highlight calculation. All images are 500 by 450 pixels. The MC techniques are:

(a) Area sampling. A sample direction ω_i is chosen uniformly (with respect to a solid angle) within the cone of directions of each light source, using 4 samples per pixel.
(b) Directional sampling. The direction ω_i is chosen with probability proportional to the BRDF $f_r(\mathbf{x}\omega_i \leftrightarrow \omega_r)d\sigma(\omega_i)$, with 4 samples per pixel.
(c) A weighted combination of the samples from (a) and (b) is computed, using the power heuristic with $\beta = 2$.

The glossy BRDF is a symmetric, energy-conserving variation of the Phong model. The Phong exponent is $n = 1/r - 1$,where $r \in (0,1)$ is a surface roughness parameter. The glossy surfaces also have a small diffusion component.

2.5 Preserving Sample Diversity in Importance Sampling

For simplicity of notation, we denote an arbitrary distribution in space Ω by $p(\mathbf{x})$. For image segmentation problems with Bayesian Inference, we observe that $p(\mathbf{x})$ has two important properties.

1. $p(\mathbf{x})$ has an enormous number of local maxima (called modes in statistics). A significant mode corresponds to a distinct interpretation of the image, and the cloud surrounding a mode contains local, small perturbations of the region boundaries or model parameters. These significant modes of $p(\mathbf{x})$, denoted by $\mathbf{x}_i, i = 1,2,...$, are well separated from each other due to the high dimensions.
2. Each mode $\mathbf{x}_i$ has a weight $\omega_i = p(\mathbf{x}_i)$, and its energy is defined as $E(\mathbf{x}_i) = -\log p(\mathbf{x}_i)$. The energies of these modes are uniformly distributed in a broad range $[E_{\min}, E_{\max}]$, say, $[1000, 10,000]$. For example, it is normal to have solutions (or local maxima) whose energy difference is of order 500 or more. Thus, their probability (weights) differ on the order of $\exp^{-500}$, and our perception is interested in those "trivial" local modes!

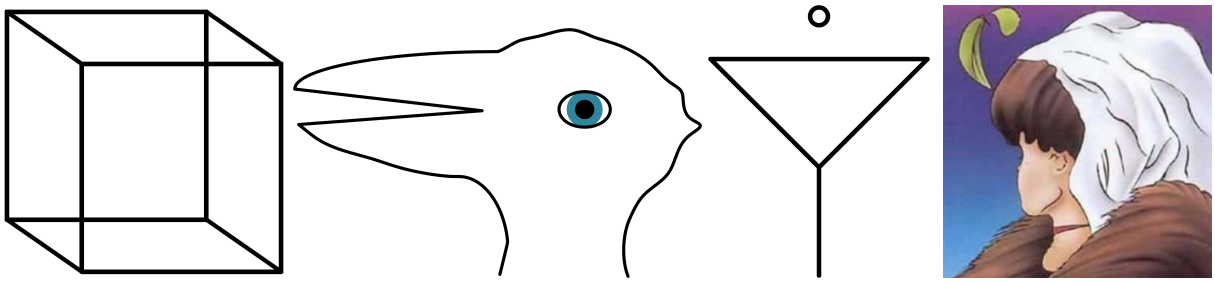


Fig. 2.15: Different images that have multiple interpretations. From left to right: the Necker cube, the duck/rabbit illusion [6], the martini/bikini ambiguity, old woman vs young lady.

Preserving sample diversity is an important issue for maintaining the modes of a probability distribution. Maintaining the modes of the probability are important for example to maintain the ambiguity of the interpretation of the images from Figure 2.15.

Intuitively, it helps to imagine that $p(\mathbf{x})$ in Ω is distributed like the mass of the universe. Each star is a local maximum of the mass density. The significant and developed stars are well separated from each other and their masses

could differ in many orders of magnitudes. This metaphor leads us to a Gaussian mixture representation of $p(\mathbf{x})$. For a large enough N we have,

$$p(\mathbf{x}) = \frac{1}{\omega}\sum_{j=1}^{N}\omega_j G(\mathbf{x}-\mathbf{x}_j,\sigma_j^2), \quad \omega = \sum_{j=1}^{N}\omega_j.$$

We denote by

$$S_o = \{(\omega_j,\mathbf{x}_j),\ j=1,2,...,N\},$$

the set of weighted particles (or modes). Thus, our task is to select a set S of $K << N$ particles from S_o. We define a mapping from the index in S to the index in S_o,

$$\tau : \{1,2,...,K\} \longrightarrow \{1,2,....,N\}.$$

Therefore,

$$S = \{(\omega_{\tau(i)},\mathbf{x}_{\tau(i)});\ i=1,2,...,K\}$$

encodes a non-parametric model for approximating $p(\mathbf{x})$ by

$$\hat{p}(\mathbf{x}) = \frac{1}{\alpha}\sum_{i=1}^{K}\omega_{\tau(i)}G(\mathbf{x}-\mathbf{x}_{\tau(i)},\sigma_{\tau(i)}^2), \quad \alpha = \sum_{i=1}^{K}\omega_{\tau(i)}.$$

Our goal is to compute

$$S^* = \arg\min_{|S|=K} D(p||\hat{p}).$$

For notational simplicity, we assume all Gaussians have the same variance in approximating $p(\mathbf{x})$, $\sigma_j = \sigma$, and $j = 1,2,...,N$. Following our analogy, all "stars" have the same volume, but differ in weight. With the two properties of $p(\mathbf{x})$, we can approximately compute $D(p||\hat{p})$ as follows. We start with an observation for the KL-divergence for Gaussian distributions.

Let $p_1(x) = G(\mathbf{x}-\mu_1;\sigma^2)$ and $p_2(x) = G(\mathbf{x}-\mu_2;\sigma^2)$ be two Gaussian distributions, then it is easy to check that

$$D(p_1||p_2) = \frac{(\mu_1-\mu_2)^2}{2\sigma^2}.$$

We partition the solution space Ω into disjoint domains

$$\Omega = \cup_{i=1}^{N} D_i, \quad D_i \cap D_j = \emptyset,\ \forall i \neq j.$$

D_i is the domain where $p(\mathbf{x})$ is decided by a particle $(\omega_i,\mathbf{x}_i)$, $i \in \{1,...,N\}$. The reason for this partition is that the particles in S are far apart from each other in high dimensional space and their energies vary significantly based on the two properties described at the beginning of this section. Within each domain, it is reasonable to assume that $p(\mathbf{x})$ is dominated by one term in the mixture and the other $N-1$ terms are neglectable.

$$p(\mathbf{x}) \approx \frac{\omega_i}{\omega}G(\mathbf{x}-\mathbf{x}_i;\sigma^2), \quad \mathbf{x} \in D_i,\ i=1,2,...,N.$$

The size of D_i is much larger than σ^2. After removing $N-K$ particles in the space, it is dominated by a nearby particle that is selected in S.

We define a second mapping function

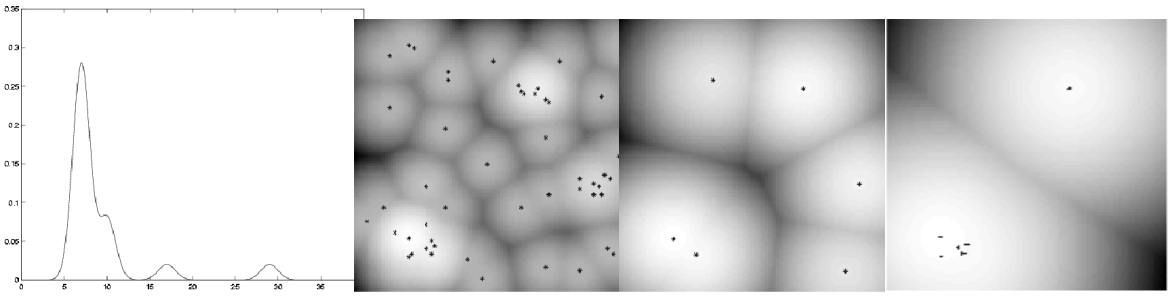
$$c : \{1,2,...,N\} \rightarrow \{1,2,...,K\},$$

so that $\hat{p}(\mathbf{x})$ in D_i is dominated by a particle $\mathbf{x}_{\tau(c(i))} \in S_K$, and

$$\hat{p}(\mathbf{x}) \approx \frac{\omega_{c(i)}}{\alpha}G(\mathbf{x}-\mathbf{x}_{\tau(c(i))};\sigma^2), \quad \mathbf{x} \in D_i, i=1,2,...,N.$$

Intuitively, the N domains are partitioned into K groups, each of which is dominated by one particle in S_K. Thus we can approximate $D(p||\hat{p})$.

$$
\begin{aligned}
D(p||\hat{p}) &= \sum_{n=1}^{N} \int_{D_n} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{\hat{p}(\mathbf{x})} d\mathbf{x} \\
&= \sum_{n=1}^{N} \int_{D_n} \frac{1}{\omega} \sum_{i=1}^{N} \omega_i G(\mathbf{x}-\mathbf{x}_i;\sigma^2) \log \frac{\frac{1}{\omega}\sum_{i=1}^{N} \omega_i G(\mathbf{x}-\mathbf{x}_i;\sigma^2)}{\frac{1}{\alpha}\sum_{j=1}^{k} \omega_{\tau(j)} G(\mathbf{x}-\mu_{\tau(j)};\sigma^2)} d\mathbf{x} \\
&\approx \sum_{n=1}^{N} \int_{D_n} \frac{\omega_n}{\omega} G(\mathbf{x}-\mathbf{x}_n;\sigma^2) [\log \frac{\alpha}{\omega} + \log \frac{\omega_n G(\mathbf{x}-\mathbf{x}_n;\sigma^2)}{\omega_{\tau(c(n))} G(\mathbf{x}-\mathbf{x}_{\tau(c(n))};\sigma^2)}] d\mathbf{x} \\
&= \sum_{n=1}^{N} \frac{\omega_n}{\omega} [\log \frac{\alpha}{\omega} + \log \frac{\omega_n}{\omega_{\tau(c(n))}} + \frac{(\mathbf{x}_n - \mathbf{x}_{\tau(c(n))})^2}{2\sigma^2}] \\
&= \log \frac{\alpha}{\omega} + \sum_{n=1}^{N} \frac{\omega_n}{\omega} [(E(\mathbf{x}_{\tau(c(n))}) - E(\mathbf{x}_n)) + \frac{(\mathbf{x}_n - \mathbf{x}_{\tau(c(n))})^2}{2\sigma^2}] = \hat{D}(p||\hat{p}).
\end{aligned}
\tag{2.15}
$$



(a) (b) (c) (d)

Fig. 2.16: (a) A 1D distribution $p(\mathbf{x})$ with four particles $\mathbf{x}_i, i = 1,2,3,4$. (b) A 2D distribution $p(\mathbf{x})$ with 50 particles, we show $\log p(\mathbf{x})$ in the image for visualization. (c) $\hat{p}_1(x)$ with 6 particles that minimizes $D(p||\hat{p})$. (d) $\hat{p}_2(x)$ with 6 particles that minimizes $|p-\hat{p}|$. © [2002] IEEE. Reprinted, with permission, from ref. [9].

Equation (2.15) has an intuitive meaning. The second term suggests that each selected $\mathbf{x}_{\tau(c(i))}$ should have large weight $\omega_{\tau(c(i))}$. The third term contains the attraction forces from particles in S_o to particles in S. Thus this term helps to pull apart the particles in S_k and also plays the role of encouraging the selection of particles with large weights like the second term.

To demonstrate the goodness of the approximation $\hat{D}(p||\hat{p})$ to $D(p||\hat{p})$, we show two experiments.

Figure 2.16.(a) displays a 1D distribution $p(\mathbf{x})$ which is a mixture of $N = 4$ Gaussians (particles). We index the centers from left to right $\mathbf{x}_1 < \mathbf{x}_2 < \mathbf{x}_3 < \mathbf{x}_4$. Suppose we want to choose $K = 3$ particles for S_K and $\hat{p}(\mathbf{x})$.

S_3:	$\{x_1,x_2,x_3\}$	$\{x_1,x_2,x_4\}$	$\{x_1,x_3,x_4\}$	$\{x_2,x_3,x_4\}$
$D(p\|\|\hat{p})$:	3.5487	1.1029	0.5373	2.9430
$\hat{D}(p\|\|\hat{p})$:	3.5487	1.1044	0.4263	2.8230
$\|p-\hat{p}\|$:	0.1000	0.1000	0.3500	1.2482

Table 2.1: Distances between $p(x)$ and $\hat{p}(x)$ for different particle set S_3 for the 1D distribution in Figure 2.16, left.

Table 2.1 displays the distances between $p(\mathbf{x})$ and $\hat{p}(\mathbf{x})$ over the four possible combinations. The second row shows the KL-divergence $D(p||\hat{p})$ and the third row is the estimate $\hat{D}(p||\hat{p})$. The approximation is very accurate given the particles are well separated.

Both measures choose (x_1,x_3,x_4) as the best S. Particle x_2 is not favored by the KL-divergence because it is near x_1, although it has much higher weight than x_3 and x_4. The fourth row shows the absolute value of the difference between

$p(x)$ and $\hat{p}(x)$. This distance favors (x_1, x_2, x_3) and (x_1, x_2, x_4). In comparison, the KL-divergence favors particles that are apart from each other and picks up significant peaks in the tails.

This idea is better demonstrated in Figure 2.16. Figure 2.16.(b) shows $\log p(\mathbf{x}) = -E(\mathbf{x})$ which is renormalized for display purposes. $p(\mathbf{x})$ consists of $N = 50$ particles whose centers are shown by the black spots. The energies $E(\mathbf{x}_i), i = 1, 2, ..., N$ are uniformly distributed in an interval $[0, 100]$. Thus their weights differ in exponential order. Figure 2.16(c) shows $\log \hat{p}(\mathbf{x})$ with $k = 6$ particles that minimize both $D(p||\hat{p})$ and $\hat{D}(p||\hat{p})$. Figure 2.16(d) shows the 6 particles that minimize the absolute difference $|p - \hat{p}|$. It is clear that Figure 2.16.(c) has more dispersed particles.

2.5.1 Parzen window discussion

A similar method for estimating the density, $p(\mathbf{x})$, using n samples from SMC is the Parzen window. For this approach we assume that the distribution in the region surrounding $\mathbf{x}$ takes a particular form. For example, assume that the "window function", $\phi(\mathbf{x})$, which bares resemblance to a kernel function, is a d-dimensional hypercube. This function is an indicator that returns a value of 1 if the sample falls within the unit hypercube centered at the origin and 0 otherwise. The number of samples, S_n, that fall into the cube with side length ℓ around $\mathbf{x}_i$ is then given by

$$S_n = \sum_{i=1}^{n} \phi\left(\frac{\mathbf{x} - \mathbf{x}_i}{\ell}\right).$$

To turn this count into an estimate of the distribution at $\mathbf{x}$ we simply take the average and scale for the volume V of the window,

$$p_n(\mathbf{x}) = \frac{1}{nV} S_n.$$

This strategy estimates the distribution at $\mathbf{x}$ by allowing the influence of other samples to grow with proximity to $\mathbf{x}$.

This is the general framework for Parzen window, where in reality we can choose any non-negative window function $\phi(\mathbf{x})$ that integrates to 1. Furthermore, we can choose different functions for different sample points. This means that the distribution can be estimated as a linear combination of window functions. Depending on the chosen window size, the distribution will be more or less in focus.

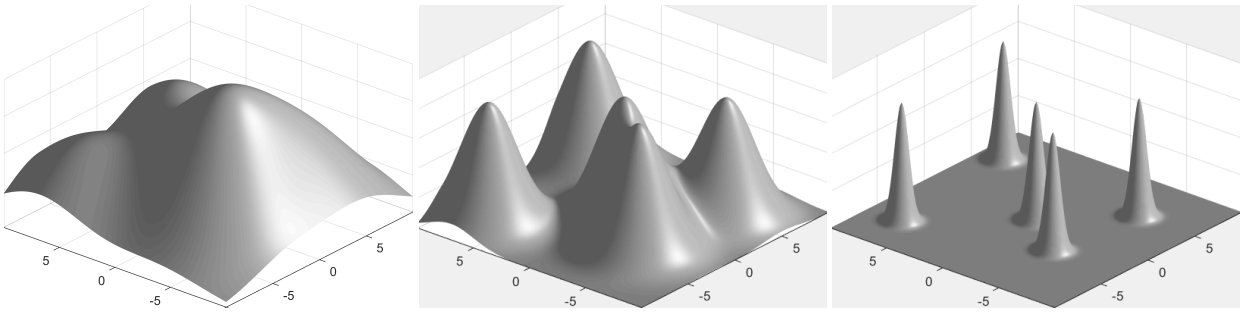


Fig. 2.17: Parzen Window estimates using 3 different window sizes. The window size decreases from left to right.

Figure 2.17 shows a series of plots of the same data using different sizes of a Gaussian window function. We see that the larger the window is the more smooth the probability function appears. For small windows the landscape appears to be a set of separated spikes.

To understand this, consider zooming in with a camera on a mountainous landscape. While zoomed out, corresponding to a smaller ℓ, the mountains are entirely visible, but appear as small, sharp peaks in comparison to the other features (lakes, fields, etc.) in the rest of the image. On the other hand, as the camera is zoomed in on the mountains, they begin to take up the entire frame and a smaller and smaller portion of the whole range is visible. The specific

details of the ridges between peaks may come into focus, but the overall landscape is harder to understand. Effectively using this technique requires finding a balance between these two ideas.

To blend this idea with SMC, we recall that our target probability $p(\mathbf{x})$ is approximated by a distribution $g(\mathbf{x})$. The samples are collected from g and each one is assigned a weight according to $\omega(\mathbf{x}_i) = \frac{p(\mathbf{x}_i)}{g(\mathbf{x}_i)}$. For each sample $\mathbf{x}_i$, we have a specific window function, say a normal distribution with mean $\mathbf{x}_i$ and variance v_i. The Parzen Window estimate of $p(\mathbf{x})$ is then given by

$$p_n(\mathbf{x}) = \sum_{i=1}^{n} w(\mathbf{x}_i) N(\mathbf{x} - \mathbf{x}_i, v_i).$$

The mean of the general Parzen Window estimate $\bar{p}_n$ above can be calculated as

$$\bar{p_n}(\mathbf{x}) = E[p_n(\mathbf{x})] = \frac{1}{n} E[\frac{1}{V} S_n] = \frac{1}{n} E[\sum_{i=1}^{n} \frac{1}{V} \phi(\frac{\mathbf{x}-\mathbf{x}_i}{\ell})] \xrightarrow{n\to\infty} \int \frac{1}{V} \phi(\frac{\mathbf{x}-\mathbf{z}}{\ell}) p(\mathbf{z}) d\mathbf{z} \tag{2.16}$$

As discussed previously, but now shown mathematically, the mean appears as a convolution of the target distribution with the window function. Asymptotically, if it is required that $\lim_{n\to\infty} V = 0$, then the mean will approach the true value. For the variance, since $p(\mathbf{x})$ is the sum of independent variables we simply sum the variances of each variable and derive

$$\begin{aligned} n\sigma_n^2(\mathbf{x}) &= n \sum_{i=1}^{n} E[(\frac{1}{nV} \phi(\frac{\mathbf{x}-\mathbf{x}_i}{\ell}) - \frac{1}{n} \bar{p}_n(\mathbf{x}))^2] \\ &= n \sum_{i=1}^{n} E[\frac{1}{n^2 V^2} \phi^2(\frac{\mathbf{x}-\mathbf{x}_i}{\ell})] - \bar{p}_n^2(\mathbf{x}) \\ &\xrightarrow{n\to\infty} \frac{1}{V} \int \frac{1}{V} \phi^2(\frac{\mathbf{x}-\mathbf{z}}{\ell}) p(\mathbf{z}) d\mathbf{z} - \left(\int \frac{1}{V} \phi(\frac{\mathbf{x}-\mathbf{z}}{\ell}) p(\mathbf{z}) d\mathbf{z} \right)^2 \end{aligned} \tag{2.17}$$

This result agrees with the mountain analogy presented previously. The variance is highly related to the volume of the window V. For a large volume, or equivalently a large value of ℓ, the window function at each point is smoothed and the variance of the resulting distribution is reduced.

2.6 Monte Carlo Tree Search

Monte Carlo tree search is a stochastic procedure for decision making in Markov decision processes, which works by searching ahead multiple scenarios and using them to accumulate support for the most promising immediate actions. **Markov decision processes (MDP)** are used in reinforcement learning where an agent performs actions in the environment, and from time to time receives rewards based on the actions performed. In a MDP with current state s, the agent performs an action a and the environment reaches a new state s' that depends only on the current state s and the action a performed. At the same time, the agent receives a reward $R(s')$. Denoting by Ω the space of all states and by A the space of possible actions, the agent performs action $a \in A$ when the system is in state $s \in \Omega$, then the system reaches a new state s' that is a sample from a distribution $p(s'|a,s)$.

Examples of MDPs are games such as backgammon, chess and Go, or robots exploring their environment. Pole balancing is another example where the reward is negative each time the pole has fallen and is zero otherwise.

The goal is to learn what action to take in each state so that the expected reward is maximized. For that, one wants to learn a *policy* $\pi : \Omega \to A$, where $\pi(s)$ represents for a state s the best action to take to maximize the expected reward. In some cases one uses a nondeterministic policy $\pi : A \times \Omega \to \mathbb{R}$ where $\pi(a|s)$ represents the probability of taking action a in state s.

The expected reward starting from each state s based on the current policy π is represented by a *state-value function* $v_\pi : \Omega \to \mathbb{R}$. One could also consider the *action-value function* $q_\pi : \Omega \times A \to \mathbb{R}$, where $q(s,a)$ represents the total expected reward when taking action a in state s.

In special cases where the action-state spaces $\Omega \times A$ are finite and not very large, the state-value and action-value functions can be estimated as the agent explores the environment, and through some dynamic-programming types of algorithms, better and better policies can be learned. In most applications however, the state space is so large that it is practically impossible to memorize the value function $v_\pi(s)$ or the action value function $q_\pi(s,a)$. In these cases approximate methods are used.

Monte Carlo Tree Search is such an approximate method that estimates the immediate action-value function $q_\pi(s_t,a)$ starting from the current state s_t by exploring different actions and obtaining support for each based on the rewards obtained along the way. Observe that a separate MCTS is run for each state s_t that is explored, as illustrated in Figure 2.18.

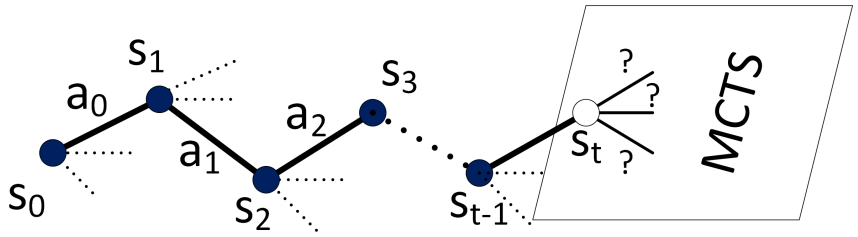


Fig. 2.18: Monte Carlo Tree Search is used for deciding on the best action to take in the current state s_t by estimating the expected reward of each of the possible immediate actions.

Different variants of Monte Carlo Tree Search have been used successfully in many applications, including Feature Selection [3], where MCTS obtained state of the art results on a number of datasets from the NIPS 2003 Feature Selection Challenge. Another application is in solving Quantified Constrained Satisfaction Problems (QCSP), where a modified MCTS described in [1] outperformed state of the art $\alpha - \beta$ search algorithms for large scale problems.

2.6.1 Pure Monte Carlo tree search

At the current state s_t, the Monte Carlo Tree Search (MCTS) procedure is used to build support for the most promising actions and decide on the best possible action to take. For that a tree is grown with the current state as a root and with the most promising actions as the immediate branches. The tree is build iteratively, at each MCTS iteration one leaf being added to the tree and a *tree policy* is updated.

The tree policy is used to navigate the tree until a leaf node is reached. The tree policy balances exploration of new tree branches with exploitation of the already existing tree branches. There are many possible tree policies that could be used, but a popular one is the *Upper Confidence Bound for Trees* (UCT), which will be described towards the end of this section. From the leaf node on, a *default policy* is used to play the game until the end of the episode (e.g. when a win/loose state is reached.)

The process that happens at each MCTS iteration can be broken down into four parts, illustrated in Figure 2.19.

a) Selection. In this phase the tree policy is played by random sampling the most promising states based on their learned winning chances, until a node L with unvisited children is reached.
b) Expansion. Unless the game ends at the node L, the node is expanded with one or more children (based on the possible actions) and one child C is chosen at random.
c) Simulation. The game is played using the default policy starting from state C until an outcome is reached (e.g. the game ends).
d) Backpropagation. The outcome is used to update the winning counts (hence the tree policy) on the path from the leaf C to the root.

These four steps are run for each iteration of the MCTS, and the number of iterations depends on the computational budget. When the computational budget is reached, the search terminates and the most up to date tree policy is used to decide on the best action to take at the root node s_t. After the action has been taken, the new state is s_{t+1} and the

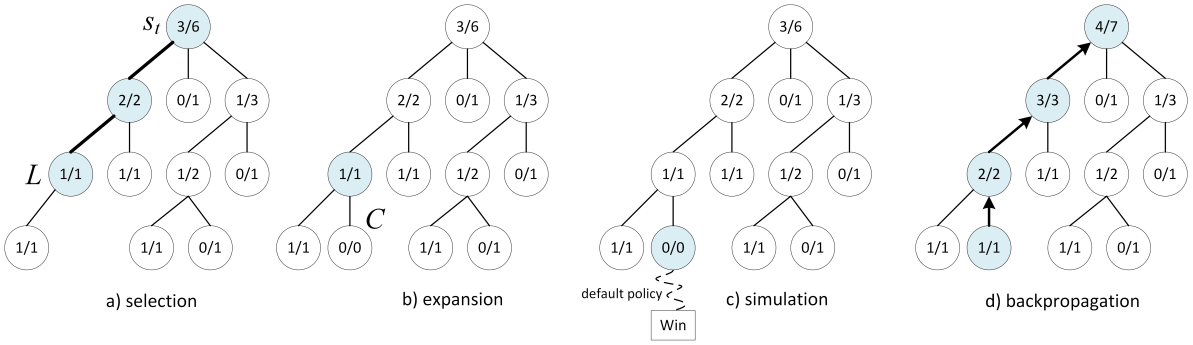


Fig. 2.19: Illustration of the four parts of a Monte Carlo tree search iteration.

MCTS is run again with s_{t+1} as the root node. The tree grown with s_t as a root node can be discarded, or better yet, the subtree with root s_{t+1} can be reused for the new MCTS.

Observe that at each MCTS iteration, at least one node is added to the tree and the total count at the root is incremented by 1. This way after many iterations, the total counts at the children of the root node are high, therefore better and better accuracy is obtained in estimating the expected reward for each of these children.

For each tree node v two values need to be maintained: the number $N(v)$ of times the node has been visited and the total reward $Q(v)$ for the playouts that passed through the node. Then the ratio $Q(v)/N(v)$ is an approximation of the expected reward for the playouts passing through node v.

The values $N(v)$ and $Q(v)$ are used to define the tree policy, which therefore changes as the tree is grown. One of the most popular tree policies is the *Upper Confidence Bound for Trees* (UCT). It aims to balance the exploration of unvisited children with the exploitation (revisiting) of the already visited children.

According to the UCT, for each tree node v, a child node j is selected to maximize:

$$UCT(j) = \frac{Q(j)}{N(j)} + c\sqrt{\frac{2\ln N(v)}{N(j)}} \tag{2.18}$$

where $c > 0$ is a constant. Observe that if $N(j) = 0$ then $UCT(j) = \infty$, so all unvisited children must be visited before the visited children nodes are explored any further. Therefore UCT is a breadth first search policy.

When the computational budget is reached, the best action to take could be chosen according to one of the following criteria, described in [7]:

1. Max child. The child with the highest estimated reward is selected.
2. Robust child. The most visited child is selected.
3. Max-Robust child. Select the child which is simultaneously most visited and highest reward. If no such child exists, MCTS is continued until a minimum visit count it achieved for the highest reward child.
4. Secure child. Select the child that maximizes a lower confidence bound.

The MCTS has many variants including many tree policies, learning strategies, etc. A comprehensive survey of the MCTS methods and applications can be found in [2].

2.6.2 AlphaGo

AlphaGo [8] is a modification of MCTS, adapted for playing the game of Go. Go is a more challenging game than chess because the space of all possible games is on the order of $250^{150} \approx 2^{1200}$ (about 250 possible moves for each configuration and a total game length of about 150), while for chess is on the order of $35^{80} \approx 2^{410}$. The sheer size of the search space makes it unfeasible to exhaustively search for the best move. Besides the large size of the search space, a possibly even bigger challenge is the difficulty in finding a good state value function that can evaluate the wining chance of any position.

A direct application of MCTS is not appropriate because of the breadth first search property of the pure MCTS and the large number of possible moves from each position. For these reasons, the authors have adopted learning based strategies for reducing both the breadth of the game (the space of possible moves from the current configuration) and the depth of the evaluation of each move.

The breadth of the game is reduced by using a tree policy that estimates the most promising position a for the next stone given the current configuration s. The depth of the evaluation of each possible move is also reduced, by using a learned value function $V(s)$ that evaluates the chance of winning from the state s.

To describe the policy and value function, we first need to describe three learned policy networks $p_\sigma(a|s), p_\rho(a|s)$, $p_\pi(a|s)$ and a value network $v_\theta(s)$ that are used in the AlphaGo MCTS policy and value functions.

- First, a policy network $p_\sigma(a|s)$ is learned in a supervised way from 160,000 games and about 30 million board moves. The model is a 13-layer CNN (Convolutional Neural Network) based on a number of features extracted from the current configuration s.
- The policy network is further improved by adding more training data obtained by self-playing the current policy network against an older version. This way an improved policy network $p_\rho(a|s)$ is obtained.
- A default (rollout) policy $p_\pi(a|s)$ is also trained as a linear model on an extended set of features. The rollout policy is about 1000 times faster than the policy network and is used for the simulation step of the Monte Carlo Tree search.
- A value network $v_\theta(s)$ is also trained as a CNN using the same features as the policy network $p_\sigma(a|s)$, plus one more feature indicating the current player color. To avoid overfitting, the training data consists of 30 million configurations obtained by self playing from separate games. The value network $v_\theta(s)$ obtains a much better position evaluation accuracy than Monte Carlo roll-outs using the rollout policy $p_\pi(a|s)$, and is comparable to Mote Carlo roll-outs using the policy network $p_\rho(a|s)$, while being 15,000 times faster.

For MCTS, with each edge (s,a) of the tree are stored the visit count $N(s,a)$ over the MCTS simulations, the action value $Q(s,a)$ and the prior probability $P(s,a) = p_\sigma(a|s)$. Then, the MCTS proceeds as follows:

a) Selection. The game is played by selecting the action

$$a_t = \underset{a}{\operatorname{argmax}}[Q(s,a) + u(s,a)] \tag{2.19}$$

at configuration s until a leaf node s_L is reached, where $u(s,a) \propto P(s,a)/(1+N(s,a))$.

b) Expansion. Unless the game ends at the leaf node s_L, the leaf node may be expanded with the legal moves a and the prior probabilities $P(s_L,a) = p_\sigma(a|s_L)$ are computed and stored.

c) Simulation. The game is played by Monte Carlo roll-outs using the fast policy $p_\pi(a|s)$ starting from state s_L until the game ends, obtaining outcome z_L. Then the value of s_L is computed as:

$$V(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_L \tag{2.20}$$

d) Backpropagation. The game outcome z_L is used to update the visit counts $N(s,a)$ and action values $Q(s,a)$ on the path from the leaf s_L to the root as described in Eq. (2.21) below.

After n MCTS simulations, the visit counts and the action values are:

$$\begin{aligned} N(s,a) &= \sum_{i=1}^{n} \mathbf{1}(s,a,i), \\ Q(s,a) &= \frac{1}{N(s,a)} \sum_{i=1}^{n} \mathbf{1}(s,a,i) V(s_L^i) \end{aligned} \tag{2.21}$$

where s_L^i is the leaf node reached in simulation i and $\mathbf{1}(s,a,i)$ is a binary indicator whether the edge (s,a) has been traversed at simulation i.

Observe that unlike the standard MCTS, the value $V(S_L)$ of the leaf is not determined entirely by the Monte-Carlo rollout, but it is a mixture of the rollout outcome and the prediction of the value network $v_\theta(s_L)$. Comparing the performance of AlphaGo using the value network only (without roll-outs) or using only roll-outs, it turns out that the

roll-outs perform better than the value network, but the combination from Eq. (2.20) with $\lambda = 0.5$ is much better than both of them.

In October 2015 AlphaGo won 5-0 against 2-dan European champion Fan Hui and in March 2016 it won 4-1 against 9-dan professional player Lee Sedol. Then in 2017 AlphaGo won 3-0 against Ke Jie, the no 1 ranked Go player in the world and was awarded professional 9-dan by Chinese Weiqi Association.

2.7 Exercises

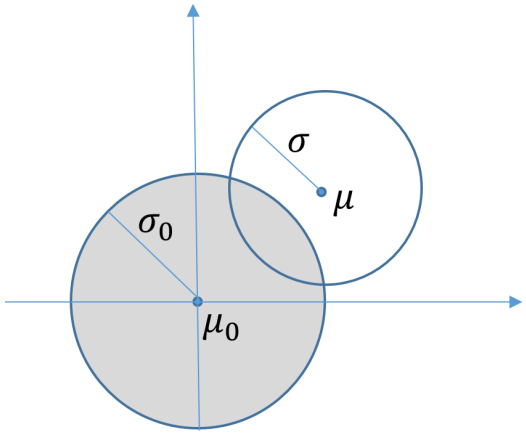


Fig. 2.20: Figure for Problem 1.

Problem 1. Importance sampling and the effective number of samples. In a 2D plane, suppose the target distribution $\pi(x,y)$ is a symmetric Gaussian with mean $\mu = (2,2)$ and standard deviation $\sigma = 1$. Suppose we use an approximate distribution $g(x,y)$ as the trial density which is a Gaussian with mean $\mu_0 = (0,0)$ with standard deviation σ_0. So

$$\pi(x,y) = \frac{1}{2\pi} e^{-1/2[(x-2)^2+(y-2)^2]}, \quad g(x,y) = \frac{1}{2\pi\sigma_o} e^{-1/(2\sigma_o^2)[x^2+y^2]}$$

We estimate the quantity $\theta = \int \sqrt{y^2+x^2}\pi(x,y)dxdy$. We compare the effectiveness of 3 reference probabilities used in importance sampling.

- Step 1, Compute $\hat{\theta}_1$: estimate θ by drawing n_1 samples directly from $\pi(x,y)$. Since the two dimensions are independent, you can sample x and y from the 1D marginal Gaussians.
- Step 2, Compute $\hat{\theta}_2$: estimate θ by drawing n_2 samples from g(x,y) with $\sigma_0 = 1$.
- Step 3, Compute $\hat{\theta}_3$: estimate θ by drawing n_3 samples from g(x,y) with $\sigma_0 = 4$.

i) Plot $\hat{\theta}_1, \hat{\theta}_2, \hat{\theta}_3$ over n (increasing n so that they converge) in one figure to compare the convergence rates. Before running the experiment, try to guess whether step 3 is more effective than step 2. [you may use a log plot at a few points $n = 10, 100, 1000, 10000, \ldots$]

ii) Estimate the value of the "effective sample size". We suggested the estimator

$$ess(n) = \frac{n}{1 + \mathrm{var}_g[\omega]}$$

in eq. (2.6), but we are not sure how good it is. Since the samples in step 1 are all "effective" samples directly drawn from the target distribution, we use $ess^*(n_1) = n_1$ as the truth and compare the effective sample sizes for step 2 and

step 3, i.e. the true $ess^*(n_2)$ and $ess^*(n_3)$ are the numbers when the estimated errors reach the same level as in step 1. Plot $ess(n_2)$ over $ess^*(n_2)$, and $ess(n_3)$ over $ess^*(n_3)$. Discuss your results.

Problem 2. Estimating the number of Self-Avoiding-Walks in an $(n+1)\times(n+1)$ grid. Suppose we always start from position $(0,0)$, i.e. lower-left corner. We design a trial (reference) probability $p(r)$ for a SAW $r=(r_1,r_2,\ldots,r_N)$ of varying length N. Then we sample a number of M SAWs from $p(r)$, and the estimation is calculated below. Some results are shown as examples in the chapter.

At each step, the trial probability $p(r)$ can choose to stop (terminate the path) or walk to the left/right/up/down as long as it does not intersect itself. Each option is associated with a probability (of your design) and these probabilities sum to 1 at each point.

1) What is the total number K of SAWs for $n=10$ [try $M=107$ to 108]? To clarify: a square is considered a 2×2 grid with $n=1$. Plot K against M (in a log-log plot) and monitor whether the Sequential Importance Sampling (SIS) process has converged. Try to compare at least 3 different designs for $p(r)$ and see which is more efficient. E.g. you may start from a path that you find before multiple times, as long as you compute the $p(r)$ correctly.

2) What is the total number of SAWs that start from $(0,0)$ and end at (n,n)? Here you can still use the same sampling procedure above, but only record the SAWs which successfully reach (n,n). The truth for this number is what we discussed: 1.5687×10^{24}.

3) For each experiment in 1) and 2), plot the distribution of the lengths N of the SAWs in a histogram (Think: Do you need to weight the SAWs in calculating the histogram?) and visualize (print) the longest SAW that you find.

References

1. Yongjoon Joe Baba Satomi, Atsushi Iwasaki, and Makoto Yokoo. Real-time solving of quantified csps based on monte-carlo game tree search. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence*, 2011.
2. Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.
3. Romaric Gaudel and Michele Sebag. Feature selection as a one-player game. In *International Conference on Machine Learning*, pages 359–366, 2010.
4. Michael Isard and Andrew Blake. Condensation: conditional density propagation for visual tracking. *International journal of computer vision*, 29(1):5–28, 1998.
5. James T Kajiya. The rendering equation. In *ACM Siggraph Computer Graphics*, volume 20, pages 143–150. ACM, 1986.
6. Jacob Porway and Song-Chun Zhu. C^ 4: Exploring multiple solutions in graphical models by cluster sampling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(9):1713–1727, 2011.
7. Frederik Christiaan Schadd. Monte-carlo search techniques in the modern board game thurn and taxis. *M. sc, Maastricht University, Netherlands*, 2009.
8. David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.
9. Zhuowen Tu and Song-Chun Zhu. Image segmentation by data-driven markov chain monte carlo. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(5):657–673, 2002.
10. Eric Veach and Leonidas J Guibas. Optimally combining sampling techniques for monte carlo rendering. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 419–428. ACM, 1995.

Chapter 3
Markov Chain Monte Carlo - the Basics

松下问童子，言师采药去。只在此山中，云深不知处。— 贾岛【779-843】

Translation:
Inquery a boy under a pine tree;
Told that the master went to search for herbal medicine;
Must be inside this mountain,
With the cloud so deep, nobody knows where he is.

— Jia Dao (779-843)

Introduction

Imagine you enter a big national park (in the poem it is a mountain), your path is essentially a Markov chain in a bounded space. The frequency that you stay at an attraction spot is proportional to its popularity. How will you predict the location x where your friends are at certain moment t? The uncertainty of the location is a distribution $p_t(x)$.

Markov

Markov chain Monte Carlo (MCMC) is a general purpose technique for generating fair samples from a probability in a *high-dimensional* space, driven by random numbers drawn from a uniform distribution over $[a,b]$. A Markov chain is designed to have a probability distribution function, $\pi(x)$, as its stationary (invariant) probability. Many stochastic systems in physics, chemistry, and economics can be simulated using MCMC. This chapter gives an overview of Markov chains and their defining properties. Additionally, theorems for the existence of a unique, stationary distribution of a Markov chain are discussed and applications to simulated annealing and webpage popularity ranking are provided.

3.1 Markov Chain Basics

A Markov Chain is a mathematical model for stochastic systems whose states, *discrete* or *continuous*, are governed by a transition probability P. The current state in a Markov chain only depends on the most recent previous states, *e.g.* for a 1^{st} order Markov chain

$$X_t|X_{t-1},\dots,X_0 \sim P(X_t|X_{t-1},\dots,X_0) = P(X_t|X_{t-1})$$

The Markovian property implies "locality" in space or time, such as for Markov random fields and Markov chains. Indeed, a discrete time Markov chain can be viewed as a special case of a Markov random field (causal and 1-dimensional).

A Markov chain is often denoted by

$$\text{MC} = (\Omega, \nu_0, K)$$

where Ω is the state space, $\nu_0 : \Omega \to \mathbb{R}$ is the initial probability distribution over the states, $K : \Omega \times \Omega \to \mathbb{R}$ is the transition probability, also called transition kernel.

Suppose that Ω is countable (or even better, finite), then K is the matrix of transition probabilities $K(X_{t+1}|X_t)$. At time n, the Markov chain state will follow a probability,

$$\nu_n = \nu_0 K^n.$$

Example 3.1. Suppose we have a finite state space, $|\Omega| = N \sim 10^{30}$, then the transition probability K would be represented by an $N \times N$ transition matrix

$$K(X_{t+1}|X_t) = \begin{bmatrix} k_{11} & \cdots & k_{N1} \\ \vdots & \ddots & \vdots \\ k_{1N} & \cdots & k_{NN} \end{bmatrix}_{(N\times N).}$$

The transition matrix is usually sparse, but not always.

Thus, in SMC we try to design the trial probability $g(x)$, and in MCMC we will try to design the transition matrix $K(X_{t+1}|X_t)$. Therefore,

$$X_n \sim \underbrace{(\cdots)_{(1\times N)}}_{\nu_n} = \underbrace{(\cdots)_{(1\times N)}}_{\nu_{n-1}} \begin{bmatrix} k_{11} & \cdots & k_{N1} \\ \vdots & \ddots & \vdots \\ k_{1N} & \cdots & k_{NN} \end{bmatrix}_{(N\times N).}$$

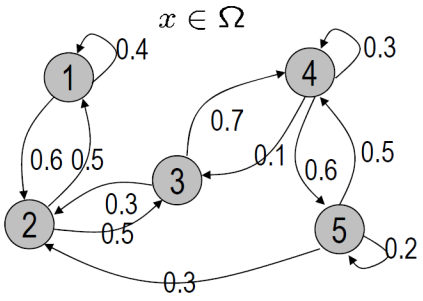


Fig. 3.1: Trade diagram for the five families.

Example 3.2. **Five Families.** Suppose there are five families on an island. There are 1,000,000 tokens used as their currency, and we normalize wealth such that a value of 1 implies possession of all tokens. Let ν_t be a 5×1 vector for the wealth of the 5 families after year t. Each family will trade with other families for goods. For example, family 1 will spend 60% of their income to buy from family 2, and save the other 40%, and so on, as shown in Figure 3.1. The question is: how will the fortune be distributed among the families after a number of years? To ask in a different

way, suppose we mark one token in a special color (say, red), and after many years, we wonder what the probability distribution is that determines who owns this token.

We convert this to a mathematical model. Denote the state space for the red token by $\Omega = \{1,2,3,4,5\}$. Then the transition kernel is

$$K = \begin{pmatrix} 0.4 & 0.6 & 0.0 & 0.0 & 0.0 \\ 0.5 & 0.0 & 0.5 & 0.0 & 0.0 \\ 0.0 & 0.3 & 0.0 & 0.7 & 0.0 \\ 0.0 & 0.0 & 0.1 & 0.3 & 0.6 \\ 0.0 & 0.3 & 0.0 & 0.5 & 0.2 \end{pmatrix}.$$

Computing the distribution of wealth starting from different initial conditions we get the results in Table 3.1.

Table 3.1: Final wealth distribution after convergence in Example 3.2 when starting from different initial distributions.

Year	A					B				
1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
2	0.4	0.6	0.0	0.0	0.0	0.0	0.3	0.0	0.7	0.0
3	0.46	0.24	0.30	0.0	0.0	0.15	0.0	0.22	0.21	0.42
4	...					...				
5	...					...				
6	0.23	0.21	0.16	0.21	0.17	0.17	0.16	0.16	0.26	0.25
...	...					...				
Final	0.17	0.20	0.13	0.28	0.21	0.17	0.20	0.13	0.28	0.21

Under certain conditions for finite state Markov chains, the state converges to an invariant probability given by

$$\lim_{n\to\infty} \nu_0 K^n = \pi.$$

In Bayesian inference, we are given a target probability π, and our objective is to design a Markov chain kernel K, such that π is the *unique*, invariant probability of K.

In general, there are infinitely many K's that have the same invariant probability.

$$\begin{array}{ccccccc} X_1 & \longrightarrow & X_2 & \longrightarrow & \cdots & \longrightarrow & X_n & \longrightarrow \\ \wr & & \wr & & & & \wr & \wr \\ \nu_1 & & \nu_2 & & \cdots & & \nu_n & \pi \end{array}$$

Suppose we are given Ω and a target probability $\pi = (\pi_1, \cdots, \pi_N)_{(1\times N)}$, our goal is to design ν_0 and K so that

1) $\pi K = \pi$;
   This is a necessary condition for the Markov chain to have the stationary probability π.
2) Fast convergence. Fast convergence can be obtained using a:
   - good initial probability ν_0.
   - good transition matrix K.

Usually the transition matrix is sparse (zero almost everywhere) due to the local connectivity, i.e. since the new state of an MCMC move is usually near the current state. There are some isolated examples where this is not the case, such as the Swendsen-Wang algorithm from Chapter 6.

3.2 Topology of Transition Matrix: Communication and Period

Now, we check the conditions for Markov chain design.

1. Stochastic Matrix. The kernel matrix K should be a stochastic matrix, i.e.

$$\sum_{j=1}^{N} K_{ij} = 1, \qquad \forall i \in \Omega, \quad K_{ij} \geqslant 0,$$

or in matrix form:

$$K\mathbf{1} = \mathbf{1},$$

where $\mathbf{1}$ is the $N \times 1$ vector of 1's: $\mathbf{1} = (1, ..., 1)^T$.

2. Global Balance. Another necessary condition is global balance:

$$\pi K = \pi \qquad \rightarrow \qquad \sum_{i=1}^{N} \pi_i K_{ij} = \pi_j \qquad \forall j \in \Omega.$$

This condition can be replaced by the *detailed balance* condition (a sufficient but not necessary condition):

$$\pi(i)K_{ij} = \pi(j)K_{ji}, \ \forall \ i, j \in \Omega. \tag{3.1}$$

Indeed, detailed balance implies stationarity:

$$\begin{aligned} \pi K &= \sum_i \pi(i)K_i = \sum_i \pi(i)\Big(K_{i1}, \ldots, K_{iN}\Big) \\ &= \sum_i \Big(\pi(1)K_{1i}, \ldots, \pi(N)K_{Ni}\Big) = \pi, \end{aligned}$$

and in particular global balance

$$\sum_i \pi(i)K_{ij} = \sum_i \pi(j)K_{ji} = \pi(j)\sum_i K_{ji} = \pi(j).$$

A kernel that satisfies the detailed balance condition is called *reversible*.

Going back to Example 3.2 we can infer that the global balance equation represents the total wealth conservation. Indeed, the total amount received by family j is $\sum_i \pi(i)K_{ij}$ and it is equal to family j's wealth $\pi(j)$, which is the amount spent by family j .

There are infinite ways to construct K given a π. In global balance, we have $2N$ equations with $N \times N$ unknowns, in detailed balance we have $\frac{N^2}{2} + N$ equations with $N \times N$ unknowns.

3. Irreducibility. A state j is said to be accessible from state i if there exists a step M, such that $(K^M)_{ij} > 0$, where

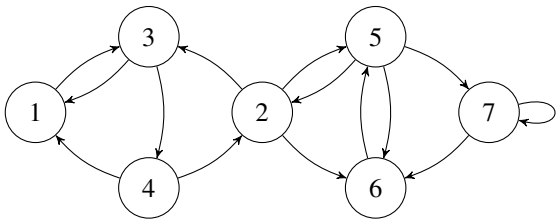
$$i \rightarrow j \qquad (K^M)_{ij} = \sum_{i_1, \ldots, i_{M-1}} K_{ii_1} \cdots K_{i_{M-1}j} > 0.$$

If i and j are accessible from each other then we write $i \leftrightarrow j$. The communication relation $\leftrightarrow$ generates a partition of the state space into disjoint equivalence(communication) classes given by

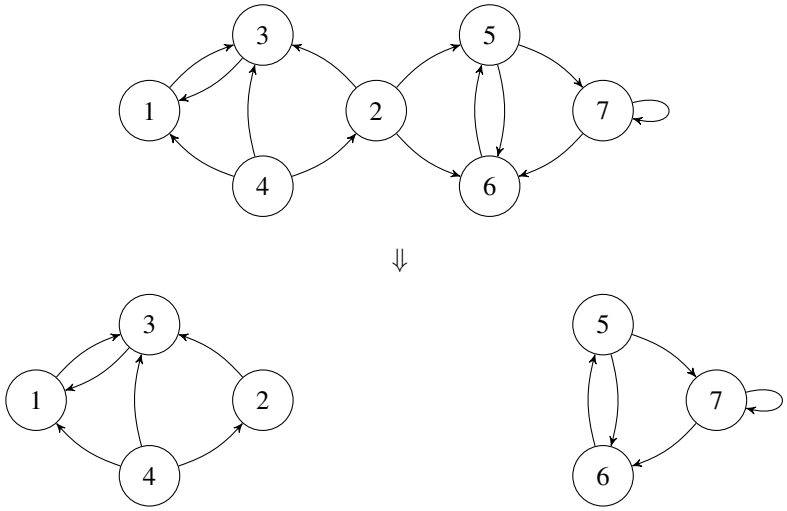
$$\Omega = \cup_{i=1}^{C} \Omega_i.$$

Definition 3.1. A Markov Chain is *irreducible* if its transition matrix K has only 1 communication class.

Example 3.3. Irreducible MC:



Example 3.4. Reducible MC:



In general, a greedy optimization algorithm has a reducible chain and will get stuck in a local optimum.

Given a target distribution, π, the ideal transition kernel would be $K = \begin{pmatrix} \pi \\ \pi \\ \dots \\ \pi \end{pmatrix}$, which always converges in one step no matter where it starts. However, in general it is difficult to sample the distribution π directly so this kernel is not very useful in practice.

4. Aperiodicity. To define aperiodicity, we first need to define a periodic Markov chain.

Definition 3.2. An irreducible Markov chain with transition matrix K has period d if there is a (unique) partition of graph G into d cyclic classes:

$$C_1, \dots, C_d, \qquad \sum_{j \in C_k} K_{ij} = 1, \qquad \forall i \in C_{k-1}.$$

Remark 3.1 *In a periodic Markov chain there is no connection between states within each individual cyclic class. The transition matrix is a block matrix of the following form.*

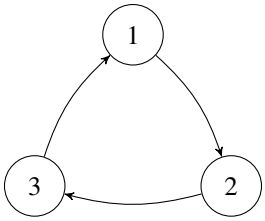
$$K = \begin{pmatrix} & \blacksquare & & \\ & & \blacksquare & \\ & & & \blacksquare \\ \blacksquare & & & \end{pmatrix}$$

Then, the transition matrix of K^d becomes a diagonal block matrix.

$$K^d = \begin{pmatrix} \blacksquare & & & \\ & \blacksquare & & \\ & & \blacksquare & \\ & & & \blacksquare \end{pmatrix}$$

This means that K has one communication class, but K^d has d communication classes.

Example 3.5. Consider the Markov chain with the following transition kernel: $K = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$.



It has period 3 and alternates between three distributions: $(1\ 0\ 0) \to (0\ 1\ 0) \to (0\ 0\ 1)$.

Definition 3.3. An irreducible Markov chain with transition matrix K is aperiodic if it's largest period is $d = 1$.

5. Stationary distribution. A Markov chain with transition kernel K has stationary distribution π if

$$\pi K = \pi.$$

There may be many stationary distributions w.r.t K. Even if there is a stationary distribution, a Markov chain may not always converge to it.

Example 3.6. Consider the Markov chain with the transition kernel: $K = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$. It has $\pi = (\frac{1}{3}\ \frac{1}{3}\ \frac{1}{3})$ as a stationary distribution, but it might never converge to it, as seen in Example 3.5.

3.3 The Perron-Frobenius Theorem

Theorem 3.1 (Perron-Frobenius) *For any primitive (irreducible and aperiodic) $N \times N$ stochastic matrix K, K has eigenvalues*

$$1 = \lambda_1 > |\lambda_2| > \cdots > |\lambda_r|$$

with multiplicities $m_1, ..., m_r$ and left and right eigenvectors $(\mathbf{u}_i, \mathbf{v}_i)$ respectively. Then $\mathbf{u}_1 = \pi, \mathbf{v}_1 = \mathbf{1}$, and

$$K^n = \mathbf{1} \cdot \pi + O(n^{m_2 - 1}|\lambda_2|^n).$$

We define $\lambda_{\text{slem}} = |\lambda_2|$, which is the second largest eigenvalue modulus. we see that the convergence rate is decided by λ_{slem}.

Remark 3.2 *If K is reducible, and has C communication classes, then it is a block diagonal matrix with C blocks. Consequently, the eigenvalue 1 has at least C distinct eigenvectors, and K does not have a unique invariant probability.*

Remark 3.3 *If K is irreducible but has period $d > 1$, then it has at least d distinct eigenvalues with modulus 1, namely, the d^{th} roots of unity. This is because its characteristic polynomial is $\det(tI - K) = \det(t^d I - K_1 ... K_d)$ as can be proved by induction, where $K_1, ..., K_d$ are the non-zero blocks. $K_1, ..., K_d$ all have eigenvalue 1, so $U = K_1 ... K_d$ has eigenvalue 1, and thus its characteristic polynomial $\det(tI - U)$ is divisible by $t - 1$. Therefore, $\det(tI - K) = \det(t^d I - U)$ is divisible by $t^d - 1$.*

Review

Suppose K is a $N \times N$ positive non-symmetric matrix, and has N eigenvalues.

$$\begin{array}{ccc} \lambda_1 & \cdots\cdots\cdots & \lambda_N \\ \widehat{u_1\ v_1} & & \widehat{u_N\ v_N} \end{array}$$

Each eigenvalue has a corresponding right and left eigenvector. λ, u, v are all complex numbers.

$$\begin{aligned} u_i K &= \lambda_i u_i, & u_i &: 1 \times N \\ K v_i &= \lambda_i v_i, & v_i &: N \times 1 \end{aligned}$$

Therefore,

$$K = \lambda_1 v_1 u_1 + \lambda_2 v_2 u_2 + \cdots + \lambda_N v_N u_N$$

$$K \cdot K = \sum_{i=1}^{N} \lambda_i v_i u_i \cdot \sum_{j=1}^{N} \lambda_j v_j u_j = \sum_{\substack{i=1 \\ j=1}}^{N} \lambda_i \lambda_j v_i u_i v_j u_j, \qquad \begin{cases} \text{if } i \neq j & u_i v_j = 0 \\ \text{if } i = j & u_i v_j = 1 \end{cases}.$$

Thus,

$$K^n = \lambda_1^n v_1 u_1 + \lambda_2^n v_2 u_2 + \cdots + \lambda_N^n v_N u_N.$$

Since we have global balance,

$$\left.\begin{array}{lll} \pi K = \pi & \Longrightarrow & \lambda_1 = 1, u_1 = \pi \\ K\mathbf{1} = \mathbf{1} & \Longrightarrow & \lambda_1 = 1, v_1 = \mathbf{1} \end{array}\right\} \Longrightarrow \lambda_1 \cdot v_1 \cdot u_1 = \begin{pmatrix} \pi \\ \pi \\ \cdots \\ \pi \end{pmatrix}.$$

Thus,

$$K^n = \begin{pmatrix} \pi \\ \pi \\ \cdots \\ \pi \end{pmatrix} + \underbrace{\varepsilon}_{\to 0} \qquad \text{, if } |\lambda_i| < 1, \forall i > 1,$$

so K^n approaches the ideal kernel $\begin{pmatrix} \pi \\ \pi \\ \cdots \\ \pi \end{pmatrix}$ as $n \to \infty$.

3.4 Convergence Measures

One thing we are interested in is the state i which is the global optimum of the probability,

$$i^* = \operatorname{argmax} \pi(x).$$

Definition 3.4. Given a Markov chain $(x_0, ..., x_n, ...)$ with transition kernel K and invariant probability π we define

i) The first hitting time of a state i (in the finite state case)

$$\tau_{\text{hit}}(i) = \inf\{n \geqslant 1; x_n = i, x_0 \sim \nu_0\}, \qquad \forall i \in \Omega.$$

$E[\tau_{\text{hit}}(i)]$ is the mean first hitting time of i for the Markov chain governed by K.

ii) The first return time of a state i

$$\tau_{\text{ret}}(i) = \inf\{n \geqslant 1; x_n = i, x_0 = i\}, \qquad \forall i \in \Omega.$$

iii) The mixing time

$$\tau_{\text{mix}} = \min_n \{\|\nu_0 K^n - \pi\|_{\text{TV}} \leqslant \varepsilon, \forall \nu_0\},$$

where the total variation is defined as

$$\|\mu - \nu\|_{\text{TV}} = \frac{1}{2}\sum_{i\in\Omega} |\mu(i) - \nu(i)| = \sum_A \Big(\mu(i) - \nu(i)\Big), \qquad A = \{i : \mu(i) \geqslant \nu(i), i \in \Omega\}.$$

Definition 3.5. The contraction coefficient for K is the maximum TV-norm between any two rows in the transition kernel and is calculated by

$$C(K) = \max_{x,y} \|K(x,\cdot) - K(y,\cdot)\|_{\text{TV}}.$$

Example 3.7. Consider the Markov kernel for the five families living in an island, where the values are different than in Example 3.2

$$K = \begin{pmatrix} 0.3, & 0.6, & 0.1, & 0.0, & 0.0 \\ 0.2 & 0.0, & 0.7, & 0.0, & 0.1 \\ 0.0, & 0.5, & 0.0, & 0.5, & 0.0 \\ 0.0, & 0.0, & 0.4, & 0.1, & 0.5 \\ 0.4, & 0.1, & 0.0, & 0.4, & 0.1 \end{pmatrix}$$

1) We plot in Figure 3.2, left, the five complex eigenvalues, in a 2D plane.

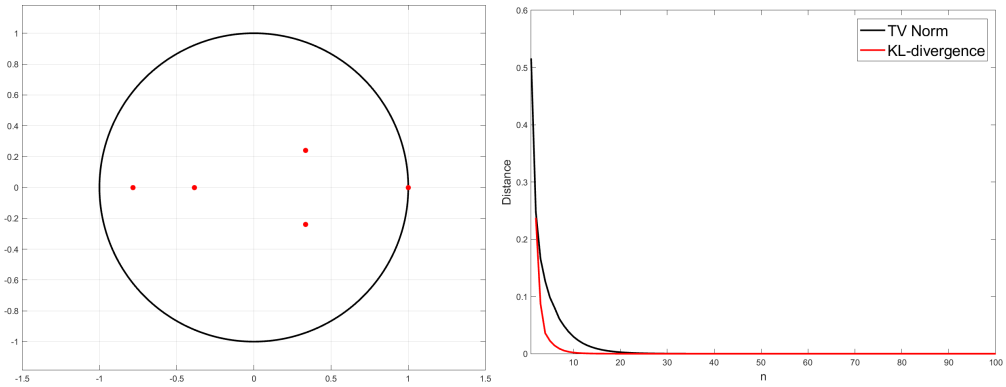


Fig. 3.2: The five families kernel of Example 3.7. Left: the five complex eigenvalues. Right: the TV norm and KL divergence between $\mu_n = \nu \cdot K^n$ and the invariant probability π.

The invariant probability is $\pi = (0.1488\ 0.2353\ 0.2635\ 0.2098\ 0.1427)$. The second largest eigenvalue has $\lambda_{slem} = \|\lambda_2\| = 0.7833$.

2) Suppose we start with an initial probabilities $\nu = (1,0,0,0,0)$ i.e. we know for sure that the initial state is at $x_0 = 1$. So, at step n, the Markov chain state follows a distribution $\mu_n = \nu \cdot K^n$. We compute the distance between μ_n and π using the TV-norm,

$$d_{\text{TV}}(n) = ||\pi - \mu_n||_{\text{TV}} = \frac{1}{2}\sum_{i=1}^{5} |\pi(i) - \mu_n(i)|,$$

or KL-divergence,

$$d_{\text{KL}}(n) = \sum_{i=1}^{5} \pi(i) \log \frac{\pi(i)}{\mu_n(i)}.$$

The plot of the two distances $d_{\text{TV}}(n)$ and $d_{\text{KL}}(n)$ for the first 100 steps is shown in Figure 3.2, right.

3) We calculate the contraction coefficient for K. Note that contraction coefficient is the maximum TV-norm between any two rows in the transition kernel,

$$C(K) = \max_{x,y} ||K(x,\cdot) - K(y,\cdot)||_{\text{TV}}.$$

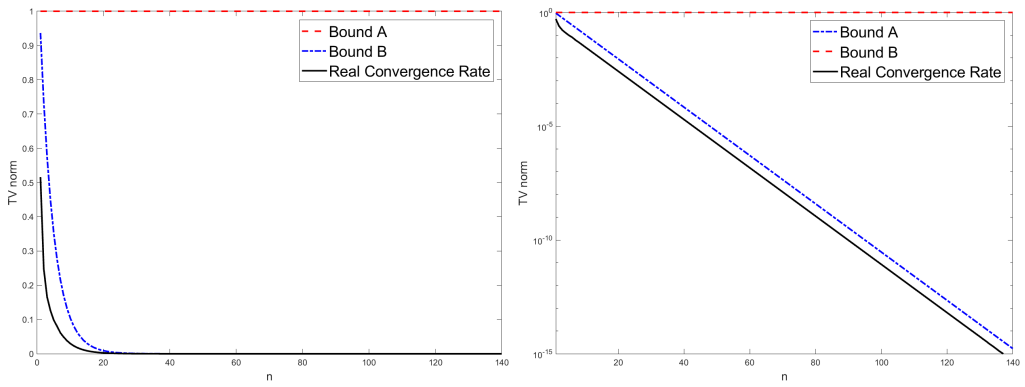


Fig. 3.3: The TV norm between $\mu_n = \nu \cdot K^n$ and the invariant probability π and the two bounds $A(n)$ and $B(n)$ from Eq. (3.2) and (3.3) respectively. Left: original scale. Right: log scale.

One can prove that

$$||\nu_1 \cdot K - \nu_2 \cdot K||_{\text{TV}} \leq C(K)||\nu_1 - \nu_2||_{\text{TV}}.$$

As $||\nu_1 - \nu_2||_{\text{TV}} \leq 1$, if $C(K) < 1$, then the convergence rate can be upper bounded by

$$A(n) = C^n(K) \geq C^n(K)||\nu_1 - \nu_2||_{\text{TV}} \geq ||\nu_1 \cdot K^n - \nu_2 \cdot K^n||_{\text{TV}}, \quad \forall \nu_1, \nu_2. \tag{3.2}$$

For this example, we can see that $C(K) = 1$, so the bound is not very useful.

4) There is another bound – the Diaconis-Hanlon bound given by

$$B(n) = \sqrt{\frac{1-\pi(x_0)}{4\pi(x_0)}}\lambda_{\text{slem}}^n \geq ||\pi - \nu K^n||_{\text{TV}}, \tag{3.3}$$

where $x_0 = 1$ is the initial state and $\pi(x_0)$ is a target probability at $x = 1$. The real convergence rate $d_{\text{TV}}(n)$ is plotted in comparison with $A(n)$ and $B(n)$ in Figure 3.3, on the original scale and the log-scale. The bound only holds until the machine precision is attained.

3.5 Markov Chains in Continuous or Heterogeneous State spaces

In the continuous case, the target distribution $\pi : \Omega \to \mathbb{R}$ is a pdf $\pi(x)$ and the transition kernel is a conditional pdf $K(x,y) = K(y|x)$, so $\int_\Omega K(x,y)\mathrm{d}y = 1$.

The global balance equation must then be satisfied for any event $A \subseteq \Omega$,

$$\pi K(A) = \int_A \int_\Omega \pi(x)K(x,y)\mathrm{d}x\mathrm{d}y = \int_A \pi(x)\mathrm{d}x = \pi(A),$$

and the detailed balance equation in the continuous case is

$$\int_A \int_B \pi(x)K(x,y)\mathrm{d}x\mathrm{d}y = \int_A \int_B \pi(y)K(y,x)\mathrm{d}x\mathrm{d}y.$$

In practice, Ω is a mixed/heterogeneous space made up of discrete/finite and continuous variables.

Example 3.8. Consider the heterogeneous space of $X = \{N, (x_i, y_i), i = 1, \ldots, N\}$, where N is the number of people in a picture and (x_i, y_i) are their positions. In such a case there are many different Markov chain processes, as illustrated in Figure 3.4. Irreducible MCMC will have many dynamics(sub chains) such as:

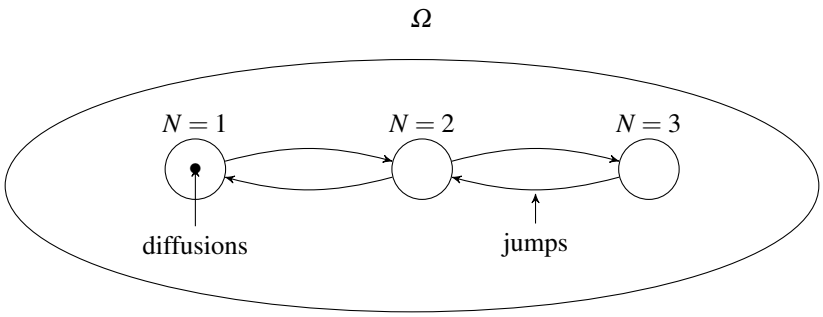


Fig. 3.4: Illustration of jump-diffusion processes for the heterogeneous space from Example 3.8.

$$\begin{cases} \text{Jumps} \begin{cases} \text{Death/Birth} \\ \text{Split/Merge} \end{cases} \text{process} \\ \text{Diffusions} \end{cases}.$$

3.6 Ergodicity Theorem

Definition 3.6. A state i is said to be *recurrent* if it has

$$P(\tau_{\text{ret}}(i) < \infty) = 1.$$

Otherwise it is *transient*. The quantity $\tau_{\text{ret}}(i)$ is the return time, the total number of steps needed to return from x to x.

Definition 3.7. A state i that satisfies

$$E[\tau_{\text{ret}}(i)] < \infty$$

is called *positively recurrent*. Otherwise it is null-recurrent. A Markov chain is positively recurrent if all of its states are positively recurrent.

Usually, positive recurrence is a condition for spaces with infinite states.

Theorem 1 (Ergodicity theorem) *For an irreducible, positively recurrent Markov chain with stationary probability π, in a state space Ω, let $f(x)$ be any real valued function with finite mean with respect to π, then for any initial probability, almost surely we have*

$$\lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} f(x_i) = \sum_{x\in\Omega} f(x)\pi(x) = E_\pi[f(x)], \qquad \forall f$$

where $x_i \sim$ Markov chain states (but do not need to be i.i.d.).

3.7 MCMC for Optimization by Simulated Annealing

An MCMC algorithm is designed to obtain samples from a posterior distribution π, $X \sim \pi$. We saw that under certain conditions (detailed balance, irreducibility and aperiodicity) the Markov chain invariant probability will converge to the stationary distribution π after a burn-in period.

MCMC can also be used for optimization by slowly changing the stationary distribution π while running the Markov chain. Suppose we want to maximize a function $f(x) : \Omega \to \mathbb{R}$. We consider the posterior probability

$$\pi(x;T) = \frac{1}{Z(T)} \exp(-f(x)/T),$$

which depends on a temperature parameter, T. When T is large, the probability $\pi(x,T)$ will have smaller peaks and local maxima making it easier to sample from. When T is very small, the probability $\pi(x,T)$ will be concentrated at its global maximum, as illustrated in Figure 3.5.

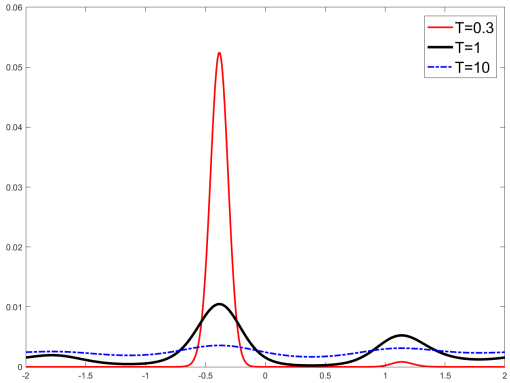


Fig. 3.5: The influence of the temperature on a probability distribution. At temperature $T = 10$ the probability is close to uniform, while at $T = 0.3$ the global optimum is clearly visible.


Scott Kirkpatrick

The annealing procedure [1] requires starting the Markov chain at a high temperature and slowly decreasing it until a very small temperature is reached. This procedure is inspired from the annealing method for producing crystalline structures in metals or other materials. In this process, the material is heated to a high temperature until it melts, and then the temperature is slowly decreased to allow the atoms to position themselves in a low energy configuration, which gives the crystalline structure. If the material is cooled too quickly, it will develop cracks or other defects such as small crystals. If the cooling is slow enough, larger crystals with fewer defects can be obtained.

Similarly, for optimization one needs to select an *annealing schedule*, which specifies the temperature used at each step of the Markov chain. The schedule starts with a high T_0 and decreases to 0 as $k \to \infty$, so $\lim_{k\to\infty} T_k = 0$. Through this annealing schedule, the probability $\pi(x,T)$ becomes $\pi(x,T_k)$, a time-dependent probability. The simulated annealing algorithm is described below, where for any $x \in \Omega$, $N(x)$ is the set of possible states that are accessible from state x in one Markov chain step.

There are two types of convergence results, based on modeling the above algorithm as a sequence of homogeneous Markov chains or as a single inhomogeneous Markov chain. The following is an inhomogeneous Markov chain result due to Mitra [2].

Theorem 3.2 *(Mitra 1986) The Markov chain associated with simulated annealing with the following update function*

$$T_k = \frac{\gamma}{\log(k + k_0 + 1)}$$

for any parameter $k_0 \geq 1$ and γ sufficiently large converges to a global optimum regardless of the initial solution $x \in \Omega$.

In reality, one cannot wait too long to find a solution, and faster annealing schedules are used in practice, decreasing linearly or even exponentially in t. With these schedules, the optimization algorithm finds a local optimum that can be good or bad depending on the annealing schedule and the MCMC algorithm used. Some MCMC algorithms, such as the Gibbs sampler, need slow annealing schedules to obtain good solutions while other algorithms, such as the Swendsen-Wang Cut, allow faster colling schedules to obtain similar solutions.

Remark 3.4 *In many computer vision problems finding the global optimum of $\pi(x)$ can be NP hard, which means a polynomial optimization algorithm is not likely to be found. In these cases, the annealing schedule will have to decrease logarithmically in t so the global optimum will be found in exponential time with respect to the problem size (e.g. dimensionality of Ω).*

Algorithm 1 Simulated Annealing

input: Initial solution $x \in \Omega$
input: Temperature cooling schedule, T_k
input: Initial temperature $T = T_0 > 0$
input: Repetition schedule M_k - the number of iterations executed at each temperature T_k
Set the temperature change counter $k = 0$
repeat
  for $m = 0$ to M_k **do**
    Generate a solution $x' \in N(x)$
    Calculate $\Delta_{x,x'} = f(x') - f(x)$
    if $\Delta_{x,x'} \leq 0$ **then**
      $x \leftarrow x'$
    else
      $x \leftarrow x'$ with probability $\exp(-\Delta_{x,x'}/T_k)$
    end if
  end for
  k=k+1
until stopping criterion is met

3.7.1 Page rank example

As one final example of the usefulness of MCMC, we consider ranking the importance of a set of webpages. We consider a directed graph $G = \langle V, E \rangle$ with the pages as the set of nodes, V and the links connecting the pages as the set of edges, E. For a specific page, x, consider two sets given by

$$out(x) = \{w | x \to w \in E\}, \quad in(x) = \{y | y \to x \in E\}.$$

In order to create a sufficient measure of the importance of a page, we need to consider two features of the pages it connects to.

1. Are links to x from pages that have many other links, or is it from pages that show only a few options?
2. Are links to x from reputable, high-traffic pages, or are they found on personal websites or blogs?

The first point tells us that our ranking measure should take into account $|out(y)|$, for all pages y that connect to x, while the second point indicates that the importance of a page, $\pi(x)$, should be defined recursively with respect to the importance of the pages that connect to it, $\pi(y)$. Taking these into consideration we define

$$\pi(x) = \sum_{y \in in(x)} \frac{\pi(y)}{|out(y)|}.$$

In order to sample from this distribution of ranked webpages we use MCMC with transition probability given by

$$K(y,x) = \frac{1}{|out(y)|}.$$

π is indeed a stationary distribution of this Markov chain since K satisfies

$$\sum_{y \in V} \pi(y)K(y,x) = \sum_{y \in in(x)} \pi(y)K(y,x) = \sum_{y \in in(x)} \frac{\pi(y)}{|out(y)|} = \pi(x).$$

While this chain does have π as a stationary distribution, it is not quite ergodic since there are likely several pages that contain no links, unless G has high connectivity. For this reason we introduce a probability α that a user types in a link and jumps to an unconnected page. If there are $N = |V|$ webpages, then the new transition probability is given by

$$K(x,y) = \begin{cases} \frac{1-\alpha}{N} & x \to y \notin E \\ \frac{1-\alpha}{N} + \frac{\alpha}{|out(y)|} & x \to y \in E \end{cases}.$$

Since this new chain is irreducible regardless of the graph connectivity, it is ergodic.

To demonstrate this idea, in Figure 3.6 is shown a graphical representation for a personal website where the transition probabilities are calculated using the formula above. The site consists of 5 pages: Homepage, About, Projects, Publications, and Contact. The links present on each page are also given in Table 3.2 below.

Table 3.2: Links present in the example webpage.

Page	Homepage	About	Projects	Publications	Contact
Links	About Projects Publications Contact	Homepage Publications Contact	Homepage Publications	Homepage About Projects	Homepage About

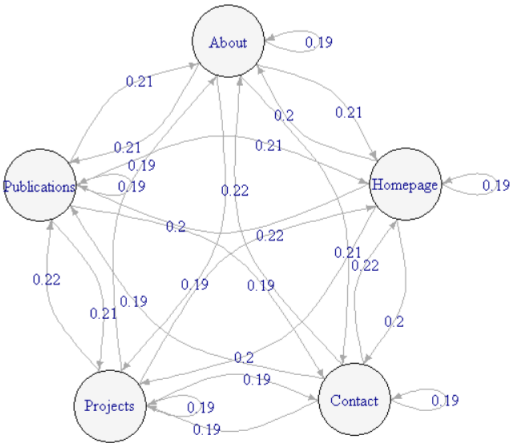


Fig. 3.6: An example graph for the pagerank application.

At all states the user can hit the back button to return to the previous page and also refresh the current page. The user always starts at the homepage, i.e., the initial probability is $(1,0,0,0,0)$. Running this Markov chain would produce the ranking of the pages by producing the stationary probability that a user visits a particular page.

3.8 Exercises

Problem 1. Consider the Markov kernel for the five families living in an island, where the numbers are changed,

$$K = \begin{pmatrix} 0.3,\ 0.6,\ 0.1,\ 0.0,\ 0.0 \\ 0.2\ \ 0.0,\ 0.7,\ 0.0,\ 0.1 \\ 0.0,\ 0.5,\ 0.0,\ 0.5,\ 0.0 \\ 0.0,\ 0.0,\ 0.4,\ 0.1,\ 0.5 \\ 0.4,\ 0.1,\ 0.0,\ 0.4,\ 0.1 \end{pmatrix}$$

1). Calculate the five eigenvalues, and their corresponding left and right eigenvectors (you can use any software package).

- Plot the 5 eigenvalues (complex numbers) in a 2D plane, i.e. show them as dots in a unit circle (draw the unit circle for reference).
- What is its invariant probability π?
- What is the value of λ_{slem}?

2). Suppose we start with an initial probabilities $\nu = (1,0,0,0,0)$ i.e. we know for sure that the initial state is at $x_0 = 1$. So, at step n, the Markov chain state follows a distribution $\mu_n = \nu \cdot K^n$. We compute the distance between μ_n and π by TV-norm,

$$d_{\text{TV}}(n) = ||\pi - \mu_n||_{\text{TV}} = \frac{1}{2}\sum_{i=1}^{5} |\pi(i) - \mu_n(i)|;$$

or KL-divergence,

$$d_{\text{KL}}(n) = \sum_{i=1}^{5} \pi(i) \log \frac{\pi(i)}{\mu_n(i)}.$$

Plot $d_{\text{TV}}(n)$ and $d_{\text{KL}}(n)$ for the first 1000 steps.

3). Calculate the contraction coefficient for K. Note that contraction coefficient is the maximum TV-norm between any two rows in the transition kernel,

$$C(K) = \max_{x,y} ||K(x,\cdot) - K(y,\cdot)||_{\text{TV}}.$$

One can prove that

$$||\nu_1 \cdot K - \nu_2 \cdot K||_{\text{TV}} \leq C(K)||\nu_1 - \nu_2||_{\text{TV}}$$

As $||\nu_1 - \nu_2||_{\text{TV}} \leq 1$, if $C(K) < 1$ then the convergence rate could be upper bounded by

$$A(n) = ||\nu_1 \cdot K^n - \nu_2 \cdot K^n||_{\text{TV}} \leq C^n(K)||\nu_1 - \nu_2||_{\text{TV}} \leq C^n(K), \quad \forall \nu_1, \nu_2.$$

Plot the bound $C^n(K)$ over $n = 1, ..., 1000$.

4). There is another bound – the Diaconis-Hanlon bound below,

$$B(n) = ||\pi - \nu K^n||_{\text{TV}} \leq \sqrt{\frac{1-\pi(x_0)}{4\pi(x_0)}}\lambda_{\text{slem}}^n$$

where $x_0 = 1$ is the initial state and $\pi(x_0)$ is a target probability at $x = 1$. Plot the real convergence rate $d_{\text{TV}}(n)$ in comparison with $A(n)$ and $B(n)$.

[Draw the three curves in the same figure to compare, then draw a 2nd figure to compare their log-plots as they are exponential rates.]

5). We define a new Markov chain with transition kernel $P = K^n$. Then draw the 5 eigenvalues of P on a 2D complex plane as you did in 1). Show how these eigenvalues move on the plane at three stages: $n = 10, 100, 1000$. You'd better draw the traces of the 5 dots (link the movements of the 5 dots to show their trajectories).

Printout the matrix P for $n = 1000$ and see whether it becomes the "ideal" transition kernel.

Problem 2. Now we consider two more transition matrices

$$K_1 = \begin{pmatrix} 0.1, & 0.4, & 0.3, & 0.0, & 0.2 \\ 0.5 & 0.3, & 0.2, & 0.0, & 0.0 \\ 0.0, & 0.4, & 0.5, & 0.1, & 0.0 \\ 0.0, & 0.0, & 0.0, & 0.5, & 0.5 \\ 0.0, & 0.0, & 0.0, & 0.7, & 0.3 \end{pmatrix}, \quad K_2 = \begin{pmatrix} 0.0, & 0.0, & 0.0, & 0.4, & 0.6 \\ 0.0 & 0.0, & 0.0, & 0.5, & 0.5 \\ 0.0, & 0.0, & 0.0, & 0.9, & 0.1 \\ 0.0, & 0.2, & 0.8, & 0.0, & 0.0 \\ 0.3, & 0.0, & 0.7, & 0.0, & 0.0 \end{pmatrix}$$

1) Are K_1 and K_2 irreducibly, aperiodic?

2) Printout the 5 eigenvalues and 5 eigenvectors of the two matrices.

3) How many and what are the invariant probabilities for each matrix?

Problem 3. A Markov chain returning time $\tau_{\text{ret}}(i)$ is the minimum steps that a Markov chain returns to state i after leaving i. Suppose we consider a random walk in the countable set of non-negative numbers $\Omega = \{0, 1, 2, ..., \}$. At a step, the Markov chain state $x_t = n$, it has probability α to go up (i.e. $x_{t+1} = n+1$) and probability $1-\alpha$ to return to $x_{t+1} = 0$. Calculate the probability for returning to state 0 in finite step

$$Prob(\tau_{\text{ret}}(0) < \infty) = \sum_{\tau(0)=1}^{\infty} Prob(\tau(0)).$$

Calculate the expected return time

$$E[\tau_{\text{ret}}(0)]$$

Problem 4. Let Ω be a finite state space with $|\Omega| = N$ states, and P an $N \times N$ Markov kernel on Ω with invariant probability π. (note that P observes the global balance equations, not necessarily the detailed balance equations) We define a reversed chain with kernel Q as a stochastic matrix that satisfies the following equations,

$$\pi(x)Q(x,y) = \pi(y)P(y,x), \quad \forall x, y.$$

Show that π is also an invariant probability of Q.

Problem 5. In a finite state space Ω, suppose we run two Markov chains $\{X_t\}_{t\geq 0}$ and $\{Y_t\}_{r\geq 0}$ in a coupling mode, i.e. the two chains share the same transition kernel P at each step. Suppose the first chain is a stationary chain with $X_t \sim \pi$, and the second chain has state probability $Y_t \sim \mu_t$. Consider the joint probability $Pr(X_t, Y_t)$, show that

$$||\pi - \mu_t||_{\text{TV}} \leq 1 - Pr(X_t = Y_t)$$

That is, the TV-norm at time t is less than one minus the probability that the two chains collapse (couple).

Hint: the TV norm can be written in the other form:

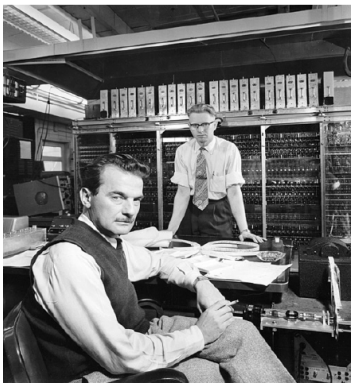
$$||\mu - \nu||_{\text{TV}} = \max_{A \subset \Omega} (\mu(A) - \nu(A)).$$

References

1. Scott Kirkpatrick, MP Vecchi, et al. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

2. D Mitra, F Romeo, and A Sangiovanni-Vincentelli. Convergence and finite-time behavior of simulated annealing. *Advances in applied probability*, 18(3):747–771, 1986.

Chapter 4
Metropolis Methods and Variants



Nicholas Metropolis seated with the MANIAC computer in the background.

"Most of us have grown so blase about computer developments and capabilities — even some that are spectacular — that it is difficult to believe or imagine there was a time when we suffered the noisy, painstakingly slow, electromechanical devices that chomped away on punched cards" - Nicholas Metropolis

Introduction

The Metropolis Algorithm [15, 16] has been declared a member of the list of top ten algorithms of the 20th century by Dongarra and Sullivan [4]. The original algorithm [15] was proposed for equations in chemical physics and has been generalized to its current form by Hastings [10]. In this chapter several variations of the original algorithm are discussed and the concepts of reversible jumps and diffusion are covered. Applications include simple image segmentation, furniture arrangement, and people counting.

4.1 The Metropolis-Hastings Algorithm

The Metropolis-Hastings algorithm is a simple method to take any algorithm that tries to jump from a current state X to a new state Y and slightly modify it by accepting the move with a probability in order for the resulting algorithm to satisfy the detailed balance equation (3.1).


Wilfred Keith Hastings

Example 4.1. The idea of the Metropolis-Hasting algorithm is illustrated in Figure 4.1. Suppose that there is a "proposed" algorithm that is trying to move between states X and Y according to the probabilities displayed in Figure 4.1, left.

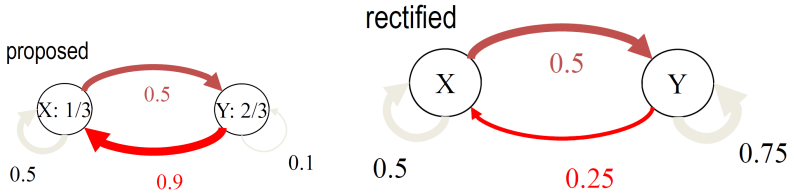


Fig. 4.1: Illustration of the Metropolis-Hastings algorithm. Left: the proposed move between states X and Y does not satisfy detailed balance. Right: the transition probabilities are rectified to satisfy the detailed balance equation.

Since $\pi(X) = 1/3$ and $\pi(Y) = 2/3$, the detailed balance equation is

$$K(X,Y)\frac{1}{3} = K(Y,X)\frac{2}{3}$$

and it is easy to check that it is not satisfied under the proposed transition probabilities.

The move between X and Y is rectified with an acceptance probability, $\alpha = \frac{0.5\times\frac{1}{3}}{0.9\times\frac{2}{3}} = \frac{5}{18}$. Only $\frac{5}{18}$ of the proposals from Y to X are allowed, and all proposals from X to Y are allowed. The acceptance probability rectifies the proposal probability, so that the MC follows the target distribution. The rectified probabilities are shown in Figure 4.1, right.

4.1.1 The original Metropolis-Hastings algorithm

The Metropolis Hastings algorithm resembles importance sampling in the sense that it uses a simpler distribution $Q(x,y)$ to generate proposal samples, which are then reweighted by an acceptance probability. In general the proposal distribution $Q(x,y)$ is simple and samples of y conditional on x can be obtained easily.

Input: Target probability distribution $\pi(x)$, current state $x^{(t)} \in \Omega$, and proposal probability distribution $Q(x,y)$.
Output: New state $x^{(t+1)} \in \Omega$
1. Propose a new state y by sampling from $Q(x^{(t)},y)$.
2. Compute the acceptance probability:

$$\alpha(x,y) = \min\left(1, \frac{Q(y,x)}{Q(x,y)} \cdot \frac{\pi(y)}{\pi(x)}\right) \tag{4.1}$$

3. With probability $\alpha(x,y)$ accept the move and make $x^{(t+1)} = y$, otherwise $x^{(t+1)} = x^{(t)}$.

Fig. 4.2: **One step of the Metropolis-Hastings Algorithm**

Theorem 4.1 (Metropolis-Hastings) *The Metropolis-Hastings algorithm from Figure 4.2 satisfies the detailed balance equation.*

Proof. We have

$$\underbrace{K(x,y)}_{\text{transition probability}} = \begin{cases} \underbrace{Q(x,y)}_{\text{proposal}} \cdot \underbrace{\alpha(x,y)}_{\text{acceptance rate}} = Q(x,y) \cdot \min\Big(1, \underbrace{\frac{Q(y,x)}{Q(x,y)}}_{\text{proposal}} \cdot \underbrace{\frac{\pi(y)}{\pi(x)}}_{\text{verification}}\Big), & \forall y \neq x. \\ 1 - \sum_{y \neq x} Q(x,y)\alpha(x,y), & y = x \end{cases}$$

Since,

$$\alpha(x,y) = \min\Big(1, \frac{Q(y,x)}{Q(x,y)} \cdot \frac{\pi(y)}{\pi(x)}\Big), \text{ and}$$
$$\alpha(y,x) = \min\Big(1, \frac{Q(x,y)}{Q(y,x)} \cdot \frac{\pi(x)}{\pi(y)}\Big),$$

we will have either $\alpha(x,y) = 1$ or $\alpha(y,x) = 1$. Thus, for detailed balance, the left hand side will be

$$\pi(x)K(x,y) = \pi(x)Q(x,y)\alpha(x,y) = \pi(x)Q(x,y)\min\Big(1, \frac{Q(y,x)}{Q(x,y)} \cdot \frac{\pi(y)}{\pi(x)}\Big) = \min\Big(\pi(x)Q(x,y), \pi(y)Q(y,x)\Big)$$

, and the right hand side will be

$$\pi(y)K(y,x) = \pi(y)Q(y,x)\alpha(y,x) = \pi(y)Q(y,x)\min\Big(1, \frac{Q(x,y)}{Q(y,x)} \cdot \frac{\pi(x)}{\pi(y)}\Big) = \min\Big(\pi(x)Q(x,y), \pi(y)Q(y,x)\Big)$$

;

therefore, the detailed balance equation is satisfied.□

4.1.2 Another version of the Metropolis-Hastings algorithm

In many cases the target probability is written as a Gibbs distribution

$$\pi(x) = \frac{1}{Z}e^{-E(x)}$$

, and the normalization constant is hard to compute. Suppose the proposal probability is symmetric($Q(x,y) = Q(y,x)$). Then the acceptance probability becomes

$$\alpha(x,y) = \min\Big(1, \frac{\pi(x)}{\pi(y)}\Big) = \min\big(1, e^{-(E(x)-E(y))}\big) = \min(1, e^{-\Delta E}).$$

Thus,

$$\alpha(x,y) = 1, \quad \text{if } \Delta E < 0, \quad \textit{i.e. } y \text{ is a lower energy state (better) than } x.$$
$$\alpha(x,y) = e^{-\Delta E} < 1, \quad \text{if } \Delta E > 0, \quad \textit{i.e. } y \text{ is a higher energy state (worse) than } x.$$

ΔE is often computed locally as the two states x and y share most of their elements. When the proposal is rejected (with probability $1-\alpha$), the Markov chain stays at state x.

The procedure is illustrated in Figure 4.3. Note that $Q(y,x)$ is designed to make informed proposals that could guide the Markov chain in the correct direction.

Remark 4.1 *We must be careful with the assumption that $Q(x,y) = Q(y,x)$, since it is usually violated at the boundary of the domain Ω.*

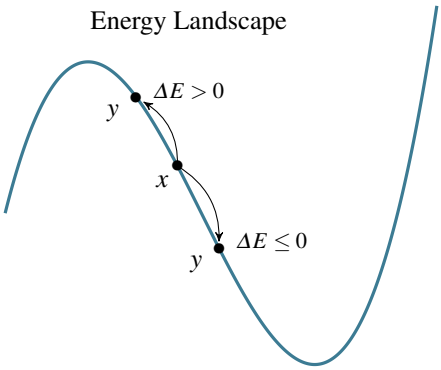


Fig. 4.3: Illustration of the Metropolis-Hastings algorithm variant for Gibbs distributions.

4.1.3 Other acceptance probability designs

There exist other designs for the acceptance rate that guarantee the detailed balance equation, such as

$$\alpha(x,y) = \frac{\pi(y)Q(y,x)}{\pi(y)Q(y,x) + \pi(x)Q(x,y)}$$

or more generally

$$\alpha(x,y) = \frac{s(x,y)}{\pi(x)Q(x,y)}$$

, where $s(x,y)$ is any symmetric function.

Remark 4.2 *Going back to Example 3.2, one could think of $\pi(x)$ as the equilibrium distribution of wealth among a number of families that frequently trade with each other. $Q(x,y)$ can be regarded as trade proposals between the families. In this context, the Metropolis-Hastings choice of acceptance probability maximizes the trade between families among all designs based on $Q(x,y)$ that satisfy detailed balance.*

4.1.4 Key issues in Metropolis design

Intuitively, the Metropolis-Hastings method allows a probability to climb out of the local minima. The key issue in designing the Metropolis algorithm is the design of the proposal probability $Q(x,y)$. Some desirable properties of $Q(x,y)$ are:

i. For any x, the set of reachable states, $\{y, Q(x,y) > 0\}$ is large, so $K(x,y)$ is more connected.
ii. For any x, the probability $Q(x,y)$ if far from uniform (*i.e.* well-informed).

4.2 The Independence Metropolis Sampler

The Independence Metropolis Sampler (IMS) is a Metropolis-Hastings type algorithm in which the proposal is independent of the current state of the chain. It has also been called Metropolized Independent Sampling (Liu [12]). The goal is to simulate a Markov chain $\{X_m\}_{m\geq 0}$ taking values in Ω and having stationary distribution $\pi = (\pi_1, \pi_2, ..., \pi_N)$ (the target probability), with a very large N, e.g. $N = 10^{20}$, in which case it is practically impossible to enumerate all the states. In this case, a new state $j \in \Omega$ is sampled in each step from the proposal probability $q = (q_1, q_2, \ldots, q_N)$ according to $j \sim q_j$, which is then accepted with probability

$$\alpha(i,j) = \min\{1, \frac{q_i}{\pi_i}\frac{\pi_j}{q_j}\}.$$

Therefore, the transition from X_m to X_{m+1} is decided by the transition kernel having the form

$$\mathcal{K}(i,j) = \begin{cases} q_j\alpha(i,j), & j \neq i, \\ 1 - \sum_{k\neq i}\mathcal{K}(i,k), & j = i. \end{cases}$$

The initial state can be either fixed or generated from a distribution whose natural choice in this case is q. In section 4.2.3, we show why it is more efficient to generate the initial state from q instead of choosing it deterministically.

It is easy to show that π is the invariant (stationary) distribution of the chain. In other words, $\pi\mathcal{K} = \pi$. Since from $q > 0$ it follows that $\mathcal{K}$ is ergodic, then π is also the equilibrium distribution of the chain. Therefore, the marginal distribution of the chain at step m, for m large enough, is approximately π.

However, instead of trying to sample from the target distribution π, one may be interested in searching for a state i^* with maximum probability: $i^* = \arg\max_{i\in\Omega}\pi_i$. This is where the mean first hitting time can come into play. $E[\tau(i)]$ is a good measure for the speed of search in general. As a special case we may want to know $E[\tau(i^*)]$ for the optimal state.

As shall be made clear later, a key quantity for this analysis is the probability ratio $w_i = q_i/\pi_i$. It measures how much knowledge the heuristic q_i has about π_i, or in other words how *informed* is q about π for state i. Therefore we define the following concepts.

Definition 4.1. A state i is said to be *over-informed* if $q_i > \pi_i$ and *under-informed* if $q_i < \pi_i$.

There are three special states defined below.

Definition 4.2. A state i is *exactly-informed* if $q_i = \pi_i$. A state i is *most-informed* (or *least-informed*) if it has the highest (or lowest) ratio $w_i = q_i/\pi_i$: $i_{\max} = \arg\max_{i\in\Omega}\{w_i\}$, $i_{\min} = \arg\min_{i\in\Omega}\{w_i\}$.

Liu [12] noticed that the transition kernel can be written in a simpler form by reordering the states increasingly according to their informedness. Since for $i \neq j$, $\mathcal{K}_{ij} = q_j\min\{1, w_i/w_j\}$, if $w_1 \leq w_2 \leq \ldots \leq w_n$ it follows that

$$\mathcal{K}_{ij} = \begin{cases} w_i\pi_j & i < j, \\ 1 - \sum_{k<i}q_k - w_i\sum_{k>i}\pi_k & i = j, \\ q_j = w_j\pi_j & i > j. \end{cases}$$

Without loss of generality, we can assume that the states are indexed such that $w_1 \leq w_2 \leq \ldots \leq w_n$, to allow for this more tractable form of the transition kernel.

4.2.1 The eigenstructure of the IMS

In the last two decades a considerable amount of work has been devoted to studying properties of the IMS. Without trying to be comprehensive, we shall briefly review some of the results. For finite state spaces, Diaconis and Hanlon [3] and Liu [12] proved various upper bounds for the total variation distance between updated and target distributions for the IMS. They showed that the convergence rate of the Markov chain is upper bounded by a quantity that depends on the second largest eigenvalue:

$$\lambda_{slem} = 1 - \min_i\{\frac{q_i}{\pi_i}\}.$$

Remark 4.3 *In the continuous case, denoted by* $\lambda^* = 1 - \inf_x\{\frac{q(x)}{p(x)}\}$*, Mengersen and Tweedie [14] showed that if* λ^* *is strictly less than* 1*, the chain is uniformly ergodic, while if* λ^* *is equal to* 1*, the convergence is not even geometric. Similar results were obtained by Smith and Tierney [19]. These results show that the convergence rate of the Markov chain for the IMS is subject to a* worst-case *scenario. For the finite case, the state corresponding to the smallest probability ratio* q_i/π_i *determines the rate of convergence. That is, just one state from a potentially huge state space decides the rate of convergence of the Markov chain, and this state might even be irrelevant to all of the tasks of MCMC! A similar situation occurs in continuous spaces.*

To illustrate this phenomenon, consider the following simple example.

Example 4.2. Let q and π be two Gaussians having equal variances and the means slightly shifted. Then q, as proposal distribution, will approximate the target π very well. However, it is easy to see that $\inf_x\{q(x)/p(x)\} = 0$, and therefore the IMS will not have a geometric rate of convergence.

This dismal behavior motivated interest for studying the mean first hitting time as a measure of "speed" for Markov chains. This is particularly appropriate where dealing with stochastic search algorithms, when the focus could be on finding individual states rather than on the global convergence of the chain. For instance, in computer vision problems, one is often searching for the most probable interpretation of a scene and, to this end, various Metropolis-Hastings type algorithms can be employed. See Tu and Zhu [20] for examples and discussions. In such a context, it is of interest to find the behavior of the first hitting time of some states, like the modes of the posterior distribution of a scene given the input images.

4.2.2 General first hitting time for finite spaces

Consider an ergodic Markov chain $\{X_m\}_m$ on the finite space $\Omega = \{1,2,\ldots,n\}$. Let $\mathcal{K}$ be the transition kernel, π its unique stationary probability, and q the starting distribution. For each state $i \in \Omega$, the *first hitting time* $\tau_{\text{hit}}(i)$ has been defined in section 3.4.

For any i, denote by $\mathcal{K}_{-i}$ the $(n-1)\times(n-1)$ matrix obtained from $\mathcal{K}$ by deleting the i^{th} column and row, that is, $\mathcal{K}_{-i}(k,j) = \mathcal{K}(k,j), \forall k \neq i,\ j \neq i$. Also let $q_{-i} = (q_1,...,q_{i-1},q_{i+1},...,q_n)$. Then it follows that $P(\tau(i) > m) = q_{-i}\mathcal{K}_{-i}^{m-1}\mathbf{1}$, where $\mathbf{1} := (1,1,...,1)'$. This leads to the following formula for the expectation:

$$E_q[\tau(i)] = 1 + q_{-i}(\mathbf{I} - \mathcal{K}_{-i})^{-1}\mathbf{1}, \tag{4.2}$$

where $\mathbf{I}$ denotes the identity matrix. The existence of the inverse of $\mathbf{I} - \mathcal{K}_{-i}$ is implied by the sub-stochasticity of $\mathcal{K}_{-i}$ and the irreducibility of $\mathcal{K}$ (Bremaud [1]).

More generally, the mean f.h.t of a subset A of Ω is given by

$$E_q[\tau(A)] = 1 + q_{-A}(\mathbf{I} - \mathcal{K}_{-A})^{-1}\mathbf{1}, \quad \forall A \subset \Omega. \tag{4.3}$$

4.2.3 Hitting time analysis for the IMS

Here, we shall capitalize on the previous result to compute the mean first hitting time for the IMS and provide bounds for it, by making use of the eigen-structure of the IMS kernel.

Theorem 4.2 *(Maciuca and Zhu, 2006) Assume a Markov chain starting from q is simulated according to the IMS transition kernel having proposal q and target probability π. Then, using previous notations:*

i) $E[\tau(i)] = \dfrac{1}{\pi_i(1-\lambda_i)}, \forall i \in \Omega,$

ii) $\dfrac{1}{\min\{q_i,\pi_i\}} \leq E[\tau(i)] \leq \dfrac{1}{\min\{q_i,\pi_i\}}\dfrac{1}{1-\|\pi - q\|_{TV}},$

where we define λ_n to be equal to zero and $\|\pi - q\|_{TV}$ denotes the total variation distance between π and q. Equality is attained for the three special states from Definition 4.2.

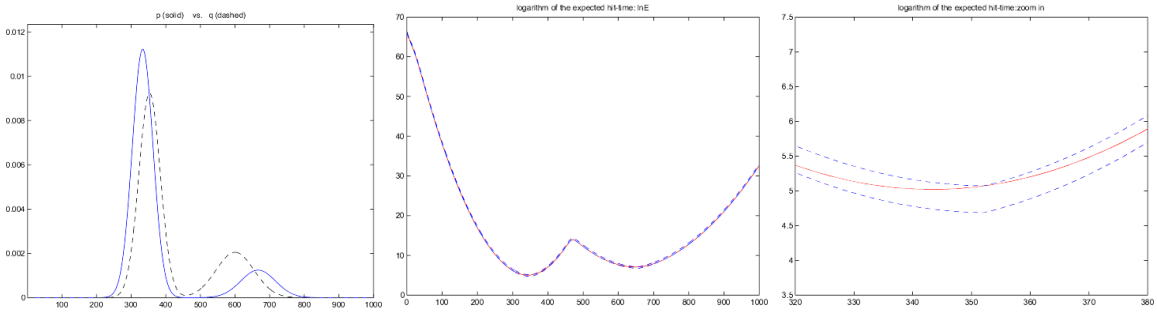
The proof is in [13]. Theorem 4.2 can be extended by considering the first hitting time of some particular sets. The following corollary holds true, proved in [13].

Corollary 4.1. *Let $A \subset \Omega$ of the form $A = \{i+1, i+2, \ldots, i+k\}$, with $w_1 \leq w_2 \leq \ldots \leq w_n$. Denote $\pi_A := \pi_{i+1} + \pi_{i+2} + \ldots + \pi_{i+k}, q_A := q_{i+1} + q_{i+2} + \ldots + q_{i+k}, w_A := q_A/\pi_A$ and $\lambda_A := (q_{i+1} + \ldots + q_n) - (\pi_{i+1} + \ldots + \pi_n)w_A$. Then i) and ii) from Theorem 4.2 hold ad-literam with i replaced by A.*

In the introduction to this section we hinted at showing why generating the initial state from q is preferable to starting from a fixed state $j \neq i$. The following result attempts to clarify this issue.

Proposition 4.3 *Assuming that $w_1 \leq w_2 \leq \ldots \leq w_n$, the following inequalities hold true:*

$$E_1[\tau(i)] \geq E_2[\tau(i)] \geq \ldots \geq E_{i-1}[\tau(i)] \geq E_{i+1}[\tau(i)] = \ldots = E_n[\tau(i)] = E[\tau(i)], \forall i \in \Omega.$$



(a) π (solid) vs q (dashed) (b) $\ln E[\tau(i)]$ and bounds (c) zoom in for (b)

Fig. 4.4: Mean first hitting time and bounds for Example 4.3. © [2006] Springer. Reprinted, with permission, from ref. [13].

Example 4.3. We can illustrate the main results in Theorem 4.2 through a simple example. We consider a space with $n = 1000$ states. Let π and q be mixtures of two discretized Gaussians with tails truncated and then normalized to one. They are plotted as solid (π) and dashed (q) curves in Fig.4.4(a). Fig.4.4(b) plots the logarithm of the expected first hitting-time $\ln E[\tau(i)]$. The lower and upper bounds from Theorem 4.2 are plotted against a logarithmic scale as dashed curves which almost coincide with the hitting-time plot. For better resolution we focused on a portion of the plot around the mode, the three curves becoming more distinguishable in Fig.4.4(c). We can see that the mode $x^* = 333$ has $\pi(x^*) \approx 0.012$ and it is hit in $E[\tau_{x^*}] \approx 162$ times on average for q. This is much smaller than $n/2 = 500$ which would be the average time for exhaustive search. In comparison, for an uninformed (i.e uniform) proposal the result is $E[\tau_{x^*}] = 1000$. Thus, it becomes visible how a "good" proposal q can influence the speed of such a stochastic sampler.

Theorem 4.4 *(Maciuca and Zhu, 2006) Let p and Q be the target probability and the proposal matrix respectively for a Metropolis-Hasting sampler. Let $M = \max_{i,j} Q_{ij}/p_j$ and $m = \min_{i,j} Q_{ij}/p_j$. We assume $m > 0$. Then for any initial distribution q, the expected first hitting times are bounded by*

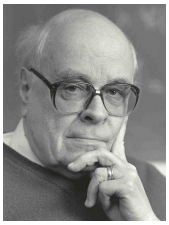
$$p_i + \frac{1-q_i}{M} \leq p_i E_q^Q[\tau(i)] \leq p_i + \frac{1-q_i}{m}, \forall i.$$

Equality is attained if $Q_{ij} = p_j, \forall i, j$.

The proof is again in [13].

4.3 Reversible Jumps and Trans-Dimensional MCMC

There are many cases in which one may need to sample a posterior probability that is defined over a union of spaces of different dimensions. For example, one could define Bayesian models for objects in images, with a variable number of parameters, and may be interested in sampling from such models to estimate the most likely observation for a give image.



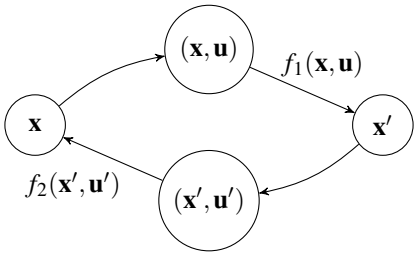
Ulf Grenander

This problem was first proposed in Grenander and Miller 1994 [8] for image analysis and in Green 1995 [7] for Bayesian model selection.

4.3.1 Reversible jumps

Let $\Omega = \cup_{i=1}^{\infty} \Omega_i$ be the solution space written as a union of subspaces of different dimensions, $\dim(\Omega_i) = d_i$, and π be a probability distribution defined on Ω. Reversible jumps are MCMC moves from a state in one space, Ω_i, to another, Ω_j, that satisfy the detailed balance equation with respect to π.


Michael Miller



General case. The reversible jump move $q(\mathbf{x} \to \mathbf{x}')$ from $\mathbf{x} \in \Omega_i$ to $\mathbf{x}' \in \Omega_j$ is obtained by first sampling j from a probability $q(j|i,\mathbf{x})$, sampling an auxiliary vector $\mathbf{u} \in \mathbb{R}^m$ (for some dimension m that needs to be specified) with pdf $q(\mathbf{u}|\mathbf{x})$, and then obtaining $\mathbf{x}'$ as a deterministic function $\mathbf{x}' = f_1(\mathbf{x},\mathbf{u})$. A reverse move $q(\mathbf{x}' \to \mathbf{x})$ can be defined in a similar way, sampling both i with probability $q(i|j,\mathbf{x}')$ and an auxiliary vector $\mathbf{u}' \in \mathbb{R}^{m'}$ from pdf $q(\mathbf{u}'|\mathbf{x}')$. There must be a bijection $f : \Omega_i \times \mathbb{R}^m \to \Omega_j \times \mathbb{R}^{m'}$ such that $f(\mathbf{x},\mathbf{u}) = (\mathbf{x}',\mathbf{u}')$. Therefore, the *dimension matching condition* $d_i + m = d_j + m'$ must be satisfied as well as $\dfrac{d\mathbf{x}'d\mathbf{u}'}{d\mathbf{x}d\mathbf{u}} = \dfrac{\partial f(\mathbf{x},\mathbf{u})}{\partial(\mathbf{x},\mathbf{u})}$. To satisfy detailed balance, the proposed move $q(\mathbf{x} \to \mathbf{x}')$ is accepted with probability

$$\alpha(\mathbf{x} \to \mathbf{x}') = \min\left(1, \frac{q(i|j,\mathbf{x}')q(\mathbf{u}'|\mathbf{x}')\pi(\mathbf{x}')}{q(j|i,\mathbf{x})q(\mathbf{u}|\mathbf{x})\pi(\mathbf{x})}\left|\det\frac{\partial f(\mathbf{x},\mathbf{u})}{\partial(\mathbf{x},\mathbf{u})}\right|\right). \tag{4.4}$$

Expansion-contraction. A special case of reversible jumps are expansion-contraction moves in which $\Omega_j = \Omega_i \times Z$. Starting from $\mathbf{x} \in \Omega_i$, one could choose $\mathbf{u} \in Z$ and f as the identity function, thus obtaining the expansion move $q(\mathbf{x} \to \mathbf{x}') = (\mathbf{x},\mathbf{u})$. Starting from $\mathbf{x}' = (\mathbf{x},\mathbf{u}) \in \Omega_j$, the contraction move just drops the $\mathbf{u}$, and thus $q(\mathbf{x}' \to \mathbf{x}) = \mathbf{x}$. The acceptance probability for the expansion move is then

$$\alpha(\mathbf{x} \to \mathbf{x}') = \min\left(1, \frac{\pi(\mathbf{x}')}{\pi(\mathbf{x})q(\mathbf{u}|\mathbf{x})}\right), \tag{4.5}$$

and for the contraction is

$$\alpha(\mathbf{x}' \to \mathbf{x}) = \min\left(1, \frac{\pi(\mathbf{x})q(\mathbf{u}|\mathbf{x})}{\pi(\mathbf{x}')}\right). \tag{4.6}$$

4.3.2 Toy example: 1D range image segmentation

Figure 4.5 shows an example of a simulated 1D range image $I(x), x \in [0,1]$. It is generated by adding Gaussian noise $N(0,\sigma^2)$ to the original surfaces I_o from Fig. 4.5b. I_o consists of an unknown number of k surfaces which could be either straight lines or circular arcs, separated by $k-1$ change points,

$$0 = x_0 < x_1 < ... < x_k = 1.$$

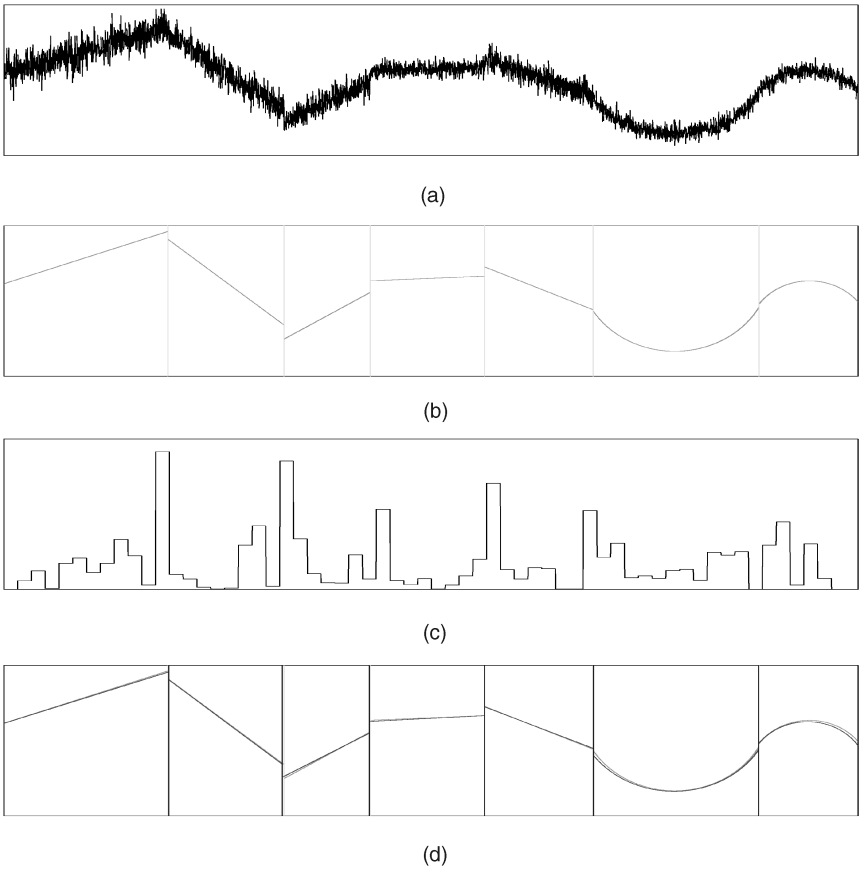


Fig. 4.5: (a) A 1D range image $I(x), x \in [0,1]$ (b) The true segmentation, W_{th}. (c) Edgeness measure $b(x), x \in [0,1]$. A large value $b(x)$ indicates a high probability for x being a change point. (d) The best solution W^* (dark gray) found by the algorithm plotted against W_{th} (light gray). © [2004] IEEE. Reprinted, with permission, from ref. [9].

Let $l_i \in \{line, circle\}$ index the surface type in interval $[x_{i-1}, x_i)$ with parameters $\theta_i, i = 1, ...k$. For a straight line, $\theta = (s, \rho)$ represents the slope s and intercept ρ. For a circular arc, $\theta = (u, v, R)$ represents the center (u, v) and radius R. Thus, the 1D "world scene" is represented by a vector of random variables,

$$W = (k, \{x_i, i = 1, ..., k-1\}, \{(l_i, \theta_i), i = 1, ..., k\}).$$

The surface I_o is fully determined by W with $I_o(x) = I_o(x, l_i, \theta_i), x \in [x_{i-1}, x_i), i = 1, ..., k$.

By the standard Bayesian formulation, we have the posterior probability

$$p(W|I) \propto \exp\left\{ -\frac{1}{2\sigma^2} \sum_{i=1}^{k} \int_{x_{i-1}}^{x_i} (I(x) - I_o(x, l_i, \theta_i))^2 dx \right\} \cdot p(k) \prod_{i=1}^{k} p(l_i) p(\theta_i | l_i). \tag{4.7}$$

The first factor above is the likelihood and the rest are prior probabilities $p(k) \propto \exp(-\lambda_0 k)$ and $p(\theta_i|l_i) \propto \exp(-\lambda \# \theta_i)$, which penalizes the number of parameters $\# \theta_i$. $p(l_i)$ is a uniform probability on the lines and arcs. Thus, the energy function is defined by

$$E(W) = \frac{1}{2\sigma^2} \sum_{i=1}^{k} \int_{x_{i-1}}^{x_i} (I(x) - I_o(x, l_i, \theta_i))^2 dx + \lambda_o k + \lambda \sum_{i=1}^{k} \# \theta_i. \tag{4.8}$$

One problem with this is that W does not have a fixed dimension. The probability $p(W|I)$ (or the energy $E(W)$) is thus distributed over a countable number of subspaces of varying dimension. The next subsection briefly introduces the jump-diffusion process for exploring such a solution space.

4.3.2.1 Jump-diffusion


Peter Green

Consider a solution space $\Omega = \cup_{n=1}^{\infty} \Omega_n$, where the subspace indexes $n = (k, l_1, ..., l_k)$ contain the discrete variables of the model. To traverse the solution space the algorithm needs two types of moves: reversible jumps between different subspaces and stochastic diffusions within each continuous subspace.

1. Reversible jumps. Let $W = (i, \mathbf{x})$, be the state of a Markov chain at time t, where $\mathbf{x} \in \Omega_i$ represents the continuous variables for the solution. In an infinitesimal time interval dt, the Markov chain jumps to a new state $W' = (j, \mathbf{x}')$ in another subspace $\Omega_j, j \neq i$. There are three types of jumps: 1) switching a line to a circular arc or vice versa, 2) merging two adjacent intervals to a line or a circle, and 3) splitting an interval into two intervals (lines or circles). The jump is realized by a Metropolis move [15] that proposes to move from W to W' by a forward proposal probability $q(W'|W) = q(i \to j)q(\mathbf{x}'|j)$. The backward proposal probability is $q(W|W') = q(j \to i)q(\mathbf{x}|i)$. The forward proposal is accepted with probability

$$\alpha(W \to W') = \min\left(1, \frac{q(j \to i)q(\mathbf{x}|i)\pi(W')}{q(i \to j)q(\mathbf{x}'|j)\pi(W)}\right). \tag{4.9}$$

The dimension is matched in the above probability ratio.

2. Stochastic diffusions. Within each subspace Ω_n with $n = (k, l_1, ..., l_k)$ fixed, the energy functional $E(\mathbf{x})$ is

$$E(\mathbf{x}) = E(x_1, ..., x_{k-1}, \theta_1, ..., \theta_k) = \frac{1}{2\sigma^2} \sum_{i=1}^{k} \int_{x_{i-1}}^{x_i} (I(x) - I_o(x, l_i, \theta_i))^2 dx + \text{const.}$$

We adopt a stochastic diffusion (or Langevin) equation to explore the subspace. The Langevin equations are the steepest descent PDE (partial differential equations) driven by Brownian motions, $dB(t)$, with temperature T. Let $\mathbf{x}(t)$ denote the variables at time t, and then

$$d\mathbf{x}(t) = -\frac{dE(\mathbf{x})}{d\mathbf{x}}dt + \sqrt{2T(t)}dw_t,\ dw_t \sim N(0, (dt)^2). \tag{4.10}$$

For example, the motion equation of a change point x_i is

$$\frac{dx_i(t)}{dt} = \frac{1}{2\sigma^2}[(I(x) - I_o(x, l_{i-1}, \theta_{i-1}))^2 - (I(x) - I_o(x, l_i, \theta_i))^2] + \sqrt{2T(t)}N(0, 1).$$

This is the 1D version of the region competition equation [23]. The movement of the point x_i is driven by the fitness of data $I(x_i)$ to the surface models of the two adjacent intervals, plus a Brownian motion. In practice, the Brownian motion is found to be useful in avoiding local pitfalls.

For computing the parameters $\theta_i, i = 1, ..., k$, running the diffusion is more robust and often faster than fitting the best θ_i for each interval $[x_{i-1}, x_i)$ deterministically since the deterministic fit is an "overcommitment." This is especially true when the current interval contains more than one object.

It is well-known [6] that the continuous Langevin equations in (4.10) simulate Markov chains with stationary density $p(\mathbf{x}) \propto \exp(-E(\mathbf{x})/T)$. This is the posterior probability within subspace Ω_n at temperature T.

3. The coordination of jumps and diffusions. The continuous diffusions are interrupted by jumps at time instances $t_1 < t_2 < \ldots < t_M \ldots$ given as Poisson events. In practice, the diffusion always runs at a discrete time step, δt. Thus, the discrete waiting time τ_j between two consecutive jumps is

$$w = \frac{t_{j+1} - t_j}{\delta_t} \sim p(w) = e^{-\tau}\frac{\tau^w}{w!},$$

where the expected waiting time $E(w) = \tau$ controls the frequency of jumps. Both jump and diffusion processes should follow an annealing scheme for lowering the temperature gradually.

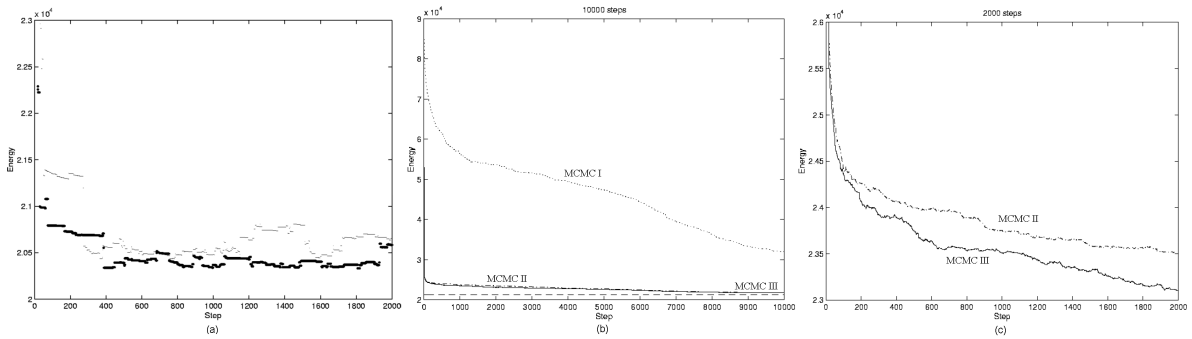


Fig. 4.6: (a) Diffusion with jumps. Energy plots for two trials (MCMC II, thin and MCMC III, thick) of the jump-diffusion processes. Continuous energy changes in diffusion are interrupted by energy jumps. (b) Average energy plot. Comparison of the energy curves in the first 10,000 steps of three Markov chains, MCMC I, II, and III, averaged over 100 randomly generated signals. (c) Zoomed-in view of MCMC II and III for the first 2,000 steps. Note that the energy scale is different from (b). © [2004] IEEE. Reprinted, with permission, from ref. [9].

To illustrate, Fig. 4.6(a) shows two trials of the jump-diffusion process running on the input 1D range data in Fig. 4.5(a). The energy plots go up and down (i.e., the algorithm is not greedy) and the continuous energy curves (diffusion) are interrupted by jumps.

4. Reversibility and global optimization. From an engineering point of view, the most important property of the jump-diffusion process is that it simulates Markov chains that traverse the complex solution space. This property distinguishes it from greedy and local methods. In theory, this Markov chain samples from the posterior probability $p(W|I)$ over the solution space Ω [8]. With an annealing scheme, it can theoretically achieve the globally optimal solution with probability close to one. The reversibility of the jumps may not be a necessary condition; however, it is a useful tool for achieving irreducibility of the Markov chain in the complex solution space.

5. The speed bottlenecks. Conventional jump-diffusion designs are limited by their computing speed. However, this problem can be overcome by designing better proposal probabilities, as we shall show in the next section. We observed that the bottlenecks in the jumps are affected by the design of the proposal probabilities. In (4.9), a proposal probability $q(\mathbf{x}'|j)$ in interval $[x_{i-1}, x_i)$ can be divided into three cases: 1) switching to a new model with $\mathbf{x}' = \theta_i$, 2) merging to form a new interval $[x_{i-2}, x_i)$ with type l and parameter $\mathbf{x}$, and 3) splitting to form two new intervals with models (l_a, θ_a) and (l_b, θ_b) respectively.

$$q(\mathbf{x}|m) = \begin{cases} q(\theta_i|l_i, [x_{i-1}, x_i)) & \text{switch } [x_{i-1}, x_i) \text{ to model } (l_i, \theta_i) \\ q(\theta|l, [x_{i-2}, x_i)) & \text{merge to a model } (l, \theta) \\ q(x|[x_{i-1}, x_i))q(\theta_a|l_a, [x_{i-1}, x))q(\theta_b|l_b, [x, x_i)) & \text{split } [x_{i-1}, x_i) \text{ into } (l_a, \theta_a) \text{ and } (l_b, \theta_b) \text{ at } x. \end{cases}$$

4.4 Application: Counting People

An application of the Metropolis-Hastings algorithm is presented in [5] for detecting and counting people in crowded scenes. The representation of the problem is in the framework of Marked Point Processes, where each person is represented as a marked point, s, consisting of a spatial process representing the image location, $p \in \mathbb{R}^2$, and a *marked process* denoted by $m = (w, h, \theta, j)$ representing the width, height, orientation and shape of the person. Together these form the marked point $s = (p, (w, h, \theta, j))$.

4.4.1 Marked point process model

The model assumes the marked point depends on the spatial location from the marked process, and therefore

$$\pi(s) = \pi(p)\pi(w,h,\theta,j|p)$$

for every marked point $s = (p,(w,h,\theta,j))$.

The prior for the point process $\pi(p)$ is a homogeneous Poisson point process, i.e. the total number of points follows a Poisson distribution and given the number of points, their location is uniform inside the region. A simulation of the prior model is shown in Figure 4.7.

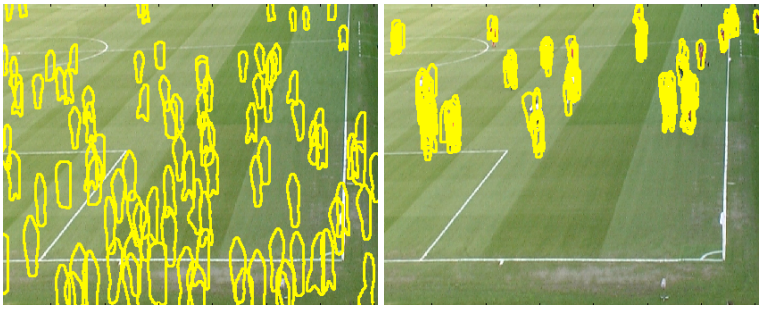


Fig. 4.7: Samples from the Poisson point process prior $\pi(s)$. © [2009] IEEE. Reprinted, with permission, from ref. [5].

The conditional mark process $\pi(w,h,\theta,j|p)$ is represented with independent Gaussians for the width, height and orientation, depending on the image location p, and a uniform distribution for the shape j from a set of possible shapes. The values of the spatially dependent means and variances are stored as look-up tables. The set of possible shapes is learned by Expectation Maximization as a mixture of Bernoulli templates from a set of manually segmented boxes.

The input image is processed so that a foreground *mask* data y is obtained where pixel i has $y_i = 1$ if the pixel is a foreground pixel and $y_i = 0$ for background. Given the current point configuration $s_1,...,s_n$, a *label* image is constructed where the pixels are labeled foreground if any of the shapes corresponding to the n marked points covers it and background otherwise. In reality both the mask and label images have soft labels and contain values in the interval $[0,1]$.

The likelihood is

$$\log \mathcal{L}(Y|X) = \sum (x_i \log y_i + (1-x_i)\log(1-y_i)).$$

4.4.2 Inference by MCMC

Given an input image with a foreground mask y, the most likely marked point configuration is obtained by Maximum A Posteriori (MAP) estimation, i.e. maximization of the posterior probability function $\pi(s|y) = \pi(y|x(s))\pi(s)$.

This is achieved using three types of reversible moves:

- Birth proposal. A marked point is proposed at uniform locations according to the foreground mask. The width, height and orientation are sampled from the respective Gaussians conditional on the point location. The type of the shape is chosen uniformly at random from the set of learned shape prototypes. The reverse of this mode is the death proposal.
- Death proposal. One point at random is removed in opposite fashion to the birth proposal.
- Update proposal. One marked point is selected at random and its position or mark parameters are modified. The position is modified as a random walk. The mark is modified by selecting one of the three parameters and sampling it from the conditional distribution given the current position, or selecting the shape type at random from the possible shapes.

The three types of moves are used with probability 0.4,0.2,0.4 respectively. About 500-3000 moves are needed for one image, starting from an empty configuration. More moves are needed for more crowded scenes.

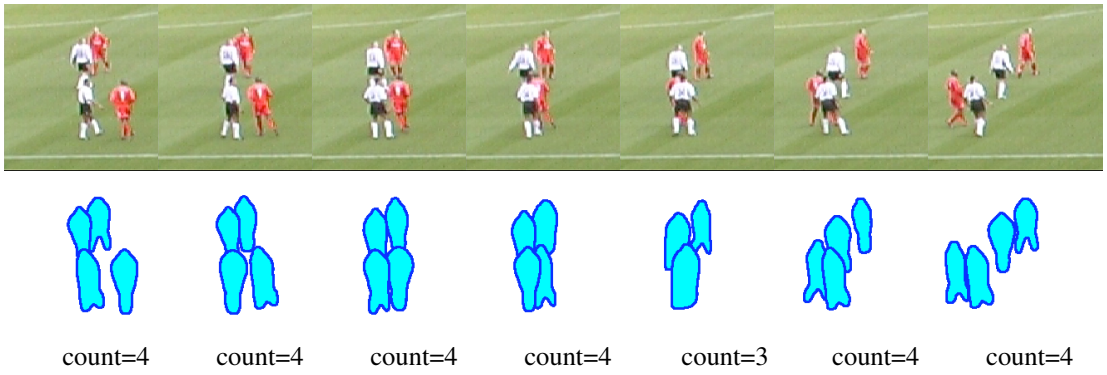


Fig. 4.8: Results on seven frames of the VSPETS sequence. The counting is precise until there is significant overlap. © [2009] IEEE. Reprinted, with permission, from ref. [5].

Fig. 4.9: Results on images of the CAVIAR dataset. © [2009] IEEE. Reprinted, with permission, from ref. [5].

4.4.3 Results

The MCMC method was tested on two benchmark sequences with ground truth annotation: the EU CAVIAR dataset [1] and the VSPETS soccer sequence [2]. Examples of results are shown in Figures 4.8 and 4.9.

4.5 Application: Furniture Arrangement

One application of the Metropolis-Hastings algorithm is furniture arrangement [21], illustrated in Figure 4.10. The process consists of two stages: (1) the extraction of spatial, hierarchical, and pairwise relationships from positive examples, and (2) the synthesis of novel furniture arrangements through optimization.

[1] http://homepages.inf.ed.ac.uk/rbf/CAVIAR/

[2] http://www.cvg.cs.rdg.ac.uk/VSPETS/vspets-db.html

Fig. 4.10: Left: Arbitrary initialization of the furniture layout. Middle and right: Two synthesized furniture arrangements optimized according to ergonomic criteria such as unobstructed accessibility and visibility. © [2011] ACM. Reprinted, with permission, from ref. [21].

1. Object representation. Optimizing furniture into a realistic and functional configuration relies on modeling various interaction factors, such as pairwise furniture relationships, spatial relationships with respect to the room, and other human factors.

Bounding surfaces: Each object in the scene is represented by a set of bounding surfaces. Apart from the top and bottom surfaces, there is a "back" surface for every object, which is the surface closest to a wall. Other surfaces are labeled as "non-back" surfaces. The back surface is used to define a reference plane for assigning other attributes.

Center and orientation: The key attributes of an object are center and orientation, denoted respectively by (p_i, θ_i), where p_i denotes the (x,y) coordinates and θ_i denotes the angle relative to the nearest wall (defined as the angle between the nearest wall and the back surface).

Accessible space: For each surface of the object, a corresponding accessible space is assigned.We define a_{ik} to be the center of coordinates of accessible space k of object i. The diagonal of the region is measured by ad_{ik}, which is used to measure how deep other objects can penetrate into the space during optimization. The size of the accessible space is set from available examples or given as input related to the size of a human body. If the space is very close to the wall in all examples, then the corresponding surface need not be accessible; otherwise, it is set to be the dimension of an average-sized adult if such a measurement is not given.

Viewing frustum: For some objects, such as the televisions and paintings, the frontal surface must be visible. A viewing frustum is assigned to this particular surface, which for an object i, is approximated by a series of rectangles with center coordinates v_{ik}, where k is the rectangle index. vd_{ik} is the diagonal of the rectangle, which is useful in defining the penetration cost akin to that for the accessible space.

Other attributes: Other attributes involved in the optimization process are the distance from p_i to the nearest wall, which is defined as d_i and the diagonal, b_i, from p_i to the corner of the bounding box.Also recorded is the z-position, z_i, of the object.

2. Cost function. The goal of the optimization process is to minimize a cost function that characterizes realistic, functional furniture arrangements. Although it is often difficult to quantify the "realism" or "functionality" of a furniture arrangement, the following basic criteria should not be violated.

Accessibility: A furniture object must be accessible in order to be functional [2, 17]. To favor accessibility, the cost increases whenever any object moves into the accessible space of another object. Suppose object i overlaps with the accessible space k of object j, then the accessibility cost is defined as

$$C_{\text{a}}(\phi) = \sum_i \sum_j \sum_k \max\left[0, 1 - \frac{\|p_i - a_{jk}\|}{b_i + ad_{jk}}\right]. \tag{4.11}$$

There are other cost terms for visibility, pathways connecting doors, pairwise constraints between certain furniture objects (eg. TV facing a sofa), and a prior to encourage configurations seen in the training examples.

3. Furniture arrangement optimization. The search space of the problem is highly complex as objects are interdependent in the optimization process. The furniture positions and orientations depend on numerous factors, such as whether the object should be visible or accessible. It is very difficult to have a global optimization scheme or a closed-form solution that yields a unique optimum.

To tackle this problem, simulated annealing [11] is used with a Metropolis-Hastings state-search step [10, 15] to search for a good approximation to the global optimum. Note, however, that given a room, a set of furniture objects, and the prior spatial and hierarchical relationships, numerous acceptably-good configurations will be possible. This is

the rationale for finding a good approximation in a reasonably short time, rather than searching exhaustively over the complex search space in order to find the global optimum of the cost function.

To explore the space of possible arrangements effectively, the proposed moves $\phi \rightarrow \phi'$ involve both local adjustment, which modify the current arrangement, and a global reconfiguration step that swaps objects, thereby altering the arrangement significantly. There are three types of moves: translation and rotation, swapping objects and moving pathway control points. More details are given in [21]. Examples of furniture arrangement results are given in Fig. 4.11.

Synthesis 1 Synthesis 2 Synthesis 3

Fig. 4.11: Selected views of synthesized results. Top to bottom: Gallery, Resort, Restaurant. © [2011] ACM. Reprinted, with permission, from ref. [21].

With the aforementioned moves, given a floor-plan and a fixed number of furniture objects that define the solution space, the configuration of a furniture object (p_i, θ_i) has a positive probability to move to any other configuration (p_i', θ_i'). Given the annealing schedule, the solution space is explored more extensively with larger moves early in the optimization, and the furniture configuration is more finely tuned with smaller moves towards the end.

4.6 Application: Scene Synthesis

Another application of Metropolis-Hastings is human-centric scene synthesis [18]. Here, an attributed spatial And-Or graph (S-AOG), illustrated in Figure 4.12 is used to represent indoor scenes. The model distributions are learned from an indoor scene dataset and new layouts can be sampled using Markov Chain Monte Carlo.

Representation of Indoor Scenes. An attributed S-AOG [22] was used to represent an indoor scene. An attributed S-AOG is a probabilistic grammar model with attributes on the terminal nodes. It combines i) a probabilistic context free grammar (PCFG), and ii) contextual relations defined on an Markov Random Field (MRF), i.e., the horizontal links among the nodes. The PCFG represents the hierarchical decomposition from scenes (top level) to objects (bottom level) by a set of terminal and non-terminal nodes, whereas contextual relations encode the spatial and functional relations through horizontal links. The structure of S-AOG is shown in Figure 4.12.

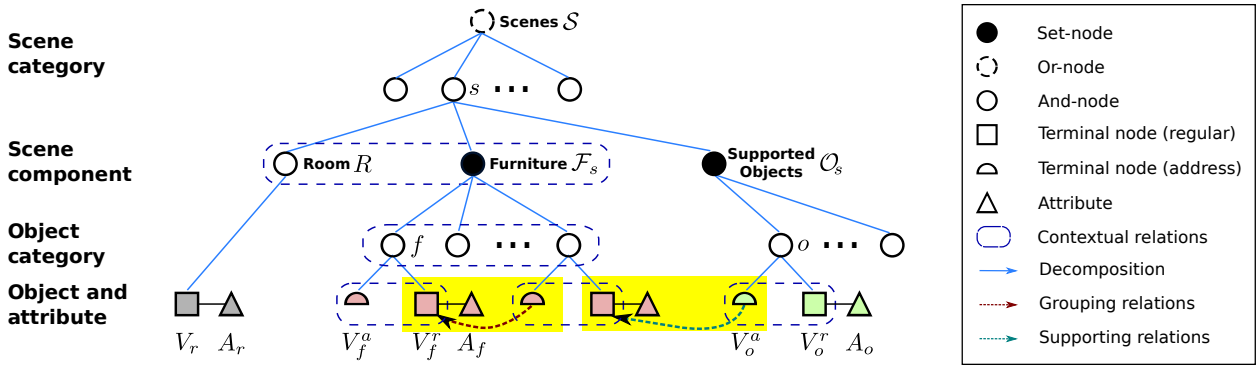


Fig. 4.12: Scene grammar as an attributed Spatial And/Or Graph. © [2018] IEEE. Reprinted, with permission, from ref. [18].

Formally, an S-AOG is defined as a 5-tuple: $\mathscr{G} = \langle S, V, R, P, E \rangle$, where S is the root node of the scene grammar, V the vertex set, R the production rules, P the probability model defined on the attributed S-AOG, and E contains the contextual relations represented as horizontal links between nodes in the same layer. [3]

Vertex Set V can be decomposed into a finite set of non-terminal and terminal nodes: $V = V_{NT} \cup V_T$.

- $V_{NT} = V^{And} \cup V^{Or} \cup V^{Set}$. The set of non-terminal nodes consists of three subsets. i) A set of **And-nodes** V^{And}, in which each node represents a decomposition of a larger entity (e.g., a bedroom) into smaller components (e.g., walls, furniture and supported objects). ii) A set of **Or-nodes** V^{Or}, in which each node branches to alternative decompositions (e.g., an indoor scene can be a bedroom or a living room), enabling the algorithm to reconfigure a scene. iii) A set of **Set nodes** V^{Set}, in which each node represents a nested And-Or relation: a set of Or-nodes serving as child branches are grouped by an And-node, and each child branch may include different numbers of objects.
- $V_T = V_T^r \cup V_T^a$. The set of terminal nodes consists of two subsets of nodes: regular nodes and address nodes. i) A **regular terminal node** $v \in V_T^r$ represents a spatial entity in a scene (e.g., an office chair in a bedroom) with attributes. Here, the attributes include internal attributes A_{int} of object sizes (w, l, h), external attributes A_{ext} of object position (x, y, z) and orientation ($x-y$ plane) θ, and sampled human positions A_h. ii) To avoid excessively dense graphs, an **address terminal node** $v \in V_T^a$ is introduced to encode interactions that only occur in a certain context but are absent in all others. It is a pointer to regular terminal nodes, taking values in the set $V_T^r \cup \{\text{nil}\}$, representing supporting or grouping relations as shown in Figure 4.12.

Contextual Relations E among nodes are represented by the horizontal links in S-AOG forming MRFs on the terminal nodes. To encode the contextual relations, different types of potential functions are defined for different cliques. The contextual relations $E = E_f \cup E_o \cup E_g \cup E_r$ are divided into four subsets: i) relations among furniture E_f; ii) relations between supported objects and their supporting objects E_o (e.g., a monitor on a desk); iii) relations between objects of a functional pair E_g (e.g., a chair and a desk); and iv) relations between furniture and the room E_r. Accordingly, the cliques formed in the terminal layer could also be divided into four subsets: $C = C_f \cup C_o \cup C_g \cup C_r$. Instead of directly capturing the object-object relations, the potentials are computed using affordances as a bridge to characterize the object-human-object relations.

A hierarchical parse tree pt is an instantiation of the S-AOG by selecting a child node for the Or-nodes as well as determining the state of each child node for the Set-nodes. A parse graph pg consists of a parse tree pt and a number of contextual relations E on the parse tree: $pg = (pt, E_{pt})$. Figure 4.13 illustrates a simple example of a parse graph and four types of cliques formed in the terminal layer.

Probabilistic Formulation of S-AOG. A scene configuration is represented by a parse graph pg, including objects in the scene and associated attributes. The prior probability of pg generated by an S-AOG parameterized by Θ is formulated as a Gibbs distribution:

[3] We use the term "vertices" instead of "symbols" (in the traditional definition of PCFG) to be consistent with the notations in graphical models.

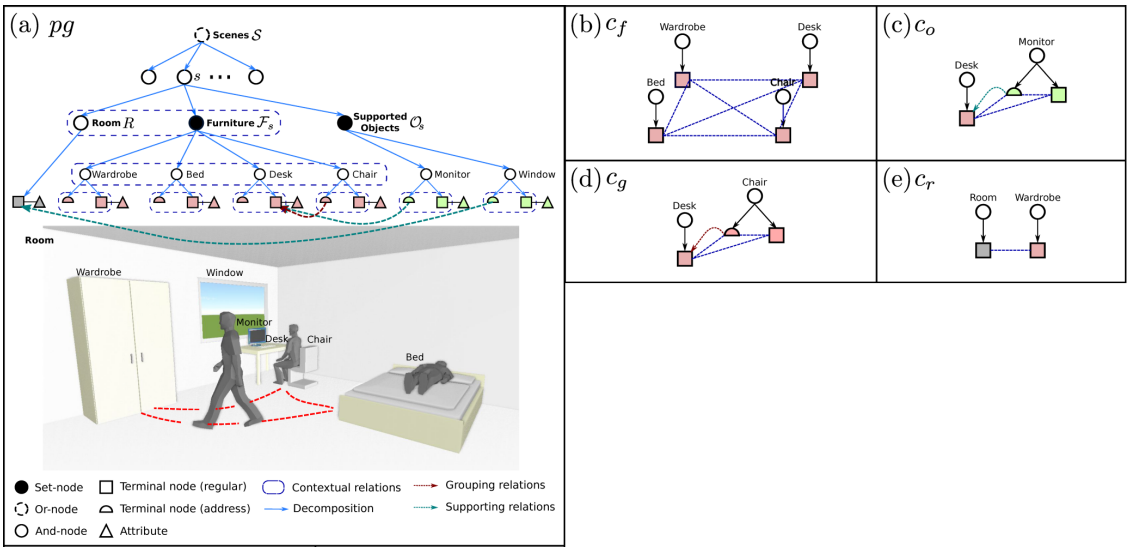


Fig. 4.13: (a) A simplified example of a parse graph of a bedroom. The terminal nodes of the parse graph form an MRF in the terminal layer. Cliques are formed by the contextual relations projected to the terminal layer. Examples of the four types of cliques are shown in (b)-(e), representing four different types of contextual relations. © [2018] IEEE. Reprinted, with permission, from ref. [18].

$$p(pg|\Theta) = \frac{1}{Z}\exp\{-\mathscr{E}(pg|\Theta)\} = \frac{1}{Z}\exp\{-\mathscr{E}(pt|\Theta) - \mathscr{E}(E_{pt}|\Theta)\}, \tag{4.12}$$

where $\mathscr{E}(pg|\Theta)$ is the energy function of a parse graph, $\mathscr{E}(pt|\Theta)$ is the energy function of a parse tree, and $\mathscr{E}(E_{pt}|\Theta)$ is the energy term of the contextual relations.

$\mathscr{E}(pt|\Theta)$ can be further decomposed into the energy functions of different types of non-terminal nodes, and the energy functions of internal attributes of both regular and address terminal nodes:

$$\mathscr{E}(pt|\Theta) = \underbrace{\sum_{v\in V^{Or}} \mathscr{E}_{\Theta}^{Or}(v) + \sum_{v\in V^{Set}} \mathscr{E}_{\Theta}^{Set}(v)}_{\text{non-terminal nodes}} + \underbrace{\sum_{v\in V_T^r} \mathscr{E}_{\Theta}^{A_{in}}(v)}_{\text{terminal nodes}}, \tag{4.13}$$

where the choice of the child node of an Or-node $v \in V^{Or}$ and the child branch of a Set-node $v \in V^{Set}$ follow different multinomial distributions. Since the And-nodes are deterministically expanded, we do not have an energy term for the And-nodes here. The internal attributes A_{in} (size) of the terminal nodes follow a non-parametric probability distribution learned by kernel density estimation.

$\mathscr{E}(E_{pt}|\Theta)$ combines the potentials of the four types of cliques formed in the terminal layer, integrating human attributes and external attributes of regular terminal nodes:

$$p(E_{pt}|\Theta) = \frac{1}{Z}\exp\{-\mathscr{E}(E_{pt}|\Theta)\} = \prod_{c\in C_f}\phi_f(c)\prod_{c\in C_o}\phi_o(c)\prod_{c\in C_g}\phi_g(c)\prod_{c\in C_r}\phi_r(c). \tag{4.14}$$

Synthesizing Scene Configurations. Synthesizing scene configurations is accomplished by sampling a parse graph pg from the prior probability $p(pg|\Theta)$ defined by the S-AOG. The structure of a parse tree pt (i.e., the selection of Or-nodes and child branches of Set-nodes) and the internal attributes (sizes) of objects can be easily sampled from the closed-form distributions or non-parametric distributions. However, the external attributes (positions and orientations) of objects are constrained by multiple potential functions, hence they are too complicated to be directly sampled from. Here, a Markov chain Monte Carlo (MCMC) sampler is used to draw a typical state in the distribution. The process of each sampling can be divided into two major steps:

1. Directly sample the structure of pt and internal attributes A_{in}: (i) sample the child node for the Or-nodes; (ii) determine the state of each child branch of the Set-nodes; and (iii) for each regular terminal node, sample the sizes and human positions from learned distributions.
2. Use an MCMC scheme to sample the values of address nodes V^a and external attributes A_{ex} by making proposal moves. A sample will be chosen after the Markov chain converges.

Two simple types of Markov chain dynamics are used at random with probabilities $q_i, i = 1,2$ to make proposal moves:

- Dynamics q_1: translation of objects. This dynamic chooses a regular terminal node, and samples a new position based on the current position x: $x \rightarrow x + \delta x$, where δx follows a bivariate normal distribution.
- Dynamics q_2: rotation of objects. This dynamic chooses a regular terminal node, and samples a new orientation based on the current orientation of the object: $\theta \rightarrow \theta + \delta\theta$, where $\delta\theta$ follows a normal distribution.

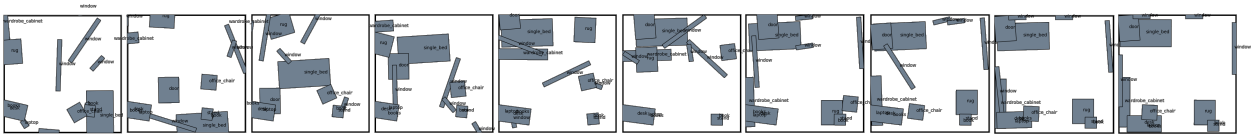
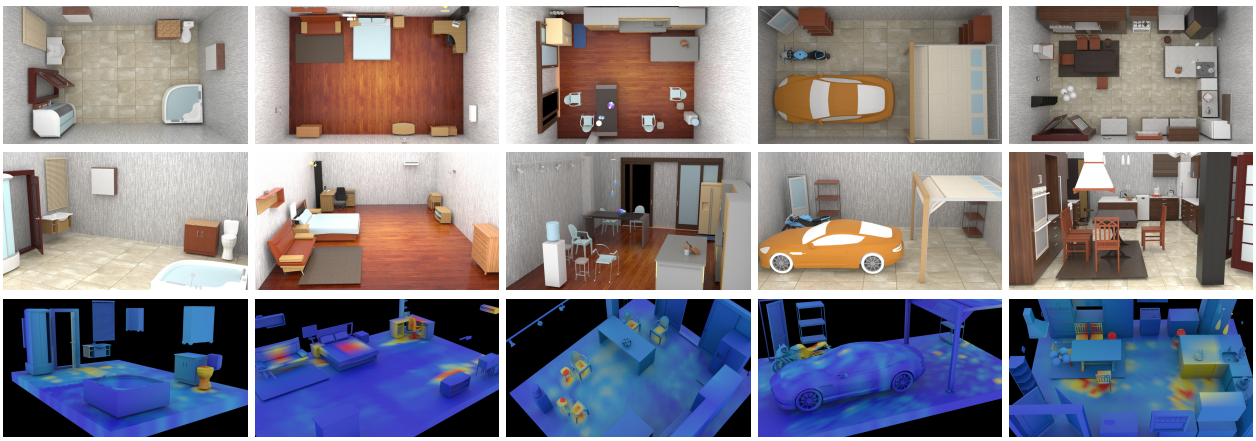


Fig. 4.14: Example of MCMC sampling. From left to right: scene configurations obtained during the simulated annealing. © [2018] IEEE. Reprinted, with permission, from ref. [18].

Adopting the Metropolis-Hastings algorithm, the proposed new parse graph pg' is accepted according to the following acceptance probability:

$$\alpha(pg'|pg,\Theta) = \min(1, \frac{p(pg'|\Theta)p(pg|pg')}{p(pg|\Theta)p(pg'|pg)}) = \min(1, \exp(\mathscr{E}(pg|\Theta) - \mathscr{E}(pg'|\Theta))), \tag{4.15}$$

where the proposal probability rate is canceled since the proposal moves are symmetric in probability. A simulated annealing scheme is adopted to obtain high probability samples. The process is illustrated in Figure 4.14 and high probability samples are shown in Figure 4.15.



bathroom bedroom dining room garage kitchen

Fig. 4.15: Examples of scenes in five different categories. Top: top-view. Middle: a side-view. Bottom: affordance heatmap. © [2018] IEEE. Reprinted, with permission, from ref. [18].

4.7 Exercises

Problem 1. Consider a space with $n = 1000$ states $1, 2, ..., 1000$ as illustrated in Example 4.3. The probability π is a mixture of Gaussians with means 330 and 670, standard deviations 40 and 50 and weights 0.9 and 0.1 respectively. The proposal probability q is also a mixture of Gaussians with means 350 and 600, standard deviations 40 and 50, and weights 0.75 and 0.25 respectively. Both probabilities are discretized to the states $1, ..., 1000$ and normalized to 1.

a) Estimate the first hitting times $E[\tau(i)]$ from all the states using 100 Markov chains for each i. Plot the estimated hitting times vs i.
b) Compute the first hitting times for all states and plot them together with the estimated times and the bounds from Theorem 4.2.

Problem 2. Suppose we have a simple 1D range image obtained as $y(x) = \alpha|x| + \varepsilon, x \in \{-100, .-99, ..., 100\}$, where α controls the signal strength and $\varepsilon \sim \mathcal{N}(0,1)$. Implement a simple reversible jump-diffusion algorithm for segmenting the image in up to two pieces. The reversible jumps are between the one segment and the two segment solutions. The diffusion in the two segment space moves the common endpoint (break) position. Use ordinary least squares to fit the best segmentation when the number of segments and the break positions are given. Try different signals strengths $\alpha \in \{0.01, 0.003, 0.1, 0.3, 1\}$ and plot the average energy (4.8) from 10 independent runs vs computation time (in seconds).

References

1. Pierre Bremaud. *Markov chains: Gibbs fields, Monte Carlo simulation, and queues*, volume 31. Springer, 1999.
2. Francis DK Ching and Corky Binggeli. *Interior design illustrated*. John Wiley & Sons, 2012.
3. Persi Diaconis and Phil Hanlon. Eigen-analysis for some examples of the metropolis algorithm. *Contemporary Mathematics*, 138:99–117, 1992.
4. Jack Dongarra and Francis Sullivan. Guest editors introduction: The top 10 algorithms. *Computing in Science & Engineering*, 2(1):22–23, 2000.
5. Weina Ge and Robert T Collins. Marked point processes for crowd counting. In *CVPR*, pages 2913–2920. IEEE, 2009.
6. Stuart Geman and Chii-Ruey Hwang. Diffusions for global optimization. *SIAM Journal on Control and Optimization*, 24(5):1031–1043, 1986.
7. Peter J Green. Reversible jump markov chain monte carlo computation and bayesian model determination. *Biometrika*, 82(4):711–732, 1995.
8. Ulf Grenander and Michael I Miller. Representations of knowledge in complex systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 549–603, 1994.
9. Feng Han, Zhuowen Tu, and Song-Chun Zhu. Range image segmentation by an effective jump-diffusion method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1138–1153, 2004.
10. W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.
11. Scott Kirkpatrick, MP Vecchi, et al. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
12. Jun S Liu. Metropolized independent sampling with comparisons to rejection sampling and importance sampling. *Statistics and Computing*, 6(2):113–119, 1996.
13. Romeo Maciuca and Song-Chun Zhu. First hitting time analysis of the independence metropolis sampler. *Journal of Theoretical Probability*, 19(1):235–261, 2006.
14. Kerrie L Mengersen, Richard L Tweedie, et al. Rates of convergence of the hastings and metropolis algorithms. *The Annals of Statistics*, 24(1):101–121, 1996.
15. Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.
16. Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949.
17. Maureen Mitton and Courtney Nystuen. *Residential interior design: a guide to planning spaces*. John Wiley & Sons, 2011.
18. Siyuan Qi, Yixin Zhu, Siyuan Huang, Chenfanfu Jiang, and Song-Chun Zhu. Human-centric indoor scene synthesis using stochastic grammar. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
19. Richard L Smith and Luke Tierney. Exact transition probabilities for the independence metropolis sampler. *Preprint*, 1996.
20. Zhuowen Tu and Song-Chun Zhu. Image segmentation by data-driven markov chain monte carlo. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(5):657–673, 2002.
21. Lap-Fai Yu, Sai Kit Yeung, Chi-Keung Tang, Demetri Terzopoulos, Tony F Chan, and Stanley Osher. Make it home: automatic optimization of furniture arrangement. *ACM Trans. Graph.*, 30(4):86, 2011.

22. Song-Chun Zhu and David Mumford. *A stochastic grammar of images*. Now Publishers Inc, 2007.
23. Song Chun Zhu and Alan Yuille. Region competition: Unifying snakes, region growing, and bayes/mdl for multiband image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 18(9):884–900, 1996.

Chapter 5
Gibbs Sampler and its Variants

"Do the difficult things while they are easy and do the great things while they are small. A journey of a thousand miles must begin with a single step." - Lao Tzu

5.1 Introduction

The Gibbs sampler [9], first created by the Geman brothers Donald and Stewart, is an MCMC algorithm for obtaining samples from distributions that are difficult to sample. Usually the distributions are written in Gibbs form:

$$\pi(\mathbf{x}) = \frac{1}{Z} e^{-E(\mathbf{x})}, \quad \mathbf{x} = (x_1, ..., x_d) \in \Omega.$$

Such distributions appear when solving constrained (hard, soft) satisfaction problems (e.g. image denoising) or in Bayesian inference.


Donald and Stuart Geman

Example 5.1. The eight-queen problem is one example of a constrained satisfaction problem. The problem is to place 8 queens on a 8×8 chess board so that none of them threaten each other: i.e. no two queens occupy the same row, column, or diagonal lines. Denoting a possible solution by $\mathbf{s} \in \{1, ..., 64\}^8$ representing the board coordinates of the eight queens, thes the solutions belong to a set

$$\Omega^* = \{\mathbf{s} \in \{1, ..., 64\}^8, h_i(\mathbf{s}) \leq 1, i = 1, ..., 46\}$$

where $h_i(\mathbf{s})$ are the $8+8+30$ constraints counting the number of queens on each row, column and diagonal.

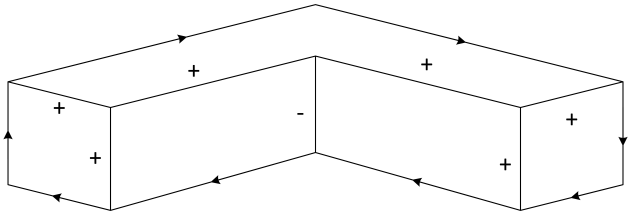


Fig. 5.1: An example of a line drawing and a possible labeling of the edges.

Example 5.2. Label the edges of a line drawing graph so that they are consistent, as illustrated in Figure 5.1. This is a constrained satisfaction problem on a graph $G = \{V, E\}$. We can define the solution set

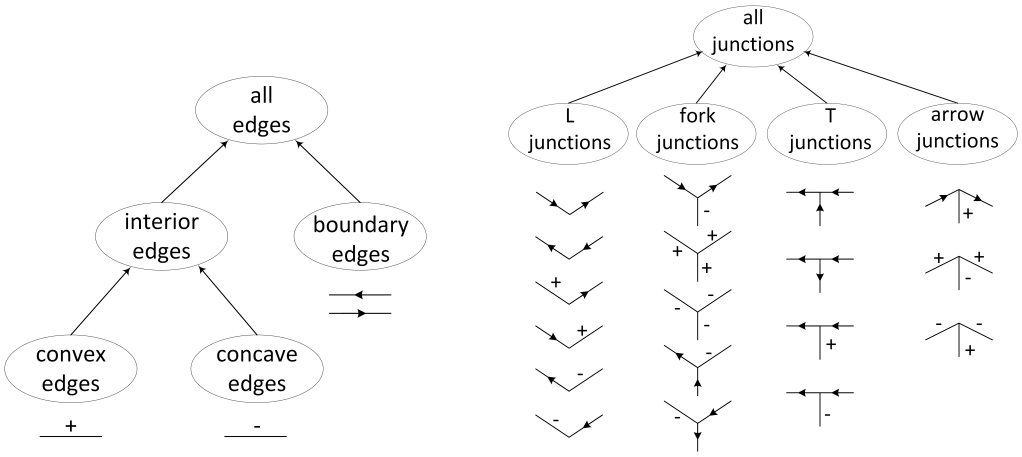


Fig. 5.2: These allowed edge labels and junction types represent strong constraints and thus prior knowledge.

$$\Omega^* = \{\mathbf{s} \in \{+,-,<,>\}^{|E|}, h_i(\mathbf{s}) = 1, i = 1,...,|V|\}$$

where $h_i(\mathbf{s})$ are the hard (logic) constraints for consistence at each vertex. Depending on the type of junction, these constraints are illustrated in Figure 5.2.

In these cases one would want to find the mode of the distribution, or certain distribution parameters such as mean, standard deviation, etc. Before the Gibbs sampler, finding the mode was done by the Relaxation Labeling algorithm [19], using an algorithm similar to the one described below.

Algorithm 2 Relaxation Algorithm

Input: Energy function $E[\mathbf{x}]$, current state $\mathbf{x}^{(t)} = (x_1,...,x_d) \in \Omega$
Output: New state $\mathbf{x}^{(t+1)} \in \Omega$
1. Select a variable $i \in \{1,...,d\}$ at random.
2. Compute

$$u = \text{argmin}\Big(E[x_i = 1|x_{-i}], \cdots, E[x_i = L|x_{-i}]\Big).$$

3. Set

$$\mathbf{x}_{-i}^{(t+1)} = \mathbf{x}_{-i}^{(t)}, x_i^{(t+1)} = u.$$

The problem with such a greedy algorithm is that it makes no guarantees for finding the global optimum. In fact, it often gets stuck in local optima. The Gibbs sampler was introduced as a stochastic version of the relaxation algorithm, which is why the Geman & Geman 1984 paper is titled "Stochastic relaxation".

In this chapter the Gibbs sampler and its issues and generalizations are discussed. The topic of data augmentation is also introduced and an application to the Julesz ensemble is studied.

5.2 Gibbs Sampler

The goal of Gibbs sampler is to sample a joint probability,

$$X = (x_1, x_2, \cdots, x_d) \sim \pi(x_1, x_2, \cdots, x_d)$$

by sampling in each dimension according to the conditional probability,

$$x_i \sim \pi(x_i|\underbrace{x_{-i}}_{\text{fixed}}) = \frac{1}{Z}\exp(-E[x_i|x_{-i}]), \qquad \forall i.$$

Here, $\pi(x_i|x_{-i})$ is the conditional probability at a site (variable) i conditional of the other variables.

Suppose Ω is d-dimensional, and each dimension is discretized into L finite states. Thus, the total number of states is L^d. The Gibbs sampler procedure is shown in the following algorithm.

Algorithm 3 Gibbs Sampler

Input: Probability function $\pi(\mathbf{x})$, current state $\mathbf{x}^{(t)} = (x_1, ..., x_d) \in \Omega$
Output: New state $\mathbf{x}^{(t+1)} \in \Omega$
1. Select a variable $i \in \{1, ..., d\}$ at random, taking L values $y_1, ..., y_L$.
2. Compute the conditional probability vector $\mathbf{u} = (u_1, ..., u_L)$ with

$$u_k = \pi(x_i = v_k|x_{-i}).$$

3. Sample $j \sim \mathbf{u}$ and set

$$\mathbf{x}_{-i}^{(t+1)} = \mathbf{x}_{-i}^{(t)}, x_i^{(t+1)} = y_j.$$

The order in which the variables are selected at step 1 above can be either randomized or follow a predefined scheme (e.g. $1, 2, ..., d$).

Definition 5.1. A *sweep* of the Gibbs sampler is a sequential visit to all of the sites (variables) once.

Although the transition matrix K_i for one Gibbs step may not be irreducible and aperiodic, it is easy to show that the total transition matrix $K = K_1 \cdot K_2 \cdots K_d$ does have these features after one sweep. Thus, the contraction coefficient satisfies $C(K) < 1$.

If $\mathbf{x}^{(t)} \sim \pi(\mathbf{x})$ at time t, and $\mathbf{x}^{(t+1)} \sim \pi(\mathbf{x})$, then K has π as its invariant probability as shown by

$$\mathbf{x}^{(t)} = (x_1, \cdots, x_i, x_{i+1}, \cdots, x_d) \sim \pi(x)$$
$$\mathbf{x}^{(t+1)} = (x_1, \cdots, y_j, x_{i+1}, \cdots, x_d).$$

The only change that takes place in moving between the two states is the replacement of x_i with y_j. However, we know that

$$\mathbf{x}^{(t+1)} \sim \pi(x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_d) \cdot \pi(y_j|x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_d) \implies \mathbf{x}^{(t+1)} \sim \pi(x).$$

In fact, one can show [3] that the periodic Gibbs sampler (which visits the sites using the predefined scheme $1, 2, ..., d$) has a geometric rate of convergence:

$$\|\mu K^n - \pi\|_{\text{TV}} \leqslant \frac{1}{2}(1 - e^{d\Delta})^n \|\mu - \pi\|_{\text{TV}}.$$

where $\Delta = \sup_i \delta_i$, with

$$\delta_i = \sup\{|E(\mathbf{x}) - E(\mathbf{y})|;\ x_j = y_j\ \forall j \neq i\}$$

Here we used $E(\mathbf{x})$, the energy of $\pi(\mathbf{x})$, i.e. $\pi(\mathbf{x}) = \frac{1}{Z}\exp\{-E(\mathbf{x})\}$. Observe that we only need to know $E(\mathbf{x})$ up to an additive constant.

5.2.1 A major problem with the Gibbs sampler

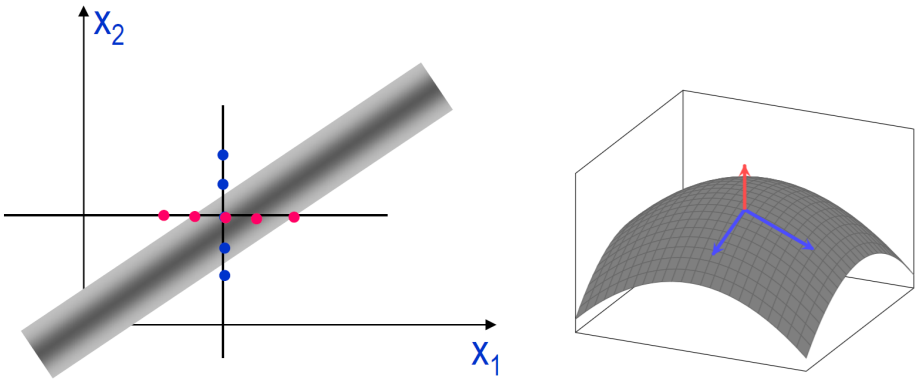


Fig. 5.3: Left: The Gibbs sampler has a hard time sampling a probability with two tightly coupled variables, as explained in Example 5.3. Right: In general the Gibbs sampler has a hard time sampling data that is concentrated on a manifold.

The problem is illustrated in the following example.

Example 5.3. For a probability $\pi(x_1, x_2)$ whose probability mass is focused on a 1D line segment, as illustrated in Figure 5.3, sampling the two dimensions iteratively is obviously inefficient. i.e. the chain is "jagging".

This problem occurs because the two variables are tightly coupled. It is best if we move along the direction of the line. In general, problems arise when the probability is concentrated in a lower dimensional manifold in the d-dimensional space. The Markov chain is not allowed to move in normal directions (off the manifold) but only in tangent directions.

As we know, Gibbs distributions are derived from constraints on the variable $\mathbf{x}$, and thus they are defined in some implicit manifold,

$$\Omega(H_0) = \{X : H_i(\mathbf{x}) = h_i, i = 1, 2, \cdots, K\}, \qquad H_0 = (h_1, h_2, \cdots, h_K).$$

Examples of Gibbs distributions that could be hard to sample using the Gibbs sampler are Markov Random Fields in general and the Ising/Potts model in particular.

Let $\mathbf{G} =< V, E >$ be an adjacency graph, such as a lattice with the 4 nearest neighbor connections. Each vertex $v_i \in V$ has a state variable x_i with a finite number of labels (or colors), $x_i \in \{1, 2, ..., \mathrm{L}\}$. The total number of labels L is predefined.

Definition 5.2. Let $\mathbf{x} = (x_1, x_2, ..., x_{|V|})$ denote the labeling of the graph, then the Ising/Potts model is a Markov random field,

$$\pi_{\mathrm{PTS}}(\mathbf{x}) = \frac{1}{Z} \exp\{- \sum_{<s,t> \in E} \beta_{st} \mathbf{1}(x_s \neq x_t)\}, \tag{5.1}$$

where $\mathbf{1}(x_s \neq x_t)$ is a Boolean function, equal to 1 if the condition $x_s \neq x_t$ is satisfied and equal to 0 otherwise. If the number of possible labels is $L = 2$, then π is called the Ising model, and if $L \geq 3$, then it is called the Potts model.

Usually we consider $\beta_{st} > 0$ for a ferro-magnetic system that prefers same colors for neighboring vertices. The Potts models and its extensions are used as *a priori* probabilities in many Bayesian inference tasks.

Example 5.4. For a single site Gibbs sampler defined on the Ising model (5.1), the boundary nodes are flipped with probability $p = 1/2$ due to the flat energy landscape illustrated in Figure 5.4. Flipping a string of length n will need on average $t \geq 1/p^n = 2^n$ steps! This implies exponential waiting time.

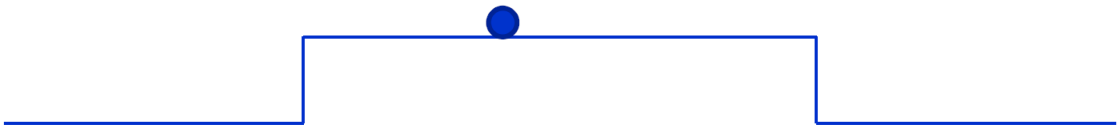


Fig. 5.4: The Ising model has a flat energy landscape that is difficult to sample.

5.3 Gibbs Sampler Generalizations

This section presents several Gibbs sampler modifications and generalizations that alleviate some of the difficulties associated with using the method for correlated variables that were highlighted in Section 5.2.1.

5.3.1 Hit-and-Run

This design selects a direction at random and samples in that direction.

Suppose the current state is $\mathbf{x}^{(t)}$.

1) Select a direction or axis $\vec{e}_t$.
2) Sample along the axis.
$$r \sim \pi(\mathbf{x}^{(t)} + r \cdot \vec{e}_t)$$
3) Update the state.
$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} + r \cdot \vec{e}_t$$

Sampling along the axis will be a continuous Gibbs sampler, which can be implemented by Multi-Try Metropolis. However, there is still one problem with this design, which is how to select the sampling direction.

5.3.2 Generalized Gibbs sampler

As a further generalization, one may not have to move in straight lines. In more general cases, a group of transformations may be used for the possible moves, as long as the moves preserve the invariant probability.

Theorem 2 (Liu and Wu, 1999 [12]) *Let $\Gamma = \{\gamma\}$ be a locally compact group that acts on the space Ω and each element multiplication is a possible move given by,*

$$\mathbf{x}^{(t)} \to \mathbf{x}^{(t+1)} = \gamma \cdot \mathbf{x}^{(t)}.$$

If $\mathbf{x} \sim \pi$ and the element $\gamma \in \Gamma$ is chosen by

$$\gamma|\mathbf{x} \sim \pi(\gamma \cdot \mathbf{x})|J_\gamma(\mathbf{x})|H(\mathrm{d}\gamma),$$

where $J_\gamma(x)$ is the Jacobian of the transformation $\mathbf{x} \to \gamma \cdot \mathbf{x}$ evaluated at $\mathbf{x}$ and $H(\mathrm{d}\gamma)$ is the left-invariant Haar measure,

$$H(\gamma \cdot B) = H(B), \qquad \forall \gamma, B,$$

then the new state follows the invariant probability

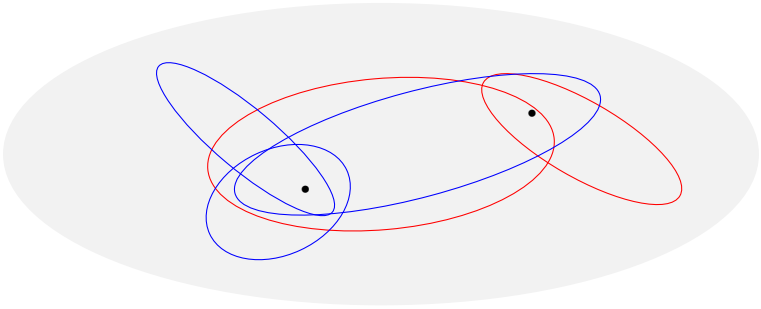
$$\mathbf{x}^{(t+1)} = \gamma \cdot \mathbf{x} \sim \pi.$$

5.3.3 Generalized hit-and-run

Conceptually, it helps to generalize the hit-and-run idea to an arbitrary partition of the space, especially in finite state spaces. This concept was created by Persi Diaconis in 2000.

Suppose a Markov chain consists of many sub-chains, and the transition probability is a linear sum,

$$K(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{N} \omega_i K_i(\mathbf{x},\mathbf{y}), \qquad \omega_i = p(i), \quad \sum_{i=1}^{N} \omega_i = 1.$$



If each sub-kernel has the same invariant probability,

$$\sum_{\mathbf{x}} \pi(\mathbf{x}) K_i(\mathbf{x},\mathbf{y}) = \pi(\mathbf{y}), \qquad \forall \mathbf{y} \in \Omega,$$

then the whole Markov chain follows $\pi(\mathbf{x})$. We denote the set of states connected to $\mathbf{x}$ by the i-th type moves with kernel K_i as

$$\Omega_i(\mathbf{x}) = \{\mathbf{y} \in \Omega : K_i(\mathbf{x},\mathbf{y}) > 0\}.$$

Now $\mathbf{x}$ is connected to the set

$$\Omega(\mathbf{x}) = \cup_{i=1}^{N} \Omega_i(\mathbf{x}).$$

The key issues with this approach are:

1. How do we decide the sampling dimensions, directions, group transforms, and sets $\Omega_i(\mathbf{x})$ in a systematic and principled way?
2. How do we schedule the visiting order governed by $p(i)$? i.e. How do we choose the moving directions, groups, and sets?

5.3.4 Sampling with auxiliary variables

We would like to sample $\mathbf{x}$ from $\pi(\mathbf{x})$, which may be hard to sample from due to correlations between variables. A systematic way to escape these correlations is to introduce an auxiliary random variable, y, such that

$$\mathbf{x} \sim \pi(\mathbf{x}) \quad \rightarrow \quad (\mathbf{x}, y) \sim \pi^{+}(\mathbf{x}, y).$$

Examples of auxiliary variables y:

- T – temperature: Simulated Tempering [11]
- S – scale: Multi-grid sampling
- w – weight: Dynamic weighting
- b – bond: Cluster sampling, Swendsen-Wang [7, 13]
- u – energy level: Slice sampling [7]

5.3.5 Simulated tempering

Let the target probability be

$$\pi(\mathbf{x}) = \frac{1}{Z}\exp\{-U(\mathbf{x})\},$$

and augment a variable I in $\{1, 2, \ldots, \mathrm{L}\}$ for L levels of temperature

$$1 = T_1 < T_2 < \cdots < T_L.$$

Then sample a joint probability, and keep the X's with $I = 1$

$$(x, I) \sim \pi^+(x, I) = \frac{1}{Z^+}\exp\{-\frac{1}{T_I}U(\mathbf{x})\}.$$

The sampler will move more freely at high temperatures, but it has difficulty crossing between different temperature levels. Suppose we run Markov chains at the L levels in parallel. Define a joint probability for all chains

$$\pi^+(\mathbf{x}_1, \ldots, \mathbf{x}_L) \propto \prod_{i=1}^{L}\exp\{-\frac{1}{T_i}U(\mathbf{x}_i)\}.$$

Now propose to permute two chains

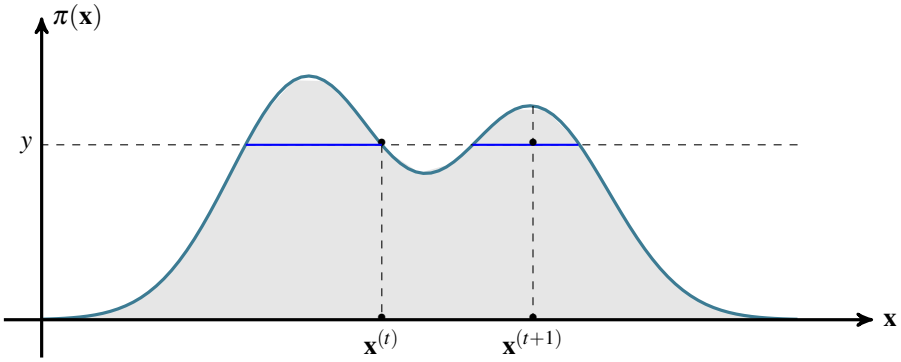
$$(\ldots, \mathbf{x}_i, \ldots, \mathbf{x}_j, \ldots) \to (\ldots, \mathbf{x}_j, \ldots, \mathbf{x}_i, \ldots).$$

Finally, accept with Metropolis-Hastings

$$\alpha = \min\left(1, \exp\{\left(\frac{1}{T_j} - \frac{1}{T_i}\right)\left(U(\mathbf{x}_j) - U(\mathbf{x}_i)\right)\}\right).$$

5.3.6 Slice sampling

Suppose $\mathbf{x} \sim \pi(\mathbf{x})$ in a 1D distribution. We introduce an auxiliary variable $y \in [0, 1]$ for the probability level. Thus, sampling $\pi(\mathbf{x})$ is equivalent to sampling uniformly from the shaded area in the $(\mathbf{x}, y)$ space.



We satisfy the conditions of

$$\sum_y \pi^+(\mathbf{x}, y) = \pi(\mathbf{x}),$$

but

$$\begin{cases} y \sim \pi^+(y|\mathbf{x}) = \text{unif}(0, \pi(\mathbf{x})) \leftarrow \text{easy to sample} \\ \mathbf{x} \sim \pi^+(\mathbf{x}|y) = \text{unif}(\overbrace{\{x; \pi(\mathbf{x}) \geqslant y\}}^{\text{level set}}) \leftarrow \text{hard to sample.} \end{cases}$$

The slice $\{x; \pi(\mathbf{x}) \geqslant y\}$ usually contains multiple components bounded by the level set $\pi(bx) = y$ and is difficult to sample. This case is illustrated in Figure 5.5.

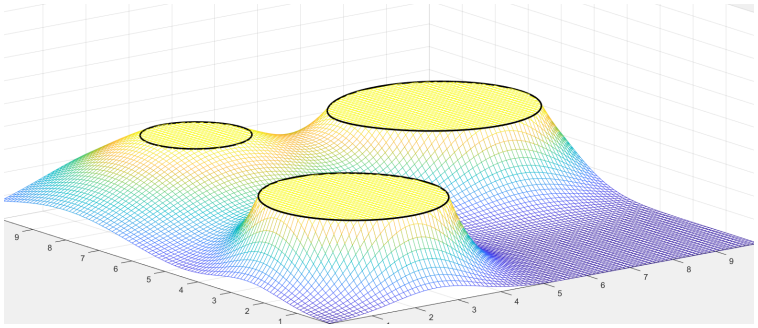


Fig. 5.5: The slice $\{x; \pi(\mathbf{x}) \geqslant y\}$ usually contains multiple components and is difficult to sample.

5.3.7 Data augmentation

The slice sampling method suggests two general conditions for the auxiliary variables given by

$$\mathbf{x} \sim \pi(\mathbf{x}) \quad \rightarrow \quad (\mathbf{x}, y) \sim \pi^+(\mathbf{x}, y).$$

1) The marginal probability is

$$\sum_y \pi^+(\mathbf{x}, y) = \pi(\mathbf{x}).$$

2) Both conditional probabilities can be factorized and are easy to sample from

$$\begin{cases} \mathbf{x} \sim \pi^+(\mathbf{x}|y) \\ y \sim \pi^+(y|\mathbf{x}). \end{cases}$$

The intuition for data augmentation is the following:
Very often the probability is focused on separated modes (areas), and hopping between these modes is difficult, since Markov chains usually move locally. Good auxiliary variables will:

1) help in the selection moving directions/groups/sets (in generalized hit-and-run).
2) enlarge the search scope.

5.3.8 Metropolized Gibbs sampler

We return to the generalized hit and run setup from section 5.3.3, where the kernel consists of a number of sub-kernels

$$K(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{N} \omega_i K_i(\mathbf{x}, \mathbf{y}), \qquad \omega_i = p(i), \quad \sum_{i=1}^{N} \omega_i = 1,$$

with the same invariant probability,

$$\sum_{\mathbf{x}} \pi(\mathbf{x}) K_i(\mathbf{x},\mathbf{y}) = \pi(\mathbf{y}), \qquad \forall \mathbf{y} \in \Omega.$$

The set of states connected to $\mathbf{x}$ by the i-th type moves is

$$\Omega_i(\mathbf{x}) = \{\mathbf{y} \in \Omega : K_i(\mathbf{x},\mathbf{y}) > 0\}.$$

Then $\mathbf{x}$ is connected to the set

$$\Omega(\mathbf{x}) = \cup_{i=1}^{N} \Omega_i(\mathbf{x}).$$

We know that there are two general designs: Gibbs and Metropolis.

1) Gibbs: We sample a probability in each set

$$y \sim [\pi]_i(\mathbf{y}), \qquad [\pi]_i(\mathbf{y}) \sim \begin{cases} \pi(\mathbf{y}) & y \in \Omega_i(\mathbf{x}), \\ 0, & y \notin \Omega_i(\mathbf{x}). \end{cases}$$

In this way, the move is symmetric with

$$\Omega_i(\mathbf{x}) = \Omega_i(\mathbf{y}).$$

2) Metropolis: We move according to an arbitrary $\Omega_i(\mathbf{x})$, but we do not know the proposal distribution q, where

$$q_i(\mathbf{x},y) = \frac{\pi(\mathbf{y})}{\sum_{y' \in \Omega_i(\mathbf{x})} \pi(y')}, \qquad \forall y' \in \Omega_i(\mathbf{x}).$$

However, we will have to check

$$q_i(y,x) = \frac{\pi(\mathbf{x})}{\sum_{x' \in \Omega_i(\mathbf{y})} \pi(\mathbf{x}')}, \qquad \forall x' \in \Omega_i(\mathbf{y}).$$

The problem now is that the move is no longer symmetric, i.e. $\Omega_i(\mathbf{x}) \neq \Omega_i(\mathbf{y})$. Although normalized, since the sets are different the detailed balance equation might not be satisfied. To rectify the move and get the correct balance, we need a condition,

$$\mathbf{y} \in \Omega_i(\mathbf{x}) \qquad \Longleftrightarrow \qquad \mathbf{x} \in \Omega_i(\mathbf{y})$$

We take the acceptance probability to be

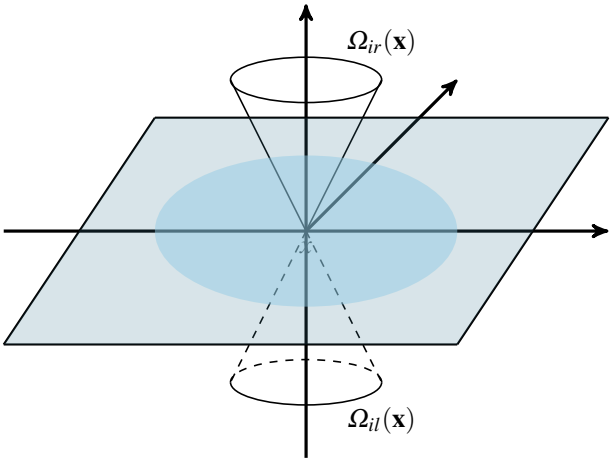
$$\alpha_i(\mathbf{x},\mathbf{y}) = \min\left(1, \frac{q_i(\mathbf{y},\mathbf{x}) \cdot \pi(\mathbf{y})}{q_i(\mathbf{x},\mathbf{y}) \cdot \pi(\mathbf{x})}\right) = \min\left(1, \frac{\frac{\pi(\mathbf{x})}{\sum_{\mathbf{x}' \in \Omega_i(\mathbf{y})} \pi(\mathbf{x}')} \cdot \pi(\mathbf{y})}{\frac{\pi(\mathbf{y})}{\sum_{y' \in \Omega_i(\mathbf{x})} \pi(\mathbf{y}')} \cdot \pi(\mathbf{x})}\right)$$

$$= \min\left(1, \frac{\overbrace{\sum_{\mathbf{y}' \in \Omega_i(\mathbf{x})} \pi(\mathbf{y}')}^{\text{total probability mass of } \Omega_i(\mathbf{x})}}{\underbrace{\sum_{\mathbf{x}' \in \Omega_i(\mathbf{y})} \pi(\mathbf{x}')}_{\text{total probability mass of } \Omega_i(\mathbf{y})}}\right).$$

The sub-kernels are designed in pairs,

$$K_i(\mathbf{x},y) = \omega_{i1} K_{il}(\mathbf{x},y) + \omega_{ir} K_{ir}(\mathbf{x},y),$$

and have their corresponding spaces $\Omega_{il}(\mathbf{x})$ and $\Omega_{ir}(\mathbf{x})$.



In this situation, the acceptance rate is

$$\alpha_i(\mathbf{x},\mathbf{y}) = \min\left(1, \frac{\sum_{\mathbf{y}'\in\Omega_{il}(\mathbf{x})}\pi(\mathbf{y}')}{\sum_{\mathbf{x}'\in\Omega_{ir}(\mathbf{y})}\pi(\mathbf{x}')}\right), \qquad \mathbf{y}\in\Omega_{il}(\mathbf{x}).$$

If the sets are symmetric, *i.e.* $\Omega_{il}(\mathbf{x}) = \Omega_{ir}(\mathbf{y})$, then we will have acceptance rate 1. If the sets are asymmetric, then we need the Metropolis acceptance step to rebalance the move.
One can improve the traditional Gibbs sampler by prohibiting the Markov chain from staying in its current state in the conditional probability. Thus, the sets are guaranteed to be asymmetric and we need the Metropolis acceptance step to rebalance. This method is called the Metropolized Gibbs Sampler(MGS). The diagonal elements in the proposal matrix are set to zero. This is a desirable property of a Markov chain as it causes the chain to have a fast mixing time.

$$q(\mathbf{x},\mathbf{y}) = \frac{\pi(\mathbf{y})}{1-\pi(\mathbf{x})}, \qquad \mathbf{y}\in\Omega(\mathbf{x}), \mathbf{x}\notin\Omega(\mathbf{x}),$$

where 1 represents the normalizing factor. Thus, the acceptance rate becomes

$$\alpha(\mathbf{x},y) = \min\left(1, \frac{1-\pi(\mathbf{x})}{1-\pi(\mathbf{y})}\right).$$

Furthermore, it has been shown that

$$K_{\text{MGS}}(\mathbf{x},\mathbf{y}) \geqslant K_{\text{Gibbs}}(\mathbf{x},\mathbf{y}), \qquad \forall \mathbf{x}\neq\mathbf{y}\in\Omega.$$

5.4 Data Association and Data Augmentation

There are many cases in which an accurate model $f(\mathbf{y},\mathbf{h}|\theta)$ could be obtained when the observed data $\mathbf{y}$ is augmented with some missing (hidden) data $\mathbf{h}$. For example a more accurate face model $f(\mathbf{y},\mathbf{h}|\theta)$ (where $\theta\in\{0,1\}$ could represent face/nonface) can be obtained if the observed intensity image is augmented with a vector $\mathbf{h}$ containing the face position, rotation, scale, 3D pose, and other variables (e.g. sun-glasses, beard, etc.).

The posterior distribution of the parameter θ conditional on the observed data is then obtained by integrating out the hidden variables

$$p(\theta|\mathbf{y}) = \int p(\theta|\mathbf{y},\mathbf{h})p(\mathbf{h}|\mathbf{y})d\mathbf{h}. \tag{5.2}$$

If we could sample $p(\mathbf{h}|\mathbf{y})$, then we could obtain a Monte Carlo approximation of $p(\theta|\mathbf{y})$ using Eq. (5.2). Tanner and Wong [14] observed that we could use an initial approximation $f(\theta)$ of the target distribution $p(\theta|\mathbf{y})$ to obtain samples of the hidden variables $\mathbf{h}_1,...,\mathbf{h}_m$ from

$$\tilde{p}(\mathbf{h}) = \int p(\mathbf{h}|\theta,\mathbf{y}) f(\theta) d\theta,$$

by first sampling $\theta_i \sim f(\theta)$ and then $\mathbf{h}_i \sim p(\mathbf{h}|\theta_i,\mathbf{y})$. These hidden samples are also called *multiple imputations*. We can use them to obtain a (hopefully) better approximation of the target distribution

$$f(\theta) = \frac{1}{m}\sum_{i=1}^{m} p(\theta|\mathbf{y},\mathbf{h}_i).$$

Thus, the original data augmentation algorithm starts with a set of hidden values $\mathbf{h}_1^{(0)},...,\mathbf{h}_m^{(0)}$ and proceeds as follows:

Algorithm 4 Data Augmentation

Initialize $\mathbf{h}_1^{(0)},...,\mathbf{h}_m^{(0)}$
for t=1 to N^{iter} **do**
  for i=1 to m **do**
    Pick k randomly from $\{1,...,m\}$
    Sample $\theta' \sim p(\theta|\mathbf{y},\mathbf{h}_k^{(t-1)})$
    Sample $\mathbf{h}_k^{(t)} \sim p(\mathbf{h}|\mathbf{y},\theta')$
  end for
end for

An important observation to make is that the DA algorithm is equivalent to the version with $m = 1$, since each element of the current generation $\mathbf{h}_1^{(t)},...,\mathbf{h}_m^{(t)}$ can be traced back to its origin. It can easily be seen that when t is sufficiently large, all samples from generation t originate from a single element. Due to the purely random way that parents are chosen, there is no bias in the selection of the common ancestor. Thus, the DA algorithm is equivalent to a Gibbs sampler type of algorithm that alternates between sampling the parameter θ with sampling the hidden variables $\mathbf{h}$:

Algorithm 5 Simplified Data Augmentation

Initialize $\mathbf{h}$
for t=1 to N^{iter} **do**
  Sample $\theta' \sim p(\theta|\mathbf{y},\mathbf{h}^{(t-1)})$
  Sample $\mathbf{h}^{(t)} \sim p(\mathbf{h}|\mathbf{y},\theta')$
end for

5.5 Julesz ensemble and MCMC sampling of texture

Let $\mathbf{I}$ be an image defined on a finite lattice $\Lambda \subset Z^2$. For each pixel $v = (x,y) \in \Lambda$, the intensity value at v is denoted by $\mathbf{I}(v) \in S$, with S being a finite interval on the real line or a finite set of quantized gray levels. We denote the space of all images on Λ by $\Omega_\Lambda = S^{|\Lambda|}$.

In modeling homogeneous texture images, we are interested in exploring a finite set of statistics of local image features. Such statistics have first been studied using co-occurrence matrices on polygons and cliques, but proven inadequate for describing real world images and irrelevant to biological vision systems. In the late 1980s, it was

recognized that real world imagery is better represented by spatial/frequency bases, such as Gabor filters [6], wavelet transforms [5], and filter pyramids.

Given a set of filters $\{F^{(\alpha)}, \alpha = 1,2,...,K\}$, a sub-band image $\mathbf{I}^{(\alpha)} = F^{(\alpha)} * \mathbf{I}$ is computed for each filter $F^{(\alpha)}$. Then statistics are extracted from the sub-band images or pyramid instead of the intensity image. From a dimension reduction perspective, the filters characterize local texture features, and as a result simple statistics of the sub-band images can capture information that would otherwise require *k*-gon or clique statistics in high dimensions.

While Gabor filters are well grounded in biological vision [4], very little is known about how visual cortices pool statistics across images. There are four popular choices of statistics in the literature.

1. Moments of a single filter response, e.g. mean and variance of $\mathbf{I}^{(\alpha)}$.
2. Rectified functions that resemble the responses of "on/off" cells [2]:

$$\mathbf{h}^{(\alpha,+)}(\mathbf{I}) = \frac{1}{|\Lambda|}\sum_{v\in\Lambda} R^{+}(\mathbf{I}^{(\alpha)}(v)), \mathbf{h}^{(\alpha,-)}(\mathbf{I}) = \frac{1}{|\Lambda|}\sum_{v\in\Lambda} R^{-}(\mathbf{I}^{(\alpha)}(v)).$$

3. One bin of the empirical histogram of $\mathbf{I}^{(\alpha)}$.
4. One bin of the full joint histogram of $(\mathbf{I}^{(1)}, \cdots, \mathbf{I}^{(k)})$.

In what follows, we will study a mathematical definition of texture –the Julesz ensemble– based on such filter statistics and an algorithm for sampling images from it.

5.5.1 The Julesz ensemble - a mathematical definition of texture

Given a set of K statistics $\mathbf{h} = \{\mathbf{h}^{(\alpha)} : \alpha = 1,2,...,K\}$ that have been normalized with respect to the size of the lattice $|\Lambda|$, an image $\mathbf{I}$ is mapped into a point $\mathbf{h}(\mathbf{I}) = (\mathbf{h}^{(1)}(\mathbf{I}),...,\mathbf{h}^{(K)}(\mathbf{I}))$ in the space of statistics. Let

$$\Omega_\Lambda(\mathbf{h}_0) = \{\mathbf{I} : \mathbf{h}(\mathbf{I}) = \mathbf{h}_0\}$$

be the set of images sharing the same statistics $\mathbf{h}_o$. The image space Ω_Λ is then partitioned into equivalence classes

$$\Omega_\Lambda = \cup_{\mathbf{h}} \Omega_\Lambda(\mathbf{h}).$$

Due to intensity quantization in finite lattices, in practice one needs to relax the constraint on statistics and define the image set as

$$\Omega_\Lambda(\mathcal{H}) = \{\mathbf{I} : \mathbf{h}(\mathbf{I}) \in \mathcal{H}\},$$

where $\mathcal{H}$ is an open set around $\mathbf{h}_0$. $\Omega_\Lambda(\mathcal{H})$ implies a uniform distribution

$$q(\mathbf{I};\mathcal{H}) = \begin{cases} \frac{1}{|\Omega_\Lambda(\mathcal{H})|} & \text{for } \mathbf{I} \in \Omega_\Lambda(\mathcal{H}), \\ 0 & \text{otherwise,} \end{cases}$$

where $|\Omega_\Lambda(\mathcal{H})|$ is the volume of the set.

Definition *Given a set of normalized statistics* $\mathbf{h} = \{\mathbf{h}^{(\alpha)} : \alpha = 1,2,...,K\}$*, a Julesz ensemble* $\Omega(\mathbf{h})$ *is the limit of* $\Omega_\Lambda(\mathcal{H})$ *as* $\Lambda \to Z^2$ *and* $\mathcal{H} \to \{\mathbf{h}\}$ *under some boundary conditions.*

A Julesz ensemble $\Omega(\mathbf{h})$ is a mathematical idealization of $\Omega_\Lambda(\mathcal{H})$ on a large lattice with $\mathcal{H}$ close to $\mathbf{h}$. As $\Lambda \to Z^2$, it makes sense to let the normalized statistics $\mathcal{H} \to \{\mathbf{h}\}$. We assume $\Lambda \to Z^2$ in the same sense as van Hove [10], i.e., the ratio between the size of the boundary and the size of Λ goes to 0, $|\partial\Lambda|/|\Lambda| \to 0$. In practice, we often consider a lattice large enough if $\frac{|\partial\Lambda|}{|\Lambda|}$ is very small, e.g. $1/15$. Thus with a slight abuse of notation and to avoid technicalities in dealing with limits, we consider a sufficiently large image (e.g. 256×256 pixels) as an infinite image. See [15] for a more careful treatment.

A Julesz ensemble $\Omega(\mathbf{h})$ defines a texture pattern on Z^2, and it maps textures into the space of feature statistics $\mathbf{h}$. To draw a comparison to color, as an electro-magnetic wave with wavelength $\lambda \in [400, 700]$nm defines a unique visible color, a statistic value $\mathbf{h}$ defines a texture pattern! [1]

A mathematical definition of texture could be different from human texture perception. The latter has very coarse precision on the statistics $\mathbf{h}$ and is often influenced by experience.

During creation of a mathematical definition of texture, modeling is posed as an inverse problem. Suppose we are given a set of observed training images $\Omega_{\text{obs}} = \{\mathbf{I}_{\text{obs},1}, \mathbf{I}_{\text{obs},2}, ..., \mathbf{I}_{\text{obs},M}\}$, which are sampled from an unknown Julesz ensemble $\Omega_* = \Omega(\mathbf{h}_*)$. The objective of texture modeling is to search for the statistics $\mathbf{h}_*$.

We first choose a set of K statistics from a dictionary B as discussed above. We then compute the normalized statistics over the observed images $\mathbf{h}_{\text{obs}} = (\mathbf{h}^{(1)}_{\text{obs}}, ..., \mathbf{h}^{(K)}_{\text{obs}})$, with

$$\mathbf{h}^{(\alpha)}_{\text{obs}} = \frac{1}{M}\sum_{i=1}^{M} \mathbf{h}^{(\alpha)}(\mathbf{I}_{\text{obs},i}), \quad \alpha = 1, 2, ..., K. \tag{5.3}$$

We then define an ensemble of texture images using $\mathbf{h}_{\text{obs}}$,

$$\Omega_{K,\varepsilon} = \{\mathbf{I} : D(\mathbf{h}^{(\alpha)}(\mathbf{I}), \mathbf{h}^{(\alpha)}_{\text{obs}}) \leq \varepsilon, \quad \forall \alpha\}, \tag{5.4}$$

where D is some distance, such as the L_1 distance for histograms. If Λ is large enough to be considered infinite, we can set ε to essentially 0, and denote the corresponding $\Omega_{K,\varepsilon}$ as Ω_K. The ensemble Ω_K implies a uniform probability distribution $q(\mathbf{I}; \mathbf{h})$ over Ω_K, whose entropy is $\log |\Omega_K|$.

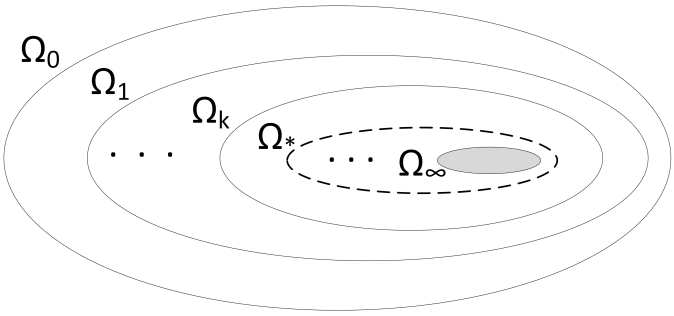


Fig. 5.6: The volume (or entropy) of Julesz ensemble decreases monotonically with more statistical constraints added. © [2000] IEEE. Reprinted, with permission, from ref. [16].

To search for the underlying Julesz ensemble Ω_*, we can adopt the pursuit strategy used by Zhu, Wu, and Mumford (1997) [17]. When $k = 0$, we have $\Omega_0 = \Omega_\Lambda$. Suppose at step k, a statistic $\mathbf{h}$ is chosen. Then, at step $k+1$ a statistic $\mathbf{h}^{(k+1)}$ is added to have $\mathbf{h}_+ = (\mathbf{h}, \mathbf{h}^{(k+1)})$. $\mathbf{h}^{(k+1)}$ is selected to have the largest entropy decrease among all statistics in the dictionary B,

$$\mathbf{h}^{(k+1)} = \arg\max_{\beta \in B} [\text{entropy}(q(\mathbf{I}; \mathbf{h})) - \text{entropy}(q(\mathbf{I}; \mathbf{h}_+)] = \arg\max_{\beta \in B} [\log |\Omega_k| - \log |\Omega_{k+1}|]. \tag{5.5}$$

The decrease of entropy is called the *information gain* of $\mathbf{h}^{(k+1)}$.

As shown in Figure 5.6, as more statistics are added, the entropy or volume of the Julesz ensemble decreases monotonically

$$\Omega_\Lambda = \Omega_0 \supseteq \Omega_1 \supseteq \cdots \supseteq \Omega_k \supseteq \cdots.$$

Obviously introducing too many statistics will lead to overfitting. In the limit of $k \to \infty$, Ω_∞ only includes the observed images in Ω_{obs} and their translated versions.

[1] We name this ensemble after Julesz to remember his pioneering work on texture. This does not necessarily mean that Julesz defined texture patterns with this mathematical formulation.

With the observed finite images, the choice of statistics $\mathbf{h}$ and the Julesz ensemble $\Omega(\mathbf{h})$ is an issue of model complexity that has been extensively studied in statistics literature. In the minimax entropy model [17, 18], an AIC criterion [1] is adopted for model selection. The intuitive idea of AIC is simple. With finite images, we should measure the fluctuation of the new statistics $\mathbf{h}^{(k+1)}$ over the training images in Ω_{obs}. Thus, when a new statistic is added, it brings new information as well as estimation error. The feature pursuit process should stop when the estimation error brought by $\mathbf{h}^{(k+1)}$ is larger than its information gain.

5.5.2 The Gibbs ensemble and ensemble equivalence

In this section we discuss the Gibbs ensemble and the equivalence between the Julesz and Gibbs ensembles. A detailed study is given in [15]. Given a set of observed images Ω_{obs} and the statistics $\mathbf{h}_{\text{obs}}$, another line of research is to pursue probabilistic texture models, in particular the Gibbs distributions or MRF models. One general class of MRF model is the FRAME model studied by Zhu, Wu, and Mumford in 1997 [17, 18]. The FRAME model derived from the maximum entropy principle has the Gibbs form

$$p(\mathbf{I};\beta) = \frac{1}{Z(\beta)} \exp\{-\sum_{\alpha=1}^{K} < \beta^{(\alpha)}, \mathbf{h}^{(\alpha)}(\mathbf{I}) >\} = \frac{1}{Z(\beta)} \exp\{< \beta, \mathbf{h}(\mathbf{I}) >\}. \tag{5.6}$$

The parameters $\beta = (\beta^{(1)}, \beta^{(2)}, ..., \beta^{(K)})$ are Lagrange multipliers. The values of β are determined so that $p(\mathbf{I};\beta)$ reproduces the observed statistics,

$$E_{p(\mathbf{I};\beta)}[\mathbf{h}^{(\alpha)}(\mathbf{I})] = \mathbf{h}^{(\alpha)}_{\text{obs}} \quad \alpha = 1, 2, ..., K. \tag{5.7}$$

The selection of statistics is guided by a minimum entropy principle.

As the image lattice becomes large enough, the fluctuations of the normalized statistics diminish. Thus as $\Lambda \to Z^2$, the FRAME model converges to a *limiting random field* in the absence of phase transition. The limiting random field essentially concentrates all its probability mass uniformly over a set of images which we call the *Gibbs ensemble*. [2]

In [15], it is shown that the Gibbs ensemble given by $p(\mathbf{I};\beta)$ is equivalent to the Julesz ensemble specified by $q(\mathbf{I};\mathbf{h}_{\text{obs}})$. The relationship between β and $\mathbf{h}_{\text{obs}}$ is expressed in equation (5.7). Intuitively, $q(\mathbf{I};\mathbf{h}_{\text{obs}})$ is defined by a hard constraint, while the Gibbs model $p(\mathbf{I};\beta)$ is defined by a soft constraint. Both use the observed statistics $\mathbf{h}_{\text{obs}}$, and the model $p(\mathbf{I};\beta)$ concentrates on the Julesz ensemble uniformly as the lattice Λ gets big enough.

The ensemble equivalence above reveals two significant facts in texture modeling.

1. Given a set of statistics $\mathbf{h}$, we can synthesize typical texture images of the fitted FRAME model by sampling from the Julesz ensemble $\Omega(\mathbf{h})$ without learning the parameters β in the FRAME models [17], which is time consuming. Thus feature pursuit, model selection, and texture synthesis can be done effectively with the Julesz ensemble.
2. For images sampled from the Julesz ensemble, a local patch of the image given its environment follows the Gibbs distribution (or FRAME model) derived by the minimax entropy principle. Therefore, the Gibbs model $p(\mathbf{I};\beta)$ provides a parametric form for the conditional distribution of $q(\mathbf{I};\mathbf{h})$ on small image patches. $p(\mathbf{I};\beta)$ should be used for tasks such as texture classification and segmentation.

The pursuit of Julesz ensembles can also be based on the minimax entropy principle. First, the definition of $\Omega(\mathbf{h})$ as the maximum set of images sharing statistics $\mathbf{h}$ is equivalent to a maximum entropy principle. Second, the pursuit of statistics in equation (5.5) uses a minimum entropy principle. Therefore a unifying picture emerges for texture modeling under the minimax entropy theory.

[2] In the computation of a feature statistic $\mathbf{h}(\mathbf{I})$, we need to define boundary conditions so that the filter responses in Λ are well defined. In case of phase transition, the limit of a Gibbs distribution is not unique, and it depends on the boundary conditions. However, the equivalence between Julesz ensemble and Gibbs ensemble holds even with a phase transition. The study of phase transitions is beyond the scope of this book.

5.5.3 Sampling the Julesz ensemble

Sampling the Julesz ensemble is by no means a trivial task! As $|\Omega_K|/|\Omega_\Lambda|$ is exponentially small, the Julesz ensemble has almost zero volume in the image space. Thus rejection sampling methods are inappropriate, and we instead use Markov chain Monte Carlo methods.

First, we define a function

$$G(\mathbf{I}) = \begin{cases} 0, & \text{if } D(\mathbf{h}^{(\alpha)}(\mathbf{I}), \mathbf{h}^{(\alpha)}_{\text{obs}}) \le \varepsilon, \quad \forall \alpha. \\ \sum_{\alpha=1}^{K} D(\mathbf{h}^{(\alpha)}(\mathbf{I}), \mathbf{h}^{(\alpha)}_{\text{obs}}), & \text{otherwise.} \end{cases}$$

Now the distribution

$$q(\mathbf{I}; \mathbf{h}, T) = \frac{1}{Z(T)} \exp\{-G(\mathbf{I})/T\} \tag{5.8}$$

converges to a Julesz ensemble Ω_K, as the temperature T goes to 0. The $q(\mathbf{I}; \mathbf{h}, T)$ can be sampled by the Gibbs sampler or other MCMC algorithms.

Algorithm 6 Sampling the Julesz Ensemble

Input: Texture images $\{\mathbf{I}_{\text{obs},i},\ i = 1, 2, ..., M\}$, K statistics (filters) $\{F^{(1)}, F^{(2)}, ..., F^{(K)}\}$.
Compute $\mathbf{h}_{\text{obs}} = \{\mathbf{h}^{(\alpha)}_{\text{obs}}, \quad \alpha = 1, ..., K\}$.
Initialize a synthesized image $\mathbf{I}$ (e.g. white noise).
$T \leftarrow T_0$.
repeat
  Randomly pick a location $v \in \Lambda$,
  for $\mathbf{I}(v) \in S$ **do**
    Calculate $q(\mathbf{I}(v) \mid \mathbf{I}(-v); \mathbf{h}, T)$.
  end for
  Randomly draw a new value of $\mathbf{I}(v)$ from $q(\mathbf{I}(v) \mid \mathbf{I}(-v); \mathbf{h}, T)$.
  Reduce T after each sweep.
  Record samples when $D(\mathbf{h}^{(\alpha)}(\mathbf{I}), \mathbf{h}^{(\alpha)}_{\text{obs}}) \le \varepsilon$ for $\alpha = 1, 2, ..., K$
until enough samples are collected.

In the above algorithm $q(\mathbf{I}(v) \mid \mathbf{I}(-v); \mathbf{h}, T)$ is the conditional probability of the pixel value $\mathbf{I}(v)$ with intensities for the rest of the lattice fixed. A sweep flips $|\Lambda|$ pixels in a random visiting scheme or flips all pixels in a fixed visiting scheme.

Due to the equivalence between the Julesz ensemble and the Gibbs ensemble [15], the sampled images from $q(\mathbf{I}; \mathbf{h})$ and those from $p(\mathbf{I}; \beta)$ share many features in that they produce not only the same statistics in $\mathbf{h}$, but also statistics extracted by any other filters, linear or nonlinear. It is worth emphasizing one key concept which has been misunderstood in some computer vision work. The Julesz ensemble is the set of "typical" images for the Gibbs model $p(\mathbf{I}; \beta)$, not the "most probable" images that minimize the Gibbs potential (or energy) in $p(\mathbf{I}; \beta)$.

This algorithm can be used for selecting statistics $\mathbf{h}$, as in [17]. That is, one can pursue new statistics by decreasing the entropy as measured in equation (5.5). An in-depth discussion is given in [15].

5.5.4 Experiment: sampling the Julesz ensemble

In this experiment, we select all 56 linear filters (Gabor filters at various scales and orientations and small Laplacian of Gaussian filters) used in [17]. The largest filter window size is 19×19 pixels. We choose $\mathbf{h}$ to be the marginal histograms of these filters and sample the Julesz ensemble using Algorithm 6. Although only a small subset of filters are often necessary for each texture pattern, we use a common filter set. It is almost impractical to learn a FRAME

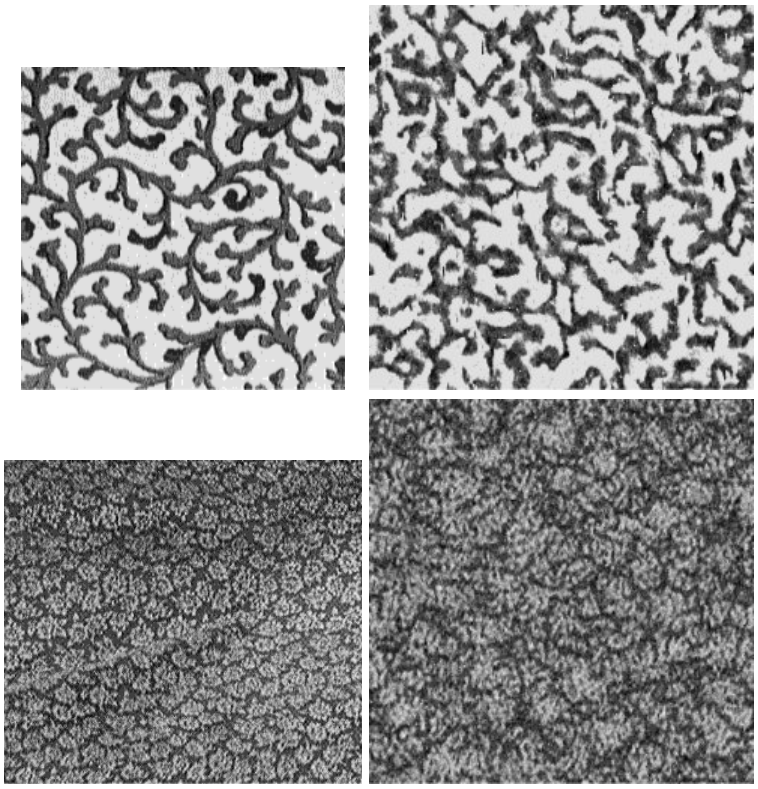


Fig. 5.7: Left column: the observed texture images, right column: the synthesized texture images that share the exact histograms with the observed ones for 56 filters. © [2000] IEEE. Reprinted, with permission, from ref. [16].

model by integrating all these 56 filters, so the computation is much easier using the simpler but equivalent model $q(\mathbf{I};\mathbf{h})$.

We run the algorithm over a broad set of texture images collected from various sources. The results are displayed in Figure 5.7. The left column shows the observed textures, and the right column displays the synthesized images whose sizes are 256×256 pixels. For these textures, the marginal statistics closely match (less than 1% error for each histogram) after about 20 to 100 sweeps, starting with a temperature $T_0 = 3$. Since the synthesized images are finite, the matching error ε cannot be infinitely small. In general, we set $\varepsilon \propto \frac{1}{|\Lambda|}$.

This experiment demonstrates that Gabor filters and marginal histograms are sufficient for capturing a wide variety of homogeneous texture patterns. For example, the cloth pattern in the top row of Figure 5.7 has very regular structures, which are reproduced fairly well in the synthesized texture image. This demonstrates that Gabor filters at various scales align without using the joint histograms explicitly. The alignment and high order statistics are accounted for through the interactions of the filters. This experiments reveals two problems.

The first problem is demonstrated in the failed example in the bottom of Figure 5.7. The observed texture patterns have large structures whose periods are longer than the biggest Gabor filter windows in our filter set. As a result, these periodic patterns are scrambled in the synthesized image, while the basic texture features are well preserved.

The second problem is with the effectiveness of the Gibbs sampler. If we scale up a checkerboard image so that each square of the checkerboard is 15×15 pixels in size, then we have to choose filters with large window sizes to learn this checkerboard pattern. It becomes infeasible to match the marginal statistics closely using the Gibbs sampler in Algorithm 6, since flipping one pixel at a time is inefficient for such large patterns. This suggests that we should search for more efficient sampling methods that can update large image patches. We believe that this problem would occur for other statistics matching methods, such as steepest descent [2, 8]. The inefficiency of the Gibbs sampler is also reflected in its slow mixing rate. After the first image is synthesized, it takes a long time for the algorithm to generate an image which is distinct from the first one. That is, the Markov chain moves very slowly in the Julesz ensemble.

5.6 Exercises

Problem 1. In a countable state space Ω, consider two transition kernels K_1 and K_2 that have the common invariant probability π. K_1 is said to *dominate* K_2 *in the Pushin order*, if

$$K_1(x,y) \geq K_2(x,y), \forall x \neq y.$$

I.e. the off-diagonal elements of K_1 are no less than those of K_2. (The samples $\{X_t^1\}_{t\geq 0}$ from K_1 are less correlated than the samples $\{X_t^2\}_{t\geq 0}$ from K_2).

Show that the Metropolised Gibbs sampler dominates the Gibbs sampler. (To fix notation, assume that we are sampling a probability $\pi(X)$ with $X = (x_1, x_2, ..., x_n)$. Both case selected a site x_i as random.)

Problem 2. We consider the design of a continuous Gibbs sampler. To simplify the problem, we only study a 1-dimensional probability $\pi(x), x \in [a,b]$. For high dimensional spaces, the same procedure will apply, because each time we still sample in a 1D conditional probability in a real interval $[a,b]$.

We divide the interval into K bins of equal length $L = \frac{b-a}{K}$. Denote these bins by $B_1, B_2, ..., B_K$. Suppose the current state is x, without loss of generality, we assume $x \in B_1$. We sample a number of $K-1$ intermediate points $Z_2, Z_3, ..., Z_K$ on the other $K-1$ bins uniformly, $Z_i \sim \text{Unif}(B_i)$. Define the total probability mass of the $K-1$ points

$$S = \pi(z_2) + \cdots + \pi(z_K).$$

Then we choose the next state y by a discrete Gibbs sampler among the K points $x, z_2, z_3, ..., z_K$.

$$y \in \{x, z_2, ..., z_K\} \sim \frac{\pi(y)}{\pi(x) + S}$$

Now when the Markov chain is at state y, it may move back to x in a similar way.

1. What is the transition probability $K(x,y)$? [hint: treat the $K-2$ other variables as 'step-stones' or auxiliary variables that need to be integrated out.]
2. If we treat the probability $K(x,y)$ as a proposal probability, and apply a Metropolis-Hastings step, show that the ratio is
$$\frac{K(x,y)}{K(y,x)} = \frac{\pi(y)}{\pi(x)}$$
Therefore the acceptance rate is always 1.
3. Will the bin number K matter in the above conclusion? If not, then what do you think may be the benefit of choosing a large K? [Think about this, and make a conjecture] Shall we use the same set of bins at states x and y? Why?

References

1. Hirotugu Akaike. On entropy maximization principle. *Application of statistics*, 1977.
2. CH Anderson and WD Langer. Statistical models of image texture. *Washington University Medical School*, 1997.
3. Pierre Bremaud. *Markov chains: Gibbs fields, Monte Carlo simulation, and queues*, volume 31. Springer, 1999.
4. Charles Chubb and Michael S Landy. Orthogonal distribution analysis: A new approach to the study of texture perception. *Computational models of visual processing*, 12:394, 1991.
5. Ingrid Daubechies et al. *Ten lectures on wavelets*, volume 61. SIAM, 1992.
6. John G Daugman. Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters. *JOSA A*, 2(7):1160–1169, 1985.
7. Robert G Edwards and Alan D Sokal. Generalization of the fortuin-kasteleyn-swendsen-wang representation and monte carlo algorithm. *Physical review D*, 38(6):2009, 1988.
8. A. Gagalowicz and S. D. Ma. Model driven synthesis of natural textures for 3d scenes. *Computers and Graphics*, 10(2):161–170, 1986.

9. Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 6:721–741, 1984.
10. Hans-Otto Georgii. *Gibbs measures and phase transitions*, volume 9. Walter de Gruyter, 2011.
11. Charles J Geyer and Elizabeth A Thompson. Annealing markov chain monte carlo with applications to ancestral inference. *Journal of the American Statistical Association*, 90(431):909–920, 1995.
12. Jun S Liu and Ying Nian Wu. Parameter expansion for data augmentation. *Journal of the American Statistical Association*, 94(448):1264–1274, 1999.
13. Robert H Swendsen and Jian-Sheng Wang. Nonuniversal critical dynamics in monte carlo simulations. *Physical Review Letters*, 58(2):86–88, 1987.
14. Martin A Tanner and Wing Hung Wong. The calculation of posterior distributions by data augmentation. *Journal of the American statistical Association*, 82(398):528–540, 1987.
15. Ying Nian Wu, Song Chun Zhu, and Xiuwen Liu. Equivalence of julesz and gibbs texture ensembles. In *ICCV*, volume 2, pages 1025–1032, 1999.
16. Song Chun Zhu, Xiu Wen Liu, and Ying Nian Wu. Exploring texture ensembles by efficient markov chain monte carlo-toward a "trichromacy" theory of texture. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(6):554–569, 2000.
17. Song Chun Zhu, Ying Nian Wu, and David Mumford. Minimax entropy principle and its application to texture modeling. *Neural Computation*, 9(8):1627–1660, 1997.
18. Song Chun Zhu, Yingnian Wu, and David Mumford. Filters, random fields and maximum entropy (frame): Towards a unified theory for texture modeling. *International Journal of Computer Vision*, 27(2):107–126, 1998.
19. Steven W. Zucker, Robert A. Hummel, and Azriel Rosenfeld. An application of relaxation labeling to line and curve enhancement. *IEEE Transactions on Computers*, 26(4):394–403, 1977.

Chapter 6
Cluster Sampling Methods

"Shapes that contain no inner components of positive/negative relationships will function better with shapes of the same nature" - Keith Haring

Introduction


Robert H. Swendsen

The Swendsen-Wang algorithm has been initially designed for addressing the critical slowing down in sampling the Ising and Potts models described below at or near the critical temperature where phase transitions occur. Fortuin and Kasteleyn [17] have mapped the Potts model to a percolation model [7]. The percolation model is a model for a porous material with randomly distributed pores though which a liquid can percolate. The model is defined on a set of nodes (e.g. organized on a lattice), each node having a label sampled independently from a Bernoulli random variable with expectation p, where label 1 represents a pore. Two adjacent nodes with both having label 1 are automatically connected by an edge. This way random clusters of nodes are obtained by sampling the node labels and automatically connecting adjacent nodes that have label 1. This is the percolation model. If the pore probability is large, then there is a non-zero probability that a large cluster will be formed that connects the left edge of the lattice to the right edge. In this case it is said that the system *percolates* (Fig. 6.1). In this chapter the Potts model and the Swendsen-Wang (SW) method are reviewed. Several interpretations, theorems, and variants of the method are also explained. The chapter concludes with a discussion of subspace clustering and the state-of-the-art C^4 algorithm. Applications include image segmentation, sparse motion segmentation, and subspace clustering.


Jian-Sheng Wang

6.1 Potts Model and Swendsen-Wang

Let $\mathbf{G} =< V, E >$ be an adjacency graph, such as a lattice with 4 nearest neighbor connections. Each vertex $v_i \in V$ has a state variable x_i with a finite number of labels (or colors), $x_i \in \{1, 2, ..., \mathrm{L}\}$. The total number of labels L is predefined. If $\mathbf{X} = (x_1, x_2, ..., x_{|V|})$ denotes the labeling of the graph, then the Ising/Potts model is a Markov random field,

$$\pi_{\mathrm{PTS}}(\mathbf{X}) = \frac{1}{Z} \exp\{- \sum_{<s,t>\in E} \beta_{st} \mathbf{1}(x_s \neq x_t)\}, \tag{6.1}$$

where $\mathbf{1}(x_s \neq x_t)$ is a Boolean function, equal to 1 if its condition $x_s \neq x_t$ is observed and equal to 0 otherwise. If the number of possible labels is $L = 2$, then π is called the Ising model, and if $L \geq 3$, then it is called the Potts model.

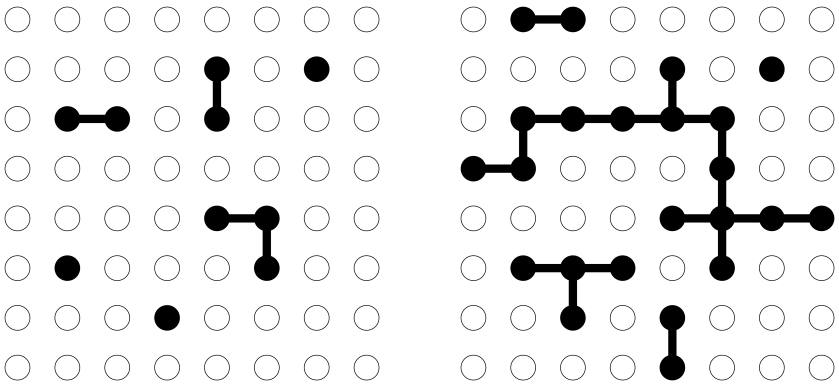
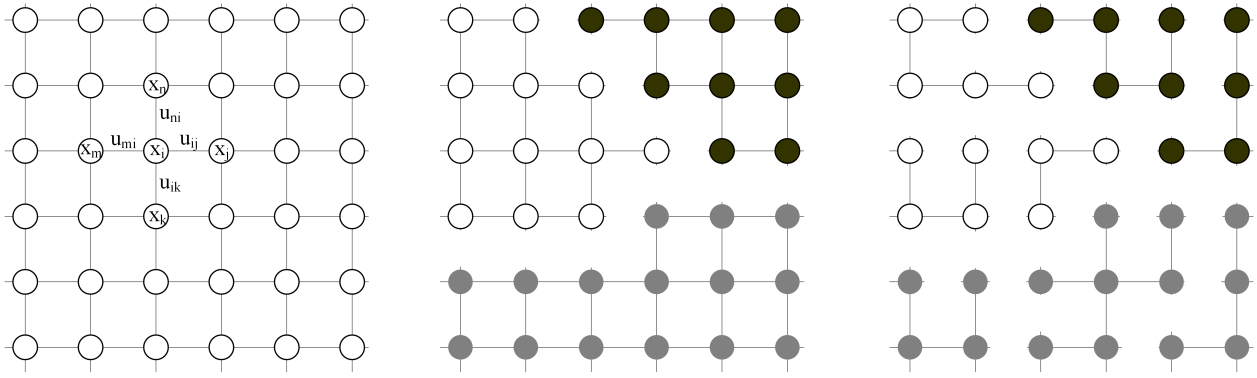


Fig. 6.1: Illustration of the percolation model. Left: the system does not percolate. Right: the system percolates.

Usually we consider $\beta_{st} > 0$ for a ferro-magnetic system that prefers same colors for neighboring vertices. The Potts models and its extensions are used as *a priori* probabilities in many Bayesian inference tasks.



(a) G_0 (b) G (c) CCP

Fig. 6.2: Illustating the SW method. (a) An adjacency graph $\mathbf{G}$ where each edge $e =< s,t >$ is augmented with a binary variable $\mu_e \in \{1,0\}$. (b) A labeling of the Graph $\mathbf{G}$ where the edges connecting vertices of different colors are removed. (c). A number of connected components after turning off some edges in (b) probabilistically. © [2007] Taylor & Francis. Reprinted, with permission, from ref. [4].

The Swendsen-Wang (SW) algorithm introduces a set of auxiliary variables on the edges, as illustrated in Figure.6.2 (a).

$$\mathbf{U} = \{\mu_e : \ \mu_e \in \{0,1\}, \ \forall e \in E\}. \tag{6.2}$$

The edge e is disconnected (or turned off) if and only if $\mu_e = 0$. The binary variable μ_e follows a Bernoulli distribution conditional on the labels of the vertices e connects, x_s, x_t.

$$\mu_e|(x_s,x_t) \sim \text{Bernoulli}(q_e \mathbf{1}(x_s = x_t)), \quad \text{with } q_e = 1 - e^{-\beta_{st}}, \ \forall e \in E. \tag{6.3}$$

Thus, $\mu_e = 1$ with probability q_e if $x_s = x_t$, and $\mu_e = 0$ with probability 1 if $x_s \neq x_t$. Using this setup the SW method iterates the two following steps.

1. **The clustering step**: Given the current labeling $\mathbf{X}$, sample the auxiliary variables in $\mathbf{U}$ according to equation (6.3). First turn off edges e according to μ_e. That is, the edge $e =< s,t >$ is turned off deterministically if $x_s \neq x_t$, as Fig.6.2 (b) shows. Now the full set of edges is given by

$$E = E_{\text{on}}(\mathbf{X}) \cup E_{\text{off}}(\mathbf{X}). \tag{6.4}$$

The remaining edges are turned off the with probability $1 - q_{st} = \exp(-\beta_{st})$. The edges e are divided into an "on" and "off" set depending on μ_e. Therefore the edge set $E_{\text{on}}(\mathbf{X})$ is further divided as,

$$E_{\text{on}}(\mathbf{X}) = E_{\text{on}}(\mathbf{U}, \mathbf{X}) \cup E_{\text{off}}(\mathbf{U}, \mathbf{X}). \tag{6.5}$$

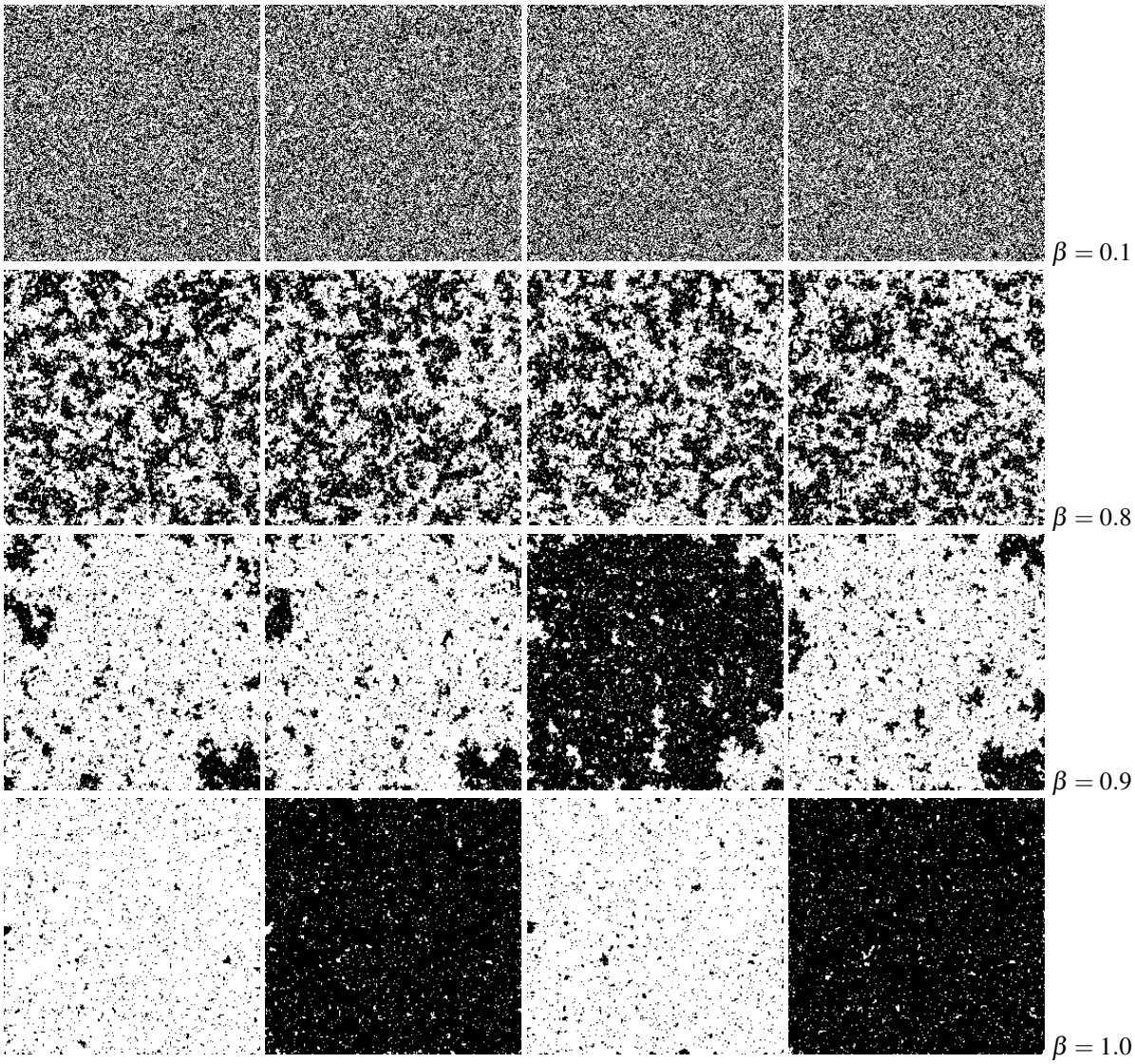


Fig. 6.3: Consecutive samples obtained by the SW algorithm on the Ising model for different values of $\beta_{ij} = \beta$. From top to bottom: $\beta = 0.1, 0.8, 0.9, 1.0$ respectively.

The edges in $E_{\text{on}}(\mathbf{U}, \mathbf{X})$ form a number of connected components, as shown in Figure 6.2 (c). We denote the set of connected components in $E_{\text{on}}(\mathbf{U}, \mathbf{X})$ by,

$$\text{CP}(\mathbf{U}, \mathbf{X}) = \{\text{cp}_i : \ i = 1, 2, ..., K, \ \text{with} \cup_{i=1}^{K} \text{cp}_i = V\}. \tag{6.6}$$

Vertices in each connected component cp_i are guaranteed to have the same color. Intuitively, strongly coupled sites have higher probability of being grouped into a connected component. These connected components are decoupled for now.

2. **The flipping step**: Select one connected component $V_o \in \text{CP}$ at random and assign a common color ℓ to all vertices in V_o. The new label ℓ follows a discrete uniform distribution,

$$x_s = \ell \ \forall s \in V_o, \quad \ell \sim \text{uniform}\{1,2,...,\text{L}\}. \tag{6.7}$$

In this step, one may choose to perform the random color flipping for some or all of the connected components in $\text{CP}(\mathbf{U})$ independently, as they are decoupled. By doing so, all possible labelings of the graph are connected in one step, just like one sweep of the Gibbs sampler.

In a modified version by Wolff [60], one may alternatively choose a vertex $v \in V$ and grow a single connected component following the Bernoulli trials on edges around v. This saves some computation time in the clustering step, but larger components have a higher chance of being selected.

Figure 6.3 shows the results of running the SW algorithm on the Ising model for a lattice of size 256×256 for different values of the parameter $\beta_{ij} = \beta$. For small values of β the samples appear random while for $\beta = 1$ most nodes have the same label. There is a value β_0 around 0.8 or 0.9 at which there is a phase transition between the random phase and the unicolor phase. The value $1/\beta_0$ is called the *critical temperature*.

Using a path coupling technique, Cooper and Frieze [10] have showed that the mixing time τ (see equation (6.40)) is polynomial in the number of vertices N if each vertex in the graph G is connected to $O(1)$ neighbors, i.e. the connectivity of each vertex does not grow with the size of V. This is usually observed in computer vision problems, such as with lattices or planar graphs. The mixing time becomes exponential at worst when the graph G is fully connected [23]. Such cases usually do not occur in vision problems.

The Ising/Potts model $p(\mathbf{x})$ can be used as a prior in the Bayesian model $p(\mathbf{x}|I) \propto p(I|\mathbf{x})p(\mathbf{x})$, where the likelihood $p(I|\mathbf{x})$ measures how well the input image is explained by $\mathbf{x}$. However, the SW algorithm slows down in the presence of the likelihood, also known as an external field. This is the result of clusters being created entirely based on the prior coefficients β_{ij}, ignoring the likelihood. Higdon introduced an auxiliary variable method named partial decoupling [25] that takes into account the likelihood when growing the clusters. However, this approach is still limited to models with Ising/Potts priors. Huber [26] developed a bounding chain method for Potts models (6.1) that can diagnose when the SW Markov chain has converged and thus obtain exact sampling or perfect sampling [45]. The number of steps for reaching exact sampling is on the order of $O(\log|E_o|)$ for values of $\frac{1}{\beta}$ that are far below or far above the critical level.

6.2 Interpretations of the SW Algorithm

There are three different interpretations of the SW algorithm, one as a Metropolis-Hastings algorithm, one as a data augmentation method with auxiliary variables and one as a slice sampling algorithm. For simplicity, in this section we assume that we are working with a homogeneous Potts model with $\beta_{st} = \beta > 0, \forall < s,t > \in E$.

6.2.1 Interpretation 1: Metropolis-Hastings perspective

The SW algorithm can be interpreted as a Metropolis-Hastings step with acceptance probability 1.

Figure 6.4 shows two partition states A and B which differ in the labels of the pixels in a connected component V_o. Suppose the current state is A in which V_o is connected to V_1 which are the remaining black vertices. The edges that are turned off probabilistically between V_o and V_1 form a cut

$$C_{01} = C(V_0, V_1) = \{e =< s,t >: \ s \in V_0, \ t \in V_1\}.$$

The cut is illustrated by the crosses in Figure 6.4. Obviously there are many ways to arrive at connected component V_o through the SW clustering step. However, every way must include turning off the edges in C_{01}. Similarly, if the

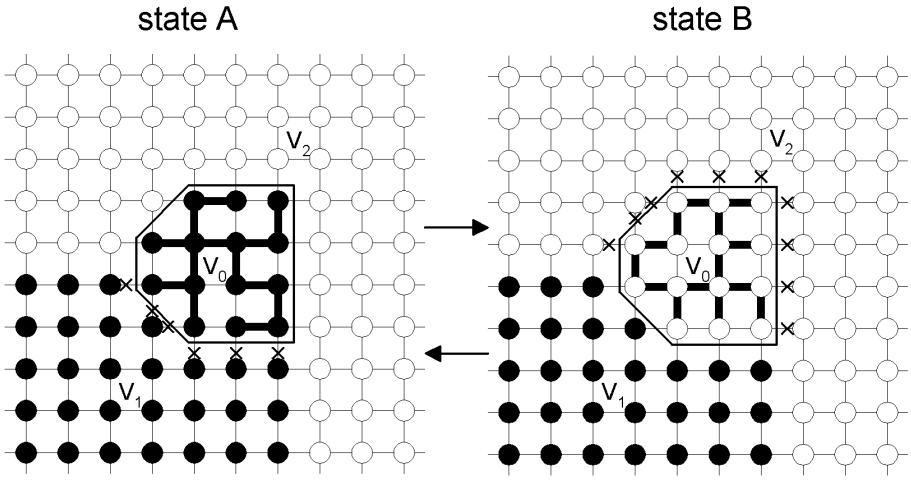


Fig. 6.4: SW algorithm flips a patch of nodes in one step for the Ising/Potts models. © [2005] IEEE. Reprinted, with permission, from ref. [3].

Markov chain is currently at state B, it also has a chance to select a connected component V_o in white. We denote the remaining white vertices as V_2, and the cut between V_0 and V_2 is

$$C_{02} = C(V_0, V_2) = \{e =< s,t >: \ \ s \in V_0,\, t \in V_2\}.$$

So far, we have a pair of states A and B possessing different labels for V_o. A Metropolis-Hastings algorithm is used to create a reversible move between them. While it is difficult to compute the proposal probabilities $Q(A \to B)$ and $Q(B \to A)$, one can compute their ratio easily through the cancellation given by

$$\frac{Q(A \to B)}{Q(B \to A)} = \frac{(1-q)^{|C_{01}|}}{(1-q)^{|C_{02}|}} = (1-q)^{|C_{01}|-|C_{02}|}. \tag{6.8}$$

In the above equation, $|C_{01}|$ and $|C_{02}|$ are the cardinalities of the sets. In other words, the probabilities for selecting V_0 in states A and B are the same, except that the cuts have different sizes. Remarkably the probability ratio $\pi(A)/\pi(B)$ is also decided by the size of the cuts as shown by

$$\frac{\pi(A)}{\pi(B)} = \frac{e^{-\beta|C_{02}|}}{e^{-\beta|C_{01}|}} = e^{\beta(|C_{01}|-|C_{02}|)}. \tag{6.9}$$

The acceptance probability for the move from A to B is then given by

$$\alpha(A \to B) = \min(1, \frac{Q(B \to A)}{Q(A \to B)} \cdot \frac{\pi(B)}{\pi(A)}) = (\frac{e^{-\beta}}{1-q})^{|C_{01}|-|C_{02}|}. \tag{6.10}$$

If the edge probability is chosen to be $q = 1 - e^{-\beta}$, then the proposal from A to B is always accepted with $\alpha(A \to B) = 1$. As β is proportional to the inverse temperature, q tends to 1 at low temperature, and the SW algorithm flips large patches at a time. Therefore, the SW algorithm can mix quickly even at critical temperatures.

Proof. (of equation (6.8)). Let $\mathbf{U}_A|(\mathbf{X} = A)$ and $\mathbf{U}_B|(\mathbf{X} = B)$ be the realizations of the auxiliary variables in states A and B respectively. Following the Bernoulli probabilities in the flipping step leads to two sets of connected components $\mathrm{CP}(\mathbf{U}_A|\mathbf{X} = A)$ and $\mathrm{CP}(\mathbf{U}_B|\mathbf{X} = B)$ respectively. We divide $\mathbf{U}_A$ into two sets for the on and off edges respectively,

$$\mathbf{U}_A = \mathbf{U}_{A,\mathrm{on}} \cap \mathbf{U}_{A,\mathrm{off}}, \tag{6.11}$$

where

$$\mathbf{U}_{A,\text{on}} = \{\mu_e \in \mathbf{U}_A : \mu_e = 1\},\ \mathbf{U}_{A,\text{off}} = \{\mu_e \in \mathbf{U}_A : \mu_e = 0\}.$$

We are only interested in the $\mathbf{U}_A$'s (and thus $\text{CP}(\mathbf{U}_A|\mathbf{X}=A)$'s) which yield the connected component V_o. We collect all such $\mathbf{U}_A$ given A in a set,

$$\Omega(V_o|A) = \{\mathbf{U}_A \text{ s.t. } V_o \in \text{CP}(\mathbf{U}_A|\mathbf{X}=A)\}. \tag{6.12}$$

In order for V_o to be a connected component in A, all edges between V_o and V_1 must be cut (turned off). We denote the set of turned off edges not part of this cut by ${}^{-}\mathbf{U}_{A,\text{off}}$,

$$\mathbf{U}_{A,\text{off}} = C(V_o,V_1) \cup {}^{-}\mathbf{U}_{A,\text{off}}, \quad \forall \mathbf{U}_A \in \Omega(V_o|A). \tag{6.13}$$

Similarly, we collect all $\mathbf{U}_B$ in state B which produce the connected component V_o,

$$\Omega(V_o|B) = \{\mathbf{U}_B \text{ s.t. } V_o \in \text{CP}(\mathbf{U}_B|\mathbf{X}=B)\}. \tag{6.14}$$

In order for V_o to be a connected component in $\mathbf{U}_B|B$, the clustering step must cut all the edges between V_o and V_2. Thus we have

$$\mathbf{U}_B = \mathbf{U}_{B,\text{on}} \cup \mathbf{U}_{B,\text{off}}, \text{ where} \tag{6.15}$$

$$\mathbf{U}_{B,\text{off}} = C(V_o,V_2) \cup {}^{-}\mathbf{U}_{B,\text{off}}, \quad \forall \mathbf{U}_B \in \Omega(V_o|B). \tag{6.16}$$

A key observation in this formulation is that there is a one-to-one mapping between $\Omega(V_o|A)$ and $\Omega(V_o|B)$. This is because any $\mathbf{U}_A \in \Omega(V_o|A)$ has a one to one correspondent $\mathbf{U}_B \in \Omega(V_o|B)$ obtained as

$$\mathbf{U}_{B,\text{on}} = \mathbf{U}_{A,\text{on}},\ \mathbf{U}_{B,\text{off}} = {}^{-}\mathbf{U}_{A,\text{off}} \cup C(V_o,V_2). \tag{6.17}$$

That is, $\mathbf{U}_A$ and $\mathbf{U}_B$ differ only in the cuts $C(V_o,V_1)$ and $C(V_o,V_2)$, where all auxiliary variables are off. Because of this, their connected components are the same

$$\text{CP}(\mathbf{U}_A|\mathbf{X}=A) = \text{CP}(\mathbf{U}_B|\mathbf{X}=B) \tag{6.18}$$

Similarly, any $\mathbf{U}_B \in \Omega(V_o|B)$ has a one to one correspondent $\mathbf{U}_A \in \Omega(V_o|A)$.

Now suppose that we choose $V_o \in \text{CP}(\mathbf{U}_A|\mathbf{X}=A)$ with uniform probability from all connected components in $\text{CP}(\mathbf{U}_A|\mathbf{X}=A)$. The probability for choosing V_o at state A is then

$$q(V_o|A) = \sum_{\mathbf{U}_A \in \Omega(V_o|A)} \frac{1}{|\text{CP}(\mathbf{U}_A|\mathbf{X}=A)|} \prod_{e \in \mathbf{U}_{A,\text{on}}} q_e \prod_{e \in {}^{-}\mathbf{U}_{A,\text{off}}} (1-q_e) \prod_{e \in C(V_o,V_1)} (1-q_e). \tag{6.19}$$

Similarly, the probability for choosing V_o at state B is

$$q(V_o|B) = \sum_{\mathbf{U}_B \in \Omega(V_o|B)} \frac{1}{|\text{CP}(\mathbf{U}_B|\mathbf{X}=B)|} \prod_{e \in \mathbf{U}_{B,\text{on}}} q_e \prod_{e \in {}^{-}\mathbf{U}_{B,\text{off}}} (1-q_e) \prod_{e \in C(V_o,V_2)} (1-q_e). \tag{6.20}$$

Dividing equation (6.19) by equation (6.20), we obtain the ratio in equation (6.8) due to cancellation through the one-to one correspondence between $\Omega(V_o|A)$ and $\Omega(V_o|B)$. In the special case where $C(V_o,V_1) = \emptyset$, $\prod_{e \in C(V_o,V_1)}(1-q_e) = 1$. □

Note that the proof holds for arbitrary design of q_e.

One slight complication occurs when there are two paths connecting the two states, as illustrated in Figure 6.5.
Path 1. Choose $V_o = V_1$. In state A, choose a new label $\ell = 2$, i.e. merge V_o into V_2, and opposingly in state B, choose a new label $\ell = 1$, i.e. split V_o from V_2.
Path 2. Choose $V_o = V_2$. In state A, choose a new label $\ell = 1$, i.e. merge V_o into V_1, and reversely in state B, choose $\ell = 2$, i.e. split V_o from V_1. In this case the proposal probability ratio is,

$$\frac{Q(B \to A)}{Q(A \to B)} = \frac{q(V_o = V_1|B)q(\mathbf{X}_{V_o} = 2|V_o,B) + q(V_o = V_2|B)q(\mathbf{X}_{V_o} = 1|V_o,B)}{q(V_o = V_1|A)q(\mathbf{X}_{V_o} = 1|V_o,A)) + q(V_o = V_2|A)q(\mathbf{X}_{V_o} = 2|V_o,A)}. \tag{6.21}$$

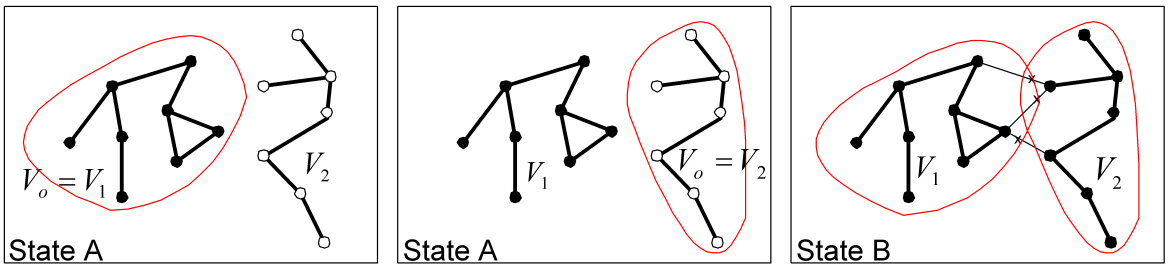


Fig. 6.5: State A has two subgraphs V_1 and V_2 which are merged in state B. In this case there are two paths to go from state A to state B, one by selecting $V_o = V_1$, the other one by selecting $V_o = V_2$. © [2005] IEEE. Reprinted, with permission, from ref. [3].

In state A, the cut $C(V_o, V_\ell \setminus V_o) = \emptyset$ for both paths, and in state B the cut is $C(V_1, V_2)$ for both paths. Following Proposition 6.6, the probability ratios for choosing $V_o = V_1$ and $V_o = V_2$ are equal and given by,

$$\frac{q(V_o = V_1|A)}{q(V_o = V_1|B)} = \frac{1}{\prod_{e \in C(V_1,V_2)}(1-q_e)} = \frac{q(V_o = V_2|A)}{q(V_o = V_2|B}. \tag{6.22}$$

Once V_o is selected such that either $V_o = V_1$ or $V_o = V_2$, the remaining partition for both A and B are the same, and it is denoted by $\mathbf{X}_{V \setminus V_o}$. In proposing the new label of V_o, we easily observe that

$$\frac{q(\mathbf{X}_{V_o} = 2|V_o = V_1, B)}{q(\mathbf{X}_{V_o} = 1|V_o = V_2, A)} = \frac{q(\mathbf{X}_{V_o} = 1|V_o = V_2, B)}{q(\mathbf{X}_{V_o} = 2|V_o = V_1, A)}. \tag{6.23}$$

Under this approach the acceptance rate remains 1.

6.2.2 Interpretation 2: data augmentation

The second interpretation follows Edward and Sokal [14], who augmented the Potts model to a joint probability for both $\mathbf{X}$ and $\mathbf{U}$,

$$p_{\mathrm{ES}}(\mathbf{X}, \mathbf{U}) = \frac{1}{Z} \prod_{e=<s,t> \in E} [(1-\rho)\mathbf{1}(\mu_e = 0) + \rho \mathbf{1}(\mu_e = 1) \cdot \mathbf{1}(x_s = x_t)] \tag{6.24}$$

$$= \frac{1}{Z}[(1-\rho)^{|E_{\mathrm{off}}(\mathbf{U})|} \cdot \rho^{|E_{\mathrm{on}}(\mathbf{U})|}] \cdot \prod_{<s,t> \in E_{\mathrm{on}}(\mathbf{U})} \mathbf{1}(x_s = x_t). \tag{6.25}$$

The second factor in equation (6.25) is in fact a hard constraint on $\mathbf{X}$ and $\mathbf{U}$. Let the space of $\mathbf{X}$ be

$$\Omega = \{1, 2, ..., \mathrm{L}\}^{|V|}. \tag{6.26}$$

Under this hard constraint, the labeling $\mathbf{X}$ is reduced to a subspace $\Omega_{\mathrm{CP}(\mathbf{U})}$ where each connected component must have the same label,

$$\prod_{<s,t> \in E_{\mathrm{on}}(\mathbf{U})} \mathbf{1}(x_s = x_t) = \mathbf{1}(\mathbf{X} \in \Omega_{\mathrm{CP}(\mathbf{U})}). \tag{6.27}$$

The joint probability $p_{\mathrm{ES}}(\mathbf{X}, \mathbf{U})$ observes two nice properties, both of which are easy to verify.

Proposition 6.1 *The Potts model is a marginal probability of the joint probability,*

$$\sum_{\mathbf{U}} p_{\mathrm{ES}}(\mathbf{X}, \mathbf{U}) = \pi_{\mathrm{PTS}}(\mathbf{X}). \tag{6.28}$$

The other marginal probability is the random cluster model π_{RCM},

$$\sum_{\mathbf{X}} p_{\text{ES}}(\mathbf{X},\mathbf{U}) = \pi_{\text{RCM}}(\mathbf{U}) = \frac{1}{Z}(1-\rho)^{|E_{\text{off}}(\mathbf{U})|}\cdot\rho^{E_{\text{on}}(\mathbf{U})}\mathrm{L}^{|\text{CP}(\mathbf{U})|}. \tag{6.29}$$

Proof. Writing $U=\{\mu_1,...,\mu_{|E|}\}$, and $(1-\rho)\mathbf{1}(\mu_e=0)+\rho\mathbf{1}(\mu_e=1)\cdot\mathbf{1}(x_s=x_t)=f(\mu_e,x_s,x_t)$ for $e=<s,t>$, we have

$$\begin{aligned}\sum_{\mathbf{U}} p_{\text{ES}}(\mathbf{X},\mathbf{U}) &= \frac{1}{Z}\sum_{\mu_1=0}^{1}...\sum_{\mu_{|E|}=0}^{1}\prod_{e=<s,t>\in E} f(\mu_e,x_s,x_t)\\ &= \frac{1}{Z}\sum_{\mu_1=0}^{1}...\sum_{\mu_{|E|}=0}^{1} f(\mu_1,x_{s_1},x_{t_1})f(\mu_2,x_{s_2},x_{t_2})\cdot...\cdot f(\mu_{|E|},x_{s_{|E|}},x_{t_{|E|}}),\end{aligned} \tag{6.30}$$

where $<s_1,t_1>$ is the edge corresponding to μ_1, $<s_2,t_2>$ to μ_2, and so on. Therefore we have

$$\begin{aligned}\sum_{\mathbf{U}} p_{\text{ES}}(\mathbf{X},\mathbf{U}) &= \frac{1}{Z}\left[\sum_{\mu_1=0}^{1} f(\mu_1,x_{s_1},x_{t_1})\right]\cdot...\cdot\left[\sum_{\mu_{|E|}=0}^{1} f(\mu_{|E|},x_{s_{|E|}},x_{t_{|E|}})\right]\\ &= \frac{1}{Z}\prod_{e=<s,t>\in E}\sum_{\mu_e=0}^{1}[(1-\rho)\mathbf{1}(\mu_e=0)+\rho\mathbf{1}(\mu_e=1)\cdot\mathbf{1}(x_s=x_t)]\\ &= \frac{1}{Z}\prod_{e=<s,t>\in E}[(1-\rho)+\rho\cdot\mathbf{1}(x_s=x_t)] = \pi_{\text{PTS}}(\mathbf{X}).\end{aligned} \tag{6.31}$$

For the second marginal, we observe

$$\begin{aligned}\sum_{\mathbf{X}} p_{\text{ES}}(\mathbf{X},\mathbf{U}) &= \sum_{\mathbf{X}}\frac{1}{Z}[(1-\rho)^{|E_{\text{off}}(\mathbf{U})|}\cdot\rho^{|E_{\text{on}}(\mathbf{U})|}]\cdot\prod_{<s,t>\in E_{\text{on}}(\mathbf{U})}\mathbf{1}(x_s=x_t)\\ &= \frac{1}{Z}[(1-\rho)^{|E_{\text{off}}(\mathbf{U})|}\cdot\rho^{|E_{\text{on}}(\mathbf{U})|}]\cdot\sum_{\mathbf{X}}\prod_{<s,t>\in E_{\text{on}}(\mathbf{U})}\mathbf{1}(x_s=x_t)\end{aligned} \tag{6.32}$$

All nodes of a connected component $c_i\in\text{CP}(\mathbf{U})$ must have the same label, otherwise the product $\prod_{<s,t>\in E_{\text{on}}(\mathbf{U})}\mathbf{1}(x_s=x_t)=0$. Furthermore, each connected component could be labeled independently with one of the L labels, therefore

$$\sum_{\mathbf{X}}\prod_{<s,t>\in E_{\text{on}}(\mathbf{U})}\mathbf{1}(x_s=x_t) = L^{|\text{CP}(\mathbf{U})|}.\square$$

Proposition 6.2 *The conditional probabilities of $p_{\text{ES}}(\mathbf{X},\mathbf{U})$ are*

$$p_{\text{ES}}(\mathbf{U}|\mathbf{X}) = \prod_{<s,t>\in E} p(\mu_e|x_s,x_t),\quad \text{with } p(\mu_e|x_s,x_t)=\text{Bernoulli}(\rho\mathbf{1}(x_s=x_t)), \tag{6.33}$$

$$p_{\text{ES}}(\mathbf{X}|\mathbf{U}) = \text{unif}[\Omega_{\text{CP}(\mathbf{U})}] = (\frac{1}{\mathrm{L}})^{|\text{CP}(\mathbf{U})|}\text{ for }\mathbf{X}\in\Omega_{\text{CP}(\mathbf{U})};\ \ 0\ \text{ otherwise.} \tag{6.34}$$

Proof. We have that $p_{\text{ES}}(\mathbf{U}|\mathbf{X})\propto\prod_{<s,t>\in E}p(\mu_{st}|x_s,x_t)$ with $p(\mu_{st}|x_s,x_t)=(1-\rho)\mathbf{1}(\mu_{st}=0)+\rho\mathbf{1}(\mu_{st}=1)\cdot\mathbf{1}(x_s=x_t)$. Therefore

$$p(\mu_{st}|x_s,x_t)\propto(1-\rho)\mathbf{1}(\mu_{st}=0)+\rho\mathbf{1}(\mu_{st}=1)\cdot\mathbf{1}(x_s=x_t) = \begin{cases}(1-\rho)\mathbf{1}(\mu_{st}=0) & \text{if } x_s\neq x_t\\ (1-\rho)\mathbf{1}(\mu_{st}=0)+\rho\mathbf{1}(\mu_{st}=1) & \text{if } x_s=x_t\end{cases} \tag{6.35}$$

Thus if $x_s\neq x_t$ we have $p(\mu_{st}|x_s,x_t)\propto(1-\rho)\mathbf{1}(\mu_{st}=0)$ so $p(\mu_{st}=1|x_s,x_t)=0$ in this case. If $x_s=x_t$ we have $p(\mu_{st}|x_s,x_t)\propto(1-\rho)\mathbf{1}(\mu_{st}=0)+\rho\mathbf{1}(\mu_{st}=1)$ so in this case $p(\mu_{st}=1|x_s,x_t)=\rho$ and $p(\mu_{st}=0|x_s,x_t)=1-\rho$. This

proves that

$$p_{\mathrm{ES}}(\mathbf{U}|\mathbf{X}) \propto \prod_{<s,t>\in E} \mathrm{Bernoulli}(\rho \mathbf{1}(x_s = x_t)) \tag{6.36}$$

Since the right hand side is a proper probability, we have that eq. (6.32) holds.

For the second conditional probability, we have

$$\begin{aligned} p_{\mathrm{ES}}(\mathbf{X}|\mathbf{U}) &= \frac{1}{Z_1} \prod_{e\in E} [(1-\rho)\mathbf{1}(\mu_e = 0) + \rho \mathbf{1}(\mu_e = 1) \cdot \mathbf{1}(x_s = x_t)] \\ &= \frac{1}{Z_1} \prod_{e\in E_{\mathrm{on}}(\mathbf{U})} [\rho \cdot \mathbf{1}(x_s = x_t)] \prod_{e\in E_{\mathrm{off}}(\mathbf{U})} (1-\rho) \\ &= \frac{\prod_{e\in E_{\mathrm{on}}(\mathbf{U})}(1-\rho)\prod_{e\in E_{\mathrm{off}}(\mathbf{U})}\rho}{Z_1} \prod_{e\in E_{\mathrm{on}}(\mathbf{U})} \mathbf{1}(x_s = x_t) = \frac{1}{Z_2} \prod_{<s,t>\in E_{\mathrm{on}}(\mathbf{U})} \mathbf{1}(x_s = x_t) \end{aligned} \tag{6.37}$$

All nodes of a connected component $c_i \in \mathrm{CP}(\mathbf{U})$ must have the same label, otherwise the product $\prod_{<s,t>\in E_{\mathrm{on}}(\mathbf{U})} \mathbf{1}(x_s = x_t) = 0$. Furthermore, each connected component could be labeled independently with one of the L labels, therefore

$$p_{\mathrm{ES}}(\mathbf{X}|\mathbf{U}) = \begin{cases} (1/L)^{|\mathrm{CP}(\mathbf{U})|} & \text{if } \mathbf{X} \in \Omega_{\mathrm{CP}(\mathbf{U})} \\ 0 & \text{otherwise} \end{cases} \square$$

Therefore, the two SW steps can be viewed as sampling the two conditional probabilities.

1. Clustering step: $\mathbf{U} \sim p_{\mathrm{ES}}(\mathbf{U}|\mathbf{X})$, i.e. $\mu_e|(x_s, x_t) \sim \mathrm{Bernoulli}(\rho \mathbf{1}(x_s = x_t))$.
2. Flipping step: $\mathbf{X} \sim p_{\mathrm{ES}}(\mathbf{U}|\mathbf{X})$, i.e. $\mathbf{X}(\mathrm{cp}_i) \sim \mathrm{Unif}\{1, 2, ..., \mathrm{L}\}$, $\forall \mathrm{cp}_i \in \mathrm{CP}(\mathbf{U})$.

Since $(\mathbf{X}, \mathbf{U}) \sim p_{\mathrm{ES}}(\mathbf{X}, \mathbf{U})$, after discarding the auxiliary variables $\mathbf{U}$, $\mathbf{X}$ follows the marginal distribution of $p_{\mathrm{ES}}(\mathbf{X}, \mathbf{U})$. The goal is achieved, and

$$\mathbf{X} \sim \pi_{\mathrm{PTS}}(\mathbf{X}). \tag{6.38}$$

The benefit of this data augmentation method (Tanner and Wong [52]) is that the labels of the connected components are completely decoupled (independent) given the auxiliary variables. As $\rho = 1 - e^{-\beta}$, it tends to choose smaller clusters if the temperature in the Potts model is high, and larger clusters if the temperature is low. So it overcomes the coupling problem of the single site Gibbs sampler.

6.3 Some Theoretical Results

Let the Markov chain have kernel $\mathcal{K}$ and initial state $\mathbf{X}_o$, after t steps the state follows probability $p_t = \delta(\mathbf{X} - \mathbf{X}_o)\mathcal{K}^t$, where $\delta(\mathbf{X} - \mathbf{X}_o)$ is given by

$$\delta(\mathbf{X} - \mathbf{X}_o) = \begin{cases} 1, \text{ if } \mathbf{X} = \mathbf{X}_o. \\ 0, \text{ otherwise.} \end{cases}$$

is the initial probability. The convergence of the Markov chain is often measured by the total variation

$$||p_t - \pi||_{\mathrm{TV}} = \frac{1}{2} \sum_{\mathbf{X}} |p_t(\mathbf{X}) - \pi(\mathbf{X})|. \tag{6.39}$$

The mixing time of the Markov chain is defined by

$$\tau = \max_{\mathbf{X}_o} \min\{t : \ ||p_t - \pi||_{\mathrm{TV}} \leq \varepsilon\}. \tag{6.40}$$

τ is a function of ε and the graph complexity $M = |\mathbf{G}_o|$ in terms of the number of vertices and connectivity. The Markov chain is said to mix rapidly if $\tau(M)$ is polynomial or logarithmic.

Empirically, the SW method is found to mix rapidly. Recently some analytical results on its performance have surfaced. Cooper and Frieze [10] used a path coupling technique to prove that SW mixes rapidly on sparsely connected graphs.

Theorem 6.3 (Cooper and Frieze 1999) *Let $n = |V|$ and Δ be the maximum number of edges at any single vertex, and* L *the number of colors in the Potts model. If* $\mathbf{G}$ *is a tree, then the SW mixing time is $O(n)$ for any β and* L*. If $\Delta = O(1)$, then there exists $\rho_o = \rho(\Delta)$ such that if $\rho \le \rho_o$ (i.e. higher than a certain temperature), then SW has polynomial mixing time for all* L*.*

A negative case was constructed by Gore and Jerrum [23] on complete graphs.

Theorem 6.4 (Gore and Jerrum 1999) *If* $\mathbf{G}$ *is a complete graph and* $\mathrm{L} > 2$*, then for $\beta = \frac{2(\mathrm{L}-1)\ln(\mathrm{L}-1)}{n(\mathrm{L}-2)}$, the SW does not mix rapidly.*

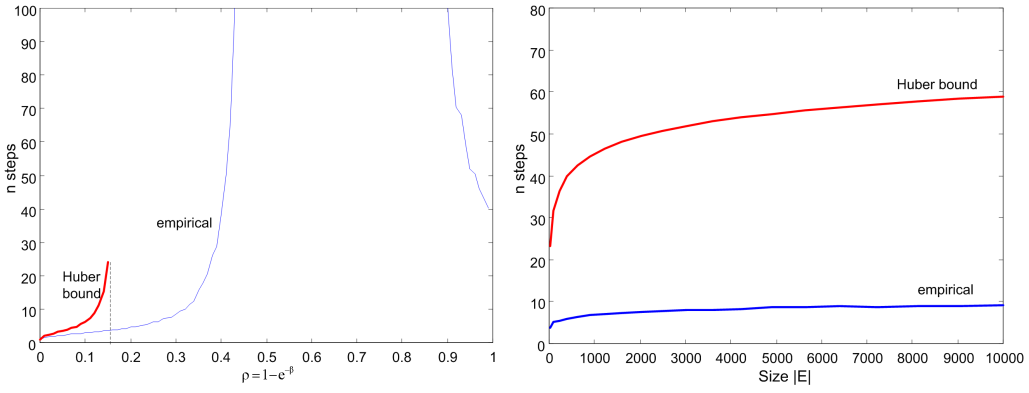
In image analysis applications, the graph often observes the Copper-Frieze condition and is far from being complete.

Most recently an exact sampling technique was developed for SW on the Potts model by Huber [26] for extreme temperatures. This approach designs a bounding chain which assumes that each vertex $s \in V$ has a set of colors S_s initialized with the full set $|S_s| = \mathrm{L}$, $\forall s$. The Bernoulli probability for the auxiliary variables μ_e is changed to

$$\mathbf{U}^{\mathrm{bd}} = \{\mu_e^{\mathrm{bd}} : \mu_e^{\mathrm{bd}} \in \{0,1\},\ \mu_e \sim \mathrm{Bernoulli}(\rho \mathbf{1}(S_s \cap S_t \neq \emptyset))\}. \tag{6.41}$$

Thus, $\mathbf{U}^{\mathrm{bd}}$ has more edges than $\mathbf{U}$ in the original SW chain, i.e. $\mathbf{U} \subset \mathbf{U}^{\mathrm{bd}}$. When $\mathbf{U}^{\mathrm{bd}}$ collapses to $\mathbf{U}$, then all SW chains starting with arbitrary initial states have collapsed into the current single chain. Thus, the chain must have converged (exact sampling). The step for collapsing is called the "coupling time".

Theorem 6.5 (Huber 2003) *Let $n = |V|$ and $m = |E|$. At higher temperature, $\rho < \frac{1}{2(\Delta-1)}$, the bounding chain couples completely by time $O(\ln(2m))$ with probability at least $1/2$. At lower temperature, $\rho \ge 1 - \frac{1}{m\mathrm{L}}$, then the coupling time is $O((m\mathrm{L})^2)$ with probability at least $1/2$.*



(a) (b)

Fig. 6.6: The coupling time empirical plots and the Huber bounds for Ising model.

In fact, the Huber bound is not as tight as one may expect. Fig. 6.6(a) plots the results on a 5×5 lattice with torus boundary condition on the Ising model for the empirical coupling time against $\rho = 1 - e^{-\beta}$. The coupling time is large near the critical temperature (not shown). The Huber bound for the high temperature starts with $\rho_o = 0.16$ and is shown by the short curve. The bound for the low temperature starts with $\rho_o > 0.99$ which is not visible. Fig.6.6.(b) plots the coupling time at $\rho = 0.15$ against the graph size $m = |E|$ and the Huber bound.

Despite the encouraging successes discussed above, the SW method is limited in two ways.

1. It is only valid for the Ising and Potts models, and furthermore it requires that the number of colorings L be known. In many applications, such as image analysis, L is the number of objects (or image regions), which has to be inferred from the input data.
2. It slows down quickly in the presence of an external field, i.e. input data. For example, in an image analysis problem, our goal is to infer the label $\mathbf{X}$ from the input image $\mathbf{I}$ and the target probability is a Bayesian posterior probability where $\pi_{\text{PTS}}(\mathbf{X})$ is used as a prior model,

$$\pi(\mathbf{X}) = \pi(\mathbf{X}|\mathbf{I}) \propto \mathscr{L}(\mathbf{I}|\mathbf{X})\pi_{\text{PTS}}(\mathbf{X}). \tag{6.42}$$

Here, $\mathscr{L}(\mathbf{I}|\mathbf{X})$ is the likelihood model, such as independent Gaussians $N(\bar{\mathbf{I}}_c, \sigma_c^2)$ for each coloring $c = 1, 2, ..., \text{L}$,

$$\mathscr{L}(\mathbf{I}|\mathbf{X}) \propto \prod_{c=1}^{\text{L}} \prod_{x_i=c} \frac{1}{\sqrt{2\pi}\sigma_c} \exp\{-\frac{(\mathbf{I}(v_i) - \bar{\mathbf{I}}_c)^2}{2\sigma_c^2}\}. \tag{6.43}$$

The slowdown is partially attributed to the fact that the Bernoulli probability $\rho = 1 - e^{-\beta}$ for the auxiliary variable is calculated independent of the input image.

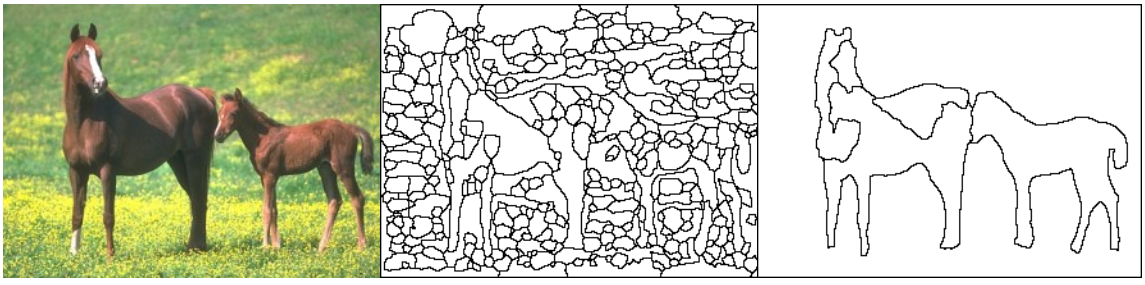
6.4 Swendsen-Wang Cuts for Arbitrary Probabilities

Adrian Barbu

In this section, we generalize the SW algorithm to arbitrary probabilities from the perspective of the Metropolis-Hastings method [24, 39]. The Swendsen-Wang Cuts (SWC) method iterates three steps: (i) a clustering step driven by data, (ii) a label flipping step which can introduce new labels, and (iii) an acceptance step for the proposed labeling. A key feature of this algorithm is the simple formula for calculating the acceptance probabilities.

We delineate the three steps in the following subsections and then show how it reduces to the original SW algorithm for Potts models.

We illustrate the algorithm with an image segmentation example. Figure 6.7.(a) is an input image $\mathbf{I}$ on a lattice Λ, which is decomposed into a number of "superpixels" in (b) to reduce the graph size in a preprocessing stage. Each superpixel has nearly constant intensity and is a vertex in the graph. Two vertices are connected if their superpixels share a boundary. Figure (c) is the result of using the SWC algorithm to optimize a Bayesian probability $\pi(\mathbf{X}) = \pi(\mathbf{X}|\text{I})$ (see section (6.6) for details). The result $\mathbf{X}$ assigns a uniform color to all vertices in each closed region which hopefully correspond to objects in the scene. Note that the number of objects or colors L is unknown, and we do not distinguish between permutations of the labels.



(a). Input image (b) Superpixels (c) Segmentation

Fig. 6.7: Example of image segmentation. (a) Input image. (b) Superpixels obtained by edge detection followed by edge tracing and contour closing. Each superpixel is a vertex in the graph. (c) Segmentation (labeling) result where each closed region is assigned a color or label. © [2007] Taylor & Francis. Reprinted, with permission, from ref. [4].

6.4.1 Step 1: data-driven clustering

We first augment the adjacency graph $\mathbf{G}$ with a set of binary variables on the edges $\mathbf{U} = \{\mu_e : e =< s,t >\in E\}$, as in the original SW method. Each μ_e follows a Bernoulli probability depending on the current state of the two vertices x_s and x_t,

$$\mu_e|(x_s,x_t) \sim \text{Bernoulli}(q_e\mathbf{1}(x_s = x_t)), \ \forall < s,t >\in E. \tag{6.44}$$

Here, q_e is a probability on edge $e =< s,t >$ which tells how likely it is that the two vertices s and t have the same label. In Bayesian inference, where the target $\pi(\mathbf{X})$ is a posterior probability, q_e can be better informed by the data.

For the image segmentation example, q_e is computed based on the similarity between image intensities at s and t (or their local neighborhood) and may be an approximation of the marginal probability of $\pi(\mathbf{X}|\mathrm{I})$,

$$q_e = q(x_s = x_t|\mathbf{I}(s),\mathbf{I}(t)) \ \approx \ \pi(x_s = x_t|\mathrm{I}). \tag{6.45}$$

There are many ways to compute $q(x_s = x_t|\mathbf{I}(v_s),\mathbf{I}(v_t))$ using so called discriminative methods, but it is beyond the scope of this book to discuss in detail.

Fig. 6.8: Three examples of connected components for the horse image computed using discriminative edge probabilities. Given that $\mathbf{X}$ is a uniform color $\mathbf{X} = c$ for all vertices. © [2007] Taylor & Francis. Reprinted, with permission, from ref. [4].

This method will work for any q_e, but a good approximation will inform the clustering step and achieve faster convergence empirically. Fig. 6.8 shows several clustering examples of the horse image. In these examples, we set all vertices to the same color ($\mathbf{X} = c$) and sample the edge probability independently,

$$\mathbf{U}|\mathbf{X} = c \ \sim \prod_{<s,t>\in E} \text{Bernoulli}(q_e). \tag{6.46}$$

The connected components in $\text{CP}(\mathbf{U})$ are shown with different colors. We repeat the clustering step three times. As we can see, the edge probabilities lead to meaningful clusters which correspond to distinct objects in the image. Such effects cannot be observed using constant edge probabilities.

6.4.2 Step 2: color flipping

Let $\mathbf{X} = (x_1,x_2,...,x_{|V|})$ be the current coloring state. The edge variables $\mathbf{U}$ are sampled conditionally on $\mathbf{X}$ and decompose $\mathbf{X}$ into a number of connected components

$$\text{CP}(\mathbf{U}|\mathbf{X}) = \{\text{cp}_i : i = 1,2,...,N(\mathbf{U}|\mathbf{X})\}. \tag{6.47}$$

Suppose we select one connected component $V_o \in \text{CP}(\mathbf{U}|\mathbf{X})$ with color $\mathbf{X}_{V_o} = \ell \in \{1,2,...,L\}$, and assign its color to be $\ell' \in \{1,2,...,L,L+1\}$ with probability $q(l'|V_o,\mathbf{X})$ (to be designed shortly). We obtain a new state $\mathbf{X}'$. There are three cases, as shown in Figure 6.9.

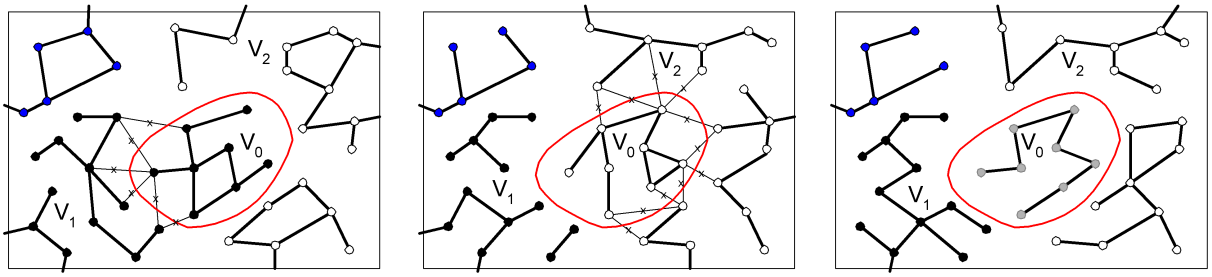


Fig. 6.9: A reversible move between three partition states $\mathbf{X}_A$(left), $\mathbf{X}_B$ (middle), and $\mathbf{X}_C$(right) that differ only in the color of the set V_0. The vertices connected by thick edges form a connected component. The thin lines marked with crosses are edges in the SW-cuts. © [2005] IEEE. Reprinted, with permission, from ref. [3].

1. The canonical case: $V_o \subset V_\ell$ and $\ell' \leq L$. That is, a portion of V_ℓ is re-grouped into an existing color $V_{\ell'}$, and the number of colors remains $\mathrm{L} = L$ in $\mathbf{X}'$. This is the move between states $\mathbf{X}_A$ and $\mathbf{X}_B$ in Figure 6.9.
2. The merge case: $V_o = V_\ell$ in $\mathbf{X}$ is the set of all vertices that have color ℓ and $\ell' \leq L$, and $\ell \neq \ell'$. Color V_ℓ is merged with $V_{\ell'}$, and the number of distinct colors reduces to $L-1$ in $\mathbf{X}'$. This is the move from state $\mathbf{X}_C$ to $\mathbf{X}_A$ or from $\mathbf{X}_C$ to $\mathbf{X}_B$ in Figure 6.9.
3. The split case: $V_o \subset V_\ell$ and $\ell' = L+1$. V_ℓ is split into two pieces and the number of distinct color increases to $L+1$ in $\mathbf{X}'$. This is the move from state $\mathbf{X}_A$ to $\mathbf{X}_C$ or from $\mathbf{X}_B$ to $\mathbf{X}_C$ in Figure 6.9.

Note that this color flipping step is also different from the original SW with Potts model as we allow new colors in each step. The number of colors L is not fixed.

6.4.3 Step 3: accepting the flip

The two previous steps have proposed a move between two states $\mathbf{X}$ and $\mathbf{X}'$ which differ in the coloring of the connected component V_o. In the third step we accept the move with probability

$$\alpha(\mathbf{X} \to \mathbf{X}') = \min\{1, \frac{q(\mathbf{X}' \to \mathbf{X})}{q(\mathbf{X} \to \mathbf{X}')} \cdot \frac{\pi(\mathbf{X}')}{\pi(\mathbf{X})}\}. \tag{6.48}$$

$q(\mathbf{X}' \to \mathbf{X})$ and $q(\mathbf{X} \to \mathbf{X}')$ are the proposal probabilities between $\mathbf{X}$ and $\mathbf{X}'$. If the proposal is rejected, the Markov chain stays at state $\mathbf{X}$. The transition kernel is

$$\mathcal{K}(\mathbf{X} \to \mathbf{X}') = q(\mathbf{X} \to \mathbf{X}')\alpha(\mathbf{X} \to \mathbf{X}'), \quad \forall \mathbf{X} \neq \mathbf{X}'. \tag{6.49}$$

For the canonical case, there is a unique path for moving between $\mathbf{X}$ and $\mathbf{X}'$ in one step – choosing V_o and changing its color. The proposal probability is the product of the probability ratio for selecting V_o as the candidate in the clustering step in both states $\mathbf{X}$ and $\mathbf{X}'$, and the probability ratio for selecting the new labels for V_o in the flipping step. This product is given by

$$\frac{q(\mathbf{X}' \to \mathbf{X})}{q(\mathbf{X} \to \mathbf{X}')} = \frac{q(V_o|\mathbf{X}')}{q(V_o|\mathbf{X})} \cdot \frac{q(\mathbf{X}_{V_o} = \ell|V_o, \mathbf{X}')}{q(\mathbf{X}_{V_o} = \ell'|V_o, \mathbf{X})}. \tag{6.50}$$

For the split and merge cases, there are two paths between $\mathbf{X}$ and $\mathbf{X}'$, but this does not change the conclusion. Now we compute the probability ratio $\frac{q(V_o|\mathbf{X}')}{q(V_o|\mathbf{X})}$ for proposing V_o.

Definition 6.1 *Let $\mathbf{X} = (V_1, V_2, ..., V_{\mathrm{L}})$ be a coloring state, and $V_o \in \mathrm{CP}(U|\mathbf{X})$ a connected component, the "cut" between V_o and V_k is a set of edges between V_o and $V_k \backslash V_o$,*

$$C(V_o, V_k) = \{< s,t >: s \in V_o, t \in V_k \backslash V_o\}, \ \forall k.$$

One of the key observations is that the ratio $\frac{q(V_o|\mathbf{X}')}{q(V_o|\mathbf{X})}$ only depends on the cuts between V_o and the rest of the vertices.

Proposition 6.6 *In the above notation, we have*

$$\frac{q(V_o|\mathbf{X})}{q(V_o|\mathbf{X}')} = \frac{\prod_{<i,j>\in C(V_o,V_\ell)}(1-q_{ij})}{\prod_{<i,j>\in C(V_o,V_{\ell'})}(1-q_{ij})}, \tag{6.51}$$

where the q_e's are the edge probabilities.

Thus the acceptance probability is given in the following Theorem.

Theorem 6.7 *The acceptance probability for the proposed swapping is*

$$\alpha(\mathbf{X}\to\mathbf{X}') = \min\{1, \frac{\prod_{<i,j>\in C(V_0,V_{\ell'})}(1-q_{ij})}{\prod_{<i,j>\in C(V_0,V_\ell)}(1-q_{ij})}\cdot\frac{q(\mathbf{X}_{V_0}=\ell|V_0,\mathbf{X}')}{q(\mathbf{X}_{V_0}=\ell'|V_0,\mathbf{X})}\cdot\frac{\pi(\mathbf{X}')}{\pi(\mathbf{X})}\}. \tag{6.52}$$

For the proof see [3].

Example 6.1. In image analysis, $\pi(\mathbf{X})$ is a Bayesian posterior $\pi(\mathbf{X}|\mathbf{I}) \propto \mathscr{L}(\mathbf{I}|\mathbf{X})p_o(\mathbf{X})$ with the prior probability $p_o(\mathbf{X})$ being a Markov random field model (like Potts in equation (6.43)). One can compute the ratio of the target probabilities in the local neighborhood ∂V_o of V_o by

$$\frac{\pi(\mathbf{X}')}{\pi(\mathbf{X})} = \frac{\mathscr{L}(\mathbf{I}_{V_o}|\mathbf{X}_{V_o}=\ell')}{\mathscr{L}(\mathbf{I}_{V_o}|\mathbf{X}_{V_o}=\ell)}\cdot\frac{p_o(\mathbf{X}_{V_o}=\ell'|\mathbf{X}_{\partial V_o})}{p_o(\mathbf{X}_{V_o}=\ell|\mathbf{X}_{\partial V_o})}. \tag{6.53}$$

Note that $\mathbf{X}_{\partial V_o} = \mathbf{X}'_{\partial V_o}$ in the above equation.

The second ratio in equation(6.52) is easy to design. For example, we can make it proportional to the likelihood,

$$q(\mathbf{X}_{V_o}=\ell|V_o,\mathbf{X}) = \mathscr{L}(\mathbf{I}_{V_o}|\mathbf{X}_{V_o}=\ell), \quad \forall\ell. \tag{6.54}$$

Therefore,

$$\frac{q(\mathbf{X}_{V_o}=\ell|V_o,\mathbf{X}')}{q(\mathbf{X}_{V_o}=\ell'|V_o,\mathbf{X})} = \frac{\mathscr{L}(\mathbf{I}_{V_o}|\mathbf{X}_{V_o}=\ell)}{\mathscr{L}(\mathbf{I}_{V_o}|\mathbf{X}_{V_o}=\ell')}. \tag{6.55}$$

Now it cancels the likelihood ratio in equation (6.53). Finally we arrive at the following proposition.

Proposition 6.8 *The acceptance probability for the proposed cluster flipping using the proposal in (6.54) is,*

$$\alpha(\mathbf{X}\to\mathbf{X}') = \min\{1, \frac{\prod_{<s,t>\in C(R,V_{\ell'})}(1-q_e)}{\prod_{e\in C(V_o,V_\ell)}(1-q_e)}\cdot\frac{p_o(\mathbf{X}_{V_o}=\ell'|\mathbf{X}_{\partial V_o})}{p_o(\mathbf{X}_{V_o}=\ell|\mathbf{X}_{\partial V_o})}\}. \tag{6.56}$$

The result above has the property that the computation is limited to a local neighborhood of V_o defined by the prior model. This result also holds if one uses the Wolff modification and grows V_o from a vertex. In image analysis experiments the SWC method is empirically $O(100)$ times faster than the single site Gibbs sampler. Consider the plots and comparison in Figures 6.11, 6.13 in Section (6.6) for details.

6.4.4 Complexity analysis

This section presents an evaluation of the computational complexity of the SWC algorithm.

Let $N = |V|$ be the number of nodes of the graph $\mathbf{G} =< V,E >$ and N^{it} the number of iterations of the SWC algorithm.

Each SWC iteration involves the following steps:

- Sampling the edges in the data driven clustering step, which is $O(|E|)$. Assuming that $\mathbf{G} =< V,E >$ is sparse, then $O(|E|) = O(N)$.

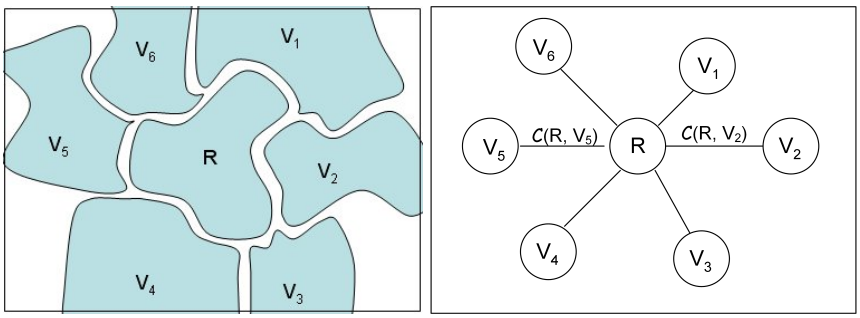
- Constructing connected components, which is $O(|E|\alpha(|E|)) = O(N\alpha(N))$ using the disjoint set forest data structure [18, 19]. The function $\alpha(N)$ is the inverse of $f(n) = A(n,n)$ where $A(m,n)$ is the fast-growing Ackerman function [1]. In fact, $\alpha(N) \leq 5$ for all practical values of N.
- Computing $\pi(\mathbf{X})$, which depends on the problem, but is usually $O(N)$.
- Flipping the label of one connected component, which is $O(N)$.

  Therefore one iteration is $O(N\alpha(N))$ and all iterations take $O(N^{it}N\alpha(N))$ time.

6.5 Variants of the cluster sampling method

In this section, we briefly discuss two variants of the cluster sampling method.

6.5.1 Cluster Gibbs sampling — the "hit-and-run" perspective



(a) (b)

Fig. 6.10: Illustrating the cluster Gibbs sampler. (a) The cluster V_o (R here) has a number of neighboring components of uniform color. (b) The cuts between V_o and its neighboring colors. The sampler follows a conditional probability modified by the edge strength defined on the cuts.

With a slight modification, we can make the cluster sampling method behave like a generalized Gibbs sampler. Suppose that $V_o \in \mathrm{CP}(U|\mathbf{X})$ is the candidate chosen in the clustering step, and Figure 6.10 shows its cuts with adjacent sets

$$C(V_o, V_k), k = 1, 2, ..., \mathrm{L}(\mathbf{X}).$$

We compute the quantity γ_k as the strength of connectivity between V_o and $V_k \backslash V_o$,

$$\gamma_k = \prod_{e \in C(V_o, V_k)} (1 - q_e). \tag{6.57}$$

Proposition 6.9 *Let $\pi(\mathbf{X})$ be the target probability, in the notation above. If V_o is relabeled probabilistically with*

$$q(\mathbf{X}_{V_o} = k | V_o, \mathbf{X}) \propto \gamma_k \pi(\mathbf{X}_{V_o} = k | \mathbf{X}_{\partial V_o}), \;\; k = 1, 2, ..., N(\mathbf{X}), \tag{6.58}$$

then the acceptance probability is always 1 in the third step.

This yields a generalized Gibbs sampler which flips the color of a cluster according to a modified conditional probability.

Cluster Gibbs Sampler

1. Cluster step: choose a vertex $v \in V$ and group a cluster V_o from v by the Bernoulli edge probability μ_e.
2. Flipping step: relabel V_o according to equation (6.58).

Remark 6.2 *The traditional single site Gibbs sampler [21] is the special case when $q_e = 0$ for all e, causing $V_o = \{v\}$ and $\gamma_k = 1$ for all k.*

One may also view the above method from the perspective of hit-and-run. In a continuous state space, a hit-and-run method (see [22]) chooses a new direction $\vec{e}$ randomly at time t and then samples in this direction by $a \sim \pi(x + a\vec{e})$. Liu and Wu [36] extended this to any compact groups of actions. In a finite state space Ω, one can choose finite sets $\Omega_a \subset \Omega$ and then apply the Gibbs sampler within the set. It is difficult to choose good directions or subsets for the hit-and-run methods. In the cluster Gibbs sampler presented above, the subset is selected by the auxiliary variables on the edges.

6.5.2 The multiple flipping scheme

Given a set of connected components $\mathrm{CP}(\mathbf{U}|\mathbf{X})$ (see equation. (6.47)) after the clustering step, instead of flipping a single component V_o, we can flip all (or any chosen number of) connected components simultaneously. There is room for designing the proposal probabilities for labeling these connected components, independently or jointly. In what follows, we assume the labels are chosen independently for each connected component $\mathrm{cp} \in \mathrm{CP}(\mathbf{U}|\mathbf{X})$, by sampling from a proposal probability $q(\mathbf{X}_{\mathrm{cp}} = l|\mathrm{cp})$.

Suppose we obtain a new label $\mathbf{X}'$ after flipping. Let $E_{\mathrm{on}}(\mathbf{X}) \subset E$ and $E_{\mathrm{on}}(\mathbf{X}') \subset E$ be the subsets of edges that connect the vertices of same color in $\mathbf{X}$ and $\mathbf{X}'$ respectively. We define two cuts by the differences of the sets

$$C(\mathbf{X} \to \mathbf{X}') = E_{\mathrm{on}}(\mathbf{X}') - E_{\mathrm{on}}(\mathbf{X}), \text{ and } C(\mathbf{X}' \to \mathbf{X}) = E_{\mathrm{on}}(\mathbf{X}) - E_{\mathrm{on}}(\mathbf{X}'). \tag{6.59}$$

We denote the set of connected components which have different colors before and after the flipping by $D(\mathbf{X}, \mathbf{X}') = \{\mathrm{cp} : \mathbf{X}_{\mathrm{cp}} \neq \mathbf{X}'_{\mathrm{cp}}\}$.

Proposition 6.10 *The acceptance probability of the multiple flipping scheme is*

$$\alpha(\mathbf{X} \to \mathbf{X}') = \min\{1, \frac{\prod_{e \in C(\mathbf{X} \to \mathbf{X}')}(1-q_e)}{\prod_{e \in C(\mathbf{X}' \to \mathbf{X})}(1-q_e)} \frac{\prod_{\mathrm{cp} \in D(\mathbf{X}, \mathbf{X}')} q(\mathbf{X}'_{\mathrm{cp}}|\mathrm{cp})}{\prod_{\mathrm{cp} \in D(\mathbf{X}, \mathbf{X}')} q(\mathbf{X}_{\mathrm{cp}}|\mathrm{cp})} \cdot \frac{p(\pi')}{p(\pi)}\}. \tag{6.60}$$

Observe that when $D = \{V_o\}$ is a single connected component, this reduces to Theorem 6.56. It is worth mentioning that if we flip all connected components simultaneously, then the Markov transition graph of $\mathcal{K}(\mathbf{X}, \mathbf{X}')$ is fully connected, i.e.

$$\mathcal{K}(\mathbf{X}, \mathbf{X}') > 0, \ \forall \mathbf{X}, \mathbf{X}' \in \Omega. \tag{6.61}$$

This means that the Markov chain can move between any two partitions in a single step.

6.6 Application: Image Segmentation

This experiment tests the cluster sampling algorithm in an image segmentation task. The objective is to partition the image into a number of disjoint regions (as Figures 6.7 and 6.8 have shown) so that each region has consistent intensity in the sense of fitting to some image models. The final result should optimize a Bayesian posterior probability $\pi(\mathbf{X}) \propto \mathscr{L}(\mathbf{I}|\mathbf{X}) p_o(\mathbf{X})$.

In such problems, $\mathbf{G}$ is an adjacency graph with vertices V being a set of superpixels. Usually $|V| = O(10^2)$. For each superpixel $v \in V$, we compute a 15-bin intensity histogram h normalized to 1. Then the edge probability is

calculated as

$$q_{ij} = p(\mu_e = \text{on}|\mathbf{I}(v_i), \mathbf{I}(v_j)) = \exp\{-\frac{1}{2}(KL(h_i||h_j) + KL(h_j||h_i))\}, \tag{6.62}$$

where $KL()$ is the Kullback-Leibler divergence between the two histograms. Usually q_e should be close to zero for e crossing an object boundary. In these experiments, the edge probability leads to good clustering as Figure 6.8 showed.

Now we briefly define the target probability in this experiment. Let $\mathbf{X} = (V_1, ..., V_{\mathrm{L}})$ be a coloring of the graph with L being an unknown variable, and the image intensities in each set V_k being consistent in terms of fitting to a model θ_k. Different colors are assumed to be independent. Therefore, we have,

$$\pi(\mathbf{X}) = \pi(\mathbf{X}|\mathbf{I}) \propto \prod_{k=1}^{\mathrm{L}} [\mathscr{L}(\mathbf{I}(V_k); \theta_k) p_o(\theta_k)] p_o(\mathbf{X}). \tag{6.63}$$

where $\mathscr{L}(\mathbf{I}(V_k); \theta_k)$ is the likelihood model with parameters θ_k and $p_o(\theta_k)$ is a model complexity prior for θ_k. These quantities are described below.

We selected three types of simple models for the likelihood models to account for different image properties. The first model is a non-parametric histogram $\mathscr{H}$, which in practice is represented by a vector of B-bins $(\mathscr{H}_1, ..., \mathscr{H}_B)$ normalized to 1. It accounts for cluttered objects, like vegetation.

$$\mathbf{I}(x, y; \theta_0) \sim \mathscr{H} \text{ iid}, \ \forall (x, y) \in V_k. \tag{6.64}$$

The other two are regression models for the smooth change of intensities in the two-dimensional image plane (x, y), and the residues follow the empirical distribution $\mathscr{H}$ (i.e. the histogram).

$$\mathbf{I}(x, y; \theta_1) = \beta_0 + \beta_1 x + \beta_2 y + \mathscr{H} \text{ iid}, \ \forall (x, y) \in V_k. \tag{6.65}$$

$$\mathbf{I}(x, y; \theta_2) = \beta_0 + \beta_1 x + \beta_2 y + \beta_3 x^2 + \beta_4 xy + \beta_5 y^2 + \mathscr{H} \text{ iid}, \ \forall (x, y) \in V_k. \tag{6.66}$$

In all cases, the likelihood is expressed in terms of the entropy of the histogram $\mathscr{H}$

$$\mathscr{L}(\mathbf{I}(V_k); \theta_k) \propto \prod_{v \in V_k} \mathscr{H}(\mathbf{I}_v) = \prod_{j=1}^{B} \mathscr{H}_j^{n_j} = \exp(-|V_k| \text{entropy}(\mathscr{H})). \tag{6.67}$$

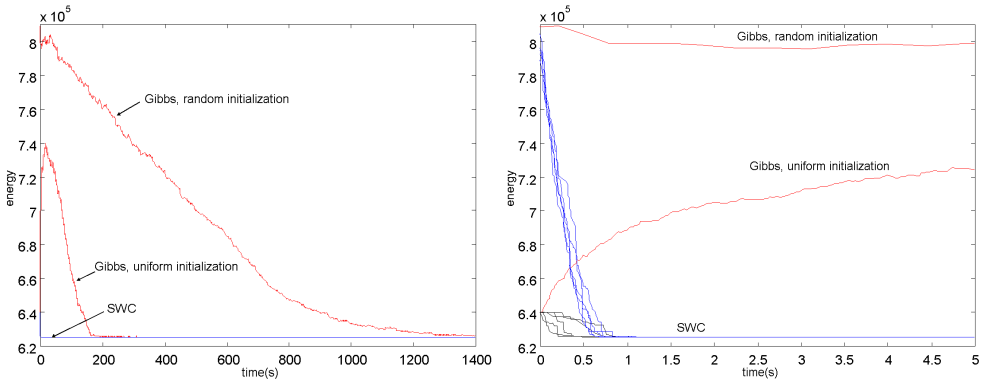
The model complexity is penalized by a prior probability $p_o(\theta_k)$ and the parameters θ in the above likelihoods are computed deterministically at each step as the best least square fit. The deterministic fitting could be replaced by the reversible jumps together with the color flipping. This was done in [56] and will be presented in Chapter 8.

The prior model $p_o(\mathbf{X})$ encourages large and compact regions with a small number of colors, as was suggested in [56]. Let $r_1, r_2, ..., r_m$, $m \geq \mathrm{L}$ be the connected components of all $V_k, k = 1, ..., \mathrm{L}$. Then the prior is

$$p_o(\mathbf{X}) \propto \exp\{-\alpha_0 \mathrm{L} - \alpha_1 m - \alpha_2 \sum_{k=1}^{m} \text{Area}(r_k)^{0.9}\}. \tag{6.68}$$

For the image segmentation example (horse) shown in Figures 6.7 and 6.8, we compare the cluster sampling method with the single-site Gibbs sampler and the results are displayed in Figure 6.11. Since our goal is to maximize the posterior probability $\pi(\mathbf{X})$, we must add an annealing scheme with a high initial temperature T_o and then decrease to a low temperature (0.05 in our experiments). We plot the $-\ln \pi(\mathbf{X})$ over CPU time in seconds. The Gibbs sampler needs to raise the initial temperature high (say $T_o \geq 100$)) and use a slow annealing schedule to reach a good solution. The cluster sampling method can run at a low temperature. We usually raise the initial temperature to $T_o \leq 15$ and use a fast annealing scheme. Figure 6.11 (a) plots the two algorithms at the first $1,400$ seconds, and Figure 6.11 (b) is a zoomed-in view for the first 5 seconds.

The two algorithms are run with two initializations. One is a random labeling of the superpixels and thus has a higher $-\ln \pi(\mathbf{X})$, and the other initialization sets all vertices to the same color. The clustering methods are run five times on both cases. They all converged to one solution (see Figure6.7 (c)) within 1 second, which is $O(10^2)$ times faster than the Gibbs sampler.



(a) convergence CPU time in seconds (b) Zoomed-in view of the first 5 seconds.

Fig. 6.11: The plot of $-\ln \pi(X)$ over computing time for both the Gibbs sampler and our algorithm for the horse image. (a) Plot of the first 1400 seconds. The Gibbs sampler needs a high initial temperature and slow annealing step to achieve the same energy level. (b) The zoomed-in view of the first 5 seconds. © [2007] Taylor & Francis. Reprinted, with permission, from ref. [4].

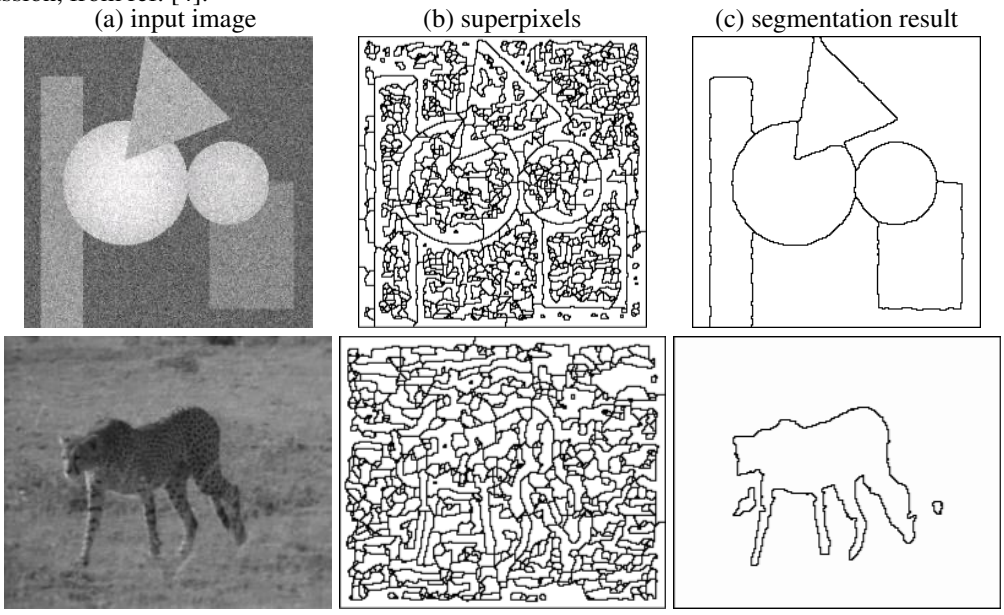


Fig. 6.12: More results for image segmentation. © [2007] Taylor & Francis. Reprinted, with permission, from ref. [4].

Figure 6.12 shows two more images. Using the sample comparison method as in the horse image, we plot $-\ln \pi(\mathbf{X})$ against running time in Figures 6.13. In experiments, we also compared the effect of the edge probabilities. The clustering algorithm is $O(100)$ times slower if we use a constant edge probability $\mu_{ij} = c \in (0,1)$ as the original SW method does. For example the single-site Gibbs sampler is an example with $q_{ij} = 0, \forall i, j$.

6.7 Multigrid and multi-level SW-cut

The essence of the SW-cut is a Markov chain $\mathcal{MC} =< \nu, \mathcal{K}, p >$ which visits a sequence of states in the partition space Ω over time t,

$$\mathbf{X}(0), \mathbf{X}(1), ..., \mathbf{X}(t) \ \in \Omega.$$

The results from the previous section ensure that the SW-cut design observes the detailed balance equations

$$p(\mathbf{X})\mathcal{K}(\mathbf{X}, \mathbf{X}') = p(\mathbf{X}')\mathcal{K}(\mathbf{X}', \mathbf{X}), \ \ \forall \mathbf{X}', \mathbf{X}. \tag{6.69}$$

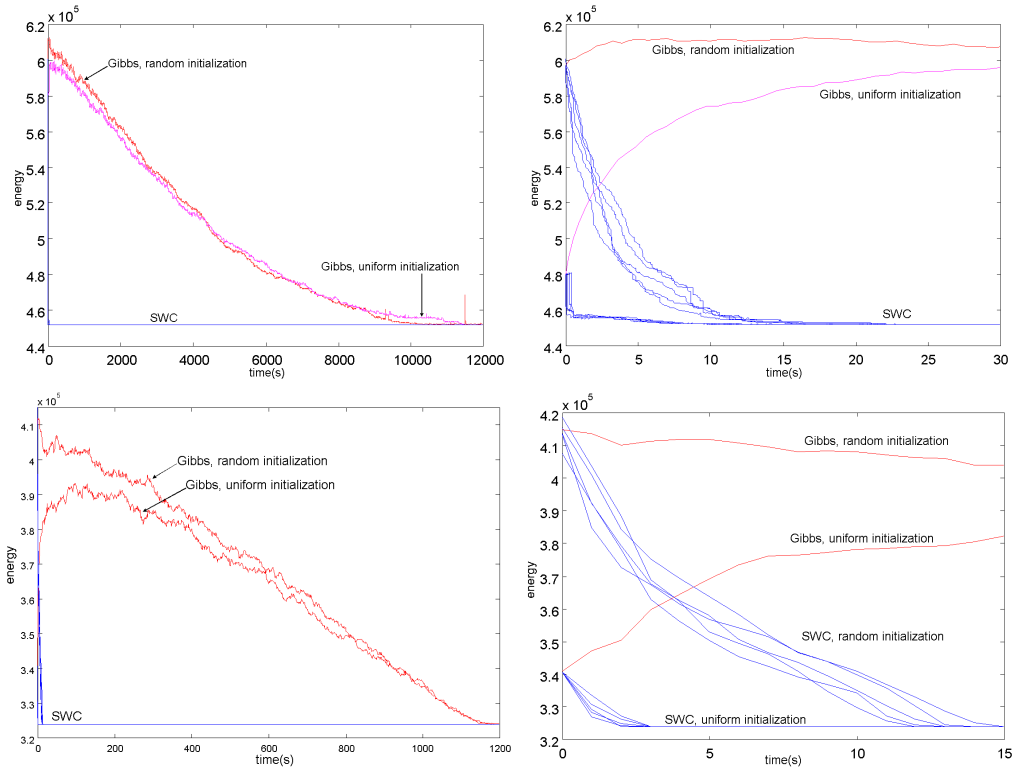


Fig. 6.13: Convergence comparison between the clustering method and Gibbs sampler in CPU time (seconds) for the two images of Fig.6.12. (Left). The first 1,200 seconds. (Right) Zoomed-in view of the first 15-30 seconds. The clustering algorithm is run 5 trials for both the random and uniform initializations. © [2005] IEEE. Reprinted, with permission, from ref. [3].

Once it converges, the SW-cut simulates fair samples from $p(\mathbf{X})$

The SW-cut is characterized by three selections in its design.

1. The discriminative proposal probabilities defined on the edges of the adjacency graph $G =< V, E >$. $q(\mathbf{X}) = \prod_{e \in E} q_e$ is a factorized approximation to $p(\mathbf{X})$ and influences the formation of connected components CP, and thus the candidate component V_o.
2. The uniform probability for selecting V_o from a connected component $V_o \in$ CP.
3. The reassignment probability $Q(\ell_{\text{new}}(V_o)|V_o, \mathbf{X}_A)$ for the new label of the connected component V_o.

We extend the SW-cut by introducing the multigrid and multilevel SW-cuts algorithms which provide more flexible means for selecting V_o's and $q(\mathbf{X})$'s. To summarize, the two extensions are new directions for sampling $p(\mathbf{X})$.

1. The multigrid SW-cuts simulates a Markov chain $\mathcal{MC}_{mg}$ with kernel $\mathcal{K}_{mg}$ by sampling the conditional probabilities of $p(\mathbf{X})$.
2. The multi-level SW-cuts simulates a Markov chain $\mathcal{MC}_{ml}$ with kernel $\mathcal{K}_{ml}$ by sampling the conditional probabilities of $p(\mathbf{X})$ at higher levels, and the full posterior at lower levels.

Both $\mathcal{MC}_{mg}$ and $\mathcal{MC}_{ml}$ satisfy the detailed balance equations in (6.69), as it will be shown in the following sections. The proofs are based on the following result.

Let $p(x,y)$ be a two-dimensional probability, and $\mathcal{K}$ be a Markov kernel sampling its conditional probability $p(x|y)$ (or $p(y|x)$). Thus, $\mathcal{K}$ observes the detailed balance equation,

$$p(x|y)\mathcal{K}(x,x') = p(x'|y)\mathcal{K}(x',x),\ \forall x,x'. \tag{6.70}$$

The kernel $\mathcal{K}$ can be naturally augmented to a kernel on (x,y) as

$$\mathcal{K}((x,y),(x',y')) = \begin{cases} \mathcal{K}(x,x') & \text{if } y = y' \\ 0 & \text{else} \end{cases}$$

Theorem 6.11 *In the above notation, $\mathcal{K}$ observes the general detailed balance equations after augmenting y, i.e.*

$$p(x,y)\mathcal{K}((x,y),(x',y')) = p(x',y')\mathcal{K}((x',y'),(x,y)).$$

Proof. If $y = y'$, then it is straightforward. If $y \neq y'$ then $\mathcal{K}((x,y),(x',y')) = \mathcal{K}((x',y'),(x,y)) = 0$. □

The conclusion of this theorem is that an algorithm which is reversible when sampling from a conditional probability is also reversible for sampling the full probability.

6.7.1 SW-cuts at multigrid

We first study the multigrid SW-cut. Recall that in each step the SW-cut turns off, probabilistically, the edges in the entire adjacency graph, and this could be less effective when G is very large. The concept of multigrid SW-cut is to allow us to select certain attention windows and run the SW-cut within the window. Thus, it provides flexibility in designing a "visiting scheme" by selecting windows of various sizes and locations over time. For example, Figure 6.14 shows windows in a multigrid arrangement.

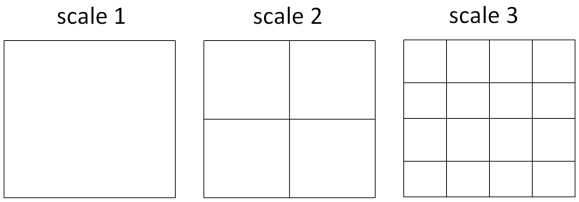


Fig. 6.14: Selecting windows in a multigrid scheme

Let $G =< V,E >$ be the adjacency graph, $\mathbf{X} = \{V_1,...,V_n\}$ the current partition, and Λ an attention window of arbitrary size and shape. Λ divides the vertices into two subsets $V = V_\Lambda \cup V_{\bar{\Lambda}}$ for vertices inside and outside the window respectively.

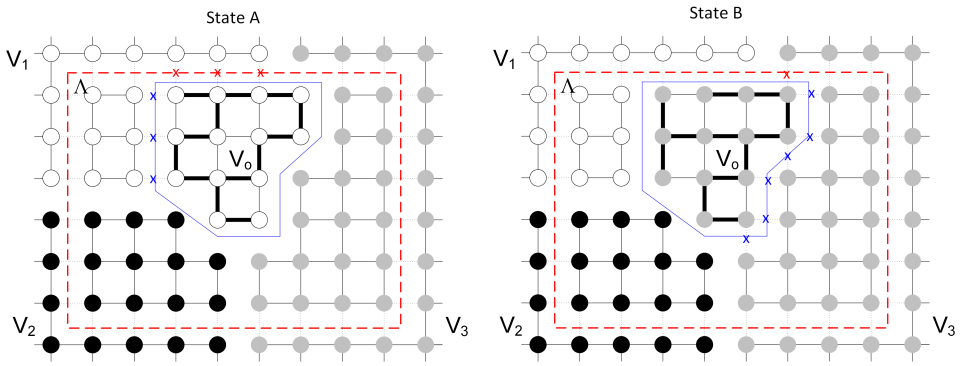


Fig. 6.15: Multigrid SW-cut: run SW-cut within an attention window Λ with the rest of the labels fixed, and realizes a reversible move between two states $\mathbf{X}_A$ and $\mathbf{X}_B$ by flipping the label of $V_o \subset V_\Lambda$. © [2007] Taylor & Francis. Reprinted, with permission, from ref. [4].

For example, Figure6.15 displays a rectangular window Λ (in red) in a lattice G. The window Λ further removes some edges within each subset $V_i, i = 1,2,...,n$, and we denote them by,

$$\mathcal{C}(V_i|\Lambda) = \{e =< s,t >: s \in V_i \cap V_\Lambda,\ t \in V_i \cap V_{\bar{\Lambda}}\}.$$

For example, in Figure6.15 the window Λ intersects with three subsets V_1 (white), V_2 (black), and V_3 (grey), and all edges crossing the (red) rectangle window are removed. We divide the labeling (coloring or partition) of the vertices V into two parts

$$\mathbf{X}(V) = (\mathbf{X}(V_\Lambda), \mathbf{X}(V_{\bar{\Lambda}})). \tag{6.71}$$

We fix $\mathbf{X}(V_{\bar{\Lambda}})$ as boundary condition, and sample the labels of vertices within the window by SW-cut. To summarize, the multigrid SW-cut iterates the following three steps:

1. It selects a window Λ of certain size and shape following a probability $\Lambda \sim q(\Lambda)$.
2. For any edges within each subgraph inside the window, $e =< s,t >, s,t \in \Lambda,\ \ell_s = \ell_t$, it turns off edge e with probability q_e. Thus, it obtains a set of connected components $\text{CP}(V_\Lambda)$.
3. It selects $V_o \in \text{CP}(V_\Lambda)$ as a connected component and flips its label according to probability

$$Q(\ell_{\text{new}}(V_o) = j | V_o, \mathbf{X}) = \frac{1}{C} \prod_{e \in \mathcal{C}_j} q_e \cdot p(\mathbf{X}^*_j),\ \forall j, \tag{6.72}$$

where $\mathbf{X}^*_j$ is the partition by assigning V_o to label j, and $\mathcal{C}_j = \mathcal{C}(V_o, V_j) - \mathcal{C}(V_j|\Lambda)$.

For example, Figure 6.15 illustrates a reversible move by flipping a connected component V_o (within the blue polygon) between two states $\mathbf{X}_A$ and $\mathbf{X}_B$. $\mathcal{C}_1$ and $\mathcal{C}_3$ are shown by the blue crosses which are removed by the random procedure. Following the same procedure as in the previous SW-cut, we can derive the proposal probability ratio for selecting V_o in the two states within Λ.

Theorem 6.12 *The probability ratio for proposing V_o as the candidate subgraph within window Λ at two states $\mathbf{X}_A$ and $\mathbf{X}_B$ is*

$$\frac{Q(V_o|\mathbf{X}_A, \Lambda)}{Q(V_o|\mathbf{X}_B, \Lambda)} = \frac{\prod_{e \in \mathcal{C}(V_o, V_1) - \mathcal{C}(V_1|\Lambda)} q_e}{\prod_{e \in \mathcal{C}(V_o, V_3) - \mathcal{C}(V_3|\Lambda)} q_e}.$$

The difference between this ratio and the ratio in theorem 6.7 is that some edges (see the red crosses in Fig.6.15) no longer participate the computation. Following the probability in equation(6.72) for the new labels, we can prove that it simulates the conditional probability,

$$\mathbf{X}(V_\Lambda) \sim p(\mathbf{X}(V_\Lambda)|\mathbf{X}(V_{\bar{\Lambda}})).$$

Theorem 6.13 *The multigrid SW-cut within window Λ simulates a Markov kernel*

$$\begin{aligned} \mathcal{K}(\Lambda) &= \mathcal{K}(\mathbf{X}(V_\Lambda), \mathbf{X}'(V_\Lambda)|\mathbf{X}(V_{\bar{\Lambda}})), \\ p(\mathbf{X}(V_\Lambda)|\mathbf{X}(V_{\bar{\Lambda}}))\mathcal{K}(\mathbf{X}, \mathbf{X}') &= p(\mathbf{X}'(V_\Lambda)|\mathbf{X}(V_{\bar{\Lambda}}))\mathcal{K}(\mathbf{X}', \mathbf{X}). \end{aligned} \tag{6.73}$$

Following theorem 6.11, we have $\mathcal{K}(\Lambda)$ satisfies the general detailed balance equation in equation(6.69).

6.7.2 SW-cuts at multi-level

Now we add a multi-level SW-cut mechanism. Suppose at state $\mathbf{X} = \{V_1, V_2, ..., V_n\}$, we freeze some subsets $A_k, k \in \{1, ..., m\}$ such that for any k, $A_k \subset V_i$ for some i. This way, the vertices in each A_k are locked to have the same label.

The subsets A_k can represent an intermediary segmentation. For example, for motion segmentation, it is useful to get an intensity segmentation A and group the intensity regions A_k into coherently moving objects. Thus, $G = G^{(1)}$ is reduced to a smaller adjacency graph $G^{(2)} =< U, F >$. U is the set of vertices

$$U = \{u_1, ..., u_m\},\ u_k = A_k, k = 1, 2, ..., m.$$

F is the adjacency relation between the subsets A_k in G, given by

$$F = \{f =< u_i, u_j >: \mathcal{C}(A_i, A_j) \neq \emptyset\}.$$

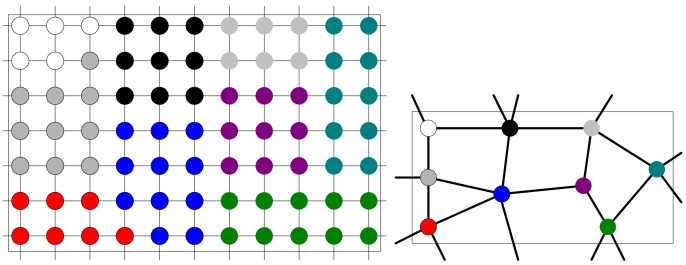


Fig. 6.16: Multi-level SW-cut with two levels. © [2007] Taylor & Francis. Reprinted, with permission, from ref. [4].

Figure 6.16 illustrates an example with $m = 9$. We run the SW-cut on level 2 based on new discriminative heuristics $q^{(2)}$ which measure the similarity of A_i, A_j, $q^{(2)}(\mathbf{X}(U)) = \prod_{f \in F} q_f^{(2)}$. In general, these heuristics are more informative than the lower level, so the SW-cuts moves are more meaningful and the convergence is faster.

The partition space for graph $G^{(2)}$ is a projection of Ω,

$$\Omega(G^{(2)}) = \{\mathbf{X} : x_s = x_t, \ \ \forall s,t \in A_i, \ \ i = 1,2,...,m\}.$$

Obviously, the SW-cut on level 2 simulates a Markov chain with kernel $\mathcal{K}^{(2)}$ which has invariant probability $p(\mathbf{X}(U)|A)$, the probability of $p(\mathbf{X})$ conditional on the relations $x_s = x_{u_i}$ for all $s \in A_i$ and all i. Following Theorem 6.11, we have that $\mathcal{K}^{(2)}$ satisfies the general detailed balance equation (6.69).

Suppose we design a visiting scheme for selecting the windows $\Lambda \sim q_w(\Lambda)$ and level $\sigma \sim q_l(\sigma)$ over time. Then the generalized SW-cut algorithm has a mixed Markov kernel

$$\mathcal{K} = \sum_{\sigma} \sum_{\Lambda} q_l(\sigma) q_w(\Lambda) \mathcal{K}^{(\sigma)}(\Lambda).$$

As each $\mathcal{K}^{(\sigma)}(\Lambda)$ observes the detailed balance equations, so is $\mathcal{K}$. When the windows cover the entire graph, it is also irreducible and its states follows $p(\mathbf{X})$ at convergence.

6.8 Subspace Clustering

Subspace clustering is the problem of grouping an unlabeled set of points into a number of clusters corresponding to subspaces of the ambient space. This problem has applications in unsupervised learning and computer vision. One such example is sparse motion segmentation, where a number of feature point trajectories need to be grouped into a small number of clusters according to their common motion model. The feature point trajectories are obtained by detecting a number of feature points using an interest point detector and tracking them through many frames using a feature point tracker or an optical flow algorithm.

A common approach in the state of the art sparse motion segmentation methods [15] [32] [34] [57] [63] is to project the feature trajectories into a lower dimensional space and use a subspace clustering method based on spectral clustering to group the projected points and obtain the motion segmentation. Even though these methods obtain very good results on standard benchmark datasets, the spectral clustering algorithm requires expensive computation of eigenvectors and eigenvalues on an $N \times N$ dense matrix where N is the number of data points. In this manner, the computation time for these subspace clustering/motion segmentation methods scales as $O(N^3)$, so it can become prohibitive for large problems (e.g. $N = 10^5 - 10^6$).

This section presents a completely different approach to subspace clustering, based on the SWC algorithm. The subspace clustering problem is formulated as a Maximum A Posteriori (MAP) optimization problem in a Bayesian framework with an Ising/Potts prior [44] and likelihood based on a linear subspace model. The SWC graph is constructed as a k-NN graph from an affinity matrix. Overall, the proposed method provides a new perspective to solve the subspace clustering problem, and demonstrates the power of the SWC algorithm in clustering problems.

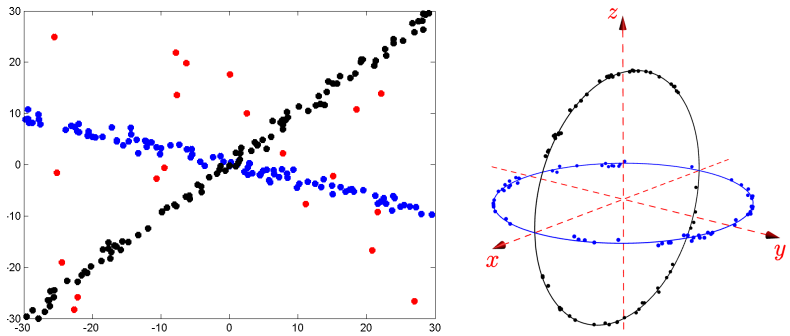


Fig. 6.17: Subspace clustering examples. Left: two 1D subspaces in 2D. Right: two 2D subspaces in 3D, where the points have been normalized to unit length. Due to noise, the points may not lie exactly on the subspace. One can observe that the angular distance finds the correct neighbors in most places except at the plane intersections. © [2015] CRC Press. Reprinted, with permission, from ref. [13].

Given a set of points $\{\mathbf{x}_1,...,\mathbf{x}_N\} \in \mathbb{R}^D$, the subspace clustering problem is to group the points into a number of clusters corresponding to linear subspaces of $\mathbb{R}^D$, as illustrated in Figure 6.17. A popular subspace clustering method [12] [32] [46] is based on spectral clustering, which relies on an affinity matrix that measures how likely any pair of points belong to the same subspace.

Spectral clustering [40, 48] is a generic clustering method that groups a set of points into clusters based on their connectivity. The point connectivity is given as an $N \times N$ affinity matrix A with A_{ij} close to 1 if point i is close to point j and close to zero if they are far away. The quality of the affinity matrix is very important for obtaining good clustering results. The affinity matrix for spectral subspace clustering is computed as follows. First, the the points are normalized to unit length [12, 32, 46], and then the following affinity measure based on the angle between the vectors has been proposed in [32]

$$A_{ij} = (\frac{x_i^T x_j}{\|x_i\|_2 \|x_j\|_2})^{2\alpha}, \tag{6.74}$$

where α is a tuning parameter. The value $\alpha = 4$ was used in [32]. It is intuitive to find that the points tend to lie in the same subspace as their neighbors in angular distance except those near the intersection of the subspaces.

6.8.1 Subspace clustering by Swendsen-Wang cuts

A subspace clustering solution can be represented as a partition (labeling) $\mathbf{X} : \{1,...,N\} \to \{1,...,M\}$ of the input points $\mathbf{x}_1,\ldots,\mathbf{x}_N \in \mathbb{R}^D$. The number $M \leq N$ is the maximum number of allowed clusters. In this section is assumed that an affinity matrix A is given, representing the likelihood for any pair of points to belong to the same subspace. One form of A has been given in (6.74) and another one will be given below.

A posterior probability can be used to evaluate the quality of any partition $\mathbf{X}$. A good partition can then be obtained by maximizing the posterior probability in the space of all possible partitions. The posterior probability is defined in a Bayesian framework as

$$p(\mathbf{X}) \propto \exp[-E_{data}(\mathbf{X}) - E_{prior}(\mathbf{X})].$$

The normalizing constant is irrelevant in the optimization since it will cancel out in the acceptance probability. The data term $E_{data}(\mathbf{X})$ is based on the fact that the subspaces are assumed to be linear. Given the current partition (labeling) $\mathbf{X}$, for each label l an affine subspace L_l is fitted in a least squares sense through all points with label l. Denote the distance of a point $\mathbf{x}$ with label l to the linear space L_l as $d(\mathbf{x}, L_l)$. Then the data term is

$$E_{data}(\mathbf{X}) = \sum_{l=1}^{M} \sum_{i,\mathbf{X}(i)=l} d(\mathbf{x}_i, L_l). \tag{6.75}$$

The prior term $E_{prior}(\mathbf{X})$ is set to encourage tightly connected points to stay in the same cluster.

$$E_{prior}(\mathbf{X}) = -\rho \sum_{<i,j>\in E, \mathbf{X}(i)\neq\mathbf{X}(j)} \log(1-A_{ij}), \tag{6.76}$$

where ρ is a parameter controlling the strength of the prior term. It will be clear in the next section that this prior is exactly the Potts model (6.1) that would have A_{ij} as the edge weights in the original SW algorithm.

The SWC algorithm could automatically decide the number of clusters; however, in this chapter, as in most motion segmentation algorithms, it is assumed that the number of subspaces M is known. Thus, the new label for the component V_o is sampled with uniform probability from the number M of subspaces

$$q(c_{V_o} = l' | V_o, \mathbf{X}) = 1/M.$$

Spectral clustering optimizes an approximation of the normalized cut or the ratio cut [58], which are discriminative measures. In contrast, the SWC-based subspace clustering approach optimizes a generative model where the likelihood is based on the assumption that the subspaces are linear. It is possible that the discriminative measures are more flexible and work better when the linearity assumption is violated.

The following affinity measure, inspired by [32], can be used.

$$A_{ij} = \exp(-m\frac{\theta_{ij}}{\bar{\theta}}), \quad i \neq j, \tag{6.77}$$

where θ_{ij} is based on the angle between the vectors x_i and x_j,

$$\theta_{ij} = 1 - (\frac{x_i^T x_j}{\|x_i\|_2 \|x_j\|_2})^2,$$

and $\bar{\theta}$ is the average of all θ. The parameter m is a tuning parameter to control the size of the connected components obtained by the SWC algorithm.

The affinity measure based on the angular information between points enables us to obtain the neighborhood graph. After the graph has been obtained, the affinity measure is also used to obtain the edge weights for making the data driven clustering proposals in the SWC algorithm as well as for the prior term of the posterior probability. The graph $G = (V, E)$ has the set of points that need to be clustered as its vertices. The edges E are constructed based on the distance measure from equation (6.77). Since this distance measure is more accurate in finding the nearest neighbors from the same subspace, the graph is constructed as the k-nearest neighbor graph, where k is a given parameter. Examples of obtained graphs are shown in Figure 6.18.

The SWC algorithm is designed for sampling the posterior probability $p(\mathbf{X})$. To use SWC for optimization, a simulated annealing scheme should be applied. With simulated annealing, the probability used by the algorithm is $p(\mathbf{X})^{1/T}$ where the temperature is large at the beginning of the optimization and is slowly decreased according to the annealing schedule. If the annealing schedule is slow enough, it is theoretically guaranteed [27] that the global optimum of the probability $p(\mathbf{X})$ will be found. In reality we use a faster annealing schedule, and the final partition $\mathbf{X}$ will only be a local optimum. We use an annealing schedule that is controlled by three parameters: the start temperature T_{start}, the end temperature as T_{end}, and the number of iterations N^{it}. The temperature at step i is calculated as

$$T_i = \frac{T_{\text{end}}}{\log\left(\frac{i}{N}[e - \exp(\frac{T_{\text{end}}}{T_{\text{start}}})] + \exp(\frac{T_{\text{end}}}{T_{\text{start}}})\right)}, i = \overline{1, N^{it}}. \tag{6.78}$$

To better explore the probability space, we also use multiple runs with different random initializations. The final algorithm is shown below.

Let N be the number of points in $\mathbb{R}^D$ that need to be clustered. The computational complexity of the SWC subspace clustering method can be broken down as follows:

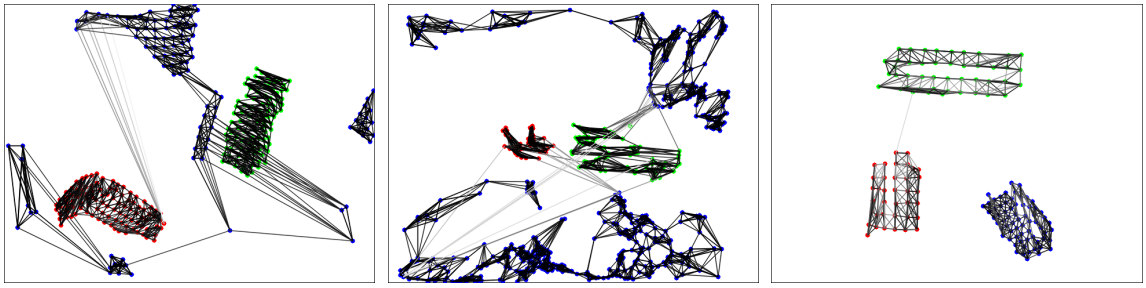
- The adjacency graph construction is $O(N^2 D \log k)$ where D is the space dimension. This is because one needs to calculate the distance from each point to the other $N-1$ points and maintain its k-nearest neighbors.

Algorithm 7 Swendsen-Wang Cuts for Subspace Clustering

Input: N points $(\mathbf{x}_1, \ldots, \mathbf{x}_N)$ from M subspaces
Construct the adjacency graph G as a k-NN graph using eq (6.77).
for $r = 1, \ldots, Q$ **do**
  Initialize the partition $\mathbf{X}$ as $\mathbf{X}(i) = 1, \forall i$.
  for $i = 1, \ldots, N^{it}$ **do**
    1. Compute the temperature T_i using eq (6.78).
    2. Run one step of the SWC algorithm using $p(\mathbf{X}|I) \propto p^{1/T_i}(\mathbf{X})$ in eq (6.52).
  end for
  Record the clustering result $\mathbf{X}_r$ and the final probability $p_r = p(\mathbf{X}_r)$.
end for
Output: Clustering result $\mathbf{X}_r$ with the largest p_r.

- Each of the N^{it} iterations of the SWC algorithm is $O(N\alpha(N))$, as discussed in section 6.4.4. Computing $E_{data}(\mathbf{X})$ involves fitting linear subspaces for each motion cluster, which is $O(D^2N + D^3)$, while computing the $E_{prior}(\mathbf{X})$ is $O(N)$. The number of iterations is fixed (e.g. $N^{it} = 2000$,) so all the SWC iterations take $O(N\alpha(N))$ time.

In conclusion, the entire algorithm complexity is $O(N^2)$ and thus it scales better than spectral clustering for large problems.



(a) 1RT2TC (b) cars3 (c) articulated

Fig. 6.18: Examples of SWC weighted graphs for a checkerboard (a), traffic (b) and articulated (c) sequence. The feature point positions in the first frame are shown. The edge intensities represent their weights from 0 (white) to 1 (black). © [2015] CRC Press. Reprinted, with permission, from ref. [13].

6.8.2 Application: sparse motion segmentation

This section presents an application of the SWC-based subspace clustering algorithm to motion segmentation. Most recent works on motion segmentation use the affine camera model, which is approximately satisfied when the objects are far from the camera. Under the affine camera model, a point on the image plane (x, y) is related to the real world 3D point X by

$$\begin{bmatrix} x \\ y \end{bmatrix} = A \begin{bmatrix} X \\ 1 \end{bmatrix}, \tag{6.79}$$

where $A \in \mathbb{R}^{2\times 4}$ is the affine motion matrix.

Let $t_i = (x_i^1, y_i^1, x_i^2, y_i^2, \ldots, x_i^F, y_i^F)^T, i = 1, \ldots .N$ be the trajectories of tracked feature points in F frames (2D images), where N is the number of trajectories. Let the *measurement matrix* $W = [t_1, t_2, \ldots, t_N]$ be constructed by assembling

the trajectories as columns. If all trajectories undergo the same rigid motion, then equation (6.79) implies that W can be decomposed into a motion matrix $M \in \mathbb{R}^{2F\times 4}$ and a structure matrix $S \in \mathbb{R}^{4\times N}$ as

$$W = MS$$

$$\begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_N^1 \\ y_1^1 & y_2^1 & \cdots & y_N^1 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^F & x_2^F & \cdots & x_N^F \\ y_1^F & y_2^F & \cdots & y_N^F \end{bmatrix} = \begin{bmatrix} A^1 \\ \vdots \\ A^F \end{bmatrix} \begin{bmatrix} X_1 & \cdots & X_N \\ 1 & \cdots & 1 \end{bmatrix},$$

where A^f is the affine object to world transformation matrix at frame f. This implies that $\text{rank}(W) \leq 4$. Since the entries of the last row of S are always 1, under the affine camera model, the trajectories of feature points from a rigidly moving object reside in an affine subspace of dimension at most 3.

(a) Checkerboard (b) Traffic (c) Articulated

Fig. 6.19: Sample images from some sequences of three categories in the Hopkins 155 database with ground truth superimposed. © [2015] CRC Press. Reprinted, with permission, from ref. [13].

Generally we are given a measurement matrix W that contains trajectories from multiple possibly nonrigid motions. The task of motion segmentation is to cluster together all trajectories coming from each motion. A popular approach [12] [32] [46] [57] is to project the trajectories into a lower dimensional space and to perform subspace clustering in that space, using spectral clustering as described earlier. These methods differ in the projection dimension D and in the affinity measure A used for spectral clustering.

Dimension reduction is an essential preprocessing step for obtaining a good motion segmentation. To perform it the truncated SVD is often applied [12, 32, 46, 57]. To project the measurement matrix $W \in \mathbb{R}^{2F\times N}$ to $X = [x_1, ..., x_N] \in \mathbb{R}^{D\times N}$, where D is the desired projection dimension, the matrix W is decomposed as $W = U\Sigma V^T$ and the first D columns of the matrix V are chosen as X^T. The value of D for dimension reduction is also a major concern in motion segmentation. This value has a large impact on the speed and accuracy of the final result, so it is very important to select the best dimension to perform the segmentation. The dimension of a motion is not fixed, but can vary from sequence to sequence, and since it is hard to determine the actual dimension of the mixed space when multiple motions are present, different methods may have different dimensions for projection. Some methods [12, 32] use an exhaustive search strategy to perform the segmentation in spaces with a range of possible dimensions and pick the best result. In this section, we use $D = 2M + 1$ which can work quite well in this application.

The computational complexity of computing the SVD of a $m \times n$ matrix U when $m >> n$ is $O(mn^2 + n^3)$ [54]. If $n >> m$, then it is faster to compute the SVD of U^T, which takes $O(nm^2 + m^3)$. Assuming that $2F << N$ means that the SVD of W can be computed in $O(NF^2 + F^3)$ operations. After projecting to the subspace of dimension $D = 2M + 1$, the SWC subspace clustering algorithm from Section 6.8.1 is applied, and the clustering result gives the final motion segmentation result.

This section presents experiments with the SWC-based motion segmentation algorithm on the Hopkins 155 motion database [55]. The database consists of 155 sequences of two and three motions. The ground-truth segmentation is also provided for evaluation purposes. Based on the content of the video, the sequences could be categorized into three

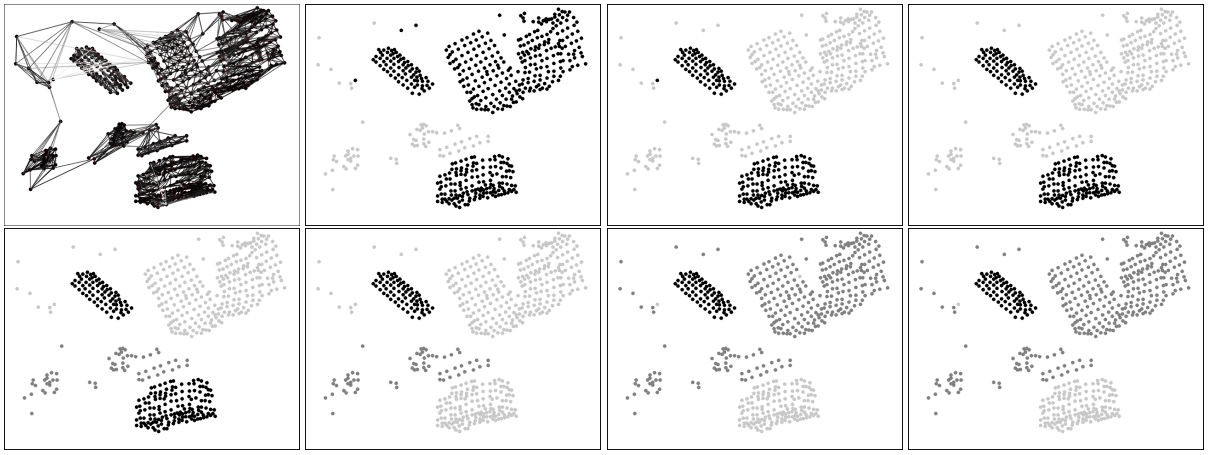


Fig. 6.20: SWC clustering of the Hopkins 155 sequence 1R2TCR, containing $M = 3$ motions. Shown are the feature point positions in the first frame, having colors as the labeling states $\mathbf{X}$ obtained while running the SWC algorithm from the initial state (top left) to the final state (bottom right). © [2015] CRC Press. Reprinted, with permission, from ref. [13].

main categories: checkerboard, traffic, and articulated sequences, with examples shown in Figure 6.19. The trajectories are extracted automatically by a tracker, so they are slightly corrupted by noise.

As already mentioned, before applying the SWC algorithm, the dimension of the data is reduced from $2F$ to $D = 2M + 1$, where M is the number of motions. After the projection, the initial labeling state in the SWC algorithm has all points having the same label.

Parameter settings. The motion segmentation algorithm has a number of tuning parameters that were held constant to the following values. The number of nearest neighbors for graph construction is $k = 7$, the parameter m in the affinity measure (6.77) is $m = 10$, and the prior coefficient in (6.76) is $\rho = 2.2$. The annealing parameters are $T_{\text{start}} = 1$, $T_{\text{end}} = 0.01$, $N^{it} = 2000$. The number of independent runs to obtain the most probable partition is $Q = 10$. An example of all the partition states during an SWC run is shown in Figure 6.20.

Results. The motion segmentation results are evaluated using the misclassification error rate given by

$$\text{Misclassification Rate} = \frac{\text{\# of misclassified points}}{\text{total \# of points}}. \tag{6.80}$$

The average and median misclassification errors are listed in Table 6.1. For accuracy, the results of the SWC algorithm from Table 6.1 are averaged over 10 runs and the standard deviations are shown in parentheses. In order to compare the SWC method with the state of the art methods, we also list the results of ALC [46], SC [32], SSC [15] and VC [12].

Table 6.1: Misclassification rates (in percent) of different motion segmentation algorithms on the Hopkins 155 dataset.

Method	ALC	SC	SSC	VC	SWC (std)	SC^4	SC^{4k}	KASP
All (2 motion)								
Average	2.40	0.94	0.82	0.96	1.49 (0.19)	11.50	7.82	4.76
Median	0.43	0.00	0.00	0.00	0.00 (0.00)	2.09	0.27	0.00
All (3 motion)								
Average	6.69	2.11	2.45	1.10	2.62 (0.13)	19.55	11.25	9.00
Median	0.67	0.37	0.20	0.22	0.81 (0.00)	18.88	1.42	1.70
All sequences combined								
Average	3.37	1.20	1.24	0.99	1.75 (0.15)	13.32	8.59	5.72
Median	0.49	0.00	0.00	0.00	0.00 (0.00)	6.46	0.36	0.31

The SWC based algorithm obtains average errors that are less than twice the errors of the other methods. In our experiments we observed that the energy of the final state is usually smaller than the energy of the ground truth state. This fact indicates that the SWC algorithm is doing a good job optimizing the model but the Bayesian model is not accurate enough in its current form and needs to be improved.

Also shown in Table 6.1 are the columns labeled SC^4 and SC^{4k} representing the misclassification errors of the SC method [32] with an affinity matrix with 4-nearest neighbors and 4k-nearest neighbors respectively. These errors are 13.32 and 8.59 respectively and indicate that the Spectral Clustering really needs a dense affinity matrix to work well, and it cannot be accelerated using sparse matrix operations.

Finally, the performance of the SWC-based algorithm is compared with the KASP algorithm [62], which is a fast approximate spectral clustering and was used in place of the spectral clustering step in the SC method [32]. The data reduction parameter used was $\gamma = 10$, which still results in a $O(N^3)$ clustering algorithm. The total misclassification error is 5.72, about three times larger than the SWC method.

Fig. 6.21: Selected frames of sequence cars10 with 1000 tracked feature points. © [2015] CRC Press. Reprinted, with permission, from ref. [13].

In order to evaluate the scalability of different algorithms, sequences with a large number of trajectories are needed. The trajectories can be generated by some optical flow algorithm, but it is difficult to obtain the ground truth segmentation and remove bad trajectories. Brox et.al. [8] provided a dense segmentation for some frames in 12 sequences in the Hopkins 155 dataset[1]. From them, we picked the cars10 sequence (Fig. 6.21) and tracked all pixels of the first frame using the Classic+NL method [50]. There are two reasons for choosing cars10. First, it has three motions, two moving cars and the background. Second, the two moving cars are relatively large in the video, so that a large number of trajectories can be obtained from each motion.

There are 30 frames in the sequence, and 3 of them have a dense manual segmentation of all pixels. We removed trajectories that have different labels on the 3 ground truth frames. The trajectories close to the motion boundaries were also removed, and we only kept the full trajectories for clustering. This way we obtained around 48,000 trajectories as a pool. From the pool, different numbers N of trajectories were subsampled for evaluation. For each given N, trajectories were randomly selected from the pool such that the number of trajectories in each of the three motions was roughly the same. For example, to generate $N = 1000$ trajectories, we would randomly pick 333 trajectories from the pool from two of the motions and 334 trajectories from the third motion. If there were not enough trajectories available from one motion, we would add more from the motion that has the most trajectories.

In Figure 6.22 is shown the computation time vs the number N of trajectories for the spectral clustering (SC) and SW Cuts (SWC) algorithms, in the original scale and in a logarithmic scale. From the figure one could find that for a small number of trajectories, SC is faster than SWC, but for more than $N = 6,000$ trajectories, the computation time of SC is greater than that of SWC, and increases much faster. In Figure 6.22, right are shown the log(time) vs. log(N) and a linear regression is fit through the data points of the two methods. If the slope of the line is α, then the computation complexity scales as $O(N^\alpha)$. We observe that the slope of SC is 2.52 while the slope for SWC is 1.29, which is consistent with the complexity analysis of Section 6.8.1.

[1] http://lmb.informatik.uni-freiburg.de/resources/datasets/

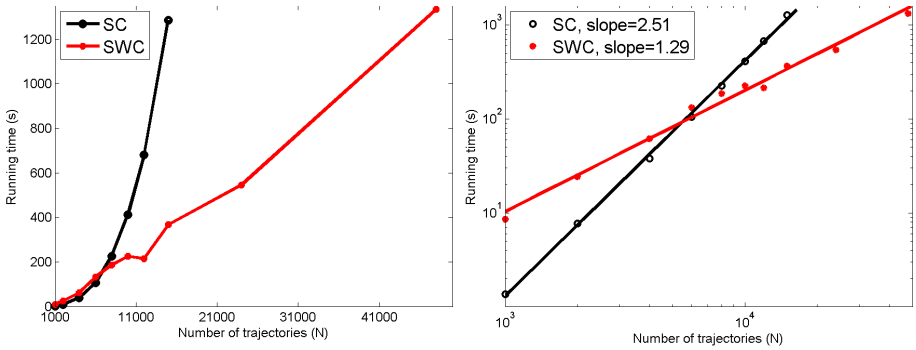


Fig. 6.22: Left. Computation time (sec) vs number of trajectories N for SC and SWC. Right: log-log plot of same data with the fitted regression lines. © [2015] CRC Press. Reprinted, with permission, from ref. [13].

6.9 C4: Clustering Cooperative and Competitive Constraints

Jake Porway

Many vision tasks, such as scene labeling [30, 43, 47], object detection/recognition [16, 53], segmentation [11, 56], and graph matching [9, 33] are formulated as energy minimization (or maximum a posteriori probability) problems defined on graphical models – Markov random fields [5, 21], conditional random fields [30, 31], or hierarchical graphs [20, 64]. These optimization problems become exceedingly difficult when there are multiple solutions, i.e. distinct modes with high probabilities or, in some cases, equal probability.

Figure 6.23 shows examples of typical scenarios that have multiple, equally likely solutions in the absence of further context. The top row shows the well-known Necker Cube which has two valid 3D interpretations. The middle row is the Wittgenstein illusion, in which the drawing can appear to be either a duck or a rabbit. Without further context, we cannot determine the correct labeling. The bottom row shows an aerial image. It can be explained as either a roof with vents or a parking lot containing cars.

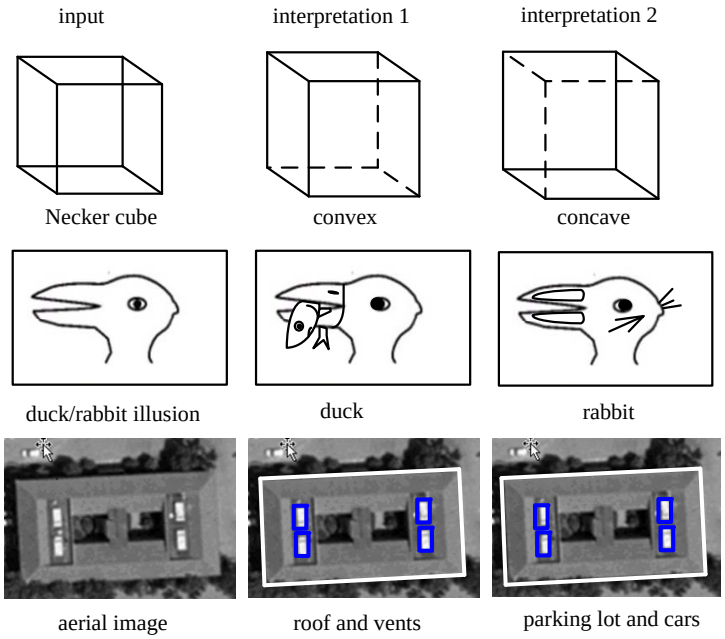


Fig. 6.23: Problems with multiple solutions: (top) the Necker Cube; (Middle) the Wittgenstein illusion; and (Bottom) An aerial image interpreted as either a roof with vents or a parking lot with cars. Ambiguities should be preserved until further context arrives. © [2011] IEEE. Reprinted, with permission, from ref. [42].

Computing multiple solutions is important for preserving the intrinsic ambiguities and avoiding early commitment to a single solution which, even if it's currently the globally optimal one, may turn out to be less favorable when later context arrives. However, it is a persistent challenge to enable algorithms to climb out of local optima and to jump between solutions far apart in the state space. Popular energy minimization algorithms, such as Iterative Conditional Modes (ICM) [5], Loopy Belief Propagation (LBP) [29, 59], and graph cuts [6, 28] compute a single solution and do not address this problem. Existing MCMC algorithms, such as various Gibbs samplers [21, 35], DDMCMC [56], and Swendsen-Wang cuts [3, 51], promise global optimization and ergodicity in the state space, but often require long waiting times to move between distinct modes, which needs a sequence of lucky moves to escape energy wells in the landscape.

In this section, we discuss an algorithm that can discover multiple solutions by jumping between equal probability states and thus preserve the ambiguities on rather general settings:

1. The graph can be flat, such as an MRF or CRF, or hierarchical, such as a parse graph.
2. The graph can have positive (cooperative) and negative (competitive or conflicting) edges for both hard or soft constraints.

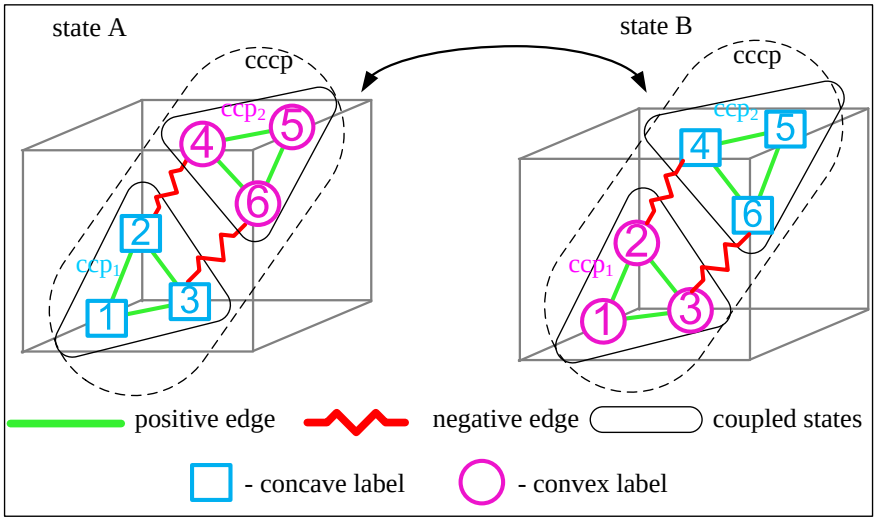


Fig. 6.24: Swapping between the two interpretations of the Necker Cube. Locally coupled labels are swapped with alternate labelings to enforce global consistency. See text for explanation. © [2011] IEEE. Reprinted, with permission, from ref. [42].

3. The probability (energy) defined on the graph can be quite general, even with energy terms involving more than two nodes.

In vision, it is safe to assume that the graph is locally connected and we do not consider the worst case scenario where graphs are fully connected.

In the 1970s, many problems, including line drawing interpretation and scene labeling, were posed as constraint satisfaction problems (CSPs). The CSPs were solved by either heuristic search methods [41] or constraint propagation methods [2, 37]. The former keeps a list of open nodes for plausible alternatives and can backtrack to explore multiple solutions. However, the open list can become too long to maintain when the graph is large. The latter iteratively updates the labels of nodes based on their neighbors. One well-known constraint propagation algorithm is the relaxation labeling method by Rosenfeld, Hummel, and Zucker in 1976 [47].

In the 1980s, the Gibbs sampler was presented in [21]. The update of labels is justified in a solid MCMC and MRF framework and thus is guaranteed to sample from the posterior probabilities. In special cases, the Gibbs sampler is equal to belief propagation [41] for polytrees and to dynamic programming in chains. The Gibbs sampler has been found to slow down critically when a number of nodes in the graph are strongly coupled.

Figure 6.24 illustrates an example of the difficulty associated with strongly coupled graphs using the Necker Cube. The six internal lines of the figure are divided into two coupling groups: (1-2-3) and (4-5-6). Lines in each group must

have the same label (concave or convex) to make a valid cube since they share the two 'Y'-junctions. Thus, updating the label of a single line in a coupled group does not make any difference unless we update the label of the whole group together, i.e. all six labels in one step. The problem is that for general scenarios with large graphs we don't know which nodes in the graph are coupled and to what extent.

6.9.1 Overview of the C^4 algorithm

In this section we present a probabilistic clustering algorithm called *Clustering Cooperative and Competitive Constraints* (C^4) for computing multiple solutions in graphical models. We consider two types of graphs. Adjacency graphs treat each node as an entity, such as a pixel, a superpixel, a line, or an object, which has to be labeled with K-classes (or colors). Most MRFs and CRFs used in computer vision are adjacency graphs. Candidacy graphs treat each node as a candidate or hypothesis, such as a potential label for an entity, or a detected object instance in a window, which has to be confirmed (on) or rejected (off). In other words, the graph is labeled with $K = 2$ colors.

As we will show in section 6.9.2, an adjacency graph can always be converted to a bigger candidacy graph. In both cases, the tasks are posed as graph coloring problems on MRF, CRF or hierarchical graphs. There are two types of edges expressing either hard or soft constraints (or coupling) between the nodes. Positive edges are cooperative constraints that favor the two nodes having the same label in an adjacency graph or being turned off and on simultaneously in a candidacy graph. Negative edges are competitive or conflicting constraints that require the two nodes to have different labels in an adjacency graph or one node to be turned on and the other turned off in a candidacy graph.

In Figure 6.24, we show that the Necker cube can be represented in an adjacency graph with each line being a node. The six internal lines are linked by six positive edges (green) and two negative edges (jagged red). Lines 2 and 4 have a negative edge between them as they intersect with each other, as do lines 3 and 6. We omit the labeling of the six outer lines for clarity.

The edges play a computational role, and are used to dynamically group nodes which are strongly coupled. On each positive or negative edge, we define an edge probability for the coupling strength using bottom-up discriminative models. Then we design a protocol for turning these edges off and on independently according to their edge probabilities respectively for each iteration. The protocol is common for all problems while the edge probabilities are problem specific. This probabilistic procedure turns off some edges, and all the edges that remain on partition the graph into some connected components (ccp's).

A ccp is a set of nodes that are connected by positive edges. For example, Figure 6.24 has two ccp's: ccp_1 includes nodes 1-2-3 and ccp_2 includes nodes 4-5-6. Each ccp is a locally coupled sub-solution. *A* $cccp$ is a composite connected component that consists of a number of ccp's connected by negative edges. For example, Figure 6.24 has one $cccp$ containing ccp_1 and ccp_2. Each $cccp$ contains some conflicting sub-solutions.

At each iteration, C^4 selects a $cccp$ and updates the labels of all nodes in the $cccp$ simultaneously so that (i) nodes in each ccp keep the same label to satisfy the positive or coupling constraints, and (ii) different ccp's in the $cccp$ are assigned different labels to observe the negative constraints. Since C^4 can update a large number of nodes in a single step, it can move out of local modes and jump effectively between multiple solutions. The protocol design groups the $cccp$'s dynamically and guarantees that each step follows the MCMC requirements, such as detailed balance equations and thus it samples from the posterior probability.

6.9.2 Graphs, coupling, and clustering

We start with a flat graph G that we will extend to a hierarchical graph in section 6.9.6,

$$G =< V, E >, \quad E = E^+ \cup E^-. \tag{6.81}$$

Here, $V = \{v_i, i = 1, 2, ..., n\}$ is a set of vertices or nodes on which variables $X = (x_1, ..., x_n)$ are defined, and $E = \{e_{ij} = (v_i, v_j)\}$ is a set of edges which is divided into E^+ and E^- for positive (cooperative) and negative (competitive or conflicting) constraints respectively. We consider adjacency and candidacy graphs for G.

An adjacency graph can always be transferred to a bigger candidacy graph by converting each node v_i into K_i nodes $\{x_{ij}\}$. $x_{ij} \in \{'\text{on}','\text{off}'\}$ represents $x_i = j$ in the adjacency graph. These nodes observe a mutual exclusion constraint to prevent fuzzy assignments to x_i. Figure 6.25 shows this conversion. The adjacency graph $G_{\text{adj}} =< V_{\text{adj}}, E_{\text{adj}} >$ has six nodes $V_{\text{adj}} = \{A, B, C, D, E, F\}$ and each has 3 to 5 potential labels. The variables are $X_{\text{adj}} = (x_A, ..., x_F)$ with $x_A \in \{1, 2, 3, 4, 5\}$ and so on. We convert it to a candidacy graph $G_{\text{can}} =< V_{\text{can}}, E_{\text{can}} >$ with 24 nodes $V_{\text{can}} = \{A_1, ..., A_5, ..., F_1, ..., F_4\}$. Node A_1 represents a candidate hypothesis that assigns $x_A = 1$. The $X_{\text{can}} = (x_{A_1}, ..., x_{F_4})$ are Boolean variables.

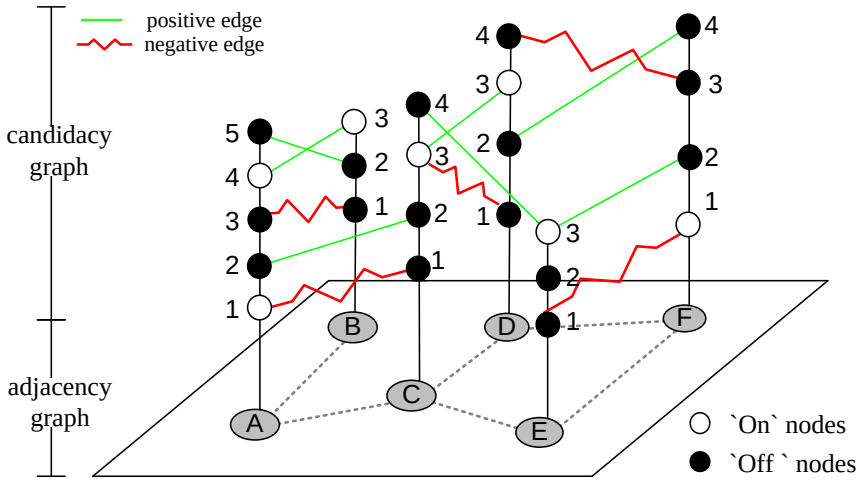


Fig. 6.25: Converting an adjacency graph to a candidacy graph. The candidacy graph has positive (straight green lines) and negative (jagged red lines) edges depending on the values assigned to the nodes in the adjacency graph. © [2011] IEEE. Reprinted, with permission, from ref. [42].

Represented by the graph G, the vision task is posed as an optimization problem that computes the most probable interpretation with a posterior probability $p(X|\mathbf{I})$ or an energy function $\mathcal{E}(X)$.

$$X^* = \arg\max p(X|\mathbf{I}) = \arg\min \mathcal{E}(X). \tag{6.82}$$

To preserve the ambiguity and uncertainty, we may compute multiple distinct solutions $\{X_i\}$ with weights $\{\omega_i\}$ to represent the posterior probability.

$$(X_i, \omega_i) \sim p(X|\mathbf{I}), \quad i = 1, 2, ..., K. \tag{6.83}$$

In conventional vision formulation, edges in the graphs are a representational concept and the energy terms in $\mathcal{E}$ are defined on the edges to express the interactions between nodes. In contrast, Swendsen-Wang [51] and Edward-Sokal [14] added a new computational role to the edges in their cluster sampling method. The edges are turned off and on probabilistically to dynamically form groups (or clusters) of nodes which are strongly coupled. We will introduce the clustering procedure shortly after the example below. In this section, we adopt this notion and the edges in graph G are characterized in three aspects:

Positive vs. negative. A positive edge represents a cooperative constraint for two nodes having the same label in an adjacency graph or being turned on (or off) simultaneously in a candidacy graph. A negative edge requires the two nodes to have different labels in an adjacency graph or requires one node to be turned on and the other turned off in a candidacy graph.

Hard vs. soft. Some edges represent hard constraints which must be satisfied, while other edge constraints are soft and can be expressed with a probability.

Position dependent vs. value dependent. Edges in adjacency graphs are generally position dependent. For example, in an Ising model an edge between two adjacent nodes poses a soft constraint that they should have the same or opposite labels. In contrast, edges in candidacy graphs are value dependent and thus have more expressive power. This is common for vision tasks, such as scene labeling, line drawing interpretation, and graph matching. As Figure 6.25

illustrates, the edges between nodes in the candidacy graph could be either positive or negative depending on the values assigned to nodes A and B in the adjacency graph.

We show in a later subsection that the positive and negative edges are crucial for generating connected components and resolving the problem of node coupling.

Figure 6.26 shows the construction of a candidacy graph G for interpreting the Necker cube. For clarity of discussion we assume the exterior lines are labeled and the task is to assign two labels (concave and convex) to the six inner lines such that all local and global constraints are satisfied. Therefore we have a total of 12 candidate assignments or nodes in G.

Based on the theory of line drawing interpretation [38, 49], the two 'Y'-junctions pose positive constraints so that lines 1-2-3 have the same label and lines 4-5-6 have the same label. We have 12 positive edges (green) in G to express these constraints. The intersection of lines 2 and 4 poses negative constraints that require lines 2 and 4 to have opposite labels which are shown in the jagged red edges in Figure 6.26. The same is true for lines 3 and 6. The two different assignments for each line should also be linked by a negative edge. These negative edges are not shown for clarity.

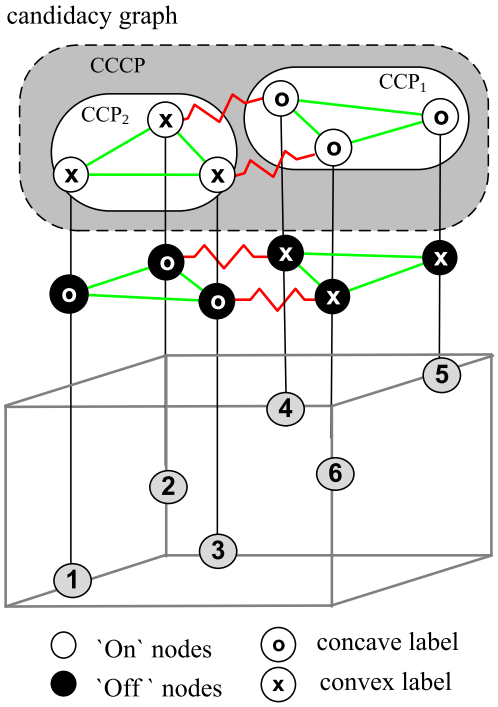


Fig. 6.26: The Necker cube example. The adjacency graph with 6 nodes (bottom) is converted to a candidacy graph of 12 nodes (top) for concave and convex label assignments respectively. 12 positive and 2 negative edges are placed between these candidate assignments to ensure consistency. © [2011] IEEE. Reprinted, with permission, from ref. [42].

In this candidacy graph, the two solutions that satisfy all constraints are represented by the 2-colors in Figure 6.26. The first has all nodes 1,2, and 3 labeled convex (x) and all nodes 4,5, and 6 labeled concave (o). This solution is currently in the on state. This would create a valid 3D interpretation where the cube is coming out of the page. The alternative solution has the opposite labeling, and creates a 3D interpretation of the cube sinking into the page.

To switch from one solution to the other we must swap the junction labels. Each set of nodes, 1-2-3 and 4-5-6, constitutes a corner of the Necker Cube and all have positive constraints between them. This indicates that we should update all of these values simultaneously. We create two connected components ccp_1 and ccp_2, comprised of the coupled nodes 1-2-3 and nodes 4-5-6 respectively. If we were simply to invert the labels of ccp_1 or ccp_2 alone we would create an inconsistent interpretation where all edges in the whole graph would now have the same label. What we need to do is simultaneously swap ccp_1 and ccp_2.

Notice that we have negative edges between nodes 2 and 4 and between nodes 3 and 6. Negative edges can be thought of as indicators of multiple competing solutions, as they necessarily dictate that groups on either end of the edge can either be (on, off) or (off, on), creating two possible outcomes. This negative edge connects nodes in ccp_1 and ccp_2, thus indicating that those nodes in the two ccp's must have different labels. We construct a composite connected component, $cccp_{12}$, encompassing nodes 1-6 and now have a full component that contains all relevant constraints. Moving from solution 1 to 2 is now as simple as flipping all the nodes simultaneously, or equivalently satisfying all of the constraints. In the next subsection, we explain how to form ccp's and $cccp$'s in a formal way.

On each positive or negative edge, we define an edge probability for the coupling strength. That is, at each edge $e \in E$, we define an auxiliary probability $u_e \in \{0,1\}$, which follows an independent probability q_e.

In Swendsen and Wang [51], the definition of q_e is decided by the energy term in the Potts model $q_e = e^{-2\beta}$ as a constant for all e. Barbu and Zhu [3] separated q_e from the energy function and defined it as a bottom-up probability, $q_e = p(l(x_i) = l(x_j)|F(x_i),F(x_j)) = p(e = on|F(x_i),F(x_j))$, with $F(x_i)$ and $F(x_j)$ being local features extracted at nodes x_i and x_j. This can be learned through discriminative training, for example, by logistic regression and boosting,

$$\frac{p(l(x_i) = l(x_j)|F(x_i),F(x_j))}{p(l(x_i) \neq l(x_j)|F(x_i),F(x_j))} = \sum_n \lambda_n h_n(F(x_i),F(x_j)).$$

On a positive edge $e = (i, j) \in E^+$, $u_e = on$ follows a Bernoulli probability,

$$u_e \sim \text{Bernoulli}(q_e \cdot 1(x_i = x_j)).$$

Therefore, at the present state X, if the two nodes have the same color, i.e. $x_i = x_j$, then the edge e is turned on with probability q_e. If $x_i \neq x_j$, then $u_e \sim \text{Bernoulli}(0)$ and e is turned off with probability 1. So, if two nodes are strongly coupled, q_e should have a higher value to ensure that they have a higher probability to stay the same color. Similarly, for negative edges $e \in E^-$, $u_e =$ 'on' also follows a Bernoulli probability,

$$u_e \sim \text{Bernoulli}(q_e 1(x_i \neq x_j)).$$

At the present state X, if the two nodes have the same color $x_i = x_j$, then the edge e is turned off with probability 1, otherwise e is turned on with probability q_e to enforce that x_i and x_j stay in different colors.

After sampling u_e for all $e \in E$ independently, we denote the sets of positive and negative edges that remain on as $E^+_{on} \subset E_+$ and $E^-_{on} \subset E^-$ respectively. Now we have formal definitions of ccp and $cccp$.

Definition 6.3 *A ccp is a set of vertices $\{v_i; i = 1,2,...,k\}$ for which every vertex is reachable from every other vertex by the positive edges in E^+_{on}.*

Definition 6.4 *A $cccp$ is a set of ccp's $\{ccp_i; i = 1,2,...,m\}$ for which every ccp is reachable from every other ccp by the negative edges in E^-_{on}.*

No two ccp's are reachable by positive edges, or else they would be a single ccp. Thus a $cccp$ is a set of isolated ccp's that are connected by negative edges. An isolated ccp is also treated as a $cccp$. In section 6.9.6, we will treat the invalid cases where a ccp contains negative edges by converting it to a $cccp$. To observe the detailed balance equations in MCMC design, we need to calculate the probabilities for selecting a ccp or $cccp$ which are determined by the edge probabilities q_e. For this purpose we define their cuts. In general, a cut is the set of all edges connecting nodes between two node sets.

Definition 6.5 *Under a current state X, a cut for a ccp is the set all positive edges between nodes in ccp and its surrounding nodes which have the same label,*

$$Cut(ccp|X) = \{e : \ e \in E^+, x_i = x_j, i \in ccp, \ j \notin ccp\}.$$

These are the edges that must be turned off probabilistically (with probability $1 - q_e$) in order to form the ccp. The cut depends on the state X.

Definition 6.6 *A cut for a $cccp$ at a state X is the set of all negative (or positive) edges connecting the nodes in the $cccp$ and its neighboring node which have different (or same) labels,*

$$Cut(cccp|X) = \{e:\ e \in E^-, i \in cccp,\ j \notin cccp, x_i \neq x_j\} \cup \{e:\ e \in E^+, i \in cccp,\ j \notin cccp, x_i = x_j\}.$$

All of these edges must be turned off probabilistically (with probability $1-q_e$) in order to form the $cccp$ at state X. As edges in E^+_{on} only connect nodes with the same label, all nodes in a ccp must have the same label. In contrast, all edges in E^-_{on} only connect nodes with different labels, and thus adjacent ccp's in a $cccp$ must have different labels.

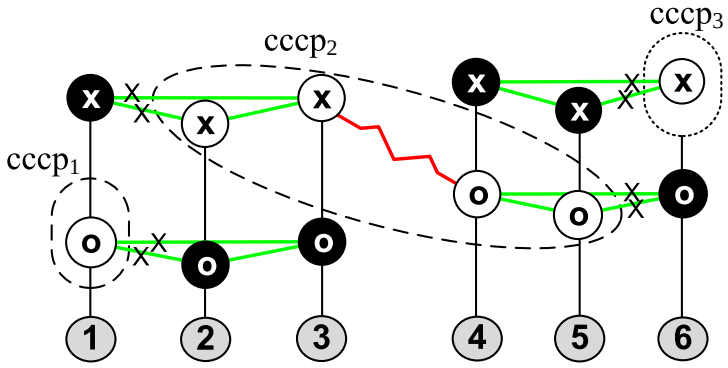


Fig. 6.27: A Necker cube candidacy graph not in a solution state. © [2011] IEEE. Reprinted, with permission, from ref. [42].

To illustrate these concepts, we show a non-solution state X for the Necker cube in Figure 6.27. By turning off some edges (marked with the crosses), we obtain three $cccp$'s for the nodes that are currently on. In this example, $q_e = 1$, as these are hard constraints that are inviolable. $cccp_1$ and $cccp_3$ have only 1 node while $cccp_2$ has two ccp's with 4 nodes. The algorithm will now arbitrarily select a $cccp$ and update its values according to its constraints. If it selects either $cccp_1$ or $cccp_3$, then we are one step closer to the solution. If it selects $cccp_2$, then all 4 vertex labels will be swapped, and we will have reached a solution state. In that case we will continue to swap back and forth between the two solutions.

6.9.3 C^4 algorithm on flat graphs

The C^4 algorithm on flat graphs works iteratively following the MCMC design. In each iteration, it generates the $cccp$'s, selects a $cccp_o$ with some probability, and reassigns labels to its ccp's such that all internal negative constraints are satisfied. As the number of ccp's in $cccp_o$ grows larger, the number of potential labelings will grow as well. One can remedy this situation in two ways:

1. Use a constraint-satisfaction problem-solver (CSP-solver) to solve a smaller, easier constraint satisfaction problem within $cccp_o$.
2. Use random or heuristic sampling to find a new valid labeling.

We will use the second approach throughout this section and the number of ccp's in $cccp_o$ will be small in general, so the label assignment is not a problem. The C^4 algorithm can be a viewed as a method that breaks a large constraint-satisfaction problem into smaller fragments which can be satisfied locally. It then propagates the solution through iterations. This assignment represents a move in MCMC which is accepted by the Metropolis-Hastings step with an acceptance probability. This acceptance probability accounts for the probabilities of generating the $cccp$'s, selecting a $cccp_o$, assigning new labels, and the posterior probability.

To summarize, a pseudo-code version of the algorithm is given below.

The C^4 Algorithm

Input: A graph $G =< V, E >$ and posterior prob. $p(X|\mathbf{I})$.
Calculate the edge probability $q_e, \forall e \in E$.

q_e is a problem specific discriminative probability.

Initialize the state $X = (x_1, x_2, ..., x_n)$.

e.g. all nodes are turned off in a candidacy graph.

for $s = 1$ to N^{iter} **do**
Denote the current X by state A.
Step 1: Generate a $cccp_o$ at state A.
$\forall e = (i, j) \in E^+$, sample $u_e \sim$ Bernoulli$(q_e 1(x_i = x_j))$.
$\forall e = (i, j) \in E^-$, sample $u_e \sim$ Bernoulli$(q_e 1(x_i \neq x_j))$.
Generate the $\{ccp\}$ and $\{cccp\}$ based on E^+_{on} and E^-_{on}
Select a $cccp_o$ from $\{cccp\}$ probabilistically
Denote the prob for selecting $cccp_o$ by $q(cccp_o|A)$.
Step 2: Assign labels to ccp's in the $cccp$ with probability: $q(l(cccp_o = L|cccp_o, A))$.
Denote the new X as state B.
Step 3: Calculate the acceptance probability:

$$\alpha(A \to B) = \min(1, \frac{q(B \to A)}{q(A \to B)} \cdot \frac{p(X = B|\mathbf{I})}{p(X = A|\mathbf{I})}).$$

end for
Output: Distinct states $\{X^*\}$ with highest probabilities.

In Markov chain design, each move between two states A and B is made reversible and observes the detailed balance equation,

$$p(X = A|\mathbf{I})\mathcal{K}(A \to B) = p(X = B|\mathbf{I})\mathcal{K}(B \to A). \tag{6.84}$$

$\mathcal{K}(A \to B)$ is the Markov chain kernel or transition probability from A to B. In the Metropolis-Hastings design,

$$\mathcal{K}(A \to B) = q(A \to B)\alpha(A \to B), \ \forall A \neq B. \tag{6.85}$$

$q(A \to B)$ is the probability for proposing state B from state A, and $\alpha(A \to B)$ is the acceptance probability,

$$\alpha(A \to B) = \min(1, \frac{q(B \to A)}{q(A \to B)} \cdot \frac{p(X = B|\mathbf{I})}{p(X = A|\mathbf{I})}). \tag{6.86}$$

It is easy to check that the design of proposal probability in equation (6.86) and the acceptance probability in equation (6.85) makes the kernel satisfy the detailed balance equation in (6.84), which in turn suffices for the invariance condition,

$$p(X = A|\mathbf{I})\mathcal{K}(A \to B) = p(X = B|\mathbf{I}). \tag{6.87}$$

So, $p(X|\mathbf{I})$ is the invariant probability of the Markov chain with kernel $\mathcal{K}$. Now we elaborate on the design of proposal and acceptance probabilities. The acceptance probability is determined by two ratios.

(i) The ratio $\frac{p(X=B|\mathbf{I})}{p(X=A|\mathbf{I})}$ is problem specific and is not part of our design. The posterior probability can be in a general form and does not have to be modified or approximated to fit the C^4 algorithm. Since states A and B only differ in their labels for nodes in $cccp_o$, this ratio can often be computed locally if the posterior probability is a MRF or CRF.

(ii) The proposal probability ratio is completely up to our design, and it includes two parts,

$$\frac{q(B \to A)}{q(A \to B)} = \frac{q(cccp_o|B)}{q(cccp_o|A)} \cdot \frac{q(l(cccp_o) = L_A|cccp_o, B)}{q(l(cccp_o) = L_B|cccp_o, A)}.$$

$q(cccp_0|A)$ and $q(cccp_0|B)$ are the probabilities for choosing $cccp_o$ at states A and B respectively. Given the chosen $cccp_o$, the assignment of new labels is independent of the surrounding neighbors of $cccp_o$ and is often as-

signed by equal probability among all valid assignments in the CSP-solver. Thus, they cancel out, and we have $\frac{q(l(cccp_o)=L_A|cccp_o,B)}{q(l(cccp_o)=L_B|cccp_o,A)}=1$.

To summarize, the key to the algorithm design is the ratio $\frac{q(cccp_o|B)}{q(cccp_o|A)}$. In single site sampling, such as Gibbs sampler, each node is a $cccp_o$ and the selection is simply a visiting scheme. In C^4, the probability for choosing $cccp_o$ at a state depends on two probabilities: (a) How likely it is to generate $cccp_o$ by sampling the edge probabilities q_e following the Bernoulli probability and (b) How likely it is to select $cccp_o$ from the set of formed $\{cccp\}$ in states A and B. These probabilities are hard to compute because there are a vast amount of partitions of the graph that include a certain $cccp_o$ by turning on/off edges. A partition is a set of $cccp$'s after turning off some edges.

Interestingly, the set of all possible partitions in state A is identical to those in state B, and all of these partitions must share the same cut $Cut(cccp_o)$. That is, in order for $cccp_o$ to be a composite connected component, its connections with its neighboring nodes must be turned off. Even though the probabilities are in complex form, their ratio is simple and clean due to cancellation. Furthermore, given the partition, $cccp_o$ is selected with uniform probability from all possible $cccp$'s.

Proposition 1 *The proposal probability ratio for selecting $cccp_o$ at states A and B is*

$$\frac{q(cccp_0|B)}{q(cccp_0|A)}=\frac{\prod_{e\in Cut(cccp_o|B)}(1-q_e)}{\prod_{e\in Cut(cccp_o|A)}(1-q_e)}. \tag{6.88}$$

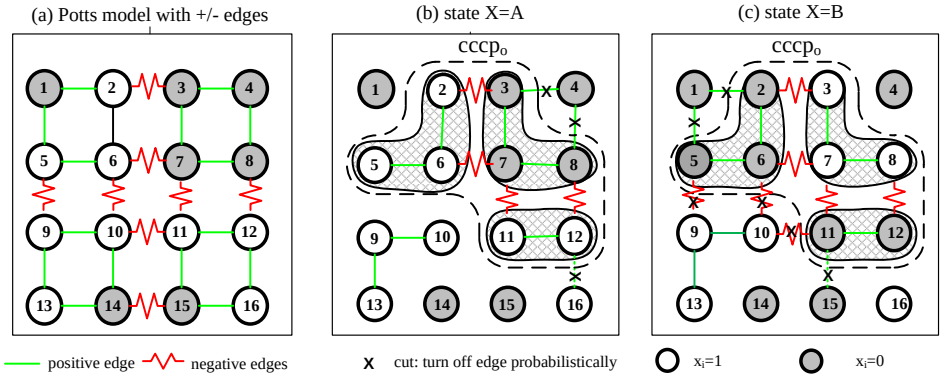


Fig. 6.28: The Potts model with negative edges. (a) Minimum energy is found in a checkerboard pattern. (b) Forming cccps. (c) $cccp_0$ consists of sub-ccps of positive edges connected by negative edges. © [2011] IEEE. Reprinted, with permission, from ref. [42].

To illustrate C^4, we derive it in more detail for a Potts model with positive and negative edges. Let X be a random field defined on a 2D lattice with discrete states $x_i \in \{0,1,2,...,L-1\}$. Its probability is specified by

$$p(X)=\frac{1}{Z}\exp\{-\mathcal{E}(X)\};\ \text{with}\ \mathcal{E}(X)=\sum_{<i,j>\in E^+}\beta\delta(x_i=x_j)+\sum_{<i,j>\in E^-}\beta\delta(x_i\neq x_j), \tag{6.89}$$

where $\beta>0$ is a constant. The edge probability will be $q_e=1-e^{-\beta}$ for all edges.

Figure 6.28 (a) shows an example on a small lattice with $L=2$ labels, which is an adjacency graph with position dependent edges. The states with checkerboard patterns will have highest probabilities. Figures 6.28 (b) and (c) show two reversible states A and B by flipping the label of a $cccp_o$ in one step. In this example, $cccp_o$ has three ccp's, $cccp_o=\{\{2,5,6\};\{3,7,8\};\{11,12\}\}$. The labels of the 8 nodes are re-assigned with uniform probability, and this leads to the difference in the cuts for $cccp_o$ at the two states, $Cut(cccp_o|A)=\{(3,4),(4,8),(12,16)\}$ and $Cut(cccp_o|B)=\{(1,2),(1,5),(5,9),(6,10),(10,11),(11,15)\}$.

Proposition 2 *The acceptance probability for C^4 on the Potts model is $\alpha(A \to B) = 1$ for any two states with different labels in $cccp_o$. Therefore, the move is always accepted.*

The proof follows two observations. Firstly, the energy terms inside and outside $cccp_o$ are the same for both A and B, and they differ only at the cuts of $cccp_o$. More precisely, let $c = |Cut(cccp_o|B)| - |Cut(cccp_o|A)$ be the difference in the sizes of the two cuts (i.e. $c = 3$ in our example). It is not too hard to show that

$$\frac{p(X = B|\mathbf{I})}{p(X = A|\mathbf{I})} = e^{-\beta c}. \tag{6.90}$$

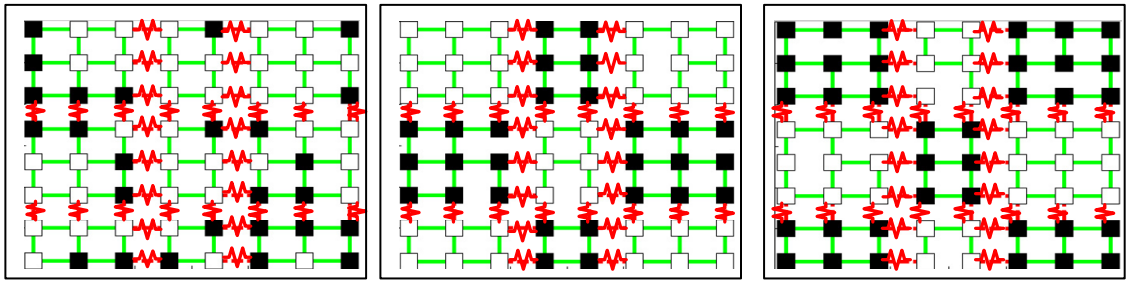
Secondly, we have the proposal probability ratio, following equation(6.88),

$$\frac{q(cccp_0|B)}{q(cccp_0|A)} = \frac{(1-q_e)^{|Cut(cccp_o|B)|}}{(1-q_e)^{|Cut(cccp_o|A)|}} = e^{\beta c}. \tag{6.91}$$

Plugging the two ratios in equation 6.86, we have $\alpha(A \to B) = 1$.

6.9.4 Experiments on flat graphs

In this section we test C^4's performance on some flat graphs (MRF and CRF) in comparison with the Gibbs sampler [21], SW method [51], iterated conditional modes (ICM), graph cuts [6], and loopy belief propagation (LBP) [29]. We choose classical examples: (i) Ising/Potts model for MRF; (ii) Line drawing interpretation for constrained-satisfaction problem using candidacy graph; (iii) scene labeling using CRF; and (iv) scene interpretation of aerial images.



(a) initial state (b) solution state 1 (c) solution state 2

Fig. 6.29: The Ising/Potts model with checkerboard constraints and two minimum energy states computed by C^4. © [2011] IEEE. Reprinted, with permission, from ref. [42].

6.9.5 Checkerboard Ising model

We first consider the Ising model on a 9×9 lattice with positive and negative edges. We tested C^4 with two parameters settings: (i) $\beta = 1$, $q_e = 0.632$; and (ii) $\beta = 5$, $q_e = 0.993$. In this lattice we have created a checkerboard pattern. We have assigned negative and positive edges so that blocks of nodes want to be the same color, but these blocks want to be different colors than their neighbors.

Figure 6.29 shows a typical initial state to start the algorithm, and two solutions with minimum (i.e. 0) energy. Figure 6.30(a) shows a plot of energy versus time for C^4, Gibbs sampler, SW, graph cuts, and LBP. C^4 converges second fastest of all five algorithms in about 10 iterations, falling behind graph cuts. Belief propagation cannot converge due to the loopiness of the graph, and Gibbs sampler and the conventional Swendsen-Wang cannot quickly satisfy the constraints as they do not update enough of the space at each iteration. This shows that C^4 has a very low burn-in time.

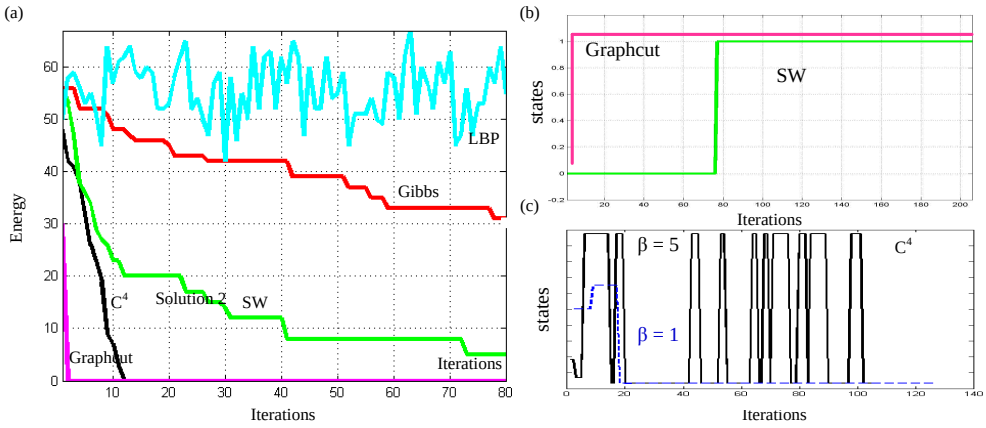


Fig. 6.30: (a) Energy plots of C^4, SW, Gibbs sampler, graph cuts, and LBP on the Ising model vs. time. (b),(c) The state (visited by the algorithms) in time for graph cuts, SW and C^4. Once SW and graph cuts hit the first solution, they get stuck while C^4 keeps swapping between the two minimum energy states. C^4 results shown for $\beta = 1$ and $\beta = 5$. © [2011] IEEE. Reprinted, with permission, from ref. [42].

Figures 6.30(b) and (c) show the states visited at each iteration. We show the states in 3 levels: the curve hits the ceiling or floor for the two minimum energy states respectively, and the middle for all other states. Here we are only comparing graph cuts, SW and C^4 as they are the only algorithms that converge to a solution in a reasonable amount of time. C^4 is clearly swapping solutions while SW and graph cuts get stuck in their first solution. This is because C^4 can group along negative edges as well as positive edges to update large portions of the system at once, while Swendsen-Wang is stuck proposing low probability moves over smaller portions of the solution space.

We also compared the results for experiments where $\beta = 1$ and $\beta = 5$. Figure 6.30(c) shows the states visited by the sampler over time. In the $\beta = 1$ case, it takes longer for C^4 to converge, because it can't form large components with high probability. As β gets large, however, C^4 very quickly takes steps in the space towards the solution and can move rapidly between solution states. We have found that an annealing schedule where $q_e = 1 - e^{-\beta/T}$ and T is adjusted such that q_e moves from 0 to 1 over the course of the experiment works quite well too.

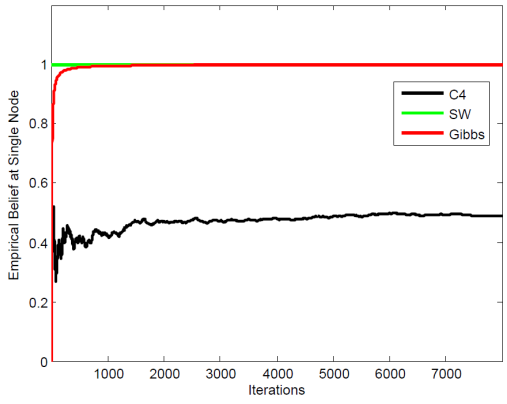


Fig. 6.31: Comparison of marginal beliefs at a single site of the Ising model for Gibbs, SW, and C^4. C^4 correctly converges toward 0.5, while the other algorithms only find a single solution state. LBP does not converge and thus has erratic beliefs that we do not show on this plot. © [2011] IEEE. Reprinted, with permission, from ref. [42].

We finally compare the estimated marginal beliefs at each node as computed by each algorithm. LBP computes these beliefs directly, but we can estimate them for Gibbs sampling, SW, and C^4 by running each algorithm and recording the empirical mean at each iteration for each node given the previous states. Figure 6.31 shows the belief for one of the Ising model sites over time for each of the 4 algorithms. LBP does not converge, so it has a noisy estimate

over time and is not plotted for clarity, Gibbs and SW converge to a probability of 1, because they get stuck in a single solution state, while C^4 approaches 0.5, as it keeps flipping between the two states.

We ran the same experiment above but this time each site could take one of seven possible colors ($L = 7$). In this example, we have a large number of equal states (in checkerboard pattern) with minimum energy.

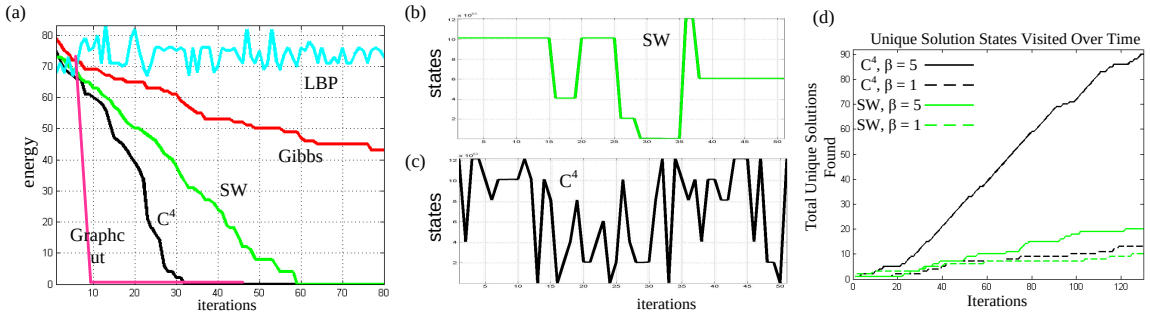


Fig. 6.32: (a) Energy plots of C^4, SW, Gibbs sampler, and LBP on the Potts model ($L = 7$) vs. time. (b) (c) The minimum energy states visited by SW and C^4 algorithms over time. (d) Total number of unique solutions found vs. time for SW and C^4 with $\beta = 1$ and $\beta = 5$. © [2011] IEEE. Reprinted, with permission, from ref. [42].

Figure 6.32(a) plots the energy convergence of each algorithm over time. Graph cuts again converges to just one of the many solutions. Unlike in the case of the $L = 2$ model, SW is able to find multiple solutions this time, as seen in Figure 6.32(b). Figure 6.32(c) shows the number of distinct states with minimum energy that have been visited by SW and C^4 over time. We see that C^4 explores more states in a given time limit which again demonstrates that C^4 is more dynamic and thus has fast mixing time – a crucial measure for the efficiency of MCMC algorithms. We also compare the case where $\beta = 1$ vs. $\beta = 5$. Once again, we see that $\beta = 1$ doesn't create strong enough connections for C^4 to move out of local minimum, so it finds roughly as many unique solutions as Swendsen-Wang does (about 13). When β is increased to 5, however, the number skyrockets from 13 to 90. We thus see that C^4 can move around the solution space much more rapidly than other methods when β is high and can discover a huge number of unique solution states.

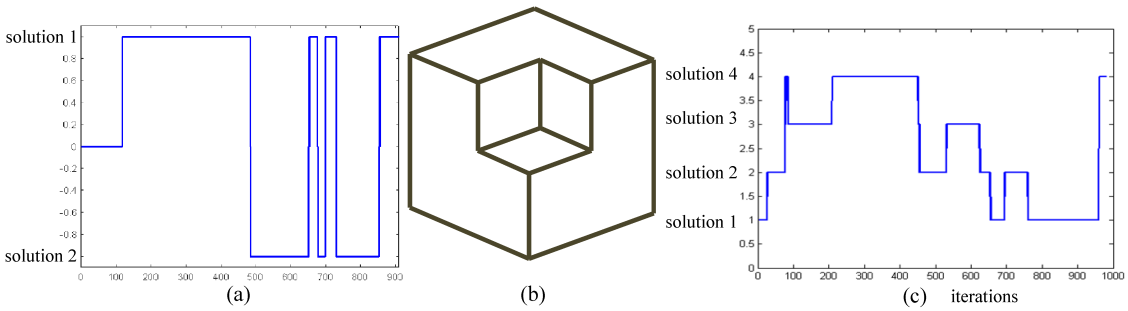


Fig. 6.33: Experimental results for swapping state between interpretations: (a) States visited by C^4 for the Necker Cube. (b) A line drawing with outer and inner cubes. (c) States visited by C^4 for the double cubes. © [2011] IEEE. Reprinted, with permission, from ref. [42].

The previous two examples are based on MRF models whose edges are position dependent. Now we consider a line drawing interpretation on candidacy graph. We use two classical examples which have multiple stable interpretations, or solutions: (i) the Necker cube in Figure 6.23 that has two interpretations and (ii) a line drawing with double cubes in Figure 6.33 that has four interpretations. Swapping between these states involves flipping of 3 or 12 lines simultaneously. Our goal is to test whether the algorithms can compute the multiple distinct solutions over time.

We adopt a Potts-like model on the candidacy graph. Each line in the line drawing is a node in the Potts model, which can take one of eight line drawing labels indicating whether the edge is concave, convex, or a depth boundary. See [49] for an in-depth discussion on labels for consistent line drawings. We add an edge in the candidacy graph

between any two lines that share a junction. At each junction, there are only a small set of valid labels for each line that are realizable in a 3D world. We add positive edges between pairs of line labels that are consistent with one of these junction types, and negative edges between line labels that are not. Thus, we model the pairwise compatibility of neighboring line labels given the type of junction they form.

For these experiments we set $\beta = 2$, resulting in $q_e = 0.865$. Figures 6.33(a) and (c) plot the states visited by the algorithms over time. Once again we see that C^4 can quickly switch between solutions in situations where CSP-solvers or other MCMC methods would likely get stuck.

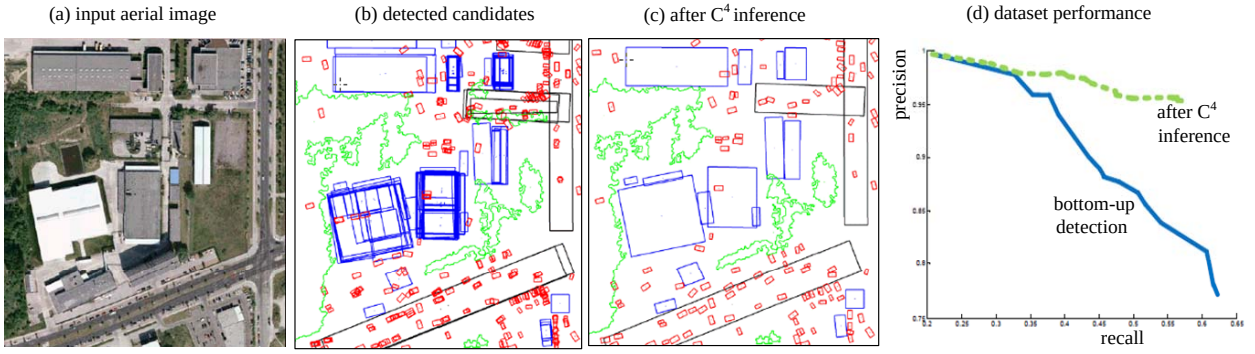


Fig. 6.34: An application of C^4 to aerial image parsing. (a) A portion of an aerial image from Google earth. (b) A set of bottom-up detections of objects with each being a candidate, i.e. a node in the candidacy graph. Note the large number of false positives that need to be turned off. (b) The final subset of proposals selected by C^4 to represent the scene. C^4 has removed the candidates that are inconsistent with the prior. (c) Precision recall curve for pixel level performance over a dataset of aerial images. © [2011] IEEE. Reprinted, with permission, from ref. [42].

In the next experiment, we use C^4 to parse aerial images. This experiment is an extension of work from [43]. In [43], aerial images are represented as collections of groups of objects, related by statistical appearance constraints. These constraints are learned automatically in an offline phase prior to inference.

We create our candidacy graph by letting each bottom-up detected window be a vertex in the graph, connected by edges with probabilities proportional to how compatible those objects are. Each candidate can be on or off, indicating whether it is in the current explanation of the scene or not.

Each edge is assigned to be positive or negative and given a probability q_e of being on by examining the energy $\varepsilon = \phi(x_i, x_j)$ between its two nodes. If $\varepsilon > t$ the edge is labeled as negative, and if $\varepsilon < t$ the edge is labeled as positive, where t is a threshold of the user's choosing. In our experiments we let $t = 0$. In this way we create data-driven edge probabilities and determine positive and negative edge types for C^4.

In this experiment we learned a prior model for likely object configurations using labeled aerial images. Object boundaries were labeled in each image from a set of over 50 images. We tested the results on five large aerial images collected from Google Earth that were also labeled by hand, so that we could measure how much C^4 improved the final detection results. Though we only use five images, each image is larger than 1000×1000 pixels and includes hundreds of objects, so one could also think of the evaluation as spanning 125 images of 200×200 pixels.

Figure 6.34 shows an example of a parsed aerial scene. The bottom-up detected windows are treated as candidates and many are false positives. After using C^4 to minimize a global energy function, however, we are left with the subset that best satisfies the constraints of the system. The false positive rates are vastly diminished after C^4 rules out incompatible proposals. Figure 6.34(d) shows the precision-recall curve for aerial image object detection using C^4 vs. just bottom-up cues. We can see that the C^4 curve, drawn in dashed green, has a much higher precision than the bottom-up detections even as the recall increases. We also compared the results of using C^4 over LBP, ICM, and SW for similar false alarm rates. The results are shown in Table 6.2

Method	False Positives per image	Detection Rate (%)
LBP	85.32	0.668
ICM	82.11	0.768
SW	87.91	0.813
C^4	**83.04**	**0.875**

Table 6.2: False positives per image and detection rate using Loopy BP, SW, ICM, and C^4 for aerial image parsing.

6.9.6 C^4 on hierarchical graphs

In this section, we discuss the consistency of the flat graphs and extend C^4 from flat graphs to hierarchical graphs. We then address high-order constraints that involve more than two sites.

In each iteration of the C^4 algorithm, suppose we have turned on edges probabilistically and the original graph $G =< V,E >$ becomes $G_{on} =< V,E_{on} >$ with $E = E_{on} \cup E_{off}$, $E_{on} = E^+_{on} \cup E^-_{on}$, and $E_{off} = E^+_{off} \cup E^-_{off}$. As we discussed in Section 6.9.2 all nodes in the graph G_{on} in each ccp shares the same label and are supposed to form a coupled partial solution. However, if the constraints in graph G are inconsistent, then some nodes in a ccp may be connected by edges in E^-_{off}. Though such negative edges are not turned on in ccp, they indicate that some nodes in the ccp may be conflicting to each other. This may not be a serious problem, for example, the negative edges may simply express soft constraints, such as overlapping windows, which is acceptable in the final solution.

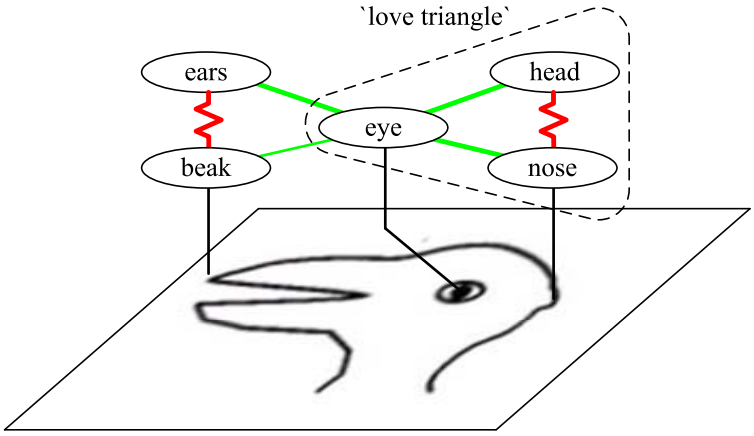


Fig. 6.35: An attempt to solve the duck/rabbit illusion using flat C^4. We see that we are very likely to form love triangles on the left and right of the graph, making constraint satisfaction very difficult. © [2011] IEEE. Reprinted, with permission, from ref. [42].

Figure 6.35 shows an example where the negative edge is a hard constraint. If we try to solve the duck/rabbit illusion using a flat candidacy graph, a ccp may contain $\{'eye','nose','head'\}$ which is inconsistent. We call this grouping a *l*ove triangle.

Definition 6.7 *In a graph G, two nodes i and j connected by a negative edge are said to be involved in a* love triangle *if there also exists a path between i, j that consists of all positive edges.*

Definition 6.8 *A ccp is said to be* consistent *in graph G if there are no negative edges in E that connect two nodes in the ccp, that is,* $\{e : i, j \in ccp\} \cap E^- = \emptyset$. *A graph G is said to be* consistent *if all of its ccp's are always consistent in* C^4.

When a graph is consistent, then we are guaranteed to get valid solutions. The existence of the so-called love triangles are the sole reason to generate inconsistent ccp's. For this we can easily prove the following proposition.

Proposition 3 *In the absence of love triangles, the graph G will be consistent.*

The essential reason for generating love triangles in a graph, mostly in candidacy graphs, is that certain nodes are overloaded with multiple labels and thus they are coupled with conflicting nodes. For example, the node 'eye' should be either a 'rabbit eye' or a 'duck eye' and it should be split into two conflicting candidates connected by an negative edge. This way it can eliminate the love triangle. Figure 6.36 illustrates that we can remove the love triangle by splitting node 1 into nodes 1 and $1'$, and thus we will have consistent *ccp*.

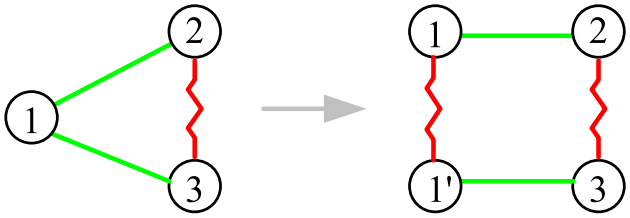


Fig. 6.36: Breaking the love triangle in a candidacy graph. © [2011] IEEE. Reprinted, with permission, from ref. [42].

One other common issue that we need to address is higher-order constraints that involve more than 2 nodes. Figure 6.37 shows a hierarchical graph representation for the duck/rabbit illusion. This is a candidacy graph with two layers. The top layer contains two hidden candidate hypotheses: duck and rabbit. The two nodes are decomposed into three parts in layer 1 respectively and thus impose high order constraints between them. Now the hypotheses for parts are specifically for duck.eye, rabbit.eye, etc. The negative edge connecting the two object nodes is inherited from their overlapping children.

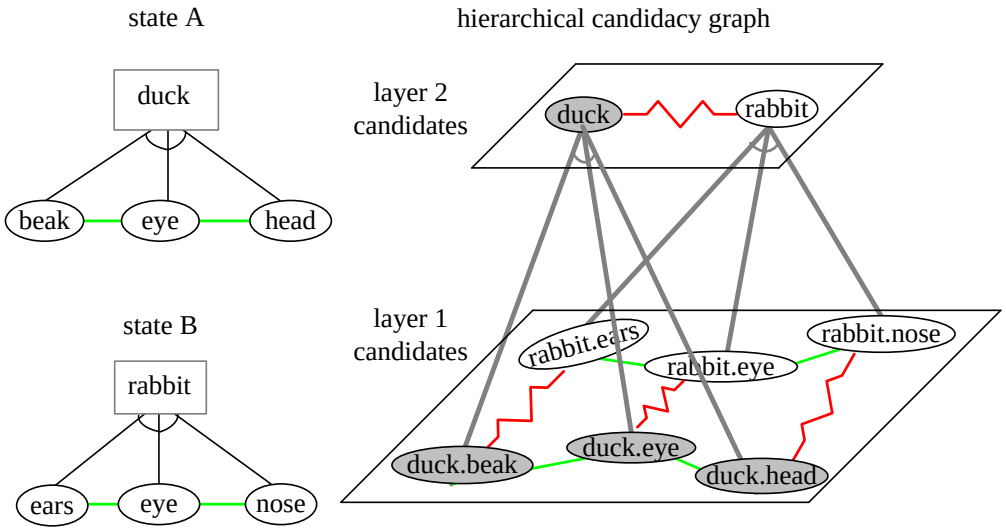


Fig. 6.37: An attempt to solve the duck/rabbit illusion using hierarchical C^4. The trees define which parts comprise each object. Nodes are grouped according to these trees, creating higher-level nodes. The higher-level nodes inherit the negative constraints. © [2011] IEEE. Reprinted, with permission, from ref. [42].

This hierarchical candidacy graph is constructed on-the-fly with nodes being generated by multiple bottom-up detection and binding processes as well as top-down prediction processes. We refer to Wu and Zhu [61] for the various bottom-up/top-down processes in object parsing. In this graph, positive and negative edges are added between nodes on the same layers in a way identical to the flat candidacy graph, while the vertical links between parent-child nodes are deterministic.

By turning on/off the positive and negative edges probabilistically at each layer, C^4 obtains *ccp*'s and *cccp*'s as in the flat candidacy graphs. In this case, a *ccp* contains a set of nodes that are coupled in both horizontal and vertical directions and thus represents a partial parse tree. A *cccp* contains multiple competing parse trees, which will

be swapped in a single step. For example, the left panel in Figure 6.37 shows two *ccp*'s for the duck and rabbit respectively which are connected with negative edges in the candidacy graph. This hierarchical representation can also eliminate the inconsistency caused by overloaded labels. That is, if a certain part is shared by multiple object or object instances, we need to create multiple instances as nodes in the hierarchical candidacy graph.

6.9.7 Experiments on hierarchical C^4

To demonstrate the advantages of hierarchical C^4 over flat C^4, we present an experiment interpreting the duck/rabbit illusion.

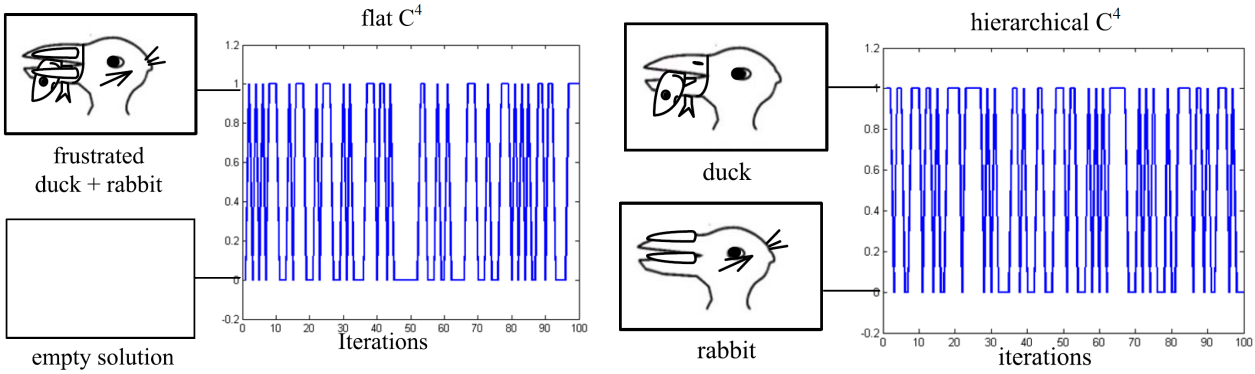


Fig. 6.38: Left: Flat C^4 results for the duck/rabbit illusion. C^4 swaps between two impossible states due to love triangles. Right: Hierarchical C^4 results for the duck/rabbit illusion. C^4 now swaps uniformly between the two correct solutions. © [2011] IEEE. Reprinted, with permission, from ref. [42].

As referenced above, C^4 on the flat candidacy graph in Figure 6.35 creates two love triangles. The left panel of Figure 6.38 shows the results of flat C^4 on the duck/rabbit illusion. C^4 continuously swaps between two states, but the two states either have all nodes on or all nodes off, neither of which are valid solutions. The right panel of Figure 6.38 shows the results of applying hierarchical C^4 to the duck/rabbit illusion. We defined a tree for the duck/rabbit illusion consisting of either a duck, $\{beak, eye, duck_{head}\}$, or a rabbit $\{ears, eye, rabbit_{head}\}$. As a result, the algorithm instantly finds both solutions and then proceeds to swap between them uniformly. These results show that hierarchical C^4 can help guide the algorithm to more robust solutions and negates the effects of love triangles.

6.10 Exercises

Problem 1. Exact sampling of the Ising/Potts model with coupled Markov Chains. We consider the Ising model in an $n \times n$ lattice ($n = 64$, or you may try $n = 128$ if you have a fast computer) with 4-nearest neighbor. X is the image defined on the lattice and the variable X_s at each site s takes values in $\{0, 1\}$. The model is

$$p(X) = \frac{1}{Z} \exp\{\beta \sum_{<s,t>} 1(X_s = X_t)\}$$

We simulate two Markov Chains with the Gibbs sampler:

- MC1 starts with all sites being 1 (call it the white chain) and its state is denoted by X^1;
- MC2 starts with all sites being 0 (call it the black chain) and its state is denoted by X^2.

At each step, the Gibbs sampler picks up a site s in both images, and calculates the conditional probabilities,

$$p(X_s^1 | X_{\partial s}^1) \text{ and } p(X_s^2 | X_{\partial s}^2)$$

It updates the variables X_s^1 and X_s^2 according to the above two conditional probabilities, and shares the same random number $r = rand[0,1]$. The two Markov Chains are said to be "coupled".
1) Prove that $X_s^1 \geq X_s^2, \forall s$ in any step. That is, the white chain is always above the black chain.
2) When the two chains meet each other, i.e. $X_s^1 = X_s^2 \forall s$ after many sweeps, they are said to "coalesce". They will stay in the same state forever as they are driven by the same random number at each step. We denote the coalescence time (number of sweeps) by τ. The images after time τ are said to be exact samples from the Ising model.

Plot the two chain states (using their total sum $\sum_s X_s^1$ and $\sum_s X_s^2$) over the sweeps as it is shown in Figure 7.8 and show the image when the two chains have coalesced. Try values $\beta = 0.6, 0.7, 0.8, 0.83, 0.84, 0.85, 0.9, 1.0$.
3) Plot the curve of τ versus β (using the parameters above) to see whether there is a critical slowing-down around $\beta = 0.84$.

Problem 2. Cluster sampling of the Ising/Potts model. For simplicity, we consider the Ising model in an $n \times n$ lattice (n is between 64 and 256) with 4-nearest neighbor. X is the image (or state) defined on the lattice and the variable X_s at each site s takes values in $\{0,1\}$. The model is

$$p(X) = \frac{1}{Z} \exp\{\beta \sum_{<s,t>} 1(X_s = X_t)\} = \frac{1}{Z} \exp\{-\beta \sum_{<s,t>} 1(X_s \neq X_t)\}$$

When $n \times n$ is large enough, we know from physics that the probability mass of $\pi(X)$ is focused on the following set uniformly with zero probability outside the set:

$$\Omega(h) = \{X : H(X) = h\}, \quad H(X) = \frac{1}{2n^2} \sum_{<s,t>} 1(X_s \neq X_t)$$

$H(X)$ is the "sufficient statistics" of X. Intuitively it measures the length of the total boundaries (cracks) in X and is normalized by the number of edges. Two images X_1 and X_2 have the same probability if $H(X_1) = H(X_2)$. Theoretically, in the absence of phase transition, there is a one-to-one correspondence between β and h, i.e $h = h(\beta)$. So, empirically, we can diagnose the convergence by monitoring whether $H(X)$ is converging to a constant value h over time.

We choose three β values: $\beta_1 = 0.6, \beta_2 = 0.8, \beta_3 = 0.84$. We have the three images X_1, X_2, X_3 at the coalescence time t_1, t_2, t_3 (sweeps). From these images we compute their sufficient statistics h_1^*, h_2^*, h_3^* respectively.

For each $\beta_i, i = 1,2,3$, we run two Markov chains using Cluster sampling.

- MC1 starts from a constant image (black or white) — $h = 0$ is the smallest; and
- MC2 starts from a checkerboard image — $h = 1$ is the largest.

Thus, when the two meet at h_i^*, we believe they have converged to $\Omega(h_i^*)$.
1) Plot the sufficient statistics $H(X)$ of the current state $X(t)$ over time t and stop when h is within an epsilon distance from h_i^* .
2) Mark the Gibbs convergence time t_1, t_2, t_3 (sweeps) in the plots for comparison between the three parameters and against the Gibbs sampler convergence in problem 1. (This comparison may be slightly unfair to Gibbs sampler as it may have converged to $\Omega(h_i^*)$ before coalescence.)
3) Plot the average sizes of the CPs (number of pixels that are flipped together at each step (sweep) and compare them at the three $\beta_i, i = 1,2,3$ settings.

Repeat the experiments using two versions of the Cluster sampling algorithm:

Version 1: Form CP's over the entire image and flip them all randomly. So each step is a sweep.

Version 2: Randomly pick up a pixel, grow a CP from it, and flip this CP only. Accumulate the number of pixels that you have flipped and divide the number by n^2 to get the sweep number.

The total number of tests is: 3 temperature $\times$ 2 initial states $\times$ 2 SW versions = 12 trials. You can also run the two MCs for $\beta_3 = 1.0$ to see fast it converges (the two MCs meet in $H(X)$).

References

1. Wilhelm Ackermann. Zum hilbertschen aufbau der reellen zahlen. *Mathematische Annalen*, 99(1):118–133, 1928.
2. Krzysztof R Apt. The essence of constraint propagation. *Theoretical computer science*, 221(1):179–210, 1999.
3. Adrian Barbu and Song-Chun Zhu. Generalizing swendsen-wang to sampling arbitrary posterior probabilities. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(8):1239–1253, 2005.
4. Adrian Barbu and Song-Chun Zhu. Generalizing swendsen–wang for image analysis. *Journal of Computational and Graphical Statistics*, 16(4), 2007.
5. Julian Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 259–302, 1986.
6. Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(11):1222–1239, 2001.
7. Simon R Broadbent and John M Hammersley. Percolation processes: I. crystals and mazes. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 53, pages 629–641. Cambridge University Press, 1957.
8. Thomas Brox and Jitendra Malik. Object segmentation by long term analysis of point trajectories. In *ECCV*, pages 282–295. 2010.
9. Haili Chui and Anand Rangarajan. A new point matching algorithm for non-rigid registration. *Computer Vision and Image Understanding*, 89(2):114–141, 2003.
10. Colin Cooper and Alan M Frieze. Mixing properties of the swendsen-wang process on classes of graphs. *Random Structures and Algorithms*, 15(3-4):242–261, 1999.
11. Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein, et al. *Introduction to algorithms*, volume 2. MIT press Cambridge, 2001.
12. L. Ding, A. Barbu, and A. Meyer-Baese. Motion segmentation by velocity clustering with estimation of subspace dimension. In *ACCV Workshop on Detection and Tracking in Challenging Environments*, 2012.
13. Liangjing Ding and Adrian Barbu. Scalable subspace clustering with application to motion segmentation. *Current Trends in Bayesian Methodology with Applications*, page 267, 2015.
14. Robert G Edwards and Alan D Sokal. Generalization of the fortuin-kasteleyn-swendsen-wang representation and monte carlo algorithm. *Physical review D*, 38(6):2009, 1988.
15. E. Elhamifar and R. Vidal. Sparse subspace clustering. In *CVPR*, 2009.
16. Pedro F Felzenszwalb and Joshua D Schwartz. Hierarchical matching of deformable shapes. In *CVPR*, pages 1–8, 2007.
17. Cornelis Marius Fortuin and Piet W Kasteleyn. On the random-cluster model: I. introduction and relation to other models. *Physica*, 57(4):536–564, 1972.
18. Michael Fredman and Michael Saks. The cell probe complexity of dynamic data structures. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 345–354, 1989.
19. Bernard A Galler and Michael J Fisher. An improved equivalence algorithm. *Communications of the ACM*, 7(5):301–303, 1964.
20. Andrew Gelman, John B Carlin, Hal S Stern, David B Dunson, Aki Vehtari, and Donald B Rubin. *Bayesian data analysis*. CRC press, 2013.
21. Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 6:721–741, 1984.
22. Walter R Gilks and Gareth O Roberts. Strategies for improving mcmc. In *Markov chain Monte Carlo in practice*, pages 89–114. Springer, 1996.
23. Vivek K Gore and Mark R Jerrum. The swendsen–wang process does not always mix rapidly. *Journal of statistical physics*, 97(1-2):67–86, 1999.
24. W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.
25. David M Higdon. Auxiliary variable methods for markov chain monte carlo with applications. *Journal of the American Statistical Association*, 93(442):585–595, 1998.
26. Mark Huber. A bounding chain for swendsen-wang. *Random Structures & Algorithms*, 22(1):43–59, 2003.
27. Scott Kirkpatrick, MP Vecchi, et al. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
28. Vladimir Kolmogorov and Carsten Rother. Minimizing nonsubmodular functions with graph cuts-a review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(7):1274–1279, 2007.
29. M Pawan Kumar and Philip HS Torr. Fast memory-efficient generalized belief propagation. In *Computer Vision–ECCV 2006*, pages 451–463. Springer, 2006.
30. Sanjiv Kumar and Martial Hebert. Man-made structure detection in natural images using a causal multiscale random field. In *CVPR*, volume 1, pages I–119. IEEE, 2003.
31. John D Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289. Morgan Kaufmann Publishers Inc., 2001.
32. F. Lauer and C. Schnörr. Spectral clustering of linear subspaces for motion segmentation. In *ICCV*, 2009.
33. Liang Lin, Kun Zeng, Xiaobai Liu, and Song-Chun Zhu. Layered graph matching by composite cluster sampling with collaborative and competitive interactions. In *CVPR*, pages 1351–1358, 2009.
34. G. Liu, Z. Lin, and Y. Yu. Robust subspace segmentation by low-rank representation. In *ICML*, 2010.
35. Jun S Liu, Wing H Wong, and Augustine Kong. Covariance structure and convergence rate of the gibbs sampler with various scans. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 157–169, 1995.

36. Jun S Liu and Ying Nian Wu. Parameter expansion for data augmentation. *Journal of the American Statistical Association*, 94(448):1264–1274, 1999.
37. Alan K Mackworth. Consistency in networks of relations. *Artificial intelligence*, 8(1):99–118, 1977.
38. AK Macworth. Interpreting pictures of polyhedral scenes. *Artificial Intelligence*, 4(2):121–137, 1973.
39. Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.
40. Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. *NIPS*, 14:849–856, 2001.
41. Judea Pearl. Heuristics. intelligent search strategies for computer problem solving. *The Addison-Wesley Series in Artificial Intelligence, Reading, Mass.: Addison-Wesley, 1985, Reprinted version*, 1, 1985.
42. Jacob Porway and Song-Chun Zhu. C^ 4: Exploring multiple solutions in graphical models by cluster sampling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(9):1713–1727, 2011.
43. Jake Porway, Qiongchen Wang, and Song Chun Zhu. A hierarchical and contextual model for aerial image parsing. *International journal of computer vision*, 88(2):254–283, 2010.
44. Renfrey Burnard Potts. Some generalized order-disorder transformations. In *Proceedings of the Cambridge Philosophical Society*, volume 48, pages 106–109, 1952.
45. James Gary Propp and David Bruce Wilson. Exact sampling with coupled markov chains and applications to statistical mechanics. *Random structures and Algorithms*, 9(1-2):223–252, 1996.
46. S. Rao, R. Tron, R. Vidal, and Y. Ma. Motion segmentation in the presence of outlying, incomplete, or corrupted trajectories. *IEEE Trans. on PAMI*, 32(10):1832–1845, 2010.
47. Azriel Rosenfeld, Robert A Hummel, and Steven W Zucker. Scene labeling by relaxation operations. *Systems, Man and Cybernetics, IEEE Transactions on*, 6:420–433, 1976.
48. Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.
49. Kokichi Sugihara. *Machine interpretation of line drawings*, volume 1. MIT press Cambridge, 1986.
50. Deqing Sun, Stefan Roth, and Michael J Black. Secrets of optical flow estimation and their principles. In *CVPR*, pages 2432–2439, 2010.
51. Robert H Swendsen and Jian-Sheng Wang. Nonuniversal critical dynamics in monte carlo simulations. *Physical Review Letters*, 58(2):86–88, 1987.
52. Martin A Tanner and Wing Hung Wong. The calculation of posterior distributions by data augmentation. *Journal of the American statistical Association*, 82(398):528–540, 1987.
53. Antonio Torralba, Kevin P Murphy, and William T Freeman. Sharing features: efficient boosting procedures for multiclass object detection. In *CVPR*, volume 2, pages II–762. IEEE, 2004.
54. Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.
55. R. Tron and R. Vidal. A benchmark for the comparison of 3-d motion segmentation algorithms. In *CVPR*, pages 1–8. IEEE, 2007.
56. Zhuowen Tu and Song-Chun Zhu. Image segmentation by data-driven markov chain monte carlo. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(5):657–673, 2002.
57. R. Vidal and R. Hartley. Motion segmentation with missing data using powerfactorization and gpca. In *CVPR*, pages II–310, 2004.
58. Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.
59. Yair Weiss. Correctness of local probability propagation in graphical models with loops. *Neural computation*, 12(1):1–41, 2000.
60. Ulli Wolff. Collective monte carlo updating for spin systems. *Physical Review Letters*, 62(4):361, 1989.
61. Tianfu Wu and Song-Chun Zhu. A numerical study of the bottom-up and top-down inference processes in and-or graphs. *International journal of computer vision*, 93(2):226–252, 2011.
62. Donghui Yan, Ling Huang, and Michael I Jordan. Fast approximate spectral clustering. In *SIGKDD*, pages 907–916, 2009.
63. J. Yan and M. Pollefeys. A general framework for motion segmentation: Independent, articulated, rigid, non-rigid, degenerate and non-degenerate. In *ECCV*, pages 94–106, 2006.
64. Song-Chun Zhu and David Mumford. *A stochastic grammar of images*. Now Publishers Inc, 2007.

Chapter 7
Convergence Analysis of MCMC

"An hour saved at a non-bottleneck is a mirage." - Eliyahu Goldratt

Introduction

One of the major issues that many practitioners run into when using MCMC is the slow convergence rate. While many MCMC methods have been shown to converge to the target distribution, the entire convergence largely depends upon the magnitude of the second largest eigenvalue of the transition matrix λ_{slem}. For this reason there are many bounds on the convergence rate of νK^n based on this quantity. In this chapter some of the most useful of these bounds are derived and implemented. These bounds are studied through the application of randomly shuffling a deck of cards. Additionally, in order to speed up the convergence process, the concepts of trade map, bottleneck, and conductance are explained. Finally, the topics of path coupling and exact sampling are covered and an application of these methods to the Ising model is included.

7.1 Key Convergence Topics

Let (ν, K, Ω) be a Markov chain with initial distribution ν, transition kernel K in the space Ω. The samples obtained by this chain at some time n follow the distribution $X(t) \sim \nu \cdot K^n \xrightarrow[n]{\infty} \pi$. The convergence of νK^n is measured using total variation $\|\nu K^n - \pi\|_{TV}$. If this quantity approaches 0 as $n \to \infty$, then the chain converges to π.

$$K^n = \sum_{i=1}^{n} \lambda_i v_i u_i$$

Recall these useful qualities of a chain, defined in Section 3.4:

i) First hitting time of a state i (in the finite state case)

$$\tau_{\text{hit}}(i) = \inf\{n \geqslant 1; x_n = i, x_0 \sim \nu_0\}, \qquad \forall i \in \Omega$$

ii) First return time of a state i

$$\tau_{\text{ret}}(i) = \inf\{n \geqslant 1; x_n = i, x_0 = i\}, \qquad \forall i \in \Omega$$

iii) Mixing time

$$\tau_{\text{mix}}(i) = \min_n\{\|\nu K^n - \pi\|_{\text{TV}} \leqslant \varepsilon, \forall \nu_0\}$$

We can also define the following concepts to characterize a chain.

Definition 7.1. The burn-in period is the expected number of steps until the Markov chain enters the subspace of typical states. The subspace of typical states is a subspace of Ω where $\pi(x)$ is concentrated.

The burn-in notion is not very precise since it is hard to estimate when the distribution of the Markov chain νK^n is sufficiently close to the target distribution π. This is especially true in high dimensional spaces.

Definition 7.2. The auto-correlation between the states of the Markov chain $\mathbf{x} = (x_0, ...,)$ is defined to be

$$\text{Corr}(\tau) = \frac{1}{T} \sum_{t=t_0+1}^{t_0+T} (x_t - \bar{\mathbf{x}})(x_{t+\tau} - \bar{\mathbf{x}}), \qquad \forall \tau \geq 0.$$

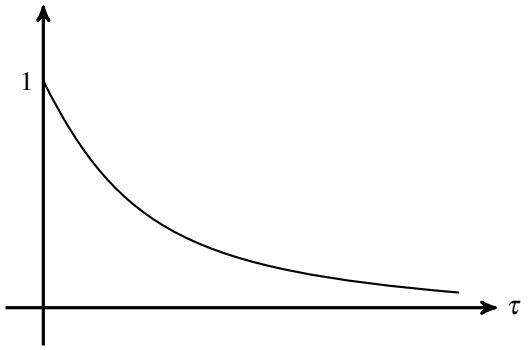


Fig. 7.1: Usually the auto-correlation between samples decreases with the lag τ.

High auto-correlation implies slow convergence, and low auto-correlation implies fast convergence. We can use the MC samples to integrate and obtain

$$\theta = \int f(x)\pi(x)\mathrm{d}x \cong \frac{1}{T} \sum_{t=t_0+1}^{t+T} f(x_t) = \hat{\theta},$$

$$\text{var}(\hat{\theta}) = E_{\text{samples}}\left[(\hat{\theta} - \theta)^2\right] = \frac{1}{m} \cdot \text{const},$$

where m is the effective number of independent samples.

7.2 Practical Methods for Monitoring

Deterministic algorithms converge to a point and the convergence to that point can usually be monitored. For example in Maximum Likelihood Estimation (MLE) we can check the likelihood function $f(x)$ to see if the algorithm has converged. In contrast, MCMC algorithms are stochastic, and it is hard to determine whether convergence has occurred or not. However, there are several methods that allow us to monitor the process of convergence, including

1. Monitoring sufficient statistics of $\pi(x)$, such as the boundary length, total energy, etc. The statistics are averaged over space (perturbations of a sample) or time (number of samples). This approach projects the state space Ω into the space of the sufficient statistics H. For any particular value of H, we have the inverse space

$$\Omega_{(h)} = \{x \in \Omega : H(x) = h\}.$$

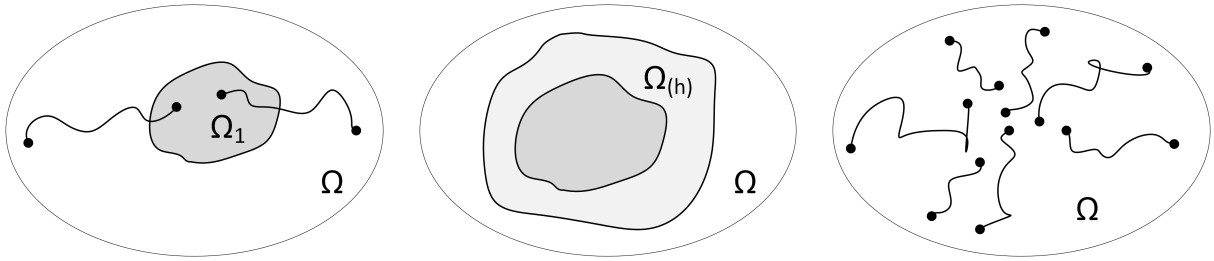


Fig. 7.2: Left: Different paths arrive in a high probability region Ω_1 after a burn-in period. Middle: A sufficient statistic H (e.g. energy) gives us inverse spaces (level sets) $\Omega_{(h)}$ for the different values h taken. Right: Running multiple Markov chains allows a better exploration of the space Ω.

2. Running many Markov chains in parallel, starting from widely dispersed initial states, and, if possible, the extreme states. Monitoring as in 1. can be done concurrently. For example, in the Ising/Potts Model, we start from a constant 0/1 (white/black) or white noise grid.

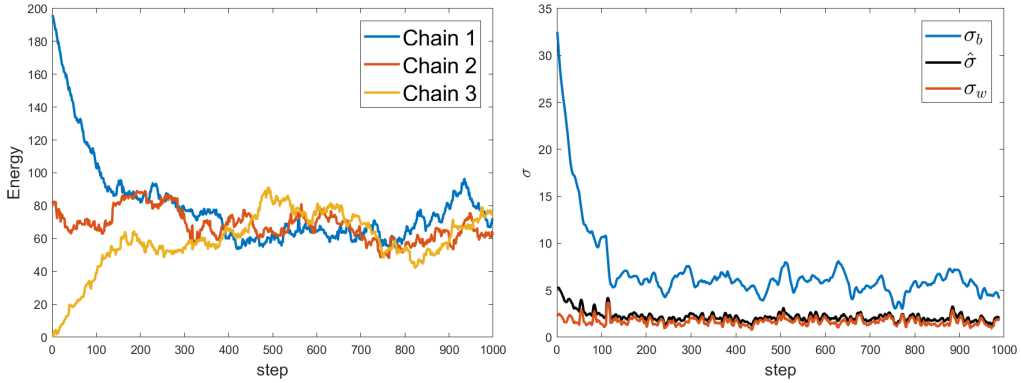


Fig. 7.3: Left: monitoring that the Markov chain has forgotten the initial points, by inspecting the sufficient statistic H, e.g. energy. Right: the σ_{w}, and σ_{b} and $\hat{\sigma}$ approximate the true σ.

3. Monitoring whether the Markov chain(s) have forgotten the past or initial points, as illustrated in Figure 7.3, left.
4. Monitoring the sampling temperature, T. Through this approach we can monitor the rejection rate of the Metropolis-Hastings algorithm as well as the entropy of $\pi(x_i|x_{-i})$ in the Gibbs sampler.
5. Simulating M different Markov chain sequences, $\{\mathrm{MC}_i; i = 1,2,\cdots,M\}$. We can compute the individual means ψ_i, and the total mean of all of the chains, $\overline{\psi}$. Thus, we have the between-chain variance

$$\sigma_{\mathrm{b}}^2 = \frac{1}{M-1}\sum_{i=1}^{M}(\psi_i - \overline{\psi})^2$$

and the within-chain variance

$$\sigma_{\mathrm{w}}^2 = \frac{1}{M}\sum_{i=1}^{M}\sigma_i^2,\ \sigma_i^2 = \frac{1}{T}\sum_{i=t_0}^{t_0+T}\Big(x_i(t) - \psi_i\Big)^2,$$

which underestimates the true variance σ.
We can then estimate the variance of the Markov chain as

$$\hat{\sigma} = \frac{T-1}{T}\sigma_{\mathrm{w}} + \frac{1}{T}\sigma_{\mathrm{b}}.$$

These quantities are illustrated in Figure 7.3, right.

7.3 Coupling Methods for Card shuffling

Shuffling a deck of cards can also be represented by a Markov chain. We can use card shuffling to study coupling methods for Markov chains, where two or more chains start independently and slowly coalesce (move identically) after a number of steps.

Suppose we have a deck with $n = 52$ cards. We can use Markov chains to answer some questions such as: When are the cards thoroughly shuffled? Are all cards randomly distributed? There are three ways to understand these problem.

1. Convergence is relative to a procedure, such as a shuffling process, since after each shuffling we get a new order. By repeating this process N times, we get N decks (arrangements), and can answer the questions by
   a) Testing the distribution of the cards that appear at a given location, i, and comparing it to a uniform distribution of cards at i.
   b) Tracking the card that started at position i, and check its position distribution.
2. Check whether the new deck has forgotten its history, so players cannot cheat by memorizing the original order.
3. Monitor some marginal statistics between the cards, so that players cannot predict the next card based on the ones that have been played.

There are many ways to shuffle the cards, two of which are presented here.

7.3.1 Shuffling to the top

Shuffling to the top is a simple method that is easy to study theoretically. At each step a card i is selected at random and placed on the top of the deck. After many moves the deck will be completely random. To couple the Markov chain associated with deck 1 and the Markov chain associated with deck 2, we find the top card of deck 1 in deck 2, and place it on top. Thus, the top cards are identical between decks 1 and 2. This procedure is repeated until all 52 cards are picked at least once. This is also known as the "coupon collector's problem". After some time T, the two decks will be identical, and are said to have coalesced. The coalescence time T has the following characteristics:

$$E[T] = n(\frac{1}{n} + \frac{1}{n-1} + \cdots + 1) \cong n\log n;$$
$$\text{var}[T] \cong 0.72n.$$

Remark 7.1 *At each step we have to select among all n cards, so that the shuffling on deck 1 is unbiased, otherwise deck 2 it is no longer random.*

Remark 7.2 *Both decks are coupled by the same card i at each move.*

7.3.2 Riffle shuffling

For riffle shuffling we divide the 52 cards into two decks according to Binomial$(n, \frac{1}{2})$, then we shuffle them. This way, a number of cards k are in deck 1 and $n-k$ are in deck 2. The number k follows the distribution

$$K \sim P(k) = \frac{1}{2^n}\binom{n}{k} = \frac{1}{2^n} \cdot \frac{n!}{k!(n-k)!}.$$

This can be better understood as an inverse shuffling process, similar to rewinding a video. This means we pick randomly the cards that belong to deck 1, then we place them at the top.

For that, at each shuffling, we simulate a binary bit for each card, $b_1, b_2, \cdots, b_n \sim$Bernoulli$(\frac{1}{2})$, n times in total, as illustrated in Figure 7.4. Then we go back and put all 0's on top of all 1's. After t shuffling operations, each card i from the original order is associated with t bits of code $x_i = b_{1i}b_{2i}\cdots b_{ti}$, as illustrated in Figure 7.5.

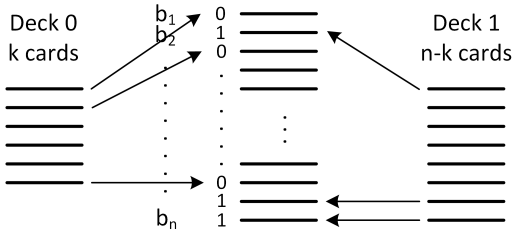


Fig. 7.4: Illustration of rifle shuffling

When not enough shuffles are done, there exist cards with identical binary codes $x_i = x_j$, as illustrated in Figure 7.5. In that case the order between these cards is the same as the order in the original deck.

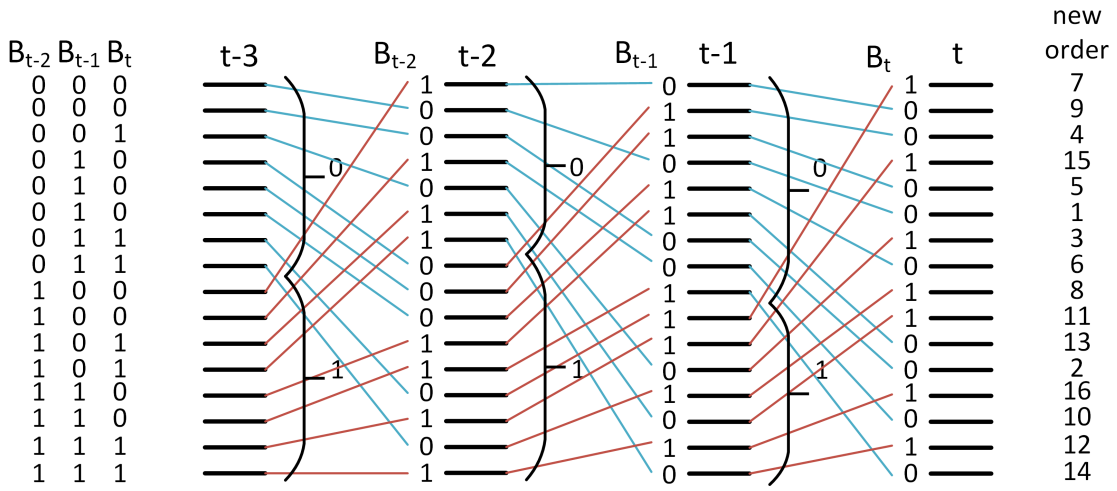


Fig. 7.5: Example of rifle shuffling and the coding bits.

However, when t is large enough so that all $\{x_i\}$ are distinct, then the order of the cards corresponds to the values of x_i, in the sense that the card that has a 0 last is above the other card in the deck. The ordering after time t is then completely random, since the order is dictated only by the bits x_i.

Calculating the number t such that all $\{x_i\}$ are distinct is related to a classic statistical problem: What is the the probability of dropping n balls in 2^t bins, so that each bin has at most 1 ball? The bins correspond to the 2^t bit combinations $(b_1, b_2, \cdots, b_t)$ without repetition, and the balls correspond to the n cards.

Example 7.1. To fix ideas consider that we have 520 decks of cards that start off completely sorted in the same way, with each card numbered 1 to 52 from top to bottom. For each deck we independently sample 52 values from Bernoulli(.5). To shuffle the cards we count the number of 0's in the vector of 52 values and take that many cards from the top of the deck. We then sequentially place them in a "new" deck such that they are positioned exactly where each of the 0's was drawn. The remaining cards are similarly placed where the 1's were drawn.

In order to test how well this method works for creating randomly shuffled decks of cards, the process is repeated t times for each of the 520 decks. After each shuffle we create 52 histograms, one for each position in the deck. In context this is equivalent to lining up all 520 decks and flipping over the top card from all of the decks before sorting them by their associated number and stacking them to get the distribution of cards at that position. Ideally we'd like to see that each of these 52 distributions is roughly uniform to represent a perfectly shuffled deck. The TV norm is used to measure how close to uniform the distributions are and the error is given by

$$err(t) = \frac{1}{52}\sum_{i=1}^{52} ||H_{t,i} - \frac{1}{52}||_{TV}.$$

In this expression $H_{t,i}$ is the probability distribution at time t for deck position i. Each iteration is expected to shrink the average error across all positions, but it is unlikely that it will converge to 0.

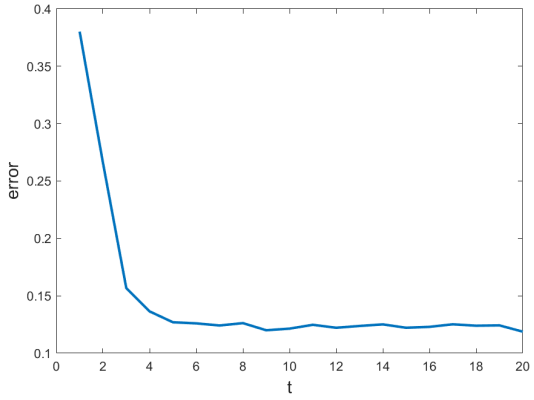


Fig. 7.6: Departure from a uniform distribution as the TV norm difference after t riffle shuffles.

As shown in Figure 7.6 above, after only 5 shuffles the error decreases to .12 and stabilizes. The recommended number of traditional riffle shuffles to obtain a random deck is between 7 and 11. However, in practice a riffle shuffle may not be exactly random, as compared to the certainly random Bernoulli process used above.

7.4 Geometric Bounds, Bottleneck and Conductance

In this section, we introduce more key concepts about MCMC convergence rates.

7.4.1 Geometric convergence

Let (ν, K, Ω) be a Markov chain. If K is irreducible and aperiodic, then

$$\|\nu \cdot K^n - \pi\|_{\mathrm{TV}} \leqslant c \cdot r^n,$$

where c is constant and $0 < r < 1$ is the geometric rate. For such a Markov chain, there exists $n_0 > 0$ such that

$$K^{n_0}(x,y) > 0, \qquad \forall x, y \in \Omega.$$

There are many methods for proving this, since r is not specified.

i) Using the contraction coefficient. The contraction coefficient for K is the maximum TV-norm between any two rows in the transition kernel and is calculated by

$$C(K) = \max_{x,y} \|K(x,\cdot) - K(y,\cdot)\|_{\mathrm{TV}}.$$

Then for any two probabilities ν_1, ν_2 and a function h defined on x such that $Kh = \sum_y h(y)K(x,y)$ the following holds:

$$\|\nu_1 K - \nu_2 K\|_{\mathrm{TV}} = \max\{\|(\nu_1 K)h - (\nu_2 K)h\| : \|h\| \leq 1\} = \max\{\|\nu_1(Kh) - \nu_2(Kh)\| : \|h\| \leq 1\} \tag{7.1}$$

$$\leq \max\{\frac{1}{2}\max_{x,y}\|Kh(x) - Kh(y)\| : \|h\| \leq 1\}\|\nu_1 - \nu_2\| \tag{7.2}$$

$$= \frac{1}{2}\max_{x,y}\max\{\|Kh(x)-Kh(y)\| : \|h\| \leq 1\}\|\nu_1-\nu_2\| = c(K)\|\nu_1-\nu_2\|. \tag{7.3}$$

Further details of this proof can be found in [3].

ii) Using Perron-Frobenius Theorem. Recall Theorem 3.1:

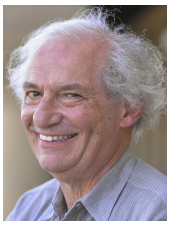
Theorem 3 (Perron-Frobenius Theorem) *For any primitive (irreducible and aperiodic) $N \times N$ stochastic matrix K, K has eigenvalues $1 = \lambda_1 > |\lambda_2| > \cdots > |\lambda_r|$ with multiplicities $m_1, ..., m_r$ and left and right eigenvectors $(\mathbf{u}_i, \mathbf{v}_i)$ respectively. Then $\mathbf{u}_1 = \pi, \mathbf{v}_1 = \mathbf{1}$, and*

$$K^n = \mathbf{1}\cdot\pi' + O(n^{m_2-1}|\lambda_2|^n).$$

From this result we have the bound on each starting state $\mathbf{x}_0 \in \Omega$

$$\|\nu K^n - \pi\|_{TV} \leqslant \sqrt{\frac{1-\pi(\mathbf{x}_0)}{4\pi(\mathbf{x}_0)}}\cdot\lambda_{\text{slem}}^n$$

This bound is known as the Diaconis-Hanlon bound [1].


Persi Diaconis

Now we need to analyze what factors bound λ_{slem}. λ_{slem} is often related to the feature of a chain that most impedes the process of convergence, i.e., the worst case scenario. This could be a state, vertex, or edge. There are several important concepts related to the understanding of λ_{slem}.

Trade map (transition graph). Returning to the analogy of a Markov chain kernel K as a trade between people (as in Example 3.2), we define a graph $G =< V, E >$ where $V = \Omega$ is the finite set of states, and $E = \{< x, y >; x, y \in V, K(x,y) > 0\}$ is the set of edges. Each edge $e =< x, y >$ is weighted by $Q(e) = \pi(x)\cdot K(x,y)$.

Several properties of this map can be used to diagnose convergence. By irreducibility, $\forall x \neq y \in \Omega$, x and y are connected through many paths. We define a weighted path $\Gamma_{xy} \stackrel{\text{def}}{=} (x, \cdots, y)$ and further require that this path contains each edge at most one time. Under this constraint we say that Γ_{xy} has effective length given by

$$\gamma_{xy} \stackrel{\text{def}}{=} \sum_{<s,t>\in\Gamma_{xy}} \frac{1}{\pi(s)K(s,t)}.$$

Thus, a lower probability to go from x to y will imply a greater effective length, and a long waiting time is required to go from x to y.

Bottleneck. The bottleneck measure of G is indicative of the graph's overall connectivity and is given by

$$\kappa = \max_{e\in E}\sum_{\Gamma_{xy}\ni e}\gamma_{xy}\cdot\pi(x)\pi(y),$$

where the sum is taken over all valid paths (regardless of where they start or end) that contain the edge e. The bottleneck itself is the edge e^* that yields the maximal value. Intuitively, e^* can be thought of as the Golden Gate Bridge or the Panama Canal of the graph. Two densely populated cities/bodies of water/nodes are connected by one, high-traffic pathway.

With the process for calculating the bottleneck of a graph established, there is a result that gives bounds on the convergence based on κ. The Poincaré inequality implies that

$$\lambda_{\text{slem}} \leqslant 1 - \kappa^{-1}. \tag{7.4}$$

Example 7.2. To fix ideas, we consider the example of finding the bottleneck in the case of the five families trading on an island. Suppose we believe that the bottleneck is the edge (3,2). There are many paths that contain the edge (3, 2). There is the one-step path, six two-step paths, and twenty-one three-step paths. Of these, only (2, 3, 2, 3) is not allowed because it repeats an edge. There are also four-step paths, five-step paths, and so forth. As we can see, the optimal edge has over 10,000 paths associated with it.

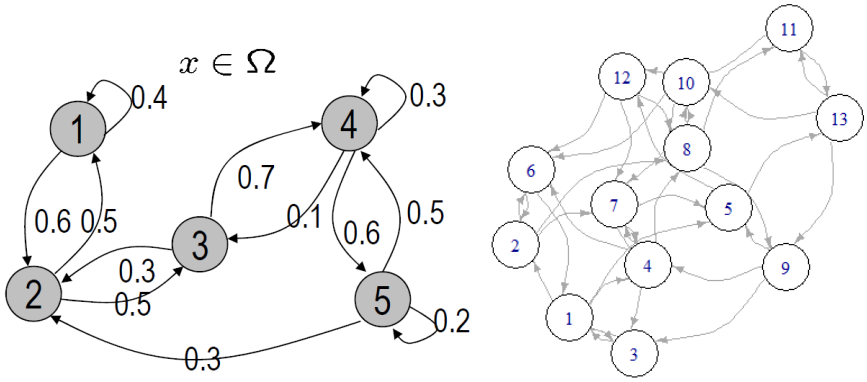


Fig. 7.7: Left: Trade diagram for the five families. Right: Unweighted, directed graph with a vertex set equivalent to the edge set of the five families graph.

To reduce the complexity of this problem, we instead calculate κ as follows. Let G be the unweighted version of the directed graph. That is, G is represented in matrix form by turning all of the nonzero $K(x,y)$ components to 1. Now convert the unweighted (directed) graph G into a new (unweighted and directed) graph G_0 according to the rule that the edge sets of G are exactly the vertices of G_0. Thus, v_1 and v_2 are edges in G_0 if and only if $v_1 = (s_1,t_1)$ and $v_2 = (s_2,t_2)$ for some vertices s_1,s_2,t_1 and t_2 of G. Finally, construct G_0 such that there is a directed edge from v_1 to v_2 if and only if $t_1 = s_2$. This is illustrated in Figure 7.7,right.

Now, consider all simple paths on this new graph G_0. A simple path is one that touches any vertex at most once. Thus, a simple path on G_0 traces a feasible path along the edges of G using each edge of G at most once. So we can easily search G_0 to find all simple paths and then throw away the ones that do not have the edge we are considering, calculate $\gamma_{x,y}\pi(x)\pi(y)$ and sum over the remaining paths. Repeat this for each vertex in G_0 (which is exactly each possible argument we are maximizing over) and choose the vertex in that maximizes the sum.

Using this algorithm we obtain the bottleneck measure $\kappa = 90508.08$ attained for edge $< 3,4 >$. Since $\lambda_{\text{slem}} = |.5443 + .1824i| = .57405$ we observe that the Poincaré inequality $\lambda_{\text{slem}} \leqslant 1 - \kappa^{-1}$ is satisfied.

Conductance. Suppose we divide the state space Ω into two subspaces such that $\Omega = S \cup S^c$. We then define the transition probability between the subspaces to be

$$K(S,S^c) =: \sum_{x\in S}\sum_{y\in S^c} K(x,y).$$

Let $\pi(S) = \sum_{x\in S}\pi(x)$ be the capacity of S and define

$$Q(S,S^c) =: \sum_{\substack{x\in S\\ y\in S^c}} \pi(x)\cdot K(x,y)$$

as the flow out of S. The conductance of G is then given by

$$h =: \min_{S:\pi(S)\leqslant\frac{1}{2}} \frac{Q(S,S^c)}{\pi(S)}.$$

If the conductance is small, then there exists an S with large $\pi(S)$ but small $Q(S,S^c)$. This definition of conductance yields Cheeger's inequality [2], which states

$$1 - 2h \leqslant \lambda_{\text{slem}} \leqslant 1 - \frac{h^2}{2}.$$

These bounds are intuitive, but do not really guide the design in practice. In practice, heuristics are used such as Data-driven MCMC (Chapter 8) or the SW cut (Chapter 6) algorithm to speed up the Markov chain.

7.5 Peskun's Ordering and Ergodicity Theorem

Now, we return to our early motivation for designing MCMC. Recall the Ergodicity Theorem from Section 3.6. Remember that this allows us to perform Monte Carlo integration to calculate the parameter θ as

$$\theta = \int f(x)\pi(x)\mathrm{d}x \cong \hat{\theta} = \frac{1}{n}\sum_{t=1}^{n} f(x^{(t)}),$$

by using samples $\{x^{(1)}, \cdots, x^{(n)}\} \sim \pi(X)$ obtained from MCMC. The efficiency of a Markov chain is ultimately measured by the variance

$$\mathrm{var}(\hat{\theta}) = \lim_{n\to\infty} \frac{1}{n}\mathrm{var}\Big\{\sum_{t=1}^{n} f(x^{(t)})\Big\}.$$

Suppose that two Markov kernels K_1 and K_2 have the same invariant probability π. We introduce a partial order among all such K's that are in the set

$$\Omega_\pi = \{K : \pi K = \pi,\ K \text{ irreducible and aperiodic}\}.$$

We say that K_1 dominates K_2, and write $K_1 \succeq K_2$, if $K_1(x,y) \geqslant K_2(x,y)$, $\forall x \neq y$.

Theorem 7.1 (Peskun) *If $K_1 \succeq K_2$, then* $\mathrm{var}(\hat{\theta}_1) \leqslant \mathrm{var}(\hat{\theta}_2)$.

Example 7.3. Consider the following two Markov chains:

MC1: $K_1(x,y) = Q(x,y)\cdot\alpha(x,y) = Q(x,y)\cdot\min\Big(1, \dfrac{Q(y,x)\pi(y)}{Q(x,y)\pi(x)}\Big)$ – Metropolis-Hastings design.

MC2: $K_2(x,y) = Q(x,y)\cdot\dfrac{\pi(y)Q(y,x)}{\pi(x)Q(x,y)+\pi(y)Q(y,x)}$ – Baker's design.

One can prove that $K_1 \succeq K_2$.

Example 7.4. Metropolized Gibbs sampler $\succeq$ Gibbs sampler.

7.6 Path Coupling and Exact Sampling

Example 7.5. We consider a $n \times n$ lattice and its graph consisting of the 4-nearest neighbors. We use the Ising model in its original physics context where it models magnetic material on a lattice of charged particles. Each particle could have one of two possible spin states, -1 (down) or 1 (up). Let X be the spin states of the lattice, so the variable X_s at each site s is the spin state, taking values in $\{-1, 1\}$. The model for spin interaction assigns positive energy to spins in opposite directions. Formally, the energy of the system is given by

$$H(X) = -\sum_{<s,t>\in C} \beta X_s X_t,$$

where C are all the 4-nearest neighbors of the lattice and β is interaction strength. This induces a probability measure for each possible state of the lattice:

$$P(X) = \frac{1}{Z}\exp^{-H(X)}.$$

We can simulate two Markov Chains with the Gibbs sampler:

1. The white chain starts with all sites being 1, and its state is denoted by X^1;
2. The black chain starts with all sites being -1, and its state is denoted by X^2.

At each step, the Gibbs sampler picks a site s in both images and calculates the conditional probabilities, $p(X_s^1|X_{\partial s}^1)$ and $p(X_s^2|X_{\partial s}^2)$. It updates the variables X_s^1 and X_s^2 according to the above two conditional probabilities and uses the

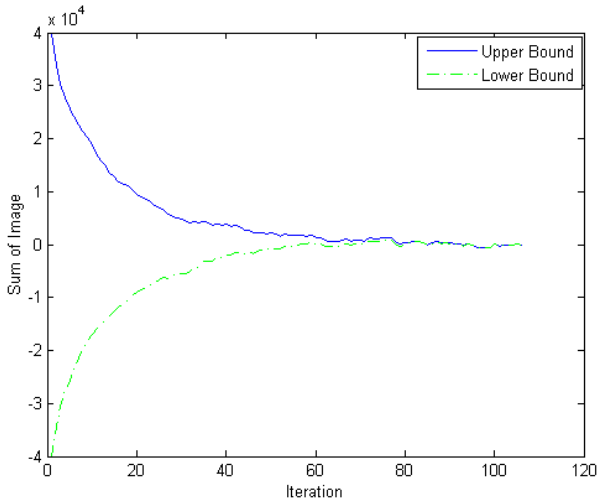


Fig. 7.8: The total magnetization $\sum_s X_s$ on the Ising model with $\beta = 0.35$ for the white chain and black chain, which coalesced at $\tau = 105$.

same random number $r \in [0,1]$ to sample the two variables. Under this process the two Markov chains are said to be coupled.

It can be shown that $X_s^1 \geq X_s^2, \forall s$ in any step. That is, the white chain always has a greater sum than the black chain. An example for $\beta = 0.35$ is shown in Figure 7.8.

Coalescence. When the two chains meet each other, i.e. $X_s^1 = X_s^2, \forall s$ after many sweeps, they are said to have coalesced. They will stay in the same state forever as they are equal and driven by the same random number at each step. We denote the coalescence time (in sweeps) by τ. The images after τ sweeps are exact samples from the Ising model.

7.6.1 Coupling from the past

One of the main concepts from exact sampling is coupling from the past. The idea is to run the simulation backwards in time from each state, keeping track of the states where each chain ends up. It is intuitive that once two states map to a single state after a simulation from time $-t$ to time 0, they will remain the same for simulations from $-t-1$ if the same random numbers are used. Coupling from the past ensures that after a finite number of rounds of simulation M, the measure $\rho(i)$ of the state i we end up with is sufficiently close to the equilibrium distribution of the chain $\pi(i)$, i.e. $||\rho(i) - \pi(i)|| < \varepsilon$. The output of a fixed-time backwards simulation is given by $F_{-M}^0(i)$, where $F_{t_1}^{t_2}$ is defined as the composition $f_{t_2-1} \circ f_{t_2-2} \circ \cdots \circ f_{t_1+1} \circ f_{t_1}$. Several important features of this output include:

1. Each $f_t(i)$ maps the state space to itself, with $-M \leq t \leq 1$.
2. F_t^0 is updated via $F_t^0 = F_{t+1}^0 \circ f_t$.
3. Coalescence is at the time point when F_t^0 becomes a constant map, $F_t^0(i) = F_t^0(i'), \forall i, i'$.

Theorem 7.2 *With probability 1 the coupling from the past procedure returns a value that is distributed according to the stationary distribution of the Markov chain.*

Example 7.6. Consider the four-state Markov chain shown in Figure 7.9. We simulate all of the states from the past. Coalescence happened after 5 iterations of simulation, as seen in Figure 7.10.

The state space S has a natural partial ordering, such that

$$x \leq y \Rightarrow \phi(x,u) \leq \phi(y,u),$$

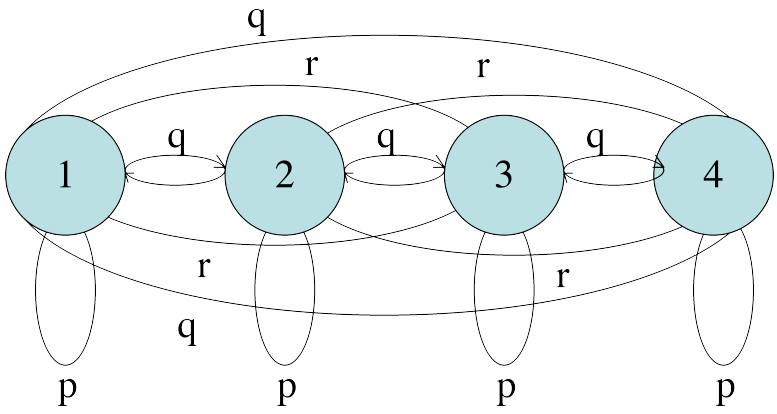


Fig. 7.9: A Markov chain with four states, where $p = \frac{1}{3}$, $q = \frac{1}{4}$, $r = 1 - p - 2q$.

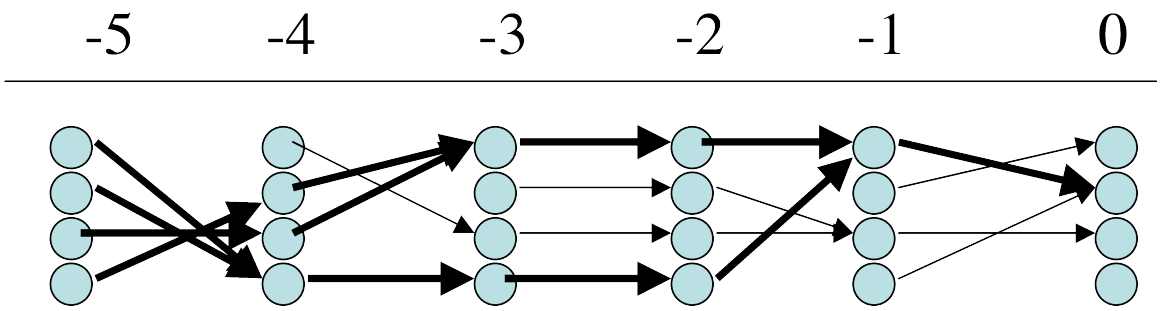


Fig. 7.10: Coupling from the past for the Markov chain from Figure 7.9. Coalescence happens after 5 iterations of simulation.

where ϕ is the update rule, and u is the source of randomness. The coalescence can be verified by tracking the maximal and minimal elements of S.

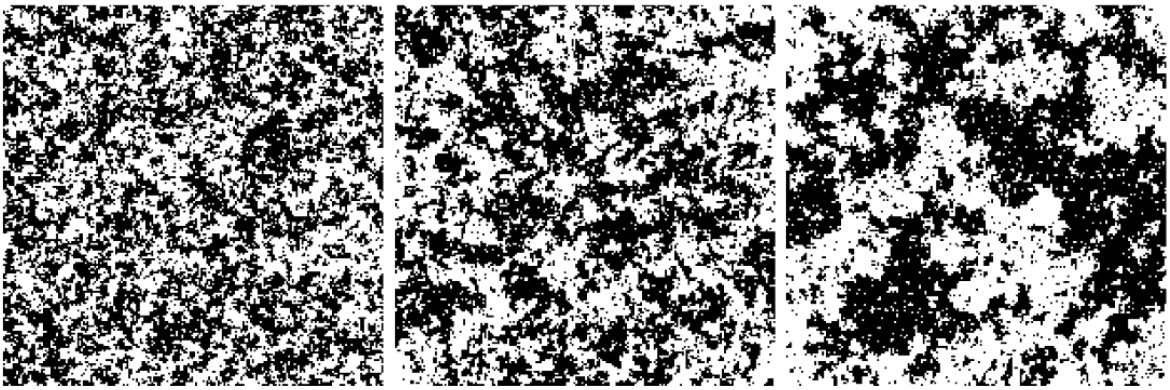
7.6.2 Application: sampling the Ising model

We now return to the Ising model from example 7.5. It is not trivial to sample the Ising model because of its high dimensionality, so we use a Gibbs sampler to update the chain based on the conditional distribution of each particular spin of the lattice, $P(s/\partial_s)$, where ∂_s is the neighbor system of s. It is very easy to sample directly from this distribution.

It has been shown that if a deterministic (or semi-deterministic) scheme for updating all of the points in the lattice with the Gibbs sampler is employed, the induced Markov chain will converge to the joint distribution for the lattice, $P(I)$. In Figure 7.11 are shown samples from the Ising model with different values of β at coalescence. Below each image is shown the value of β and the coalescence time τ. In Figure 7.8 is shown the total magnetization $\sum_s X_s$ for the MC1 (white chain) and MC2 (black chain), for $\beta = 0.35$. The energy at coalescence can vary for different random sequences, as shown in Figure 7.12. Figure 7.13 shows the correlation between the states at each time i and time $i+t$ given by

$$R(t) = \frac{1}{N}\sum_{i=1}^{N} < X^{(i)}, X^{(i+t)} > .$$

If an external field X^{obs} is added to the model, the potential $H(X)$ becomes



$\beta = 0.35, \tau = 105$ $\beta = 0.40, \tau = 780$ $\beta = 0.43, \tau = 5265$

Fig. 7.11: Samples of 2D Ising model at different temperatures. Lattice size: 200×200.

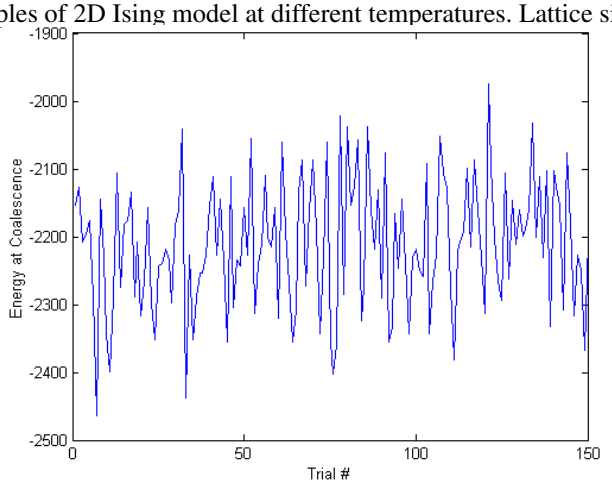


Fig. 7.12: The energy at coalescence of 150 trials. $\beta = 0.40$, lattice size: 50×50.

$$H(X) = -\sum_{<s,t>\in C} \alpha X_s X_t - \sum_{<s>\in I} X_s X_s^{obs}.$$

The Ising model with external field can be used for image denoising, by using the observed noisy image as the external field X^{obs} and the sampled image X as the denoised image. Figure 7.14 shows a sampled image X obained by coupling from the past with the external field shown in top left, and different values of the interaction strength parameter β. The total magnetization for the upper and lower bound chains for $\beta = 1$ is shown in Figure 7.15.

———-

7.7 Exercises

Problem 1. Consider the Markov kernel for the five families living in a pacific island, with slightly changed numbers.

$$K = \begin{pmatrix} 0.1, & 0.8, & 0.1, & 0.0, & 0.0 \\ 0.3 & 0.0, & 0.7, & 0.0, & 0.0 \\ 0.1, & 0.6, & 0.0, & 0.3, & 0.0 \\ 0.0, & 0.0, & 0.1, & 0.6, & 0.3 \\ 0.0, & 0.0, & 0.2, & 0.4, & 0.4 \end{pmatrix}$$

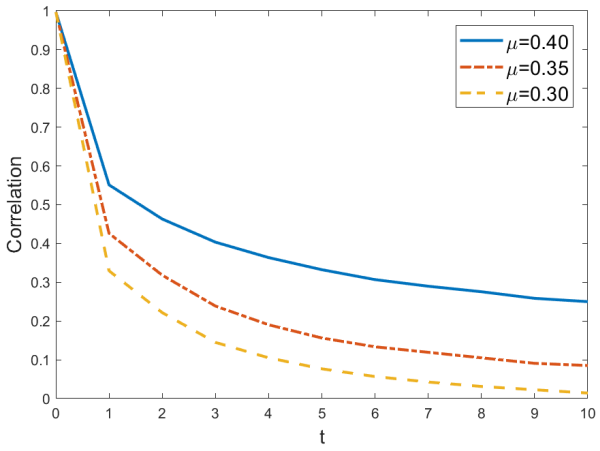


Fig. 7.13: Correlation of equilibrium states at different temperatures.

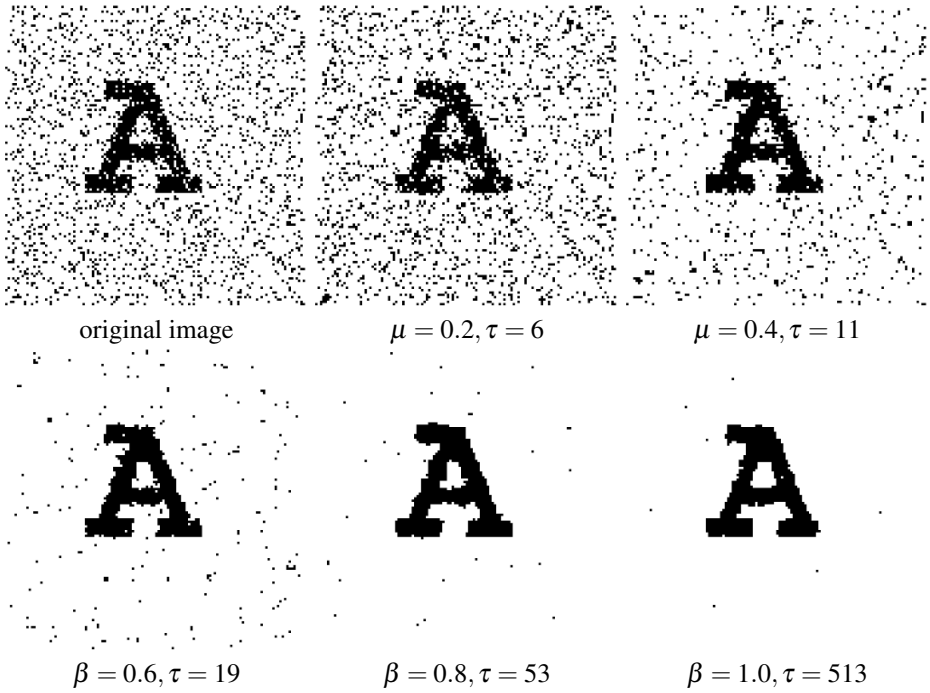


Fig. 7.14: Simulation of the coupled Markov chains with external field for noise reduction.

This transition matrix defines a directed graph $G =< V, E >$ where $V = \{1,2,3,4,5\}$ is the set of states, and $E = \{e = (x,y) : K(x,y) > 0\}$ is a set of directed edges.

1. Draw the graph G and calculate the invariant probability $\pi(x)$ for the five states $x \in \{1,2,3,4,5\}$; and compute the value of λ_{slem}.
   Now we'll try to verify the bounds of λ_{slem} by the following two concepts that we studied in this chapter – bottleneck and conductance.
2. Which edge $e = (x,y)$ is the bottleneck of G? (you may make a guess based on the graph connectivity first, and then calculate by its definition); and calculate the Bottleneck κ of the graph G. Verify the Poincare inequality:

$$\lambda_{\text{slem}} \leq 1 - \frac{1}{\kappa}.$$

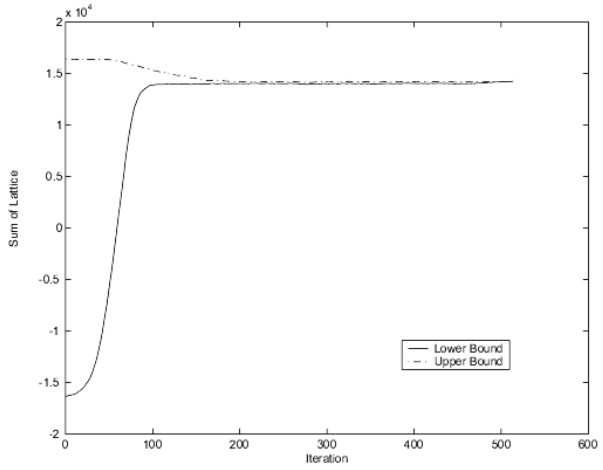


Fig. 7.15: The upper bound and lower bound coalesced at $\tau = 513$, $\beta = 1$.

3. Calculate the conductance h of the graph G. Verify Cheeger's inequality:

$$1 - 2h \leq \lambda_{\text{slem}} \leq 1 - \frac{h^2}{2}.$$

4. Now since we know π, we can design the "dream" matrix K^* that converges in one step. Then $\lambda_{\text{slem}} = 0$ for K^*. Rerun your code above to calculate the conductance h for K^*. Verify the Cheeger's inequalities.

Problem 2. In the riffle shuffling of cards, we mentioned two bounds: 7 and 11 as the expected number of shuffles to make the 52 cards random. Before proving bounds, it is often a good idea to empirically plot the convergence curve.

Suppose we label the 52 cards as 1,2,...,52 and start with a deck (or state) X_0 which is sorted from 1 to 52. Then we simulate the following riffle shuffling process iteratively from X_{t-1} to X_t.

Simulate 52 independent Bernoulli trials with probability 1/2 to be 0 or 1. Thus we obtain a binary vector 0,1,1,0....0. Suppose there are n zero's and 52-n one's. We take the top n cards from deck X_{t-1} and sequentially put them in the positions of the zero's and the remaining 52-n cards are sequentially put in the positions of one's.

Now, let's check whether the deck X_t is random as t increases. You may design your own methods to test randomness. Below is a default method if you don't have better ideas.

We always start with a sorted deck X_0 and repeat the shuffling process K times. Thus at each time t we record a population of K decks:

$$\{X_t^k : k = 1, 2, ..., K\}$$

For each card position $i = 1, 2..., 52$, we calculate the histogram (marginal distribution) of the K cards at position i in the K decks. Denote it by $H_{t,i}$ and normalize it to 1. This histogram has 52 bins, so we may choose $K = 52 \times 10$. Then we compare this 52-bin histogram to a uniform distribution by TV-norm and average them over the 52 positions as a measure of randomness at time t

$$\text{err}(t) = \frac{1}{52} \sum_{i=1}^{52} ||H_{t,i} - \text{uniform}||_{\text{TV}}.$$

Plot $\text{err}(t)$ over time t to verify the convergence steps. Based on your plot, how many times do we really need to shuffle the cards?

Problem 3. In a finite state space Ω, suppose at step t a Markov chain MC has state X following a probability ν. By applying the Markov kernel P once, its state in $t+1$ is Y which follows a probability $\mu = \nu \cdot P$. We know that P observes the detailed balance equations with invariant probability π, i.e.

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

Show that the Kullback-Leibler divergence decreases monotonically,

$$KL(\pi||\nu) - KL(\pi||\mu) = E[KL(P(Y,X)||Q(Y,X))] \geq 0.$$

where $P(Y,X)$ is a conditional probability and

$$Q(Y,X) = \frac{P(X,Y)\nu(X)}{\mu(Y)}$$

is a reverse step of P.

(Note that the KL-divergence is $KL(p_1||p_2) = \sum_x p_1(x) \log \frac{p_1(x)}{p_2(x)}$.

Hint: Compute the joint probability of two consecutive states (X,Y) according to two chains: the stationary chain $\pi(X,Y)$ and the current chain. Then compute the KL-divergence between the two.)

Problem 4. Let K be a stochastic matrix in a finite space Ω, let μ and ν be two initial probabilities on Ω, show that

$$||\mu K - \nu K||_{\text{TV}} \leq C(K)||\mu - \nu||_{\text{TV}},$$

where $C(K) \leq 1$ is the contraction coefficient, i.e. the maximum TV-norm between any two rows in the transition kernel,

$$C(K) = \max_{x,y} ||K(x,\cdot) - K(y,\cdot)||_{\text{TV}}.$$

The TV-norm is $||\mu - \nu||_{\text{TV}} = \frac{1}{2}\sum_{x\in\Omega} |\mu(x) - \nu(x)|$.

References

1. Persi Diaconis and Phil Hanlon. Eigen-analysis for some examples of the metropolis algorithm. *Contemporary Mathematics*, 138:99–117, 1992.
2. Persi Diaconis and Daniel Stroock. Geometric bounds for eigenvalues of markov chains. *The Annals of Applied Probability*, pages 36–61, 1991.
3. G. Winkler. *Image Analysis, Random Fields and Dynamic Monte Carlo Methods*. Springer, 2003.

Chapter 8
Data Driven Markov Chain Monte Carlo

"Data is the new oil." - Clive Humby

Introduction


Zhuowen Tu

Data Driven Markov chain Monte Carlo (DDMCMC) provides a principled way to use low-level information from processes such as edge detection and clustering to guide the MCMC search by making informed jumps in the solution space, achieving significant speed-ups in convergence to the modes of the posterior probability. The data-driven information obtained from edge detection and intensity clustering in represented as weighted samples (particles) and used as importance proposal probabilities for MCMC jumps using the Metropolis-Hastings method.

In the image segmentation application, these data-driven jumps are used in the split-merge operations, which together with diffusion-like operations such as region growing, snake/balloon, and region competition, achieve an efficient and ergodic exploration of the solution space.

8.1 Issues with Segmentation and Introduction to DDMCMC

Image segmentation is a long standing problem in computer vision due to the persistence of two major challenges.

The first challenge is the fundamental complexity associated with modeling a vast amount of visual patterns that appear in generic images. The objective of image segmentation is to parse an image into its constituent components. These consist of various stochastic processes, such as attributed points, lines, curves, textures, lighting variations, and deformable objects. Thus, a segmentation algorithm must incorporate many families of image models, and its performance is upper bounded by the accuracy of the chosen models.


Song-Chun Zhu

The second challenge is the intrinsic ambiguities in image perception, especially when there is no specific task to guide the attention. Real world images are fundamentally ambiguous, and our perception of an image changes over time. Furthermore, an image often demonstrates details at multiple scales. Thus, the more one looks at an image, the more one sees. Therefore, it must be wrong to think that a segmentation algorithm outputs only one result. Image segmentation should be considered a computing process not a vision task. It should output multiple distinct solutions dynamically and endlessly so that these solutions best preserve the intrinsic ambiguity.

Motivated by the above two observations, a stochastic computing paradigm called *data-driven Markov chain Monte Carlo* (DDMCMC) was created for image segmentation. The algorithm proceeds in five steps.

1. Formulates the problem in a Bayesian/MDL framework [32, 34, 69] with seven families of image models which compete to explain various visual patterns in an image, for example, flat regions, clutter, texture, smooth shading, etc.
2. Decomposes the solution space into an union of many subspaces of varying dimensions, and each subspace is a product of a number of subspaces for the image partition and image models (see Figure 8.10 for a space structure). The Bayesian posterior probability is distributed over such a heterogeneously structured space.
3. Designs ergodic Markov chains to explore the solution space and sample the posterior probability. The Markov chains consist of two types of dynamics: jumps and diffusion. The jump dynamics simulate reversible split-and-merge, and model switching. The diffusion dynamics simulate boundary deformation, region growing, region competition [69], and model adaptation. The split and merge processes are reversible such that the ergodicity and reversibility enable the algorithm to achieve nearly global optimal solutions independent of initial segmentation conditions.
4. Utilizes data-driven techniques to guide the Markov chain search in order to achieve tremendous speedup in comparison to other MCMC algorithms [24, 26, 27]. In the literature, there are various techniques for improving the Markov chain speed, such as multi-resolution approaches [7, 64], causal Markov models [7, 47], and clustering [3, 21, 54, 64]. In the DDMCMC paradigm, data-driven methods such as edge detection [10] and tracing, and data clustering [12, 13] are used. The results of these algorithms are expressed as weighted samples (or particles), which encode non-parametric probabilities in various subspaces. These probabilities respectively approximate the marginal probabilities of the Bayesian posterior probability, and they are used to design importance proposal probabilities to drive the Markov chains.
5. Implements mathematical principles and the K-adventurers algorithm for selecting and pruning a set of important and distinct solutions from the Markov chain sequence and at multiple scales of detail. The set of solutions encode an approximation to the Bayesian posterior probability. The multiple solutions are computed to minimize a Kullback-Leibler divergence from the approximate posterior to the true posterior, and they preserve the ambiguities in image segmentation.

In summary, the DDMCMC paradigm is about effectively creating particles (by bottom-up clustering/edge detection), composing particles (by importance proposals), and pruning particles (by the K-adventurers algorithm), and these particles represent hypotheses of various levels of granularity in the solution space. Conceptually, the DDMCMC paradigm also reveals the roles of some well-known segmentation algorithms. Algorithms such as split-and-merge, region growing, Snake [30] and balloon/bubble [52], region competition [69], variational methods [32], and PDEs [49] can be viewed as various MCMC jump-diffusion dynamics with minor modifications. Other algorithms, such as edge detection [10] and clustering [13, 18] compute importance proposal probabilities.

8.2 Simple Illustration of the DDMCMC

We use possibly the simplest example – the "Ψ-world" to illustrate the concepts of DDMCMC. The "Ψ-world" consists of only four types of objects: background pixels, line segments, circular arcs, and the Greek symbol Ψ, labeled by B, L, C, P respectively. The observed images $\mathbf{I}$ on a lattice Λ are generated through superposing n objects on the background with additive Gaussian noise. n is subject to a Poisson distribution, and the sizes and locations of the objects are subject to some uniform distributions. Figure 8.1 displays two *typical images* under the specified distributions.

The Ψ-world is described by a vector,

$$W = (n, \{(\ell_i, \theta_i); i = 0, 1, ..., n, \ \alpha).$$

$n \in \{0, 1, 2, .., |\Lambda|\}$ is the number of objects other than background, with $|\Lambda|$ the number of pixels in an image. $\ell_i \in \{B, L, C, P\}$ is the label and θ_i is a vector valued parameter describing the i-th object. We have only one background object, $\ell_0 = B$.

The parameters are defined in the following.

1. Type B: θ_0 is just μ_0 for the grey level of pixels.

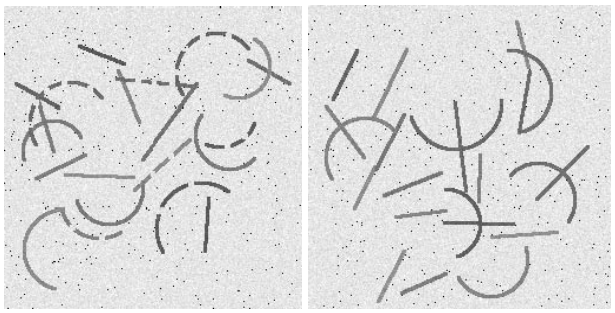


Fig. 8.1: Examples of the randomly generated images with varying number of objects. © [2000] IEEE. Reprinted, with permission, from ref. [70].

2. Type L: θ_i includes $(\rho_i, \tau_i, s_i, e_i, \mu_i)$. (ρ_i, τ_i) describes a straight line, and s_i, e_i the starting and ending points. μ_i is the intensity level of the line.

3. Type C: θ_i includes $(x_i, y_i, r_i, s_i, e_i, \mu_i)$ denoting the center, radius, starting point, ending point, and intensity level for arc objects.

4. Type P: θ_i includes $(x_i, y_i, r_i, \tau_i, \mu_i)$ denoting the center and radius of the half circle and the angle of the line segment, and the intensity level. By definition, the arc in Ψ must be a half circle.

Another important variable in W is the α-map for occlusion.

$$\alpha : \Lambda \to \{0, 1, 2, ..., n\}, \quad \alpha \in \Omega_\alpha.$$

For a pixel $(x, y) \in \Lambda$, $\alpha(x, y)$ indexes the top-most object which is the only visible object at this pixel.

We denote by $\varpi_g = [0, 255]$ the space for image intensity levels, and the solution space of the Ψ-world is,

$$\Omega = \Omega_\alpha \times \varpi_g \times \cup_{n=0}^{|\Lambda|} \Omega_o^n,$$

where Ω_o^n is the subspace with exactly n objects excluding the background.

$$\Omega_o^n = \cup_{k+l+m=n} \Omega_{k,l,m}, \; k, l, m \geq 0,$$

where $\Omega_{k,l,m}$ is the subspace that has exactly k lines, l arcs, and m Ψ-objects respectively.

$$\Omega_{k,l,m} = \underbrace{\Omega_L \times \cdots \times \Omega_L}_{k} \times \underbrace{\Omega_C \times \cdots \times \Omega_C}_{l} \times \underbrace{\Omega_\Psi \times \cdots \times \Omega_\Psi}_{m}.$$

We call Ω_L, Ω_C, and Ω_Ψ the *object spaces*.

These object spaces are further decomposed into five *atomic spaces*, which are denoted by lower case Greek symbols.

1. ϖ_g: the space for pixel intensities μ.
2. ϖ_c: the space for circle variables x, y, r.
3. ϖ_l: the space for line variables ρ, τ.
4. ϖ_e: the space for starting and ending points s, e.
5. ϖ_τ: the space for the orientations of Ψ.

Thus the object spaces are products of atomic spaces.

$$\Omega_l = \varpi_l \times \varpi_e \times \varpi_g, \quad \Omega_c = \varpi_c \times \varpi_e \times \varpi_g, \quad \Omega_\Psi = \varpi_c \times \varpi_\tau \times \varpi_g.$$

Figure 8.2 illustrates the structures of the solution space Ω. The triangles, squares and hexagons represent the three object spaces Ω_L, Ω_C, and Ω_Ψ respectively. The small circles of various shadows and sizes represent the five atomic spaces. The arrows indicate the reversible jumps between subspaces discussed in the next section.

Object recognition in the Ψ-world is posed as a Bayes inference problem,

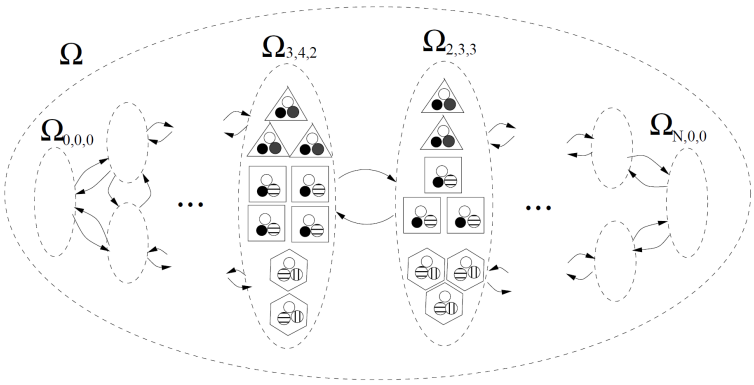


Fig. 8.2: The solution space Ω for the Ψ-world. © [2000] IEEE. Reprinted, with permission, from ref. [70].

$$W \sim p(W|\mathbf{I}) \propto p(\mathbf{I}|W)p(W), \;\; W \in \Omega.$$

$p(W)$ is the product of a Poisson distribution on the object number n and some uniform densities over the object parameters $\theta_i, i = 1,...,n$. Each W with all the parameters θ_i and α-map deterministically specify a clean image $\mathbf{I}_o$, and $p(\mathbf{I}|W) = p(\mathbf{I}|\mathbf{I}_o)$ is simply the product of iid Gaussian noise. Due to space limitation, we choose not to define the probabilities in details.

Notice that *the probability mass of $p(W|\mathbf{I})$ is distributed over many subspaces of varying dimensions*, in the next section, we simulate stochastic Markov chain dynamics which can travel in such heterogeneously structured space and achieve two general goals.

1. To compute the global optimum W^* in Ω.
2. To compute M distinct solutions (or explanations) $S = \{W_1, W_2, ..., W_M\}$ for robustness.

8.2.1 Designing MCMC – the basic issues

We design the MCMC in two steps.

Firstly, to be ergodic and aperiodic, we choose five types of MCMC dynamics.

Type I: diffusion process. This process changes the parameters θ_i, e.g. moving and extending a line segment etc.

Type II: death process. This process eliminates an existing object, and jumps to a subspace of lower dimension.

Type III: birth process. This process adds a new object, and jumps to a subspace of higher dimension.

Type IV: composition process. This process composes two existing objects into a new object, and jumps to a different subspace. For example, composing two short lines into a long line, or combining a line with a circle into a Ψ.

Type V: decomposition process. This process decomposes an existing object into two.

For example, in Figure 8.2 the move from $\Omega_{3,4,2}$ to $\Omega_{2,3,3}$ composes a line object and a circle object into a Ψ-object.

The five types of dynamics are applied in random sequence decided by throwing dice. It is easy to prove that the Markov chain with the five types of moves is reversible, ergodic, and aperiodic.

Secondly, we discuss how to balance the Markov chain dynamics. The diffusion process can be implemented by the stochastic Langevin equation, which is a steep ascent dynamics maximizing $p(W|\mathbf{I})$ with respect to θ_i plus a Brownian motion. It can also be implemented by a continuous Gibbs sampler.

Due to space limitation, we only discuss the balance between types II and III. The dynamics of type IV and V can be done in a similar way. Suppose at a certain time step t, we *propose* to eliminate an existing arc object specified by $\theta_i = (x_i, y_i, r_i, s_i, e_i, \mu_i)$:

$$W = (n, \theta_i, w) \longrightarrow (n-1, w) = W'.$$

w denotes the objects which remain unchanged during this move. In order to realize the death move, we must compute *how likely the same object can be born immediately – an inverse move by the birth process*. Notice that this is a pair

of moves that jump between two subspaces of different dimensions. We use the Metropolis-Hastings method. Let $G(W \to dW')$ and $G(W' \to dW)$ be the proposal probabilities for the two moves respectively, then the death move is accepted with probability,

$$A(W \to dW') = \min(1, \frac{G(W' \to dW)p(W'|\mathbf{I})dW'}{G(W \to dW')p(W|\mathbf{I})dW}). \tag{8.1}$$

The transition probability $P(W \to dW') = G(W \to dW')A(W \to dW')$ for $W \neq W'$. The ratio of the posterior probabilities is often the dominating factor which erects the influence of the posterior to balance the possible biases from the proposal probabilities.

The death proposal probability is

$$G(W \to dW') = q(\text{II})q_o(i)dw. \tag{8.2}$$

$q(\text{II}) \in (0,1)$ is the probability for using type II dynamics at time t, and $q_o(i)$ is the probability for choosing the circle object θ_i. The birth proposal is

$$G(W' \to dW) = q(\text{III})q(\theta_i)d\theta_i dw. \tag{8.3}$$

It first chooses type III with probability $q(\text{III})$, then chooses a new circle object θ_i with probability $q(\theta_i)d\theta_i$.

Since $dW = d\theta_i dw$ and $dW' = dw$, the dimensions of the nominator and denominator in equation (8.1) are matched. Designing $q(\text{II})$ and $q(\text{III})$ is easy and often not crucial for the speed. For example, one may use type II more frequently at the beginning. The key issue here is to compute $q(\theta_i)$!

In the statistical literature, the jump-dynamics were first studied in [26, 27], where variables in new dimensions are proposed by prior models. In our case, this chooses $q(\theta_i)$ to be a uniform distribution like blind search. Obviously such proposals are most likely to be rejected. This is a main reason for the MCMC to be inefficient! Intuitively, $q(\theta_i)$s should be able to predict where the new objects are likely to be in the object spaces. This is where the data driven (bottom-up) techniques step in.

8.2.2 Computing proposal probabilities in the atomic spaces – atomic particles

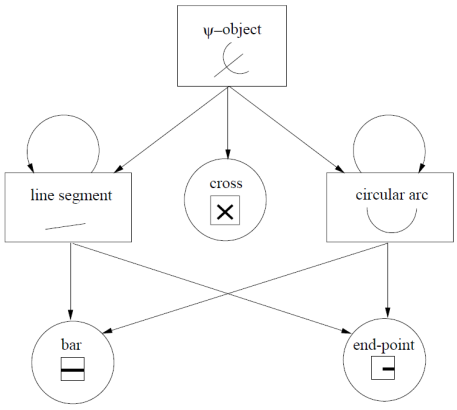


Fig. 8.3: Hierarchy in the Ψ world. © [2000] IEEE. Reprinted, with permission, from ref. [70].

Figure 8.3 displays the hierarchy of objects in the Ψ-world. The terminal (circles) nodes represent the elements of the features: bar, end point, and cross, while the arrows mean compositional relations. We use three types of feature detectors: 3 cross detectors, 6 bar detectors, and 12 end point detectors at various orientations.

Using the bar-detection map, we compute the Hough transforms for both lines and circles. Local maxima are computed using a mean-shift algorithm [13].We denote these local maxima by $(\rho_l^{(i)}, \tau_l^{(i)})$, $i = 1, 2, ..., n_l$.

Thus we compute an empirical density on the atomic space ϖ_l, expressed as a set of weighted samples.

$$q_l(\rho,\tau) = \sum_{i=1}^{n_l} w_l^{(i)} \delta(\rho - \rho_l^{(i)}, \tau - \tau_l^{(i)}), \sum_{i=1}^{n_l} w_l^{(i)} = 1.$$

$\delta(\rho - \rho_l^{(i)}, \tau - \tau_l^{(i)})$ is a window function centered at $(\rho_l^{(i)}, \tau_l^{(i)})$. We call $q_l(\rho,\tau)$ the *importance proposal probability* in atomic space ϖ_l, and $\{(\rho_l^{(i)}, \tau_l^{(i)}), i = 1,2,...,n_l\}$ the *atomic particles*.

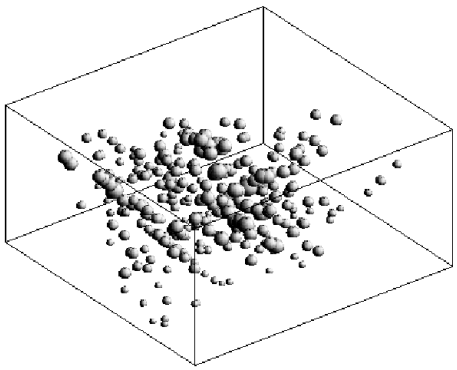


Fig. 8.4: 293 weighted samples (x,y,r) for circle particles in the atomic space ϖ_c. © [2000] IEEE. Reprinted, with permission, from ref. [70].

Similarly Figure 8.4 displays the atomic particles in the space ϖ_c. They are the local maxima of the Hough transform results for circles. The sizes of the spheres represent the weights $w_c^{(i)}$.

So we have an atomic proposal probability on ϖ_c,

$$q_c(x,y,r) = \sum_{i=1}^{n_c} w_c^{(i)} \delta(x - x_c^{(i)}, y - y_c^{(i)}, r - r_c^{(i)}), \sum_{i=1}^{n_c} w_l^{(i)} = 1.$$

In a similar way, one can compute proposal probabilities in other atomic spaces. 1) In ϖ_g we compute the intensity histogram $q_g(\mu)$. 2) In ϖ_e, we have computed the end-point maps. 3) In ϖ_τ, we can simply project $q_l()$ onto the τ axis. *For robustness and reversibility, the atomic proposal probabilities are continuous and non-zero everywhere in atomic spaces.*

8.2.3 Computing proposal probabilities in object spaces – object particles

Because the object spaces are products of atomic spaces, proposal probabilities in the three object spaces Ω_l, Ω_c and Ω_Ψ can be computed by the probabilities in the five atomic spaces. We discuss three methods.

Method I: conditional binding. This composes object particles by using the atomic features sequentially. For example, for a line object $\theta = (\rho,\tau,s,e,\mu)$, we compute

$$q(\theta) = q_l(\rho,\tau) q_e(s|\rho,\tau) q_e(e|\rho,\tau,s) q_g(\mu|\rho,\tau,s,e). \tag{8.4}$$

A set of line segments $\theta_l^{(i)} = (\rho_l^{(i)}, \tau_l^{(i)}, s_l^{(j)}, e_l^{(k)})$ are sampled from $q(\theta)$,

$$\{\theta_l^{(i)} \ : \ i = 1,2,...,n_L\}.$$

We call them the *object particles* in Ω_l. In a similar way, we can generate object particles in Ω_c.

These object particles appear similar in spirit to hypotheses in engineering methods. However, there is one crucial difference. Each object particle represents a windowed domain in the object space not just one point. The union of

the windows of the object particles covers the entire object space. To make the Markov chain reversible, each time we propose a new object by sampling the proposal probability *not just choosing from the set of particles*.

Object particles should also be generated by recursively composing object particles as shown by the arrows in Figure 8.3.

Method II: off-line composition. One may compose a particle by combining two other particles if the latter are *compatible*. The composition occurs off-line, that is, before we start run MCMC. But this is extremely expensive because of the exponential number of possible combinations.

Method III: on-line composition. This differs from method II by binding objects during the MCMC computation when two compatible objects appear (or are "alive") in the current W.

Atomic particles are computed once in a bottom-up process, whereas object particles are assembled dynamically during the MCMC process. Figure 8.5 shows three rows of object particles. One for birth candidates, one for death candidates and decomposition candidates, and one for compatible composition pairs that are alive in W. We must also consider the interactions between objects through the effects of occlusion captured by the α map. For example, suppose a long line segment in the image is covered, at present, by two short line objects $\theta_l^{(i)}$ and $\theta_l^{(j)}$. Then adding the long line object $\theta_l^{(k)}$ will have very little gain in fitting the image because of occlusion effect. Therefore the weights of these object particles must be updated on-line.

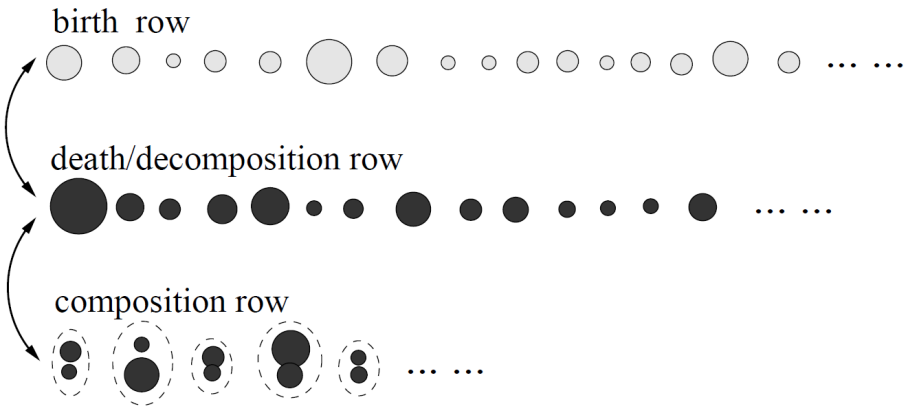


Fig. 8.5: Object particles are organized in three rows which drive the MCMC dynamics. Dark particles are alive. © [2000] IEEE. Reprinted, with permission, from ref. [70].

8.2.4 Computing multiple, distinct solutions – scene particles

To build robust vision systems for object recognition, the traditional MAP (*maximum a posteriori*) estimator

$$W^* = \arg\max_{W\in\Omega} p(W|\mathbf{I})$$

is not enough. Instead we should sample $p(W|\mathbf{I})$ and compute a set of representative solutions. However, when Ω is complicated and high dimensional, simply sampling $p(W|\mathbf{I})$ only generates solutions that are all from a single mode and have trivial differences from each other. Therefore a mathematical criterion must be derived for preserving important, distinctive local modes.

Let $S = \{W_1, W_2, ..., W_M\}$ be M solutions with weights $\omega_i \propto p(W|\mathbf{I}), \forall i$. The M weighted samples encode $p(W|\mathbf{I})$ in a non-parametric form.

$$\hat{p}(W|\mathbf{I}) = \sum_{i=1}^{M} \omega_i G(W - W_i), \quad \sum_{i=1}^{M} \omega_i = 1.$$

by some Gaussian window function G. In this section, we use the following criterion for extending the traditional MAP,

$$S^* = \{W_1, W_2, ..., W_M\} = \arg\min_{|S|=M} D(p||\hat{p}). \tag{8.5}$$

We call $\{W_1, W_2, ..., W_M\}$ the *scene particles*. They are chosen to minimize the Kullback-Leibler divergence $D(p||\hat{p})$ so that $\hat{p}$ "best" preserves p – the true posterior distribution under the constraint of complexity M.

In practice, p is represented as a mixture of Gaussian model like $\hat{p}$ by a large number $N >> M$ of particles recorded during the MCMC process. Thus we choose M distinct solutions during the MCMC computation so that $D(p||\hat{p})$ is minimized.

A mathematical deduction of the $D(p||\hat{p})$ leads us to three beautiful principles guiding the selection of scene particles.

1. One of the particles must be the global optimum W^*. Missing W^* causes substantial increase in $D(p||\hat{p})$.
2. The scene particles should minimize the total energy sum (or maximize the product of probabilities).
3. The scene particles should also maximize the sum of distances from each other.

The last two are conflicting, and thus the particles must "occupy" distinct modes in the solution space Ω.

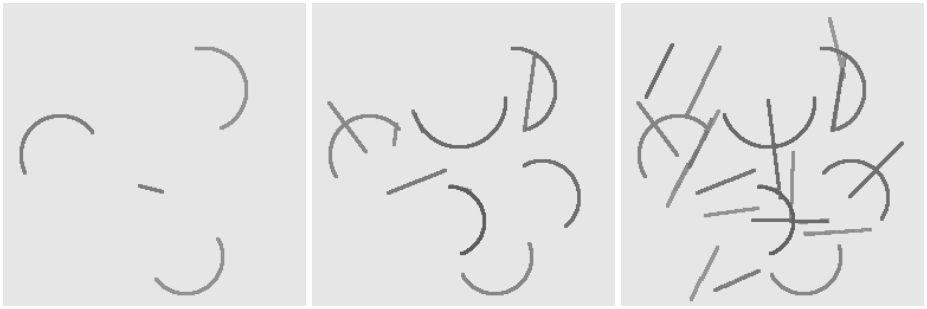


Fig. 8.6: Solution W visited by MCMC at three time steps. © [2000] IEEE. Reprinted, with permission, from ref. [70].

8.2.5 The Ψ-world experiment

We used an ensemble of randomly generated images for experiments, two of which are shown in Figure 8.1. Figure 8.6 displays the steps in an MCMC algorithm by showing the $\mathbf{I}_o$ decided by W at step t.

We are mainly interested in studying the improvement of the MCMC efficiency by comparing four Markov chains.

MCMC I: the Markov chain uses uniform proposal probabilities, as in the literature [26, 27].

MCMC II: the Markov chain uses atomic proposal probabilities with Hough transform without end point detection.

MCMC III: the Markov chain uses both Hough transform and end point detection, and samples new objects randomly from the set of particles.

MCMC IV: the Markov chain uses both Hough transform and end point detection. But it is different from MCMC III in that the weights of object particles are evaluated on-line.

Figure 8.7 plots the energy level, which is $-\log p(W|\mathbf{I})$, of the MCMC state at step t. Each of the four Markov chains runs 10 times for each of the 8 images, and the energy curves are thus averaged over 80 runs. The dotted, dash-doted, dashed, and solid curves are respectively for the MCMC I,II, III, and IV. The horizontal line at the bottom is the averaged energy for the truth W^*s. It is clear that with more data-driven methods used the Markov chains can approach the solution at much faster speed. MCMC IV reaches the solution with 2.3% relative error in 200 steps. The error is mainly a problem of fine-tuning by diffusion. In contrast, MCMC III needs about $5,000$ steps. MCMC I does not go down after even millions of steps. We have also compared the "mix" rate by measuring *how broad* the obtained solutions S are distributed in Ω.

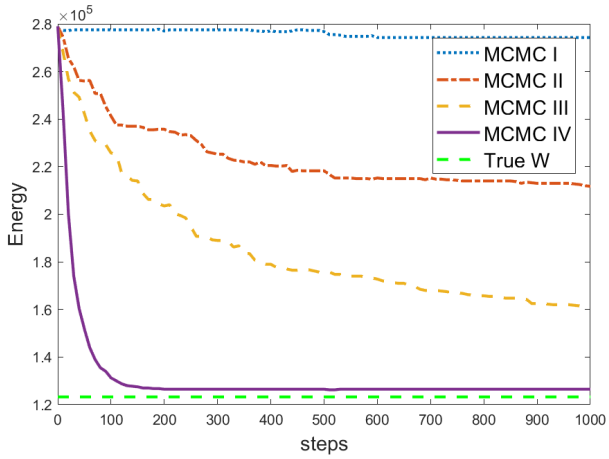


Fig. 8.7: The energy of the solution W plotted vs MCMC iterations. © [2000] IEEE. Reprinted, with permission, from ref. [70].

8.3 Problem Formulation and Image Models

In this section, the problem is formulated in a Bayesian framework, and the prior and likelihood models are discussed.

8.3.1 Bayesian formulation for segmentation

Let $\Lambda = \{(i,j) : 1 \le i \le L, 1 \le j \le H\}$ be an image lattice, and $\mathbf{I}_\Lambda$ an image defined on Λ. For any point $v \in \Lambda$, $\mathbf{I}_v \in \{0,...,G\}$ is the pixel intensity for a grey level image, or $\mathbf{I}_v = (L_v, U_v, V_v)$ for a color image.[1] The problem of image segmentation refers to partitioning the lattice into an unknown number K of disjoint regions

$$\Lambda = \cup_{i=1}^{K} R_i, \quad R_i \cap R_j = \emptyset, \quad \forall i \neq j. \tag{8.6}$$

Each region $R \subset \Lambda$ need not be connected. Let $\Gamma_i = \partial R_i$ denote the boundary of R_i. As a slight complication, two notations are used interchangeably in the literature. One considers region $R \in \Lambda$ as a discrete label map, and the other considers region boundary $\Gamma(s) = \partial R$ as a continuous contour parameterized by s. The continuous representation is convenient for diffusions while the label map representation is better for maintaining the topology. The level set method [48, 49] provides a good compromise between the two.

Each image region $\mathbf{I}_R$ is supposed to be coherent in the sense that $\mathbf{I}_R$ is a realization from a probabilistic model $p(\mathbf{I}_R; \Theta)$. Θ represents a stochastic process whose type is indexed by ℓ. Thus, a segmentation is denoted by a vector of hidden variables W, which describes the world state for generating the image $\mathbf{I}$ and is given by

$$W = (K, \{(R_i, \ell_i, \Theta_i);\ i = 1,2,...,K\}).$$

In a Bayesian framework, we make inference about W from $\mathbf{I}$ over a solution space Ω such that

$$W \sim p(W|\mathbf{I}) \propto p(\mathbf{I}|W)p(W),\ W \in \Omega.$$

As we mentioned before, the first challenge in segmentation is to obtain realistic image models. In the following subsection we briefly discuss the prior and likelihood models.

[1] We transfer the (R,G,B) representation to (L^*, u^*, v^*) for better color distance measure.

8.3.2 The prior probability

The prior probability $p(W)$ is a product of the following four probabilities.

1. An exponential model for the number of regions $p(K) \propto e^{-\lambda_0 K}$.
2. A general smoothness Gibbs prior for the region boundaries $p(\Gamma) \propto e^{-\mu \oint_\Gamma ds}$.
3. A model that encourages large regions to form $p(A) \propto e^{-\gamma A^{0.9}}$, where γ is a scale factor that controls the scale of the segmentation.
4. A prior for the parameters Θ of an image model that penalizes model complexity $p(\Theta|\ell) \propto e^{-\nu|\Theta|}$.

In summary, we have the following prior model

$$p(W) \propto p(K)\prod_{i=1}^{K} p(R_i)p(\ell_i)p(\Theta_i|\ell_i) \propto \exp\{-\lambda_0 K - \sum_{i=1}^{K}[\mu \oint_{\partial R_i} ds + \gamma|R_i|^c + \nu|\Theta_i|]\}. \tag{8.7}$$

8.3.3 The likelihood for grey level images

Visual patterns in different regions are assumed to be independent stochastic processes specified by $(\Theta_i, \ell_i), i = 1, 2, ..., K$. Thus the likelihood is,[2]

$$p(\mathbf{I}|W) = \prod_{i=1}^{K} p(\mathbf{I}_{R_i}; \Theta_i, \ell_i).$$

The choice of models needs to balance model sufficiency and computational efficiency. Four types of regions appear most frequently in real world images. Figure 8.8 shows examples of each of the following: a) flat regions with no distinct image structures, b) cluttered regions, c) regions with homogeneous textures, and d) inhomogeneous regions with globally smooth shading variations.

(a) uniform (b) clutter (c) texture (d) shading

Fig. 8.8: Four types of regions in the windows are typical in real world images. © [2002] IEEE. Reprinted, with permission, from ref. [59].

We adopt the following four families of models for the four types of regions. The algorithm can switch between them by Markov chain jumps. The four families are indexed by $\ell \in \{g_1, g_2, g_3, g_4\}$ and denoted by ϖ_{g_1}, ϖ_{g_2}, ϖ_{g_3}, and ϖ_{g_4} respectively. Let $G(0; \sigma^2)$ be a Gaussian density centered at 0 with variance σ^2.

1. Gaussian model $\ell = g_1$**:** ϖ_{g_1}. This assumes that pixel intensities in a region R are subject to independently and identically distributed (iid) Gaussian distribution,

$$p(\mathbf{I}_R; \Theta, g_1) = \prod_{v \in R} G(\mathbf{I}_v - \mu; \sigma^2), \quad \Theta = (\mu, \sigma) \in \varpi_{g_1}. \tag{8.8}$$

2. Histogram model $\ell = g_2$**:** ϖ_{g_2}. This is a non-parametric intensity histogram $h()$. In practice $h()$ is discretized as a step function expressed by a vector $(h_0, h_1, ..., h_G)$. n_j is the number of pixels in R with intensity level j.

[2] As a slight notation complication, Θ, ℓ could be viewed as parameters or hidden variables in W. We use $p(\mathbf{I}; \Theta, \ell)$ in both situations for simplicity.

$$p(\mathbf{I}_R;\Theta,g_2)=\prod_{v\in R}h(\mathbf{I}_v)=\prod_{j=0}^{G}h_j^{n_j},\quad \Theta=(h_0,h_1,...,h_G)\in\varpi_{g_2}. \tag{8.9}$$

3. Pseudo model $\ell=g_3$: ϖ_{g_3}. This is a texture FRAME model [68] with pixel interactions captured by a set of Gabor filters. To facilitate the computation, we choose a set of 8 filters and formulate the model in pseudo-likelihood form [67]. The model is specified by a long vector $\Theta=(\beta_1,\beta_2,...,\beta_m)\in\varpi_{g_3}$, where m is the total number of bins in the histograms of the 8 Gabor filtered images. Let ∂v denote the Markov neighborhood of $v\in R$, and $\mathbf{h}(\mathbf{I}_v|\mathbf{I}_{\partial v})$ the vector of 8 local histograms of filter responses in the neighborhood of pixel v. Each filter histogram counts the filter responses at pixels whose filter windows cover v. Thus we have

$$p(\mathbf{I}_R;\Theta,g_3)=\prod_{v\in R}p(\mathbf{I}_v|\mathbf{I}_{\partial v};\Theta)=\prod_{v\in R}\frac{1}{Z_v}\exp\{-<\Theta,\mathbf{h}(\mathbf{I}_v|\mathbf{I}_{\partial v})>\}, \tag{8.10}$$

The normalizing constant can be computed exactly and Θ can be estimated easily from images. We refer to [67] for discussions on the computation of this model and its variations, such as patch likelihood.

4. Surface model g_4: ϖ_{g_4}. The first three models are homogeneous, which fail in characterizing regions with shading effects, such as sky, lake, wall, perspective texture, etc. In the literature, such smooth regions are often modeled by low order Markov random fields, which again do not model the inhomogeneous pattern over space and often lead to over-segmentation. A 2D Bezier-spline model with sixteen equally spaced control points on Λ (i.e. we fix the knots) can instead be adopted. This is a generative type model. Let $B(x,y)$ be the Bezier surface, for any $v=(x,y)\in\Lambda$,

$$B(x,y)=U_{(x)}^T\times M\times U_{(y)}, \tag{8.11}$$

where $U_{(x)}=((1-x)^3,3x(1-x)^2,3x^2(1-x),x^3))^T$ and $M=(m_{11},m_{12},m_{13},m_{14};...;m_{41},...,m_{44})$. Therefore, the image model for a region R is,

$$p(\mathbf{I}_R;\Theta,g_4)=\prod_{v\in R}G(\mathbf{I}_v-B_v;\sigma^2),\quad \Theta=(M,\sigma)\in\varpi_{g_4}. \tag{8.12}$$

These four types of models compete to explain a grey intensity region. Whichever fits the region better will have a higher likelihood. The grey level model space is denoted by ϖ_Θ^g and given by

$$\Theta\in\varpi_\Theta^g=\varpi_{g_1}\cup\varpi_{g_2}\cup\varpi_{g_3}\cup\varpi_{g_4}.$$

8.3.4 Model calibration

The four image models should be calibrated for two reasons. First, for computational efficiency, simple models with fewer parameters are preferred. However, penalizing the number of parameters is not enough in practice. When a region is of size over ~ 100 pixels, the data term dominates the prior and demands more complex models. Second, the pseudo-likelihood models in family ϖ_{g_3} are not true likelihoods as they depend on a rather big neighborhood, and are thus not directly comparable to the other three types of models.

To calibrate the likelihood probabilities an empirical study can be used. We collected a set of typical regions from natural images and manually divided them into four categories. For example, Figure 8.9 shows four typical images in the first column, which are cropped from the images in Figure 8.8. The four images are denoted by $\mathbf{I}_i^{\text{obs}}, i=1,2,3,4$ on a lattice Λ_o. For each image $\mathbf{I}_i^{\text{obs}}$, we compute its per pixel coding length (minus log-likelihood) according to an optimal model within family ϖ_{g_j} computed by a maximum likelihood estimation for $j=1,2,3,4$ given in general by

$$L_{ij}=\min_{\varpi_{g_j}\ni\Theta}-\frac{\log p(\mathbf{I}_i^{\text{obs}};\Theta,g_j)}{|\Lambda_o|},\quad \text{for } 1\le i,j\le 4. \tag{8.13}$$

If $\Theta_{ij}^*\in\varpi_{gj}$ is an optimal fit within each family, then we can draw a typical sample (synthesis) from each fitted model,

$$\mathbf{I}_{ij}^{\text{syn}}\sim p(\mathbf{I};\Theta_{ij}^*,g_j),\quad \text{for } 1\le i,j\le 4.$$

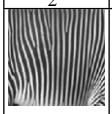
observed	ϖ_{g_1}	ϖ_{g_2}	ϖ_{g_3}	ϖ_{g_4}
				
$\mathbf{I}_1^{\text{obs}}$	$\mathbf{I}_{11}^{\text{syn}}, L_{11} = 1.957$	$\mathbf{I}_{12}^{\text{syn}}, L_{12} = 1.929$	$\mathbf{I}_{13}^{\text{syn}}, L_{13} = 1.680$	$\mathbf{I}_{14}^{\text{syn}}, L_{14} = 1.765$
				
$\mathbf{I}_2^{\text{obs}}$	$\mathbf{I}_{21}^{\text{syn}}, L_{21} = 3.503$	$\mathbf{I}_{22}^{\text{syn}}, L_{22} = 3.094$	$\mathbf{I}_{23}^{\text{syn}}, L_{23} = 2.749$	$\mathbf{I}_{24}^{\text{syn}}, L_{24} = 3.422$
				
$\mathbf{I}_3^{\text{obs}}$	$\mathbf{I}_{31}^{\text{syn}}, L_{31} = 3.852$	$\mathbf{I}_{32}^{\text{syn}}, L_{32} = 3.627$	$\mathbf{I}_{33}^{\text{syn}}, L_{33} = 2.514$	$\mathbf{I}_{34}^{\text{syn}}, L_{34} = 3.658$
				
$\mathbf{I}_4^{\text{obs}}$	$\mathbf{I}_{41}^{\text{syn}}, L_{41} = 3.121$	$\mathbf{I}_{42}^{\text{syn}}, L_{42} = 3.050$	$\mathbf{I}_{43}^{\text{syn}}, L_{43} = 1.259$	$\mathbf{I}_{44}^{\text{syn}}, L_{44} = 0.944$

Fig. 8.9: Comparison study of four families of models. The first column contains the original image regions cropped from four real world images shown in figure 8.8. Columns 2-5 are synthesized images $\mathbf{I}_{ij}^{\text{syn}} \sim p(\mathbf{I}_R; \Theta_{ij}^*)$ sampled from the four families respectively each after an MLE fitting. The number below each synthesized image shows the per-pixel coding bits L_{ij} using each model family. © [2002] IEEE. Reprinted, with permission, from ref. [59].

$\mathbf{I}_i^{\text{obs}}$, $\mathbf{I}_{ij}^{\text{syn}}$, and L_{ij} are shown in Figure 8.9 for $1 \le i, j \le 4$.

The results in Figure 8.9 show that the spline model has the shortest coding length for the shading region, while the texture model fits the best for the three other regions. We can rectify these models by a constant factor e^{-c_j} for each pixel v,

$$\hat{p}(\mathbf{I}_v; \Theta, g_j) = p(\mathbf{I}_v; \Theta, g_j) e^{-c_j}, \quad \text{for } j = 1, 2, 3, 4.$$

The c_j are chosen so that the rectified coding length $\hat{L}_{ij}$ reaches minimum when $i = j$. Effectively, uniform regions, clutter regions, texture regions, and shading regions are best fitted by the models in ϖ_1, ϖ_2, ϖ_3, and ϖ_4 respectively.

8.3.5 Image models for color

In practice, we work on both grey level and color images. For color images, we adopt a (L^*, u^*, v^*) color space and three families of models indexed by $\ell \in \{c_1, c_2, c_3\}$. Let $G(0; \Sigma)$ denote a 3D Gaussian density.

1. Gaussian model c_1: ϖ_{c_1}. This is an i.i.d. Gaussian model in (L^*, u^*, v^*) space.

$$p(\mathbf{I}_R; \Theta, c_1) = \prod_{v \in R} G(\mathbf{I}_v - \mu; \Sigma), \quad \Theta = (\mu, \Sigma) \in \varpi_{c_1}. \tag{8.14}$$

2. Mixture model c_2: ϖ_{c_2}. This is a mixture of two Gaussians and is used for modeling textured color regions,

$$p(\mathbf{I}_R; \Theta, c_2) = \prod_{v \in R} [\alpha_1 G(\mathbf{I}_v - \mu_1; \Sigma_1) + \alpha_2 G(\mathbf{I}_v - \mu_2; \Sigma_2)].$$

Thus $\Theta = (\alpha_1, \mu_1, \Sigma_1, \alpha_2, \mu_2, \Sigma_2) \in \varpi_{c_2}$ are the parameters.

3. Bezier model c_3: ϖ_{c_3}. We use three Bezier spline surfaces (see equation (8.11)) for L^*, u^*, and v^* respectively to characterize regions with gradually changing colors such as sky, walls, etc. Let $\mathbf{B}(x, y)$ be the color value in (L^*, u^*, v^*) space for any $v = (x, y) \in \Lambda$,

$$\mathbf{B}(x, y) = (U_{(x)}^T \times M_L \times U_{(y)}, U_{(x)}^T \times M_u \times U_{(y)}, U_{(x)}^T \times M_v \times U_{(y)})^T.$$

Thus the model is

$$p(\mathbf{I}_R;\Theta,c_3)=\prod_{v\in R}G(\mathbf{I}_v-\mathbf{B}_v;\Sigma),$$

where $\Theta=(M_L,M_u,M_v,\Sigma)$ are the parameters.
These three types of models compete to explain a color region. Whichever fits the region better will have higher likelihood. We denote the color model space by ϖ_Θ^c such that

$$\varpi_\Theta^c=\varpi_{c_1}\cup\varpi_{c_2}\cup\varpi_{c_3}.$$

8.4 Anatomy of Solution Space

Before we go into the details of the algorithm, we need to study the structures of the solution space Ω in which the posterior probability $p(W|\mathbf{I})$ is distributed. We start with the *partition space* for all possible partitions of a lattice Λ. When a lattice Λ is segmented into k disjoint regions, we call it a *k-partition* denoted by π_k,

$$\pi_k=(R_1,R_2,...,R_k),\quad \cup_{i=1}^k R_i=\Lambda,\quad R_i\cap R_j=\emptyset,\ \forall i\neq j. \tag{8.15}$$

If all pixels in each region are connected, then π_k is a connected component partition [64]. The set of all k-partitions, denoted by ϖ_{π_k}, is a quotient space of the set of all possible k-colorings divided by a permutation group $\mathcal{PG}$ for the labels given by

$$\varpi_{\pi_k}=\{(R_1,R_2,...,R_k)=\pi_k;\quad |R_i|>0,\ \forall i=1,2,...,k\}/\mathcal{PG}. \tag{8.16}$$

Thus, we have a general partition space ϖ_π with the number of regions $1\leq k\leq|\Lambda|$,

$$\varpi_\pi=\cup_{k=1}^{|\Lambda|}\varpi_{\pi_k}.$$

The solution space for W is a union of subspaces Ω_k, and each Ω_k is a product of one k-partition space ϖ_{π_k} and k spaces for the image models

$$\Omega=\cup_{k=1}^{|\Lambda|}\Omega_k=\cup_{k=1}^{|\Lambda|}[\ \varpi_{\pi_k}\times\underbrace{\varpi_\Theta\times\cdots\times\varpi_\Theta}_{k}\ ], \tag{8.17}$$

where $\varpi_\Theta=\cup_{i=1}^4\varpi_{gi}$ for grey level images, and $\varpi_\Theta=\cup_{i=1}^3\varpi_{ci}$ for color images.

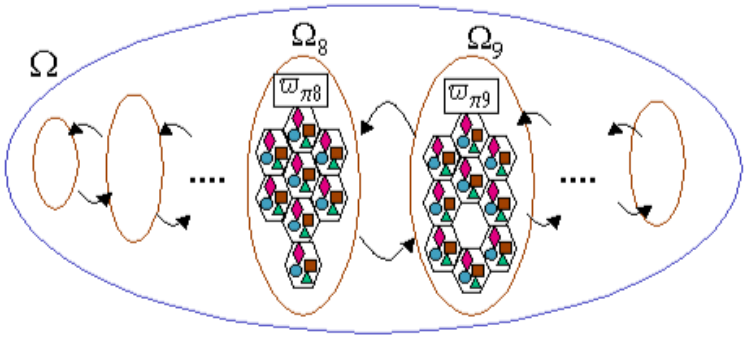


Fig. 8.10: The anatomy of the solution space. The arrows represent Markov chain jumps. The reversible jumps between two subspace Ω_8 and Ω_9 realize the split-and-merge of a region. © [2002] IEEE. Reprinted, with permission, from ref. [59].

Figure 8.10 illustrates the structures of the solution space. The four image families $\varpi_\ell,\ell=g_1,g_2,g_3,g_4$ are represented by the triangles, squares, diamonds and circles respectively. $\varpi_\Theta=\varpi_\Theta^g$ is represented by a hexagon containing

the four shapes. The partition space ϖ_{π_k} is represented by a rectangle. Each subspace Ω_k consists of a rectangle and k hexagons, and each point $W \in \Omega_k$ represents a k-partition plus k image models for k regions. We call Ω_k the scene spaces. ϖ_{π_k} and $\varpi_\ell, \ell = g_1, g_2, g_3, g_4$ (or $\ell = c_1, c_2, c_3$) are the basic components for constructing Ω and thus are called the atomic spaces. Sometimes we call ϖ_π a partition space and $\varpi_\ell, \ell = g_1, g_2, g_3, g_4, c_1, c_2, c_3$ the cue spaces.

8.5 Exploring the Solution Space by Ergodic Markov Chains

The solution space in Figure 8.10 is typical for vision problems. The posterior probability $p(W|\mathbf{I})$ not only has an enormous number of local maxima but is distributed over subspaces of varying dimensions. To search for globally optimal solutions in such spaces, we adopt the Markov chain Monte Carlo (MCMC) techniques.

8.5.1 Five Markov chain dynamics

We adopt five types of Markov chain dynamics which are used at random with probabilities $p(1), ..., p(5)$ respectively. Dynamics 1-2 are diffusion, and dynamics 3-5 are reversible jumps.

Dynamic 1: boundary diffusion/competition. For mathematical convenience, we switch to a continuous boundary representation for regions $R_i, i = 1, ..., K$. These curves evolve to maximize the posterior probability through a region competition equation [69]. Let Γ_{ij} be the boundary between R_i, R_j, $\forall i, j$, and Θ_i, Θ_j the models for the two regions respectively. The motion of points $\Gamma_{ij}(s) = (x(s), y(s))$ follows the steepest ascent equation of the $\log p(W|\mathbf{I})$ plus a Brownian motion dB along the curve normal direction $\vec{n}(s)$. By variational calculus, the resulting equation is [69],

$$\frac{d\Gamma_{ij}(s)}{dt} = [f_{prior}(s) + \log \frac{p(\mathbf{I}(x(s), y(s)); \Theta_i, \ell_i)}{p(\mathbf{I}(x(s), y(s)); \Theta_j, \ell_j)} + \sqrt{2T(t)}dB]\, \vec{n}(s).$$

The first two terms are derived from the prior and likelihood respectively. The Brownian motion is a normal distribution whose magnitude is controlled by the temperature $T(t)$ which decreases with time. The Brownian motion helps to avoid local pitfalls. The log-likelihood ratio requires that the image models are comparable. Dynamic 1 realizes diffusion within the atomic (or partition) space ϖ_{π_k} (i.e. moving within a rectangle of Figure 8.10).

Dynamic 2: model adaptation. This simply fits the parameters of a region by steepest ascent. One can add a Brownian motion, but it does not make much a difference in practice. The dynamic is given by

$$\frac{d\Theta_i}{dt} = \frac{\partial \log p(\mathbf{I}_{R_i}; \Theta_i, \ell_i)}{\partial \Theta_i}.$$

This realizes diffusion in the atomic (or cue) spaces $\varpi_\ell, \ell \in \{g_1, g_2, g_3, g_4, c_1, c_2, c_3\}$ (move within a triangle, square, diamond, or circle of Figure 8.10).

Dynamics 3-4: split and merge. Suppose at a certain time step, a region R_k with model Θ_k is split into two regions R_i and R_j with models Θ_i and Θ_j, or vice versa. This realizes a jump between two states W to W' as shown by the arrows in Figure 8.10.

$$W = (K,\ (R_k, \ell_k, \Theta_k),\ W_-) \longleftrightarrow (K+1,\ (R_i, \ell_i, \Theta_i), (R_j, \ell_j, \Theta_j),\ W_-) = W',$$

where W_- denotes the remaining variables that are unchanged during the move. By the Metropolis-Hastings method [43], we need two proposal probabilities $G(W \to dW')$ and $G(W' \to dW)$. $G(W \to dW')$ is a conditional probability for how likely the Markov chain proposes to move to W' at state W, and $G(W' \to dW)$ is the proposal probability for coming back. The proposed split is then accepted with probability

$$\alpha(W \to dW') = \min(1, \frac{G(W' \to dW)p(W'|\mathbf{I})dW'}{G(W \to dW')p(W|\mathbf{I})dW}).$$

There are two routes for computing the split proposal $G(W \to dW')$. In route 1, one first chooses a split move with probability $q(3)$, then chooses region R_k from a total of K regions at random. We denote this probability by $q(R_k)$. Given R_k, one chooses a candidate splitting boundary Γ_{ij} within R_k with probability $q(\Gamma_{ij}|R_k)$. Then, for the two new regions R_i and R_j one chooses two new model types ℓ_i and ℓ_j with probabilities $q(\ell_i)$ and $q(\ell_j)$ respectively. One then chooses $\Theta_i \in \varpi_{\ell_i}$ with probability $q(\Theta_i|R_i,\ell_i)$ and chooses Θ_j with probability $q(\Theta_j|R_j,\ell_j)$. Thus,

$$G(W \to dW') = q(3)q(R_k)q(\Gamma_{ij}|R_k)q(\ell_i)q(\Theta_i|R_i,\ell_i)q(\ell_j)q(\Theta_j|R_j,\ell_j)dW'. \tag{8.18}$$

In route 2, one first chooses two new region models Θ_i and Θ_j, and then decides the boundary Γ_{ij}. Thus,

$$G(W \to dW') = q(3)q(R_k)q(\ell_i)q(\ell_j)q(\Theta_i,\Theta_j|R_k,\ell_i,\ell_j)q(\Gamma_{ij}|R_k,\Theta_i,\Theta_j)dW'. \tag{8.19}$$

We shall discuss in later subsection that either of the two routes can be more effective than the other depending on the region R_k.

Similarly we have the merge proposal probability,

$$G(W' \to dW) = q(4)q(R_i,R_j)q(\ell_k)q(\Theta_k|R_k,\ell_k)dW, \tag{8.20}$$

where $q(R_i,R_j)$ is the probability of choosing to merge two regions R_i and R_j at random.

Dynamic 5: switching image models. This switches the image model within the four families (three for color images) for a region R_i. For example, from texture description to a spline surface we have

$$W = (\ell_i, \Theta_i, W_-) \longleftrightarrow (\ell_i', \Theta_i', W_-) = W'.$$

The proposal probabilities are

$$G(W \to dW') = q(5)q(R_i)q(\ell_i')q(\Theta_i'|R_i,\ell_i')dW', \tag{8.21}$$

$$G(W' \to dW) = q(5)q(R_i)q(\ell_i)q(\Theta_i|R_i,\ell_i)dW. \tag{8.22}$$

8.5.2 The bottlenecks

The speed of a Markov chain depends critically on the design of its proposal probabilities in the jumps. In our experiments, the proposal probabilities, such as $q(1),...,q(5)$, $q(R_k)$, $q(R_i,R_j)$, $q(\ell)$ are easy to specify and do not influence the convergence significantly. The real bottlenecks are caused by two proposal probabilities in the jump dynamics.

1. $q(\Gamma|R)$ in equation (8.18): What is a good Γ for splitting a given region R? $q(\Gamma|R)$ is a probability in the atomic space ϖ_π.
2. $q(\Theta|R,\ell)$ in equations (8.18), (8.20) and (8.22): For a given region R and a model family $\ell \in \{g_1,...,g_4,c_1,c_2,c_3\}$, what is a good Θ? $q(\Theta|R,\ell)$ is a probability in the atomic space ϖ_ℓ.

It is worth mentioning that both probabilities $q(\Gamma|R)$ and $q(\Theta|R,\ell)$ cannot be replaced by deterministic decisions like were used in region competition [69] and others [34]. Otherwise, the Markov chain will not be reversible and thus reduce to a greedy algorithm. On the other hand, if we choose uniform distributions, it is equivalent to blind search, and the Markov chain will experience exponential waiting time before each jump. In fact, the length of the waiting time is proportional to the volume of the cue spaces. The design of these probabilities needs to strike a balance between speed and robustness (non-greediness).

While it is hard to analytically derive a convergence rate for these complicated algorithms, it is revealing to observe the following theorem [42] in a simple case.

Theorem 8.1. *Given a target density $p(x)$ to sample by independence Metropolis-Hastings algorithm with proposal probability $q(x)$, let $P^n(x_o,y)$ be the probability of a random walk to reach point y in at most n steps. If there exists $\rho > 0$ such that,*

$$\frac{q(x)}{p(x)} \geq \rho, \quad \forall x,$$

then the convergence can be measured by a L_1 norm distance

$$||P^n(x_o,\cdot) - p|| \leq (1-\rho)^n.$$

This theorem states that the proposal probability $q(x)$ should be very close to $p(x)$ for fast convergence. In our case, $q(\Gamma|R)$ and $q(\Theta|R,\ell)$ should be equal to the conditional probabilities of some marginal probabilities of the posterior $p(W|\mathbf{I})$ within the atomic spaces ϖ_π and ϖ_ℓ respectively. That is,

$$q^*_\Gamma(\Gamma_{ij}|R_k) = p(\Gamma_{ij}|\mathbf{I},R_k), \quad q^*_\Theta(\Theta|R,\ell) = p(\Theta|\mathbf{I},R,\ell), \ \forall \ell. \tag{8.23}$$

Unfortunately, q^*_Γ and q^*_Θ have to integrate information from the entire image $\mathbf{I}$, and thus are intractable. We must seek approximations, and this is where the data-driven methods come in.

In the next section, we discuss data clustering for each atomic space ϖ_ℓ, $\ell \in \{c_1,c_2,c_3,g_1,g_2,g_3,g_4\}$ and edge detection in ϖ_π. The results of clustering and edge detection are expressed as non-parametric probabilities for approximating the ideal marginal probabilities q^*_Γ and q^*_Θ in these atomic spaces respectively.

8.6 Data-Driven Methods

8.6.1 Method I: clustering in atomic spaces

Given an image $\mathbf{I}$ (grey or color) on lattice Λ, we extract a feature vector F^ℓ_v at each pixel $v \in \Lambda$. The dimension of F^ℓ_v depends on the image model indexed by ℓ. We then have a collection of vectors

$$\mathcal{V}^\ell = \{F^\ell_v : \ v \in \Lambda\}.$$

In practice, v can be subsampled for computational ease. The set of vectors are clustered by either an EM method [17] or a mean-shift clustering [12, 13] algorithm to $\mathcal{V}^\ell$. The EM-clustering approximates the point density in $\mathcal{V}^\ell$ by a mixture of m Gaussians, and it extends from the m-mean clustering by a soft cluster assignment to each vector F_v. The mean-shift algorithm assumes a non-parametric distribution for $\mathcal{V}^\ell$ and seeks the modes (local maxima) in its density (after some Gaussian window smoothing). Both algorithms return a list of m weighted clusters $\Theta^\ell_1,\Theta^\ell_2,...,\Theta^\ell_m$ with weights $\omega^\ell_i, i = 1,2,...,m$, and we denote these by

$$\mathcal{P}^\ell = \{ \ (\omega^\ell_i,\Theta^\ell_i) : \ i = 1,2,...,m. \ \}. \tag{8.24}$$

We call $(\omega^\ell_i,\Theta^\ell_i)$ a weighted atomic (or cue) particle in ϖ_ℓ for $\ell \in \{c_1,c_3,g_1,g_2,g_3,g_4\}$ [3]. The size m is chosen to be conservative, or it can be computed in a coarse-to-fine strategy with a limit $m = |\mathcal{V}^\ell|$. More details are given in [12, 13].

In the clustering algorithms, each feature F^ℓ_v and thus its location v is classified to a cluster Θ^ℓ_i such that

$$S^\ell_{i,v} = p(F^\ell_v;\Theta^\ell_i), \qquad \text{with } \sum_{i=1}^m S^\ell_{i,v} = 1, \quad \forall v \in \Lambda, \ \forall \ell.$$

This is a soft assignment and can be computed by the distance from F_v to the cluster centers. We call

$$S^\ell_i = \{S^\ell_{i,v} : \ v \in \Lambda\}, \quad \text{for } i = 1,2,...,m, \ \forall \ell \tag{8.25}$$

a saliency map associated with cue particle Θ^ℓ_i.

Computing cue particles in ϖ_{c_1}. For color images, we take $F_v = (L_v,U_v,V_v)$ and apply a mean-shift algorithm [12, 13] to compute color clusters in ϖ_{c_1}. For example, Figure 8.11 shows a few color clusters (balls) in a cubic space for a

[3] The atomic space ϖ_{c_2} is a composition of two ϖ_{c_1}, and thus is computed from ϖ_{c_1}.

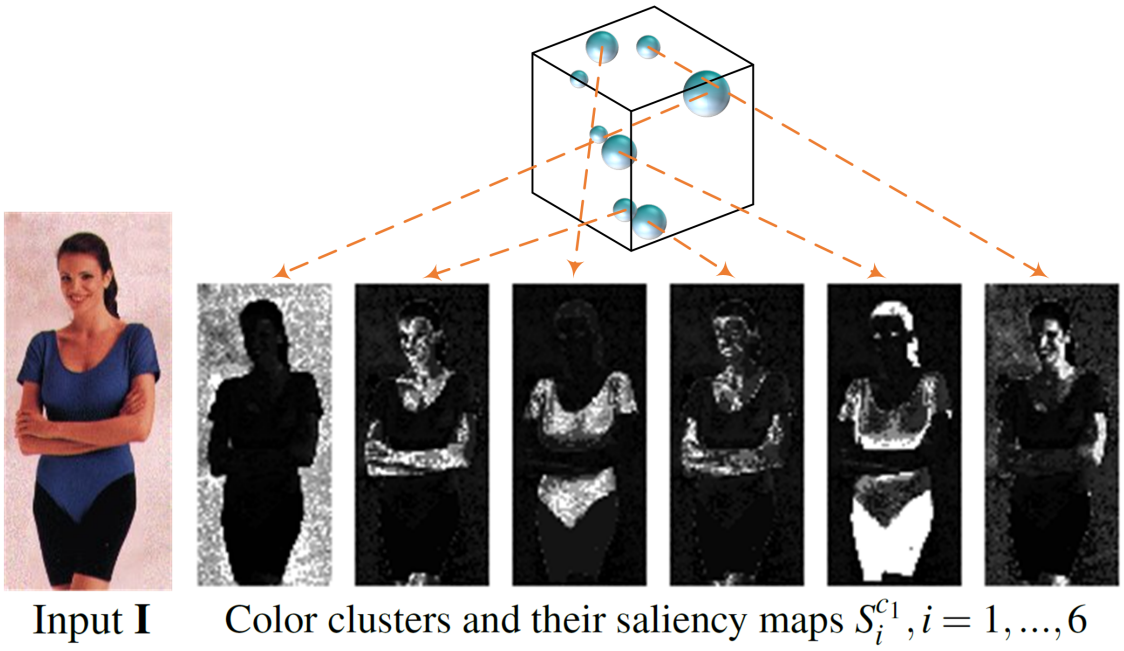
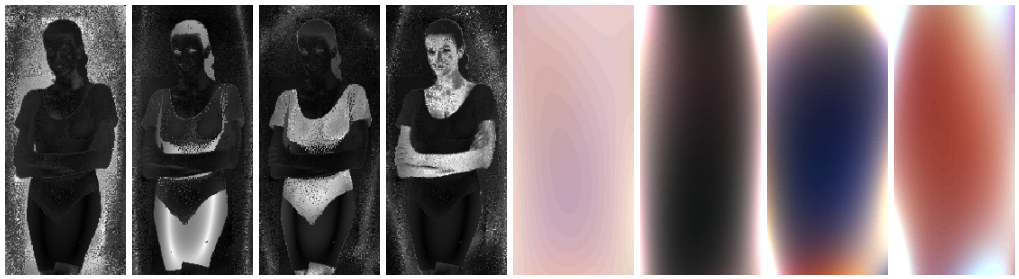


Fig. 8.11: A color image and its clusters in (L^*, u^*, v^*) space for ϖ_{c_1}, the second row contains six of the saliency maps associated with the color clusters. © [2002] IEEE. Reprinted, with permission, from ref. [59].

simple color image. The size of the balls represents the weights $\omega_i^{c_1}$. Each cluster is associated with a saliency map $S_i^{c_1}$ for $i = 1,2,...,6$ with bright areas indicating high probabilities. From left to right the maps are background, skin, shirt, shadowed skin, pant and hair, highlighted skin respectively.



(a) (b) (c) (d) (e) (f) (g) (h)

Fig. 8.12: (a)-(d) are saliency maps associated with four clusters in ϖ_{c_3}. (e)-(h) are the color spline surfaces for the four clusters. © [2002] IEEE. Reprinted, with permission, from ref. [59].

Computing cue particles in ϖ_{c_3}. Each point v contributes its color $\mathbf{I}_v = (L_v, U_v, V_v)$ as "surface heights", and we apply EM-clustering to find the spline surface models. Figure 8.12 shows the clustering result for the woman image. Figures 8.12 (a-d) are saliency maps $S_i^{c_3}$ for $i = 1,2,3,4$. Figures 8.12 (e-h) are the four reconstructed images according to fitted spline surfaces which recover some global illumination variations.

Computing cue particles in ϖ_{g_1}. In this model, the feature space $F_v = \mathbf{I}_v$ is simply the intensity, and $\mho^{g_1}$ is the image intensity histogram. We simply apply a mean-shift algorithm to get the modes (peaks) of the histogram and the breadth of each peak decides its variance.

Figure 8.13 shows six saliency maps $S_i^{g_1}, i = 1,2,...,6$ for the zebra image shown in Figure 8.21 (a). In the clustering map on the left, each pixel is assigned to its most likely particle.

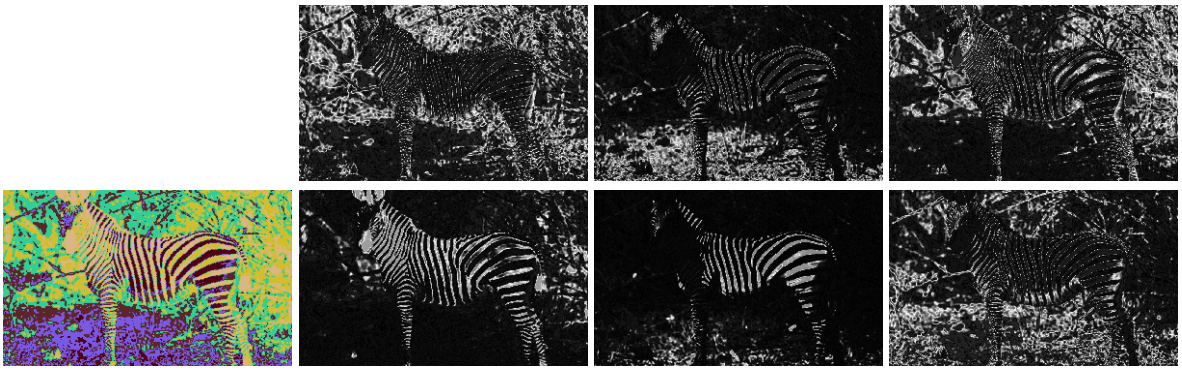


Fig. 8.13: A clustering map (left) for ϖ_{g_1} and six saliency maps $S_i^{g_1}, i = 1,...,6$ of a zebra image. © [2002] IEEE. Reprinted, with permission, from ref. [59].

Computing the cue particles in ϖ_{g_2}. For clustering in ϖ_{g_2}, at each subsampled pixel $v \in \Lambda$, we compute F_v as a local intensity histogram $F_v = (h_{v0},...,h_{vG})$ accumulated over a local window centered at v. EM clustering is then applied to compute the cue particles, and each particle $\Theta_i^{g_2}, i = 1,...,m$ is a histogram. This model is used for clutter regions.

Figure 8.14 shows the clustering results on the same zebra image.

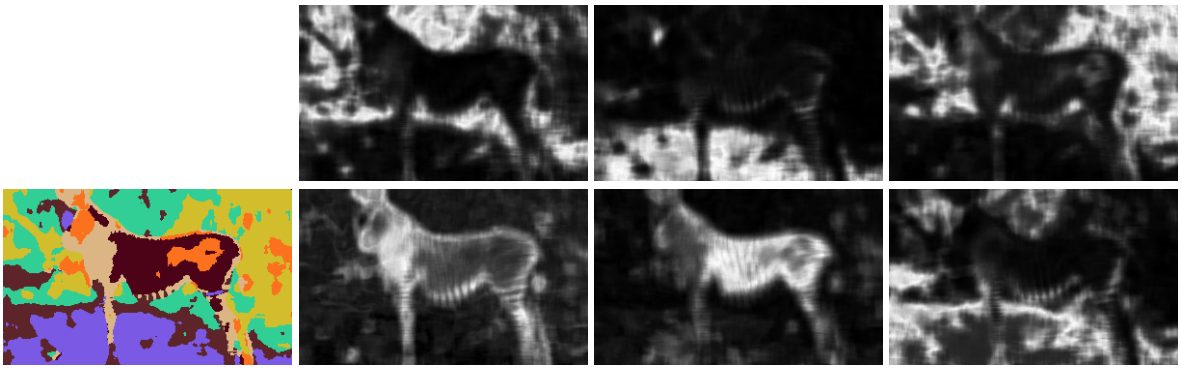


Fig. 8.14: A clustering map (left) for ϖ_{g_2} and six saliency maps $S_i^{g_2}, i = 1..6$ of a zebra image. © [2002] IEEE. Reprinted, with permission, from ref. [59].

Computing cue particles in ϖ_{g_3}. At each sub-sampled pixel $v \in \Lambda$, we compute a set of 8 local histograms for 8 filters over a local window of 12×12 pixels. We choose 8 filters for computational convenience: one δ filter, two gradient filters, one Laplacian of Gaussian filter, and four Gabor filters. Each histogram has 9 bins. $F_v^{g_3} = (h_{v,1,1},...,h_{v,8,9})$ is then the feature. EM clustering is applied to find the m mean histograms $\bar{\mathbf{h}}_i, i = 1,2,...,m$. We can compute the cue particles for texture models $\Theta_i^{g_3}$ from $\bar{\mathbf{h}}_i$ for $i = 1,2,...,m$. A detailed account of this transform is given in [67].

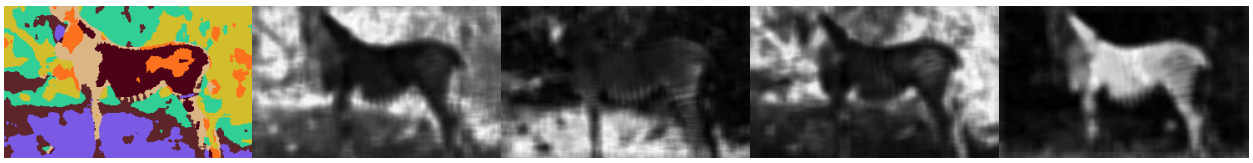
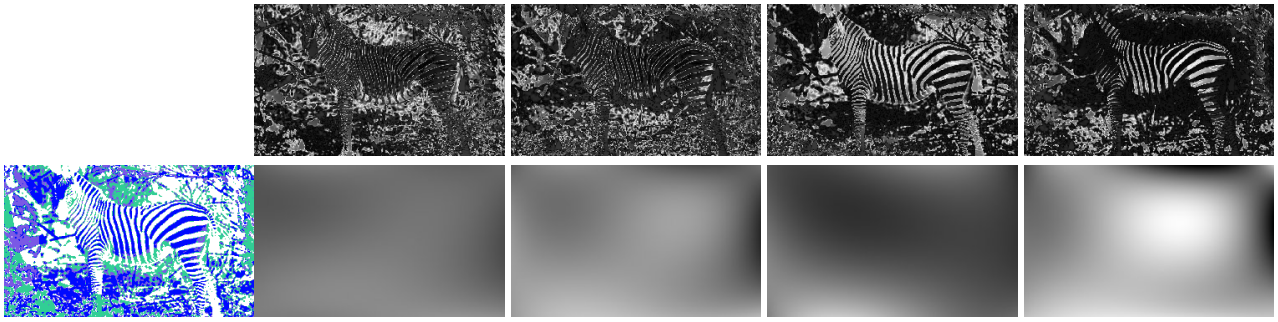


Fig. 8.15: Texture clustering. A clustering map (left) and four saliency maps for four particles $\Theta_i^{g_3}, i = 1,2,...,4$. © [2002] IEEE. Reprinted, with permission, from ref. [59].

Figure 8.15 shows the texture clustering results on the zebra image with one clustering map on the left, and four saliency maps for four particles $\Theta_i^{g_3}, i = 1,2,...,4$.

Computing cue particles in ϖ_{g4}. Each point v contributes its intensity $\mathbf{I}_v = F_v$ as a "surface height", and we apply EM clustering to find the spline surface models. Figure 8.16 shows a clustering result for the zebra image with four surfaces. The second row shows the four surfaces which recover some global illumination variations. Unlike the texture clustering results which capture the zebra stripes as a whole region, the surface models separate the black and white stripes as two regions – another valid perception. Interestingly, the black and white stripes in the zebra skin both have shading changes which are fitted by the spline models.



A clustering map Four saliency maps and surfaces

Fig. 8.16: Clustering result on the zebra image under Bezier surface model. The left image is the clustering map. The first row of images on the right side are the saliency maps. The second row shows the fitted surfaces using the surface height as intensity. © [2002] IEEE. Reprinted, with permission, from ref. [59].

8.6.2 Method II: edge detection

We detect intensity edges using Canny edge detector [10] and color edges using a method in [35]. The edges are then traced to form a partition of the image lattice. We choose edges at three scales according to edge strength, and thus compute the partition maps in three coarse-to-fine scales. We choose not to discuss the details, but show some results using the two running examples: the woman and zebra images.

Figure 8.17 shows a color image and three scales of partitions. Since this image has strong color cues, the edge maps are very informative about where the region boundaries are. In contrast, the edge maps for the zebra image are very messy, as Figure 8.18 shows.

(a) (b) (c) (d)

Fig. 8.17: Partition maps at three scales of details for a color image. (a) Input image. (b) Scale 1. (c) Scale 2. (d) Scale 3. © [2002] IEEE. Reprinted, with permission, from ref. [59].

(a) (b) (c) (d)

Fig. 8.18: A grey level image and three partition maps at three scales. (a) Input image. (b) Scale 1. (c) Scale 2. (d) Scale 3. © [2002] IEEE. Reprinted, with permission, from ref. [59].

8.7 Computing Importance Proposal Probabilities

It is generally acknowledged that clustering and edge detection algorithms can sometimes produce good segmentations or even perfect results for some images, but very often they are far from being reliable for generic images, as the experiments in Figures 8.11-8.18 demonstrate. It is also true that sometimes one of the image models and edge detection scales could do a better job in segmenting some regions than other models and scales, but we do not know a priori what types of regions are present in a generic image. Thus, we compute all models and edge detection at multiple scales and then utilize the clustering and edge detection results probabilistically. MCMC theory provides a framework for integrating this probabilistic information in a principled way under the guidance of a globally defined Bayesian posterior probability.

Computing importance proposal probability $q(\Theta|R,\ell)$. The clustering method in an atomic space ϖ_ℓ outputs a set of weighted cue particles $\mathcal{P}^\ell$. $\mathcal{P}^\ell$ encodes the non-parametric probability in ϖ_ℓ,

$$q(\Theta|\Lambda,\ell) = \sum_{i=1}^{m} \omega_i^\ell G(\Theta - \Theta_i^\ell), \quad \text{with} \ \sum_{i=1}^{m} \omega_i^\ell = 1, \tag{8.26}$$

where $G(x)$ is a Parzen window centered at 0. As a matter of fact, $q(\Theta|\Lambda,\ell) = q(\Theta|\mathbf{I})$ is an approximation to a marginal probability of the posterior $p(W|\mathbf{I})$ on cue space $\varpi_\ell, \ell \in \{g_1, g_2, g_3, g_4, c_1, c_3\}$, since the partition π is integrated out in EM-clustering.

$q(\Theta|\Lambda,\ell)$ is computed once for the whole image, and from this $q(\Theta|R,\ell)$ is computed for each R at run time. This method proceeds in the following way. Each cluster $\Theta_i^\ell, i = 1,2,...,m$ receives a real-valued vote from the pixel $v \in R$ in region R, and the cumulative vote is the summation of the saliency map S_i^ℓ associated with Θ_i^ℓ. Symbolically, p_i is given by

$$p_i = \frac{1}{|R|} \sum_{v \in R} S_{i,v}^\ell, \quad i = 1,2,...,m, \ \forall \ell.$$

Obviously the clusters which receive more votes should have a higher chance to be chosen. Thus, we sample a new image model Θ for region R,

$$\Theta \sim q(\Theta|R,\ell) = \sum_{i=1}^{m} p_i G(\Theta - \Theta_i^\ell). \tag{8.27}$$

Equation (8.27) explains how we choose (or propose) an image model for a region R. We first draw a cluster i at random according to probability $p = (p_1, p_2, ..., p_m)$ and then take a random perturbation at Θ_i^ℓ. Thus, any $\Theta \in \varpi_\ell$ has a non-zero probability to be chosen for robustness and ergodicity. Intuitively the clustering results with local votes propose the "hottest" portions of the space in a probabilistic way to guide the jump dynamics. In practice, one could implement a multi-resolution clustering algorithm over smaller local windows. In this case the clusters $\Theta_i^\ell, i = 1,2,...,m$ will be more effective at the expense of some overhead computing.

Computing importance proposal probability $q(\Gamma|R)$. By edge detection and tracing, we obtain partition maps denoted by $\Delta^{(s)}$ at multiple scales $s = 1,2,3$. In fact, each partition map $\Delta^{(s)}$ consists of a set of "meta-regions" $r_i^{(s)}$ such that

$$\Delta^{(s)}(\Lambda) = \{r_i^{(s)} : \ i = 1,2,...,n, \cup_{i=1}^{n} r_i^{(s)} = \Lambda\}, \quad \text{for } s = 1,2,3.$$

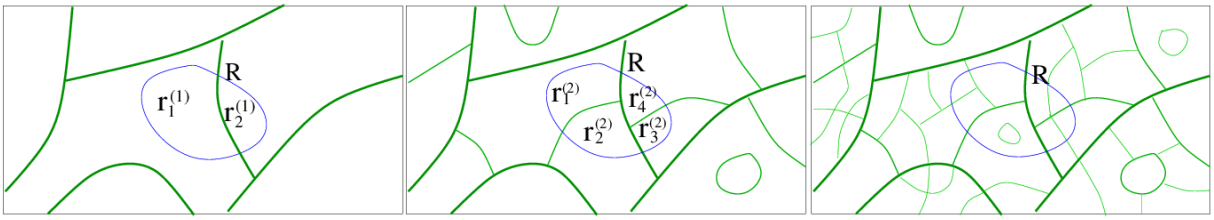


Fig. 8.19: A candidate region R_k is superimposed on the partition maps at three scales for computing a candidate boundary Γ_{ij} for the pending split. © [2002] IEEE. Reprinted, with permission, from ref. [59].

These meta-regions are used in combination to form $K \leq n$ regions $R_1^{(s)}, R_2^{(s)}, ..., R_K^{(s)}$,

$$R_i^{(s)} = \cup_j r_j^{(s)}, \quad \text{with } r_j^{(s)} \in \Delta^{(s)}, \ \forall i = 1, 2, ..., K.$$

One could further require that all meta-regions in a region $R_i^{(s)}$ are connected.

Let $\pi_k^{(s)} = (R_1^{(s)}, R_2^{(s)}, ..., R_k^{(s)})$ denote a k-partition based on $\Delta^{(s)}$. $\pi_k^{(s)}$ is different from the general k-partition π_k because regions $R_i^{(s)}, i = 1, ..., K$ in $\pi_k^{(s)}$ are limited to the meta-regions. We denote the set of all k-partitions based on a partition map $\Delta^{(s)}$ by

$$\Pi_k^{(s)} = \{(R_1^{(s)}, R_2^{(s)}, ..., R_k^{(s)}) = \pi_k^{(s)} : \quad \cup_{i=1}^k R_i^{(s)} = \Lambda\}. \tag{8.28}$$

We call each $\pi_k^{(s)}$ in $\Pi_k^{(s)}$ a k-partition particle in atomic (partition) space ϖ_{π_k}. Like the clusters in a cue space, $\Pi_k^{(s)}$ is a sparse subset of ϖ_{π_k}, and it narrows the search to the most promising portions. Therefore, each partition map $\Delta^{(s)}$ encodes a probability in the atomic (partition) space ϖ_{π_k}, and

$$q^{(s)}(\pi_k) = \frac{1}{|\Pi_k^{(s)}|} \sum_{j=1}^{|\Pi_k^{(s)}|} G(\pi_k - \pi_{k,j}^{(s)}), \quad \text{for } s = 1, 2, 3. \ \forall k. \tag{8.29}$$

$G()$ is a smooth window centered at 0 and its smoothness accounts for boundary deformations and forms a cluster around each partition particle, and $\pi_k - \pi_{k,j}^{(s)}$ measures the difference between two partition maps π_k and $\pi_{k,j}^{(s)}$. In the finest resolution, all meta-regions reduce to pixels, and $\Pi_k^{(s)}$ is then equal to the atomic space ϖ_{π_k}. In summary, the partition maps at all scales encode a non-parametric probability in ϖ_{π_k},

$$q(\pi_k) = \sum_s q(s) q^{(s)}(\pi_k), \quad \forall k.$$

This $q(\pi_k)$ can be considered as an approximation to the marginal posterior probability $p(\pi_k|\mathbf{I})$.

The partition maps $\Delta^{(s)}, \forall s$ (or $q(\pi_k), \forall k$ implicitly) are computed once for the whole image and then the importance proposal probability $q(\Gamma|R)$ is computed from $q(\pi_k)$ for each region as a conditional probability at run time, like in the cue spaces. Figure 8.19 illustrates an example. We show partition maps $\Delta^{(s)}(\Lambda)$ at three scales, and the edges are shown at width $3, 2, 1$ respectively for $s = 1, 2, 3$. A candidate region R is proposed to split. $q(\Gamma|R)$ is the probability for proposing a splitting boundary Γ. We superimpose R on the three partition maps. The intersections between R and the meta-regions generate three sets

$$\Delta^{(s)}(R) = \{r_j^{(s)} : \ r_j^{(s)} = R \cap r_j \text{ for } r_j \in \Delta^{(s)}(\Lambda), \text{ and } \cup_i r_i^{(s)} = R\}, \quad s = 1, 2, 3.$$

For example, in Figure 8.19, $\Delta^{(1)}(R) = \{r_1^{(1)}, r_2^{(1)}\}$, $\Delta^{(2)}(R) = \{r_1^{(2)}, r_2^{(2)}, r_3^{(2)}, r_4^{(2)}\}$, and so on.

Thus, we can define $\pi_c^{(s)}(R) = (R_1^{(s)}, R_2^{(s)}, ..., R_c^{(s)})$ as a c-partition of region R based on $\Delta^{(s)}(R)$, and define a c-partition space of R as

$$\Pi_c^{(s)}(R) = \{(R_1^{(s)}, R_2^{(s)}, ..., R_c^{(s)}) = \pi_c^{(s)}(R) : \quad \cup_{i=1}^{c} R_i^{(s)} = R\}, \quad \forall s. \tag{8.30}$$

We can define distributions on $\Pi_c^{(s)}(R)$ as

$$q^{(s)}(\pi_c(R)) = \frac{1}{|\Pi_c^{(s)}(R)|} \sum_{j=1}^{|\Pi_c^{(s)}(R)|} G(\pi_c - \pi_{c,j}^{(s)}(R)), \quad \text{for } s = 1,2,3, \ \forall c. \tag{8.31}$$

Thus one can propose to split R into c pieces, in a general case,

$$\pi_c(R) \sim q(\pi_c(R)) = \sum_s q(s) q^{(s)}(\pi_c(R)).$$

That is, we first select a scale s with probability $q(s)$. $q(s)$ depends on R. For example, for a large region R we can choose a coarse scale with higher probability, and choose a fine scale for small regions. We then choose a c-partition from the set $\Pi_c^{(s)}(R)$. In our implementation, $c = 2$ is chosen as a special case for easy implementation. It is trivial to show that an arbitrary c-partition of region R, $\pi_c(R)$, can be generated through composing $\pi_2(R)$ multiple times. Obviously there is a big overhead for choosing large c.

Computing $q(\Theta_i, \Theta_j | R, \ell_i, \ell_j)$ and $q(\Gamma_{ij} | R, \Theta_i, \Theta_j)$. In some cases, we find the second route, which we discussed for designing MCMC dynamics 3-4 (see equation (8.19)), useful for splitting a region. For example, there are two ways to perceive the zebra in Figure 8.21. One perceives the zebra as one textured region (by a model in ϖ_{g_3}). The other sees it as one region of black stripes plus one region of white stripes and thus uses the models in ϖ_{g_1} or ϖ_{g_4}. The Markov chain should be able to switch between the two perceptions effectively (see results in Figure 8.21.b-d). This is necessary and typical for the transitions between any texture regions and intensity regions.

Because the number of stripes in such textures is large, the first split procedure (route 1) is very ineffective, and it works on one stripe at a time. This motivates the second pathway for split dynamics. For a candidate region R, we first propose two new region models (we always assume the same labels $\ell_i = \ell_j$), this can be done by twice sampling the importance proposal probabilities $q(\Theta | R, \ell)$, so

$$(\Theta_i, \Theta_j) \sim q(\Theta_i, \Theta_j | R, \ell_i, \ell_j) = q(\Theta_i | R, \ell_i) q(\Theta_j | R, \ell_j).$$

Obviously we exclude Θ_i from the candidate set when we select Θ_j. We then decide on the boundary $\Gamma \sim q(\Gamma_{ij} | R, \Theta_i, \Theta_j)$ by randomly labeling the pixels in R according to probabilities of the saliency maps.

A unifying framework. To summarize this section, the DDMCMC paradigm provides a unifying framework for understanding the roles of many existing image segmentation algorithms. Firstly, edge detection and tracing methods [10, 35] compute implicitly a marginal probability $q(\pi | \mathbf{I})$ on the partition space ϖ_π. Secondly, clustering algorithms [12, 13] compute a marginal probability on the model space ϖ_ℓ for various models ℓ. Thirdly, the split-and-merge and model switching [3] realize jump dynamics. Fourthly, region growing and competition methods [49, 69] realize diffusion dynamics for evolving the region boundaries.

8.8 Computing Multiple Distinct Solutions

8.8.1 Motivation and a mathematical principle

The DDMCMC algorithm samples solutions from the posterior $W \sim p(W | \mathbf{I})$ endlessly. To extract an optimal result, an annealing strategy can be used with the conventional *maximum a posteriori* (MAP) estimator

$$W^* = \arg\max_{W \in \Omega} p(W | \mathbf{I}).$$

The desirable and often critical reasons for computing multiple distinct solutions are detailed below.

1. Natural scenes are intrinsically ambiguous, and for an image $\mathbf{I}$ many competing organizations and interpretations exist in visual perception.
2. For robustness, decisions should be left to the last stage of computation when a segmentation process is integrated with a specific task. Therefore, it is best to maintain a set of typical solutions.
3. Preserving multiple solutions is necessary when the prior and likelihood models are not perfect because the globally optimal solution may not be semantically more meaningful than some other inferior local maxima.

However, simply keeping a set of samples from the Markov chain sequence is not enough, because it often collects a set of segmentations that are trivially different from each other. Instead a mathematical principle for computing important and distinctive solutions in the space Ω can be used which relies on the techniques presented in Section 2.5 for preserving sample diversity in importance sampling.

Let $S = \{(\omega_i, W_i) : i = 1, ..., K\}$ be a set of K weighted solutions which we call scene particles, with their weights being their posterior probabilities $\omega_i = p(W|\mathbf{I}), i = 1, 2, ..., K$. (Note that there is a slight abuse of notation, we use K for the number of regions in W before. Here it is a different K). S encodes a non-parametric probability in Ω and

$$\hat{p}(W|\mathbf{I}) = \sum_{i=1}^{K} \frac{\omega_i}{\omega} G(W - W_i), \quad \sum_{i=1}^{K} \omega_i = \omega,$$

where G is a Gaussian window in Ω.

As all image ambiguities are captured in the Bayesian posterior probability, to reflect the intrinsic ambiguities, we should compute the set of solutions S that best preserve the posterior probability. Thus, we let $\hat{p}(W|\mathbf{I})$ approach $p(W|\mathbf{I})$ by minimizing the Kullback-Leibler divergence $D(p||\hat{p})$ under a complexity constraint $|S| = K$,

$$S^* = \arg\min_{|S|=K} D(p||\hat{p}) = \arg\min_{|S|=K} \int p(W|\mathbf{I}) \log \frac{p(W|\mathbf{I})}{\hat{p}(W|\mathbf{I})} dW. \tag{8.32}$$

This criterion extends the conventional MAP estimator.

8.8.2 A K-adventurers algorithm for multiple solutions

Fortunately, the KL-divergence $D(p||\hat{p})$ can be estimated fairly accurately by a distance measure $\hat{D}(p||\hat{p})$, which is computable, thanks to two observations of the multi-modal posterior probability $p(W|\mathbf{I})$. We can always represent $p(W|\mathbf{I})$ by a mixture of Gaussian, i.e. a set of N particles with N large enough. By ergodicity, the Markov chain is supposed to visit these significant modes over time! Thus our goal is to extract K distinct solutions from the Markov chain sampling process.

In order to accomplish this goal, an algorithm known as K-adventurers can be used. [4] Suppose we have a set of K particles S at step t. At time $t+1$, we obtain a new particle (or a number of particles) by MCMC, usually following a successful jump. We augment the set S to S_+ by adding the new particle(s). We then eliminate one particle (or a number of particles) from S_+ to get S_{new} by minimizing the approximative KL divergence $\hat{D}(p_+||p_{\text{new}})$.
In practice, we run multiple Markov chains and add new particles to the set S in a batch fashion.

8.9 Image Segmentation Experiments

The DDMCMC paradigm was tested extensively on many grey level, color, and textured images. This section shows some examples and more are available on our website[5]. It was also tested in a benchmark dataset of 50 natural images

[4] The name follows a statistics metaphor told by Mumford to S.C. Zhu. A team of K adventurers want to occupy K largest islands in an ocean while keeping apart from each other's territories.

[5] See http://vcla.stat.ucla.edu/old/Segmentation/Segment.htm

Algorithm 8 The K-adventurers algorithm

1. Initialize S and $\hat{p}$ by repeating one initial solution K times.

repeat

2. Compute a new particle $(\omega_{K+1}, \mathbf{x}_{K+1})$ by DDMCMC after a successful jump.
3. $S_+ \leftarrow S \bigcup \{(\omega_{K+1}, \mathbf{x}_{K+1})\}$.
4. $\hat{p} \leftarrow S_+$.

for $i = 1, 2, \ldots, K+1$ **do**

5. $S_{-i} \leftarrow S_+ / \{(\omega_i, \mathbf{x}_i)\}$.
6. $\hat{p}_{-i} \leftarrow S_{-i}$.
7. $d_i = D(p||\hat{p}_{-i})$.

end for

8. $i^* = \arg\min_{i \in \{1,2,\ldots,K+1\}} d_i$.
9. $S \leftarrow S_{-i^*}, \ \hat{p} \leftarrow \hat{p}_{-i^*}$.

until desired number of iterations

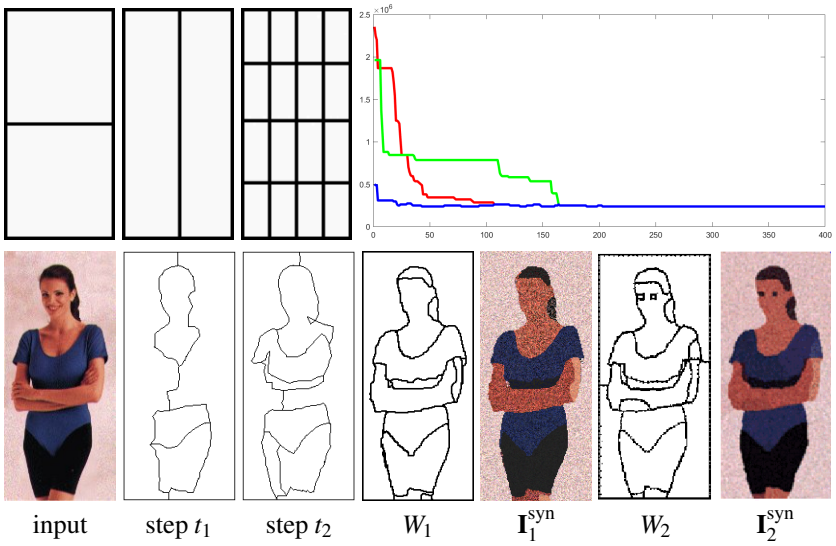


Fig. 8.20: Segmenting a color image by DDMCMC with two solutions. See text for explanation. © [2002] IEEE. Reprinted, with permission, from ref. [59].

in both color and grey level [41] by the Berkeley group[6], where the results by DDMCMC and other methods such as [53] are displayed in comparison to those by a number of human subjects. Each tested algorithm uses the same parameter setting for all the benchmark images and thus the results were obtained purely automatically.

We first show our working example on the color woman image. Following the importance proposal probabilities for the edges in Figure 8.17 and for color clustering in Figure 8.11, we simulated three Markov chains with three different initial segmentations shown in Figure 8.20 (top row). The energy changes ($-\log p(W|\mathbf{I})$) of the three MCMCs are plotted in Figure 8.20 against time s. Figure 8.20 shows two different solutions W_1, W_2 obtained by a Markov chain using the K-adventurers algorithm. To verify the computed solution W_i, we synthesized an image by sampling from the likelihood $\mathbf{I}_i^{syn} \sim p(\mathbf{I}|W_i), i = 1, 2$. The synthesis is a good way to examine the sufficiency of models in segmentation.

Figure 8.21 shows three segmentations on a gray level zebra image. As we discussed before, the DDMCMC algorithm in this section has only one free parameter γ which is a "clutter factor" in the prior model (See equation (8.7)). It controls the extents of segmentations. A big γ encourages coarse segmentation with large regions. We normally extract results at three scales by setting $\gamma = 1.0, 2.0, 3.0$ respectively. In our experiments, the K-adventurers algorithm is effective only for computing distinct solutions in a certain scale. We expect the parameter γ can be fixed to a constant if we form an image pyramid with multiple scales and conduct segmentation with the K-adventurers algorithm at each scale, and then propagate and refine the results to the next finer scale sequentially.

[6] See https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/

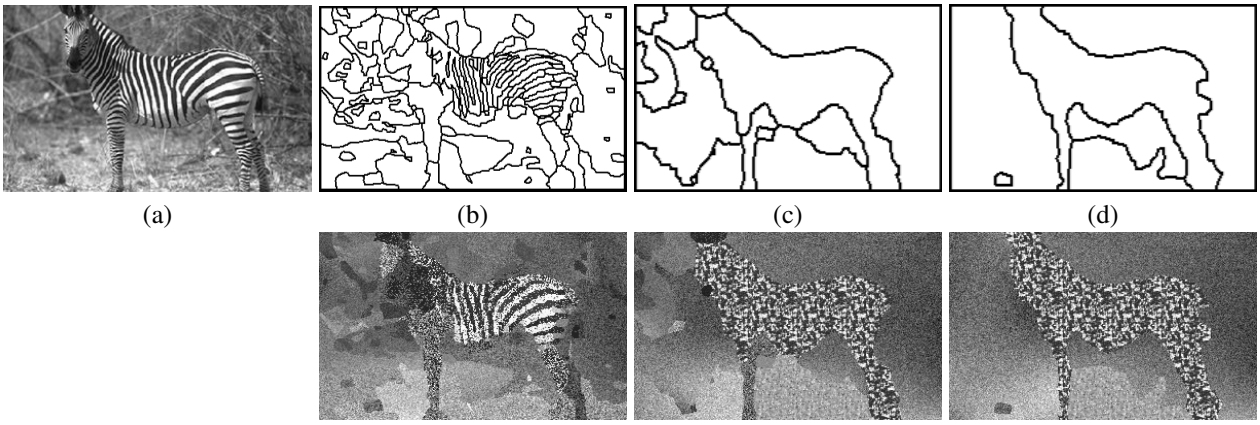


Fig. 8.21: Experiments on the grey level zebra image with three solutions. (a) input image. (b)-(d) are three solutions, $W_i, i = 1,2,3$, for the zebra image. (e)-(g) are synthesized images $\mathbf{I}_i^{\text{syn}} \sim p(\mathbf{I}|W_i^*)$ for verifying the results. © [2002] IEEE. Reprinted, with permission, from ref. [59].

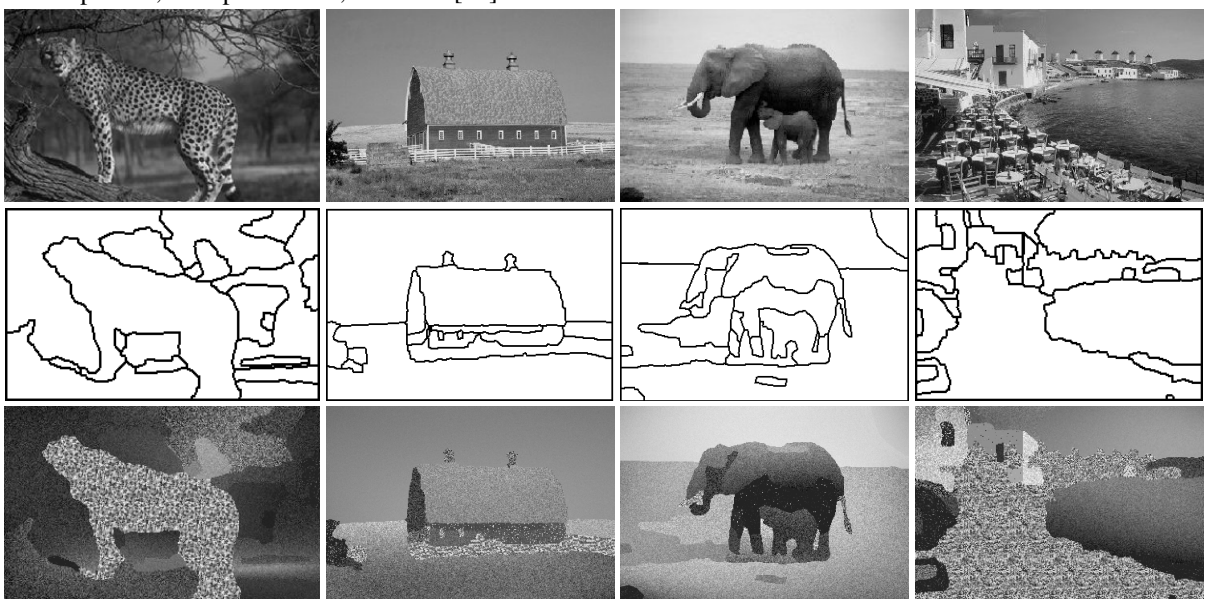


Fig. 8.22: Grey level image segmentation by DDMCMC. Top: input images, middle: segmentation results W, bottom: synthesized images $\mathbf{I}^{\text{syn}} \sim p(\mathbf{I}|W)$ with the segmentation results W. © [2002] IEEE. Reprinted, with permission, from ref. [59].

For the zebra image, W_1 segments out the black and white stripes while W_2 and W_3 treat the zebra as a texture region. The synthesized images $\mathbf{I}_i^{\text{syn}} \sim p(\mathbf{I}|W_i), i = 1,2,3$ show that the texture model is not sufficient because we choose only 8 small filters for computational ease. Also the spline surface model plays an important role in segmenting the ground and background grass, and this is verified by the global shading changes in $\mathbf{I}_2^{\text{syn}}$ and $\mathbf{I}_3^{\text{syn}}$.

Figures 8.22 and 8.23 display some other grey level and color images using the same algorithm. We show the input, a segmentation starting with arbitrary initial conditions and a synthesized image drawn from the likelihood $\mathbf{I}^{\text{syn}} \sim p(\mathbf{I}|W)$. The γ values for these images are mostly set up as 1.5 with a few obtained at 1.0-3.5. It took about 10-30 minutes, depending upon the complexity of image contents, on a Pentium III PC to segment an image with medium size such as 350×250 pixels after learning the pseudo-likelihood texture models at the beginning.

The synthesis images show that we need to engage more stochastic models such as point and curve processes, and objects like faces etc. For example, in the first row of Figure 8.23. The music band in a football stadium forms a point process which is not captured. The face is also missing in the synthesis.

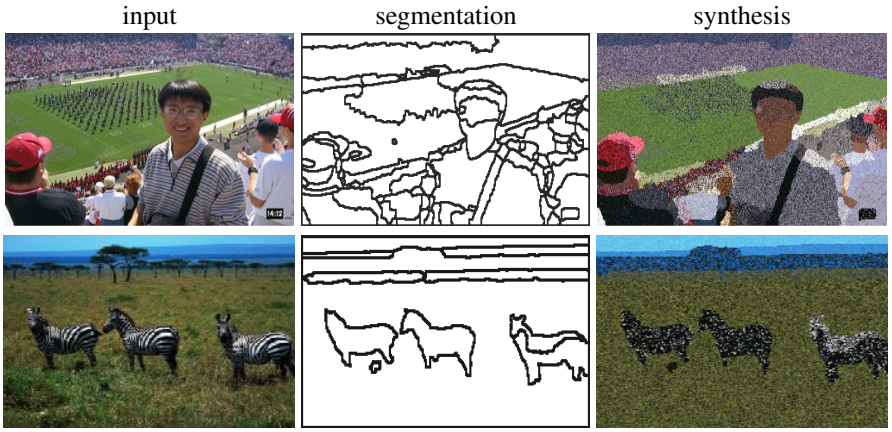


Fig. 8.23: Color image segmentation by DDMCMC. Left: input images, middle: segmentation results W, right: synthesized images $\mathbf{I}^{syn} \sim p(\mathbf{I}|W)$ with the segmentation results W. © [2002] IEEE. Reprinted, with permission, from ref. [59].

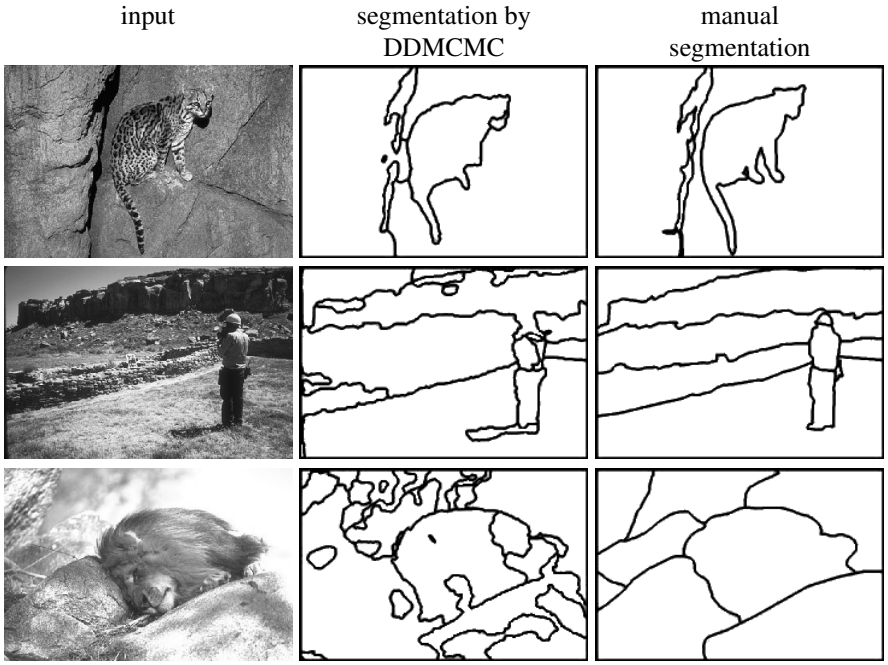


Fig. 8.24: Some segmentation results by DDMCMC for the benchmark test by Martin. The errors for the above results by DDMCMC (middle) compared with the results by a human subject (right) are 0.1083, 0.3082, and 0.5290 respectively according to their metrics. © [2002] IEEE. Reprinted, with permission, from ref. [59].

Figure 8.24 shows three grey level images out of the 50 natural images in both color and grey level for the benchmark study. The input (left), the segmentation results by DDMCMC (middle), and the manual segmentation by a human subject (right) are displayed.

8.10 Application: Image Parsing

We define image parsing to be the task of decomposing an image $\mathbf{I}$ into its constituent visual patterns. The output is represented by a hierarchical graph W – called the "parse graph". The goal is to optimize the Bayesian posterior

probability $p(W|\mathbf{I})$. Figure 8.25 illustrates a typical example where a football scene is first divided into three parts at a coarse level: a person in the foreground, a sports field, and the spectators. These three parts are further decomposed into nine visual patterns in the second level: a face, three texture regions, some text, a point process (the band on the field), a curve process (the markings on the field), a color region, and a region for nearby people. In principle, we can continue decomposing these parts until we reach a resolution criterion. The parse graph is similar in spirit to the parsing trees used in speech and natural language processing [39] except that it can include horizontal connections (see the dashed curves in Figure 8.25) for specifying spatial relationships and boundary sharing between different visual patterns.

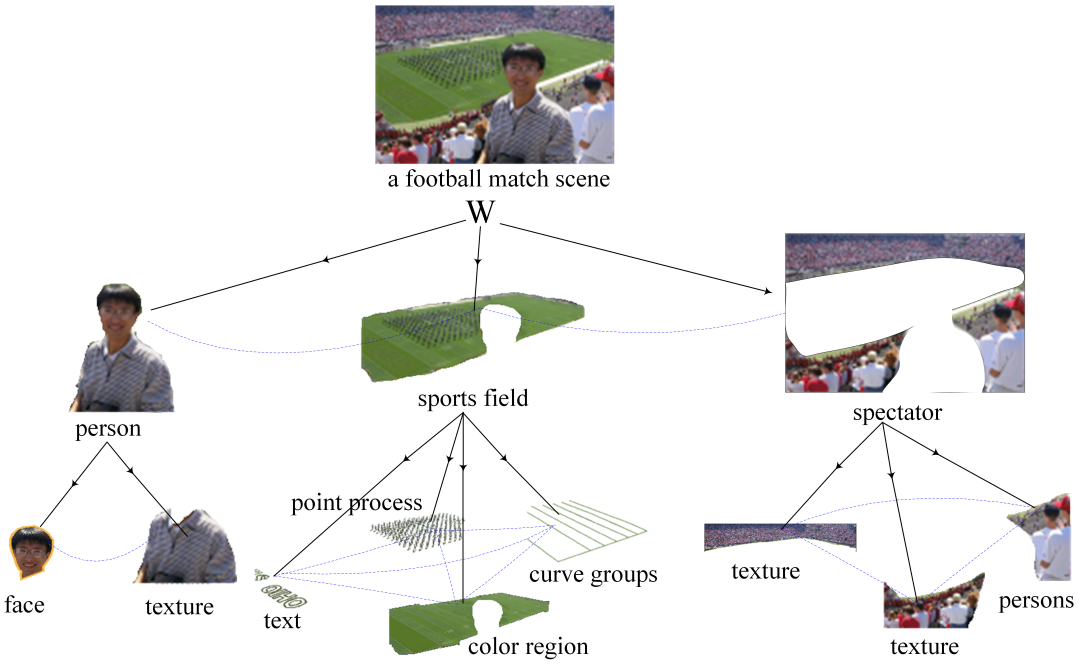


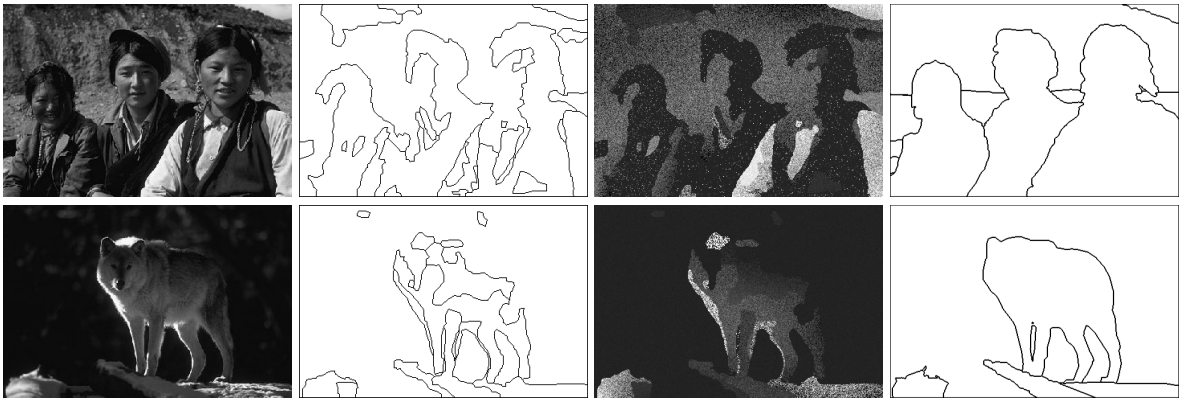
Fig. 8.25: Image parsing example. The parse graph is hierarchical and combines generative models (downward arrows) with horizontal connections (dashed lines), which specify spatial relationships between the visual patterns. See Figure 8.28 for a more abstract representation including variables for the node attributes. © [2005] Springer. Reprinted, with permission, from ref. [57].

As in natural language processing, the parse graph is not fixed and depends on the input image(s). An image parsing algorithm must *construct* the parse graph on the fly[7]. Our image parsing algorithm consists of a set of reversible Markov chain jumps [26] with each type of jump corresponding to an operator for *reconfiguring* the parse graph (i.e. creating or deleting nodes or changing the values of node attributes). These jumps combine to form an ergodic and reversible Markov chain in the space of possible parse graphs. The Markov chain probability is guaranteed to converge to the invariant probability $p(W|\mathbf{I})$ and the Markov chain will simulate fair samples from this probability.[8]. This approach is built on previous work on Data-Driven Markov Chain Monte Carlo (DDMCMC) for recognition [70], segmentation [59], grouping [60] and graph partitioning [1, 2].

Image parsing seeks a full generative explanation of the input image in terms of generative models, $p(\mathbf{I}|W)$ and $p(W)$, for the diverse visual patterns that occur in natural images, see Figure 8.25. This differs from other computer vision tasks, such as segmentation, grouping, and recognition. These are usually performed by isolated vision modules which only seek to explain parts of the image. The image parsing approach enables these different modules to cooperate and compete to give a consistent interpretation of the entire image.

[7] Unlike most graphical inference algorithms in the literature which assume fixed graphs, see belief propagation [66].

[8] For many natural images the posterior probabilities $P(W|\mathbf{I})$ are strongly peaked and so stochastic samples are close to the maxima of the posterior. So in this section we do not distinguish between sampling and inference (optimization).



a. Input image b. Segmentation c. Synthesized image d. Manual segmentation

Fig. 8.26: Examples of image segmentation failure by DDMCMC, which uses only generic visual patterns (i.e. only low-level visual cues). The results (b) show that low-level visual cues are not sufficient to obtain good intuitive segmentations. The limitations of using only generic visual patterns are also clear in the synthesized images (c) which are obtained by stochastic sampling from the generative models after the parameters have been estimated by DDMCMC. The right panels (d) show the segmentations obtained by human subjects who, by contrast to the algorithm, appear to use object specific knowledge when doing the segmentation (though they were not instructed to) [41]. We conclude that to achieve good segmentation on these types of images requires combining segmentation with object detection and recognition. © [2005] Springer. Reprinted, with permission, from ref. [57].

But combining different visual modules requires a common framework that ensures consistency. Despite the effectiveness of discriminative methods for computing scene components, such as object labels and categories, they can also generate redundant and conflicting results. Mathematicians have argued [6] that discriminative methods must be followed by more sophisticated processes to (i) remove false alarms, (ii) amend missing objects by global context information, and (iii) reconcile conflicting (overlapping) explanations through model comparison. In this section, we impose such processes by using generative models for the entire image.

As we will show, the image parsing algorithm is able to integrate discriminative and generative methods so as to take advantage of their complementary strengths. Moreover, modules such as segmentation and object detection can be coupled by the choice of the set of visual patterns used to parse the image. In this section, we focus on two types of patterns: – generic visual patterns for low/middle level vision, such as texture and shading, and object patterns at high level vision, such as frontal human faces and text.

These two types of patterns illustrate different ways in which the parse graph can be constructed (see Figure 8.40 and the related discussion). The object patterns (face and text) have comparatively little variability so they can often be effectively detected as a whole by bottom-up tests and their parts can be located subsequentially. Thus their parsing sub-graphs can be constructed in a "decompositional" manner from whole to parts. By contrast, a generic texture region has arbitrary shape and its intensity pattern has high entropy. Detecting such a region by bottom-up tests will require an enormous number of tests to deal with all this variability, and so will be computationally impractical. Instead, the parsing subgraphs should be built by grouping small elements in a "compositional" manner [5].

We illustrate the image parsing algorithm on natural images of complex city scenes and give examples where image segmentation can be improved by allowing object specific knowledge to disambiguate low-level cues, and conversely object detection can be improved by using generic visual patterns to explain away shadows and occlusions.

8.10.1 Bottom-up and top-down processing

A major element of DDMCMC is integrating discriminative and generative methods for inference. Top-down and bottom-up procedures can be broadly categorized into two popular inference paradigms – *generative* methods for top-down and *discriminative* methods for bottom-up, illustrated in Figure 8.27. From this perspective, integrating generative and discriminative models is equivalent to combining bottom-up and top-down processing.

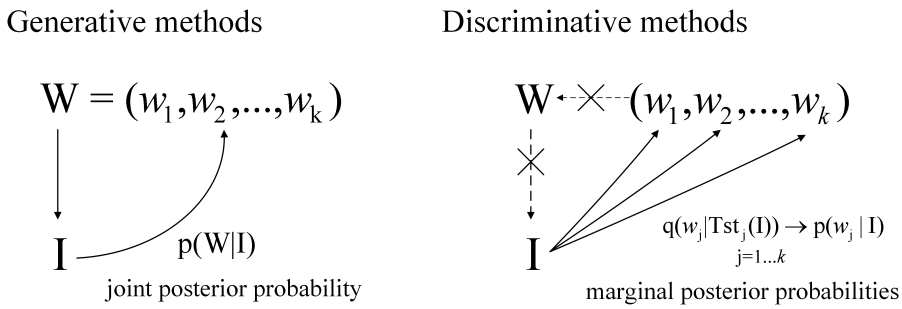


Fig. 8.27: Comparison of two inference paradigms: Top-down *generative* methods versus bottom-up *discriminative* methods. The generative methods specify how the image $\mathbf{I}$ can be synthesized from the scene representation W. On the other hand, the discriminative methods work by performing tests $Tst_j(\mathbf{I})$ and are not guaranteed to yield consistent solutions. Crosses are explained in the text.

The role of bottom-up and top-down processing in vision has been greatly discussed. There is growing evidence (see [36, 55]) that humans can perform high level scene and object categorization tasks as fast as low level texture discrimination and other so-called pre-attentive vision tasks. This suggests that humans can detect both low and high level visual patterns at early stages in visual processing. This stands in contrast to traditional bottom-up feedforward architectures [40], which start with edge detection, followed by segmentation/grouping, before proceeding to object recognition and other high-level vision tasks. The experiments also relate to long standing conjectures about the role of the bottom-up/top-down loops in the visual cortical areas [45, 62], visual routines and pathways [61], the binding of visual cues [56], and neural network models such as the Helmholtz machine [16]. DDMCMC unifies both methods by using discriminative methods to perform rapid inference of the parameters of generative models. From a computer vision perspective, DDMCMC combines bottom-up processing, implemented by the discriminative models, together with top-down processing by the generative models. The rest of this section gives more details about this feature of the algorithm.

8.10.2 Generative and discriminative methods

Generative methods specify how the image $\mathbf{I}$ is generated from the scene representation $W \in \Omega$. It combines a prior $p(W)$ and a likelihood function $p(\mathbf{I}|W)$ to give a joint posterior probability $p(W|\mathbf{I})$. These can be expressed as probabilities on graphs, where the input image $\mathbf{I}$ is represented on the leaf nodes and W denotes the remaining nodes and node attributes of the graph. The structure of the graph, for example the number of nodes, is unknown and must be estimated for each input image. Inference can be performed by stochastic sampling W from the posterior

$$W \sim p(W|\mathbf{I}) \propto p(\mathbf{I}|W)p(W). \tag{8.33}$$

This enables us to estimate $W^* = \arg\max P(W|\mathbf{I})$.[9] But the dimension of the sample space Ω is very high and so standard sampling techniques are computationally expensive.

By contrast, discriminative methods are very easy to compute. They do not specify models for how the image is generated. Instead they give discriminative (conditional) probabilities $q(w_j|\mathrm{Tst}_j(\mathbf{I}))$ for components $\{w_j\}$ of W based on a sequence of bottom-up tests $\mathrm{Tst}_j(\mathbf{I})$ performed on the image. The tests are based on local image features $\{F_{j,n}(\mathbf{I})\}$, which can be computed from the image in a cascade manner (e.g. AdaBoost filters, see Section (8.10.5.3)),

$$\mathrm{Tst}_j(\mathbf{I}) = (F_{j,1}(\mathbf{I}), F_{j,2}(\mathbf{I}), ..., F_{j,n}(\mathbf{I})), \quad j = 1, 2, ..., K. \tag{8.34}$$

The following theorem shows that the KL-divergence between the true marginal posterior $p(w_j|\mathbf{I})$ and the optimal discriminant approximation $q(w_j|\mathrm{Tst}(\mathbf{I}))$ using test $\mathrm{Tst}(\mathbf{I})$ will decrease monotonically as new tests are added[10].

[9] We are assuming that there are no known algorithms for estimating W^* directly.

[10] The optimal approximation occurs when $q(w_j|\mathrm{Tst}(\mathbf{I}))$ equals the probability $p(w_j|\mathrm{Tst}(\mathbf{I}))$ induced by $P(\mathbf{I}|W)P(W)$.

Theorem 8.1 *The information gained for a variable w by a new test* $\text{Tst}_+(\mathbf{I})$ *is the decrease of Kullback-Leibler divergence between* $p(w|\mathbf{I})$ *and its best discriminative estimate* $q(w|\text{Tst}_t(\mathbf{I}))$ *or the increase of mutual information between w and the tests.*

$$\begin{aligned} & E_{\mathbf{I}}[KL(p(w|\mathbf{I}) \,||\, q(w|\text{Tst}(\mathbf{I})))] - E_{\mathbf{I}}[KL(p(w|\mathbf{I}) \,||\, q(w|\text{Tst}(\mathbf{I}), \text{Tst}_+(\mathbf{I})))] \\ = & \, MI(w \,||\, \text{Tst}, \text{Tst}_+) - MI(w \,||\, \text{Tst}) = E_{\text{Tst},\text{Tst}_+} KL(q(w \,|\, \text{Tst}_t, \text{Tst}_+) \,||\, q(w \,|\, \text{Tst}_t) \geq 0, \end{aligned}$$

where $E_{\mathbf{I}}$ *is the expectation with respect to* $P(\mathbf{I})$*, and* $E_{\text{Tst},\text{Tst}_+}$ *is the expectation with respect to the probability on the test responses* $(\text{Tst}, \text{Tst}_+)$ *induced by* $P(\mathbf{I})$*. The decrease of the Kullback-Leibler divergence equals zero if and only if* $\text{Tst}(\mathbf{I})$ *are sufficient statistics with respect to w.*

In practice discriminative methods, particularly standard computer vision algorithms – see subsection (8.10.4.5), will typically only use a small number of features for computational practicality. Also the discriminative probabilities $q(w_j|\text{Tst}(\mathbf{I}))$ will often not be optimal. Fortunately, the image parsing algorithm in this section only requires the discriminative probabilities $q(w_j|\text{Tst}(\mathbf{I}))$ to be rough approximations to $p(w_j|\mathbf{I})$.

The difference between discriminative and generative models is illustrated in Figure 8.27. Discriminative models are easy to compute and can be run in parallel because different components are computed independently (see arrows in Figure 8.27). But the components $\{w_i\}$ may not yield a consistent solution W and, moreover, W may not specify a consistent model for generating the observed image $\mathbf{I}$. These inconsistencies are indicated by the crosses in Figure 8.27. Generative models ensure consistency but require solving a difficult inference problem. It is an open problem whether discriminative models can be designed to infer the entire state W for the complicated generative models that we are dealing with. Mathematicians [6] have argued that this will not be practical and that discriminative models will always require additional post-processing.

8.10.3 Markov chain kernels and sub-kernels

Formally, the DDMCMC image parsing algorithm simulates a Markov chain $\mathcal{MC} =< \Omega, \nu, \mathcal{K} >$ with kernel $\mathcal{K}$ in space Ω and with probability ν for the starting state. $W \in \Omega$ is a parse graph and we let the set of parse graphs Ω be finite as images have finite pixels and finite grey levels. We proceed by defining a set of moves for reconfiguring the graph. These include moves to create nodes, delete nodes, and change node attributes. We specify stochastic dynamics for these moves in terms of transition kernels[11].

For each move we define a Markov Chain sub-kernel by a transition matrix $\mathcal{K}_a(W'|W:\mathbf{I})$ with $a \in \mathcal{A}$ being an index. This represents the probability that the system makes a transition from state W to state W' when sub-kernel a is applied (i.e. $\sum_{W'} \mathcal{K}_a(W'|W:\mathbf{I}) = 1, \, \forall \, W$). Kernels which alter the graph structure are grouped into reversible pairs. For example, the sub-kernel for node creation $\mathcal{K}_{a,r}(W'|W:\mathbf{I})$ is paired with the sub-kernel for node deletion $\mathcal{K}_{a,l}(W'|W:\mathbf{I})$. This can be combined into a paired sub-kernel $\mathcal{K}_a = \rho_{ar}\mathcal{K}_{a,r}(W'|W:\mathbf{I}) + \rho_{al}\mathcal{K}_{a,l}(W'|W:\mathbf{I})$ ($\rho_{ar} + \rho_{al} = 1$). This pairing ensures that $\mathcal{K}_a(W'|W:\mathbf{I}) = 0$ if, and only if, $\mathcal{K}_a(W|W':\mathbf{I}) = 0$ for all states $W, W' \in \Omega$. The sub-kernels (after pairing) are constructed to obey the detailed balance equation

$$p(W|\mathbf{I})\mathcal{K}_a(W'|W:\mathbf{I}) = p(W'|\mathbf{I})\mathcal{K}_a(W|W':\mathbf{I}). \tag{8.35}$$

The full transition kernel is expressed as

$$\mathcal{K}(W'|W:\mathbf{I}) = \sum_a \rho(a:\mathbf{I})\mathcal{K}_a(W'|W:\mathbf{I}), \;\; \sum_a \rho(a:\mathbf{I}) = 1, \;\; \rho(a:\mathbf{I}) > 0. \tag{8.36}$$

To use this kernel, at each time step the algorithm selects a move a with probability $\rho(a:\mathbf{I})$ and then uses kernel $\mathcal{K}_a(W'|W;\mathbf{I})$ to select the transition from state W to state W'. Note that both probabilities $\rho(a:\mathbf{I})$ and $\mathcal{K}_a(W'|W;\mathbf{I})$ depend on the input image $\mathbf{I}$. This distinguishes the DDMCMC methods from conventional MCMC computing [9, 37].

The full kernel obeys detailed balance (equation (8.35)), because every sub-kernel does. It will also be ergodic, provided the set of moves is sufficient (i.e. so that we can transition between any two states $W, W' \in \Omega$ using these

[11] We choose stochastic dynamics because the Markov chain probability is guaranteed to converge to the posterior $P(W|\mathbf{I})$. The complexity of the problem means that deterministic algorithms for implementing these moves risk getting stuck in local minima.

moves). These two conditions ensure that $p(W|\mathbf{I})$ is the invariant (target) probability of the Markov Chain [9] in the finite space Ω. Applying the kernel $\mathcal{K}_{a(t)}$ updates the Markov chain state probability $\mu_t(W)$ at step t to $\mu_{t+1}(W')$ at $t+1$,

$$\mu_{t+1}(W') = \sum_{W} \mathcal{K}_{a(t)}(W'|W:\mathbf{I})\mu_t(W). \tag{8.37}$$

In summary, the DDMCMC image parser simulates a Markov chain $\mathcal{MC}$ with a unique invariant probability $p(W|\mathbf{I})$. At time t, the Markov chain state (i.e. the parse graph) W follows a probability μ_t which is the product of the sub-kernels selected up to time t,

$$W \sim \mu_t(W) = \nu(W_o) \cdot [\mathcal{K}_{a(1)} \circ \mathcal{K}_{a(2)} \circ \cdots \circ \mathcal{K}_{a(t)}](W_o, W) \longrightarrow p(W|\mathbf{I}), \tag{8.38}$$

where $a(t)$ indexes the sub-kernel selected at time t. As the time t increases, $\mu_t(W)$ approaches the posterior $p(W|\mathbf{I})$ monotonically [9] at a geometric rate [19]. The following convergence theorem is useful for image parsing because it helps quantify the effectiveness of the different sub-kernels.

Theorem 8.2 *The Kullback-Leibler divergence between the posterior $p(W|\mathbf{I})$ and the Markov chain state probability decreases monotonically when a sub-kernel $\mathcal{K}_{a(t)}, \forall\, a(t) \in \mathcal{A}$ is applied,*

$$KL(p(W|\mathbf{I}) \,||\, \mu_t(W)) - KL(p(W|\mathbf{I}) \,||\, \mu_{t+1}(W)) \geq 0. \tag{8.39}$$

The decrease of KL-divergence is strictly positive and is equal to zero only after the Markov chain becomes stationary, i.e. $\mu = p$.

The proof of this result is found in [57]. The theorem is related to the second law of thermodynamics [15], and its proof makes use of the detailed balance equation (8.35). This *KL* divergence gives a measure of the "power" of each sub-kernel $\mathcal{K}_{a(t)}$ and so it suggests an efficient mechanism for selecting the sub-kernels at each time step. By contrast, classic convergence analysis shows that the convergence of the Markov chain is exponentially fast but does not give measures for the power of sub-kernels.

8.10.4 DDMCMC and proposal probabilities

We now describe how to design the sub-kernels using proposal probabilities and discriminative models. This is at the heart of DDMCMC. Each sub-kernel[12] is designed to have the form of Metropolis-Hastings [29, 43] such that

$$\mathcal{K}_a(W'|W:\mathbf{I}) = Q_a(W'|W:\mathrm{Tst}_a(\mathbf{I}))\min\{1, \frac{p(W'|\mathbf{I})Q_a(W|W':\mathrm{Tst}_a(\mathbf{I}))}{p(W|\mathbf{I})Q_a(W'|W:\mathrm{Tst}_a(\mathbf{I}))}\},\ \ W' \neq W, \tag{8.40}$$

where a transition from W to W' is proposed (stochastically) by the proposal probability $Q_a(W'|W:\mathrm{Tst}_a(\mathbf{I}))$ and accepted (stochastically) by the acceptance probability

$$\alpha(W'|W:\mathbf{I}) = \min\{1, \frac{p(W'|\mathbf{I})Q_a(W|W':\mathrm{Tst}_a(\mathbf{I}))}{p(W|\mathbf{I})Q_a(W'|W:\mathrm{Tst}_a(\mathbf{I}))}\}. \tag{8.41}$$

Metropolis-Hastings ensures that the sub-kernels obey detailed balance (after pairing).

The proposal probability $Q_a(W'|W:\mathrm{Tst}_a(\mathbf{I}))$ is a product of discriminative probabilities $q(w_j|\mathrm{Tst}_j(\mathbf{I}))$ for the respective elements w_j changed in the move between W and W'. $\mathrm{Tst}_a(\mathbf{I})$ is the set of bottom-up tests used in the proposal probabilities $Q_a(W'|W:\mathrm{Tst}_a(\mathbf{I}))$ and $Q_a(W|W':\mathrm{Tst}_a(\mathbf{I}))$. The proposal probabilities must be easy to compute (because they should be evaluated for all possible states W' that sub-kernel a can reach) and should propose transitions to states W' where the posterior $p(W'|\mathbf{I})$ is likely to be high. The acceptance probabilities are more computationally expensive, due to their dependence on $p(W'|\mathbf{I})$, but they only need to be evaluated for the proposed state.

The design of the proposals involves a trade-off. Ideally the proposals would be sampled from the posterior $p(W'|\mathbf{I})$, but this is impractical. Instead the trade-off requires choosing between fast computations and large movements to states

[12] Except for one that evolves region boundaries.

with high posterior probabilities. More formally, we define the scope $\Omega_a(W) = \{W' \in \Omega : \mathcal{K}_a(W'|W : \mathbf{I}) > 0\}$ to be the set of states which can be reached from W in one time step using sub-kernel a. We want the scope $S_a(W)$ to be large so that we can make large moves in at each time step (i.e. jump towards the solution, not crawl). The scope should also, if possible, include states W' with high posterior $p(W'|\mathbf{I})$ (i.e. the scope should also be in the right part of Ω).

The proposals $Q_a(W'|W : \text{Tst}_a(\mathbf{I}))$ should be chosen so as to approximate

$$\frac{p(W'|\mathbf{I})}{\sum_{W'' \in \Omega_a(W)} p(W''|\mathbf{I})} \text{ if } W' \in \Omega_a(W), \text{ or 0 otherwise.} \tag{8.42}$$

They will be functions of the discriminative models for the components of W' and of the generative models for the current state W (because it is computationally cheap to evaluate the generative models for the current state). The details of the model $p(W|\mathbf{I})$ will determine the form of the proposals and how large we can make the scope while keeping the proposals easy to compute and able to approximate equation (8.42). See the detailed examples given in Section (8.10.5).

This overview gives the bare bones of DDMCMC. We refer to [60] for a more sophisticated discussion of these issues from an MCMC perspective.

8.10.4.1 Generative models and Bayesian formulation

This section describes the parse graph and the generative models used for the image parsing algorithm.

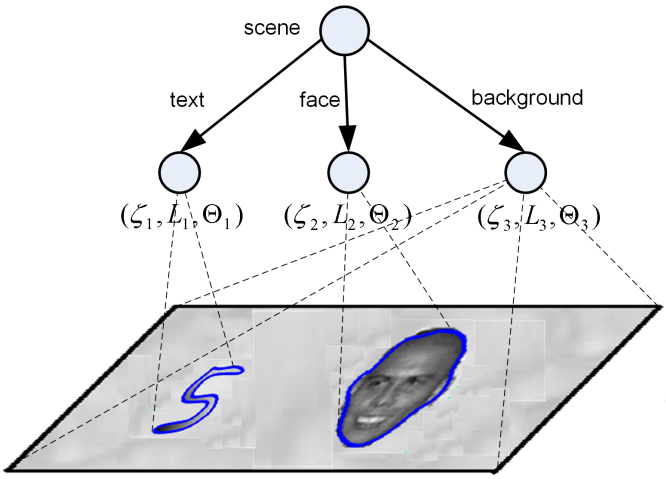


Fig. 8.28: Abstract representation of the parse graph used in this section. The intermediate nodes represent the visual patterns. Their child nodes correspond to the pixels in the image. © [2005] Springer. Reprinted, with permission, from ref. [57].

Figure 8.25 illustrates the general structure of a parse graph. In this section, we consider the simplified two-layer-graph illustrated in Figure 8.28, which is fully specified in a generative sense. The top node ("root") of the graph represents the whole scene (with a label). It has K intermediate nodes for the visual patterns (face, text, texture, and shading). Each visual pattern has a number of pixels at the bottom ("leaves"). In this graph no horizontal connections are considered between the visual patterns except that they share boundaries and form a partition of the image lattice.

The number K of intermediate nodes is a random variable, and each node $i = 1, ..., K$ has a set of attributes (L_i, ζ_i, Θ_i) defined as follows. L_i is the shape descriptor and determines the region $R_i = R(L_i)$ of the image pixels covered by the visual pattern of the intermediate node. Conceptually, the pixels within R_i are child nodes of the intermediate node i. (Regions may contain holes, in which case the shape descriptor will have internal and external boundaries). The remaining attribute variables (ζ_i, Θ_i) specify the probability models $p(\mathbf{I}_{R(L_i)}|\zeta_i, L_i, \Theta_i)$ for generating the sub-image $\mathbf{I}_{R(L_i)}$ in region $R(L_i)$. The variables $\zeta_i \in \{1, ..., 66\}$ indicate the visual pattern type (3 types of generic visual patterns, 1 face pattern, and 62 text character patterns), and Θ_i denotes the model parameters for the corresponding visual pattern. The complete scene description can be summarized by:

$$W = (K, \{(\zeta_i, L_i, \Theta_i) : i = 1, 2, ..., K\}).$$

The shape descriptors $\{L_i : i = 1, ..., K\}$ are required to be consistent so that each pixel in the image is a child of one, and only one, of the intermediate nodes. The shape descriptors must provide a partition of the image lattice $\Lambda = \{(m,n) : 1 \leq m \leq Height(\mathbf{I}), 1 \leq n \leq Width(\mathbf{I})\}$ and hence satisfy the condition

$$\Lambda = \cup_{i=1}^{K} R(L_i), \quad R(L_i) \cap R(L_j) = \emptyset, \quad \forall i \neq j.$$

The generation process from the scene description W to $\mathbf{I}$ is governed by the likelihood function

$$p(\mathbf{I}|W) = \prod_{i=1}^{K} p(\mathbf{I}_{R(L_i)}|\zeta_i, L_i, \Theta_i).$$

The prior probability $p(W)$ is defined by

$$p(W) = p(K) \prod_{i=1}^{K} p(L_i) p(\zeta_i|L_i) p(\Theta_i|\zeta_i).$$

Under the Bayesian formulation, parsing the image corresponds to computing the W^* that maximizes *a posteriori* probability over Ω, the solution space of W,

$$W^* = \arg\max_{W \in \Omega} p(W|\mathbf{I}) = \arg\max_{W \in \Omega} p(\mathbf{I}|W) p(W). \tag{8.43}$$

It remains to specify the prior $p(W)$ and the likelihood function $p(\mathbf{I}|W)$. We set the prior terms $p(K)$ and $p(\Theta_i|\zeta_i)$ to be uniform probabilities. The term $p(\zeta_i|L_i)$ is used to penalize high model complexity and was estimated for the three generic visual patterns from training data in [59].

8.10.4.2 Shape models

We use two types of shape descriptors. The first is used to define shapes of generic visual patterns and faces. The second defines the shapes of text characters. In the first case, the shape descriptor represents the boundary[13] of the image region by a list of pixels $L_i = \partial R_i$. The prior is defined as

$$p(L_i) \propto \exp\{-\gamma |R(L_i)|^{\alpha} - \lambda |L_i|\}. \tag{8.44}$$

Usually $\alpha = 0.9$. For computational reasons, we use this prior for face shapes even though a more complicated prior [14] could be applied.

In the case of text characters, we model the characters by sixty-two templates corresponding to the ten digits and the twenty-six letters in both upper and lower cases. These templates are defined by sixty-two prototype characters and a set of deformations. The prototypes are represented by an outer boundary and, at most, two inner boundaries. Each boundary is modeled by a B-spline using twenty-five control points. The prototype characters are indexed by $c_i \in \{1, ..., 62\}$ and their control points are represented by a matrix $TP(c_i)$.

We now define two types of deformations on the templates. One is a global affine transformation, and the other is a local elastic deformation. First, we allow the letters to be deformed by an affine transform M_i. We put a prior $p(M_i)$ to penalize severe rotation and distortion. This is obtained by decomposing M_i as

$$M_i = \begin{pmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & h \\ 0 & 1 \end{pmatrix},$$

where θ is the rotation angle, σ_x and σ_y denote scaling, and h is for shearing. The prior on M_i is

$$p(M_i) \propto \exp\{-a|\theta|^2 + b(\frac{\sigma_x}{\sigma_y} + \frac{\sigma_y}{\sigma_x})^2 + ch^2\},$$

[13] The boundary can include an "internal boundary" if there is a hole inside the image region explained by a different visual pattern.

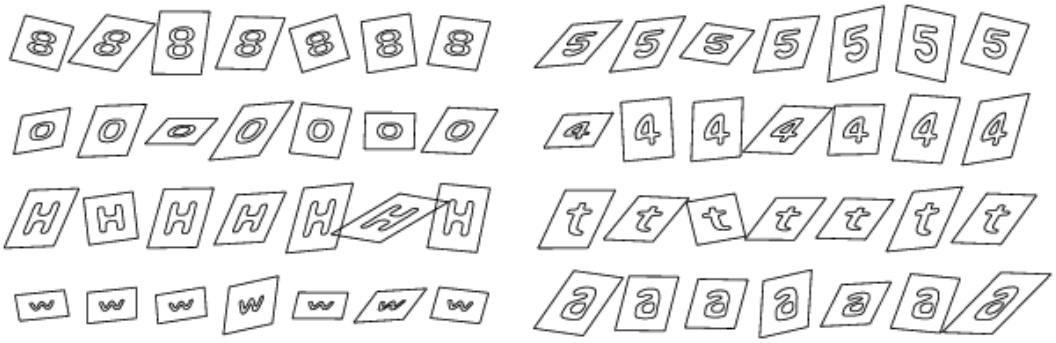


Fig. 8.29: Random samples drawn from the shape descriptors for text characters. © [2005] Springer. Reprinted, with permission, from ref. [57].

where a, b, and c are parameters.

Next, we allow local deformations by adjusting the positions of the B-spline control points. For a digit/letter c_i and affine transform M_i, the contour points of the template are given by $G_{TP}(M_i, c_i) = U \times M_s \times M_i \times TP(c_i)$. Similarly the contour points on the shape with control points S_i are given by $G_S(M_i, c_i) = U \times M_s \times S_i$ (U and M_s are the B-Spline matrices). We define a probability distribution $p(S_i|M_i, c_i)$ for the elastic deformation given by S_i,

$$p(S_i|M_i, c_i) \propto \exp\{-\gamma|R(L_i)|^{\alpha} - D(G_S(M_i, c_i)||G_{TP}(M_i, c_i))\},$$

where $D(G_S(M_i, c_i)||G_{TP}(M_i, c_i))$ is the overall distance between contour template and the deformed contour. These deformations are small, so the correspondence between points on the curves can be obtained by nearest neighbor matches. See [58] for how we can refine this. Figure 8.29 shows some samples drawn from the above model.

In summary, each deformable template is indexed by $c_i \in \{1, ..., 62\}$ and has a shape descriptor $L_i = (c_i, M_i, S_i)$, where the prior distribution on L_i is specified by $p(L_i) = p(c_i)p(M_i)p(S_i|M_i, c_i)$. Here, $p(c_i)$ is a uniform distribution on all the digits and letters (we do not place a prior distribution on text strings, though it is possible to do so [31]).

8.10.4.3 Generative intensity models

We use four families of generative intensity models for describing patterns with (approximately) constant intensity, clutter/texture, shading, and face. The first three features are similar to those defined in [59] and earlier in this chapter.

1. Constant intensity model $\zeta = 1$.

This model assumes that pixel intensities in a region R are subject i.i.d. Gaussian distributions and are given by

$$p_1(\mathbf{I}_{R(L)}|\zeta = 1, L, \Theta) = \prod_{v \in R(L)} G(\mathbf{I}_v - \mu; \sigma^2), \quad \Theta = (\mu, \sigma).$$

2. Clutter/texture model $\zeta = 2$.

This is a non-parametric intensity histogram $h()$ discretized to take G values (i.e. it is expressed as a vector $(h_1, h_2, ..., h_G)$). Let n_j be the number of pixels in $R(L)$ with intensity value j, then

$$p_2(\mathbf{I}_{R(L)}|\zeta = 2, L, \Theta) = \prod_{v \in R(L)} h(\mathbf{I}_v) = \prod_{j=1}^{G} h_j^{n_j}, \quad \Theta = (h_1, h_2, ..., h_G).$$

3. Shading model $\zeta = 3$ and $\zeta = 5, ..., 66$.

This family of models is used to describe generic shading patterns, and text characters. We use the quadratic form

$$J(x, y; \Theta) = ax^2 + bxy + cy^2 + dx + ey + f,$$

with parameters $\Theta = (a, b, c, d, e, f, \sigma)$. Therefore, the generative model for pixel (x, y) is

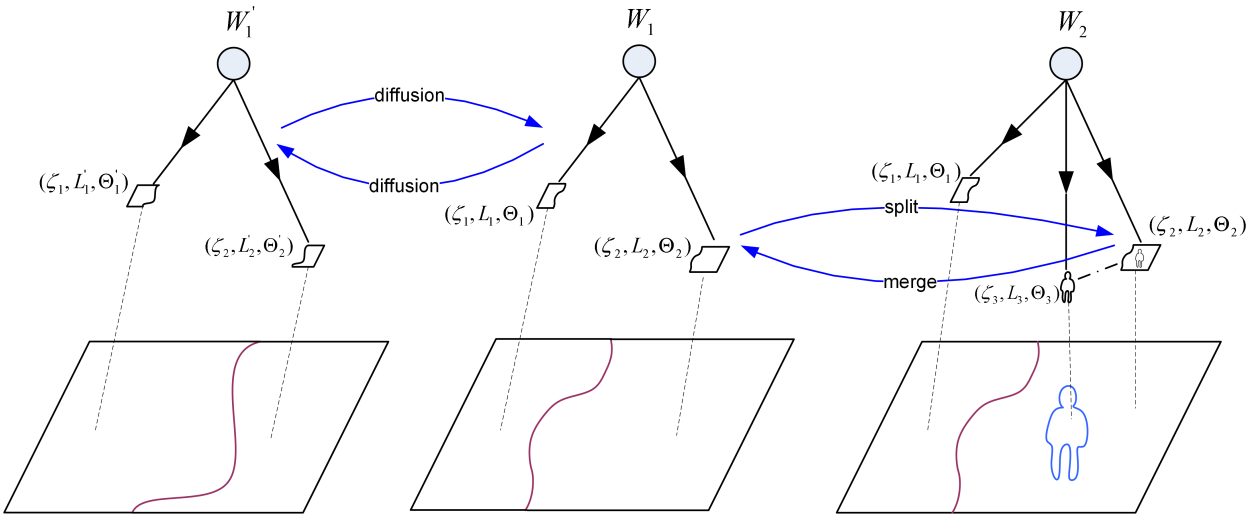


Fig. 8.30: Examples of Markov chain dynamics that change the graph structure or the node attributes of the graph giving rise to different ways to parse the image. © [2005] Springer. Reprinted, with permission, from ref. [57].

$$p_3(\mathbf{I}_R(L)|\zeta \in \{3,(5,...,66)\},L,\Theta) = \prod_{v\in R(L)} G(\mathbf{I}_v - J_v;\sigma^2), \quad \Theta = (a,b,c,d,e,f,\sigma).$$

4. The PCA face model $\zeta = 4$.

The generative model for faces is simple and uses Principal Component Analysis (PCA) to obtain representations of the faces in terms of principal components $\{B_i\}$ and covariances Σ. Lower level features, also modeled by PCA, can be added [44]. We can also add other features such as those described in Hallinan et al [28]. The model is then given by

$$p_4(\mathbf{I}_R(L)|\zeta = 4,L,\Theta) = G(\mathbf{I}_{R(L)} - \sum_i \lambda_i B_i;\Sigma), \quad \Theta = (\lambda_1,..,\lambda_n,\Sigma).$$

8.10.4.4 Overview of the Algorithm

This section gives the control structure of the image parsing algorithm. Figure 8.31 shows a diagram of the algorithm. Our algorithm must construct the parse graph on the fly and estimate the scene interpretation W.

Figure 8.30 illustrates how the algorithm selects the Markov chain moves (dynamics or sub-kernels) to search through the space of possible parse graphs of the image by altering the graph structure (by deleting or adding nodes) and by changing the node attributes. An equivalent way of visualizing the algorithm is in terms of a search through the solution space Ω. See [59, 60] for more details of this viewpoint.

We first define the set of moves to reconfigure the graph. These are birth or death of face nodes, birth or death of text characters, splitting or merging of regions, switching node attributes (region type ζ_i and model parameters Θ_i), and boundary evolution (altering the shape descriptors L_i of nodes with adjacent regions). These moves are implemented by sub-kernels. The first four moves are reversible jumps [26], and will be implemented by the Metropolis-Hastings equation (8.40). The fifth move, boundary evolution, is implemented by a stochastic partial differential equation.

The sub-kernels for these moves require proposal probabilities driven by elementary discriminative methods, which we review in the next subsection. The proposal probabilities are designed using the criteria in subsection (8.10.4), and full details are given in Section (8.10.5). The control structure of the algorithm is described in Section (8.10.4.6). The full transition kernel for the image parser is built by combining the sub-kernels, as described in subsection (8.10.3) and Figure 8.31. The algorithm proceeds (stochastically) by selecting a sub-kernel, choosing where in the graph to apply it, and then deciding whether or not to accept the operation.

8.10.4.5 Discriminative Methods

The discriminative methods give approximate posterior probabilities $q(w_j|\text{Tst}_j(\mathbf{I}))$ for the elementary components w_j of W. For computational efficiency, these probabilities are based only on a small number of simple tests $\text{Tst}_j(\mathbf{I})$. We briefly overview and classify the discriminative methods used in this implementation. Section (8.10.5) shows how these discriminative methods are combined to give proposals for making moves in the parse graph.

1. Edge Cues. These are based on edge detectors [10], [8], [33]. They are used to give proposals for region boundaries (i.e. the shape descriptor attributes of the nodes). Specifically, we run the Canny detector at three scales followed by edge linking to give partitions of the image lattice. This gives a finite list of candidate partitions which are assigned weights, see [59]. The discriminative probability is represented by this weighted list of particles. In principle, statistical edge detectors [33] would be preferable to Canny because they give discriminative probabilities $q(w_j|\text{Tst}_j(\mathbf{I}))$ learned from training data.

2. Binarization Cues. These are computed using a variant of Niblack's algorithm [46]. They are used to propose boundaries for text characters (i.e. shape descriptors for text nodes), and are used in conjunction with proposals for text detection. The binarization algorithm, and an example of its output, are given in Section (8.10.5.3). Like edge cues, the algorithm is run for different parameters settings and represents the discriminative probability by a weighted list of particles indicating candidate boundary locations.

3. Face Region Cues. These are learned by a variant of AdaBoost [51], [63] which outputs discriminative probabilities [23]. They propose the presence of faces in sub-regions of the image. These cues are combined with edge detection to propose the localization of faces in an image.

4. Text Region Cues. These are also learned by a probabilistic version of AdaBoost. The algorithm is applied to image windows at a range of scales. It outputs a discriminative probability for the presence of text in each window. Text region cues are combined with binarization to propose boundaries for text characters.

5. Shape Affinity Cues. These act on shape boundaries, produced by binarization, to propose text characters. They use shape context cues [4] and information features [58] to propose matches between the shape boundaries and the deformable template models of text characters.

6. Region Affinity Cues. These are used to estimate whether two regions R_i, R_j are likely to have been generated by the same visual pattern family and model parameters. They use an affinity similarity measure [53] of the intensity properties $\mathbf{I}_{R_i}, \mathbf{I}_{R_j}$.

7. Model Parameter and Visual Pattern Family cues. These are used to propose model parameters and visual pattern family identity. They are based on clustering algorithms, such as mean-shift [13]. The clustering algorithms depend on the model types and are described in [59].

Usually bottom-up tests $\text{Tst}_j(\mathbf{I}), j = 1, 2, ..., K$ are conducted at an early stage for all the discriminative models $q_j(w_j|\text{Tst}_j(\mathbf{I}))$, and the results are then combined to form composite tests $\text{Tst}_a(\mathbf{I})$ for each subkernel $\mathcal{K}_a$ in equations (8.40,8.41).

8.10.4.6 Control Structure of the Algorithm

The control strategy used by the image parser is illustrated in Figure 8.31. The image parser explores the space of parse graphs by an MCMC sampling algorithm. This algorithm uses a transition kernel $\mathcal{K}$ which is composed of sub-kernels $\mathcal{K}_a$ corresponding to different ways of reconfiguring the parse graph. These sub-kernels come in reversible pairs[14] (e.g. birth and death) and are designed so that the target probability distribution of the kernel is the generative posterior $p(W|\mathbf{I})$. At each time step, a sub-kernel is selected stochastically. The sub-kernels use the Metropolis-Hasting sampling algorithm, equation (8.40), which proceeds in two stages. First, it proposes a reconfiguration of the graph by sampling from a proposal probability. It then accepts or rejects this reconfiguration by sampling the acceptance probability.

To summarize, we outline the algorithm below. At each time step, it specifies (stochastically) which move to select (i.e. which sub-kernel), where to apply it in the graph, and whether to accept the move. The probability to select moves $\rho(a:\mathbf{I})$ was first set to be independent of $\mathbf{I}$, but we got better performance by using discriminative cues to estimate the number of faces and text characters in the image. The choice of where to apply the move is specified by the sub-kernel. For some sub-kernels it is selected randomly and for others is chosen based on a fitness factor, which measures how well the current model fits the image data. Some annealing is required to start the algorithm because of the limited

[14] Except for the boundary evolution sub-kernel which will be described separately. See Section 8.10.5.1.

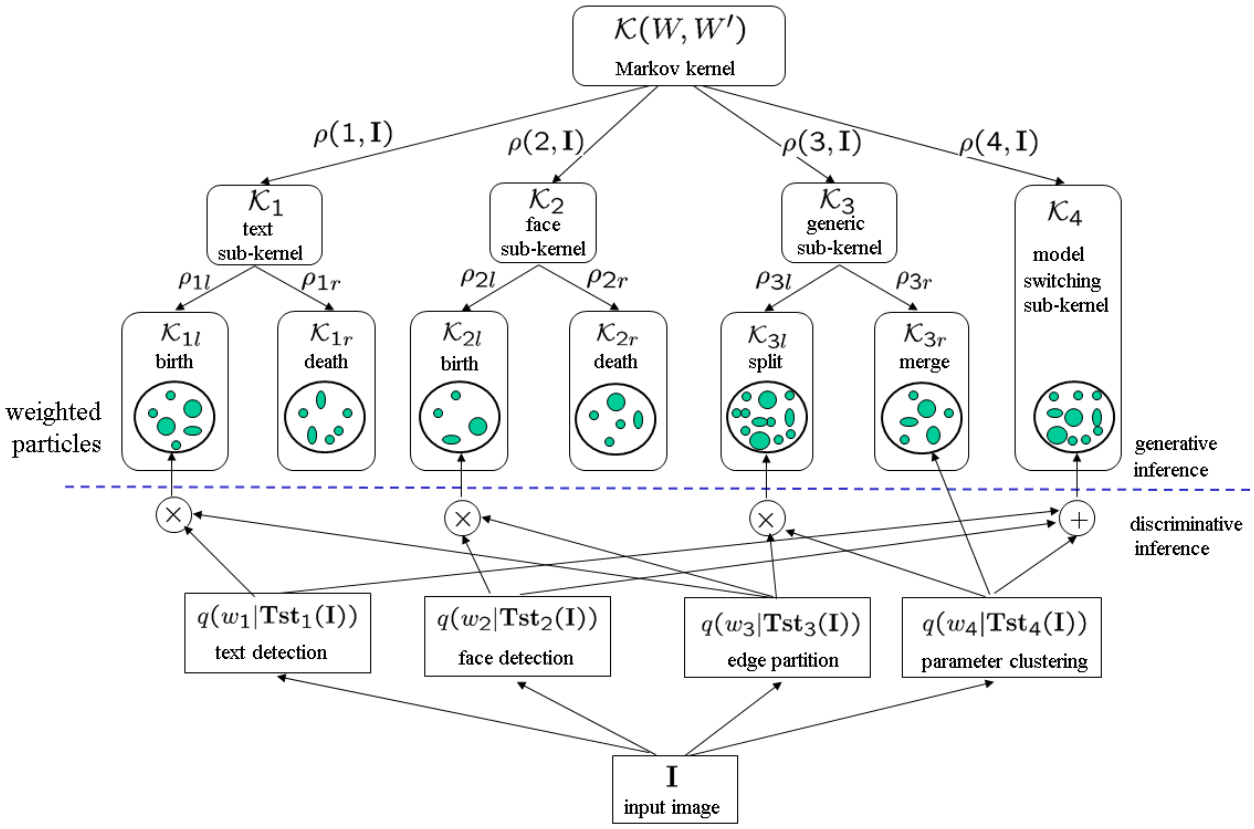


Fig. 8.31: This diagram illustrates the main points of the image parser. The dynamics are implemented by an ergodic Markov chain $\mathcal{K}$, whose invariant probability is the posterior $p(W|\mathbf{I})$, and which is composed of reversible sub-kernels $\mathcal{K}_a$ for making different types of moves in the parse graph. At each time step the algorithm selects a sub-kernel stochastically. The selected sub-kernel proposes a specific move (e.g. to create or delete specific nodes) and this move is then evaluated and accepted stochastically, see equation (8.40). The proposals are based on both bottom-up (discriminative) and top-down (generative) processes. The bottom-up processes compute discriminative probabilities $q(w_j|\text{Tst}_j(\mathbf{I})), j = 1,2,3,4$ from the input image $\mathbf{I}$ based on feature tests $\text{Tst}_j(\mathbf{I})$. © [2005] Springer. Reprinted, with permission, from ref. [57].

scope of the moves in this implementation (the need for annealing will be reduced if the compositional techniques described in [2]) are used).

We can improve the effectivenss of the algorithm by making the move selection adapt to the image (i.e. by making $\rho(a : \mathbf{I})$ depend on $\mathbf{I}$). In particular, we could increase the probability of birth and death of faces and text, $\rho(1)$ and $\rho(2)$, if the bottom-up (AdaBoost) proposals suggested that there are many objects in the scene. For example, let $N(\mathbf{I})$ be the number of proposals for faces or text above a threshold T_a. We then modify the probabilities in the table by $\rho(a_1) \mapsto \{\rho(a_1) + kg(N(\mathbf{I}))\}/Z$, $\rho(a_2) \mapsto \{\rho(a_2) + kg(N)\}/Z$, $\rho(a_3) \mapsto \rho(a_3)/Z$, $\rho(a_4) \mapsto \rho(a_4)/Z$, where $g(x) = x$, $x \leq T_b$ $g(x) = T_b$, $x \geq T_b$ and $Z = 1 + 2k$ is chosen to normalize the probability.

The basic control strategy of the image parsing algorithm

1. Initialize W (e.g. divide the image into four regions), setting their shape descriptors, and assigning the remaining node attributes at random.
2. Set the temperature to be T_{init}.
3. Select the type of move a by sampling from a probability $\rho(a)$, with $\rho(1) = 0.2$ for faces, $\rho(2) = 0.2$ for text, $\rho(3) = 0.4$ for splitting and merging, $\rho(4) = 0.15$ for switching region model (type or model parameters), and $\rho(5) = 0.05$ for boundary evolution.
4. If the selected move is boundary evolution, then select adjacent regions (nodes) at random and apply stochastic steepest descent.

5. If the jump moves are selected, then a new solution W' is randomly sampled as follows:
   - For the birth or death of a face or character, we create or delete an existing one. This includes a proposal for where to do this.
   - For region splitting a region (node) is randomly chosen based on its fitness factor. There are proposals for where to split it and for the attributes of the resulting two nodes.
   - For region merging two neighboring regions (nodes) are selected based on a proposal probability. There are proposals for the attributes of the resulting node.
   - For region switching a region is selected randomly according to its fitness factor and a new region type and/or model parameter set is proposed.

   - The full proposal probabilities, $Q(W|W:\mathbf{I})$ and $Q(W'|W:\mathbf{I})$ are computed.
   - The Metropolis-Hastings algorithm is applied to accept or reject the proposed move.
6. Reduce the temperature $T = 1 + T_{init} \times \exp(-t \times c|R|)$, where t is the current iteration step, c is a constant and $|R|$ is the size of the image.
7. Repeat the above steps until the convergence criterion is satisfied by reaching the maximum number of allowed steps or by lack of decrease of the negative log posterior.

8.10.5 The Markov chain kernels

8.10.5.1 Boundary Evolution

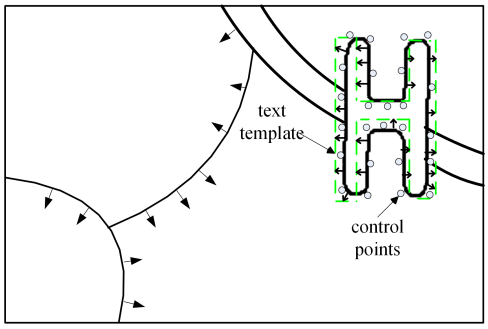


Fig. 8.32: The evolution of region boundaries is implemented by stochastic partial differential equations which are driven by models competing for ownership of the regions. © [2005] Springer. Reprinted, with permission, from ref. [57].

These moves evolve the positions of the region boundaries but preserve the graph structure. They are implemented by a stochastic partial differential equation (Langevin equation) driven by Brownian noise and can be derived from a Markov chain [25]. The deterministic component of the PDE is obtained by performing steepest descent on the negative log-posterior, as derived in [69].

We illustrate this approach by deriving the deterministic component of the PDE for the evolution of the boundary between a letter T_j and a generic visual pattern region $\mathbf{R}_i$ (Fig. 8.32). The boundary will be expressed in terms of the control points $\{S_m\}$ of the shape descriptor of the letter. Let v denote a point on the boundary, i.e. $v(s) = (x(s), y(s))$ on $\Gamma(s) = \partial R_i \cap \partial R_j$. The deterministic part of the evolution equation is obtained by taking the derivative of the negative log-posterior, $-\log p(W|\mathbf{I})$, with respect to the control points.

The relevant parts of the negative log-posterior are given by $E(\mathbf{R}_i)$ and $E(T_j)$ where

$$E(\mathbf{R}_i) = \int\int_{R_i} -\log p(\mathbf{I}(x,y)|\theta_{\zeta_i})dxdy + \gamma|R_i|^{\alpha} + \lambda|\partial R_i|,$$

and

$$E(T_j) = \int\int_{L_j} \log p(\mathbf{I}(x,y)|\theta_{\zeta_j})dxdy + \gamma|R(L_j)|^{\alpha} - \log p(L_j).$$

Differentiating $E(R_i)+E(T_j)$ with respect to the control points $\{S_m\}$ yields the evolution PDE

$$\begin{aligned}\frac{dS_m}{dt} &= -\frac{\delta E(\mathbf{R}_i)}{\delta S_m} - \frac{\delta E(T_j)}{\delta S_m} \\ &= \int [-\frac{\delta E(\mathbf{R}_i)}{\delta v} - \frac{\delta E(T_j)}{\delta v}] \frac{1}{|\mathbf{J}(s)|} ds \\ &= \int \mathbf{n}(v)[\log \frac{p(\mathbf{I}(v);\theta_{\zeta_i})}{p(\mathbf{I}(v);\theta_{\zeta_j})} + \alpha\gamma(\frac{1}{|D_j|^{1-\alpha}} - \frac{1}{|D_i|^{1-\alpha}}) - \lambda\kappa + D(G_{S_j}(s)||G_T(s))] \frac{1}{|\mathbf{J}(s)|} ds,\end{aligned}$$

where $\mathbf{J}(s)$ is the Jacobian matrix for the spline function. (Recall that $\alpha = 0.9$ in the implementation).
The log-likelihood ratio term implements the competition between the letter and the generic region models for ownership of the boundary pixels.

8.10.5.2 Markov chain sub-kernels

Changes in the graph structure are realized by Markov chain jumps implemented by four different sub-kernels.

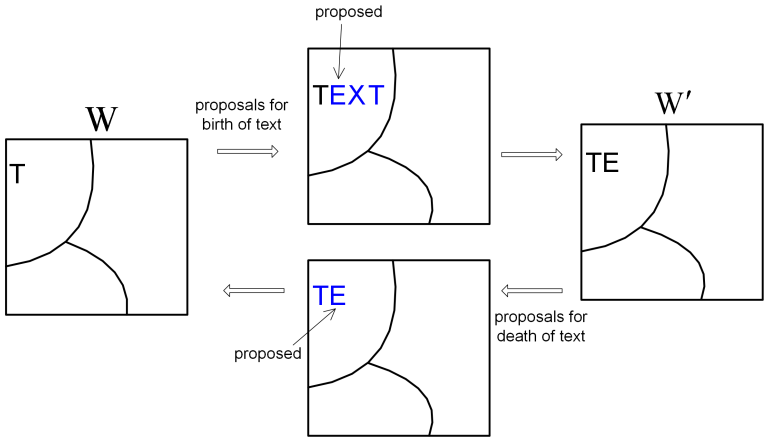


Fig. 8.33: An example of the birth-death of text. State W consists of three generic regions and a character "T". Proposals are computed for 3 candidate characters, "E", "X", and "T", obtained by AdaBoost and binarization methods. One is selected, see arrow, which changes the state to W'. Conversely, there are 2 candidate in state W' and the one selected, see arrow, returns the system to state W. © [2005] Springer. Reprinted, with permission, from ref. [57].

Sub-kernel I: birth and death of text. This pair of jumps is used to create or delete text characters (Fig. 8.33). We start with a parse graph W and transition into parse graph W' by creating a character. Conversely, we transition from W' back to W by deleting a character.

The proposals for creating and deleting text characters are designed to approximate the terms in equation (8.42). We obtain a list of candidate text character shapes by using AdaBoost to detect text regions followed by binarization to detect candidate text character boundaries within text regions. This list is represented by a set of particles which are weighted by the similarity to the deformable templates for text characters

$$S_{1r}(W) = \{ (z_{1r}^{(\mu)}, \omega_{1r}^{(\mu)}) : \mu = 1,2,...,N_{1r}\}.$$

Similarly, we specify another set of weighted particles for removing text characters

$$S_{1l}(W') = \{ (z_{1l}^{(\nu)}, \omega_{1l}^{(\nu)}) : \nu = 1,2,...,N_{1l}\}.$$

Here, $\{z_{1r}^{(\mu)}\}$ and $\{z_{1l}^{(\nu)}\}$ represent the possible (discretized) shape positions and text character deformable templates for creating or removing text, and $\{\omega_{1r}^{(\mu)}\}$ and $\{\omega_{1l}^{(\nu)}\}$ are their corresponding weights. The particles are then used to compute proposal probabilities

$$\mathbf{Q_{1r}}(\mathbf{W}'|\mathbf{W}:\mathbf{I}) = \frac{\omega_{\mathbf{1r}}(\mathbf{W}')}{\sum_{\mu=\mathbf{1}}^{\mathbf{N_{1r}}} \omega_{\mathbf{1r}}^{(\mu)}}, \;\; \mathbf{Q_{1l}}(\mathbf{W}|\mathbf{W}',\mathbf{I}) = \frac{\omega_{\mathbf{1l}}(\mathbf{W})}{\sum_{\nu=\mathbf{1}}^{\mathbf{N_{1l}}} \omega_{\mathbf{1l}}^{(\nu)}}.$$

The weights $\omega_{1r}^{(\mu)}$ and $\omega_{1l}^{(\nu)}$ for creating new text characters are specified by shape affinity measures, such as shape contexts [4] and informative features [58]. For deleting text characters we calculate $\omega_{1l}^{(\nu)}$ directly from the likelihood and prior on the text character. Ideally these weights will approximate the ratios $\frac{p(W'|\mathbf{I})}{p(W|\mathbf{I})}$ and $\frac{p(W|\mathbf{I})}{p(W'|\mathbf{I})}$.

Sub-kernel II: birth and death of faces. The sub-kernel for the birth and death of faces is very similar to the sub-kernel of birth and death of text. We use AdaBoost methods discussed in section (8.10.5.3) to detect candidate faces. Candidate face boundaries are obtained directly from edge detection. The proposal probabilities are computed similarly to those for sub-kernel I.

Sub-kernel III: splitting and merging of regions. This pair of jumps is used to create or delete nodes by splitting and merging regions (nodes). We start with a parse graph W and transition into parse graph W' by splitting node i into nodes j and k. Conversely, we transition back to W by merging nodes j and k into node i. The selection of which region i to split is based on a robust function on $p(\mathbf{I}_{R_i}|\zeta_i, L_i, \Theta_i)$ (i.e. the worse the model for region R_i fits the data, the more likely we are to split it). For merging, we use a region affinity measure [53] and propose merges between regions that have high affinity. Formally, we define W, W':

$$W = (K, (\zeta_k, L_k, \Theta_k), W_-) \;\; \rightleftharpoons \;\; W' = (K+1, (\zeta_i, L_i, \Theta_i), (\zeta_j, L_j, \Theta_j), W_-),$$

where W_- denotes the attributes of the remaining $K-1$ nodes in the graph.

We obtain proposals by seeking approximations to equation (8.42) as follows. We first obtain three edge maps. These are given by Canny edge detectors [10] at different scales (see [59] for details). We use these edge maps to create a list of particles for splitting $S_{3r}(W)$. A list of particles for merging is denoted by $S_{3l}(W')$. Formally,

$$S_{3r}(W) = \{\, (z_{3r}^{(\mu)}, \omega_{3r}^{(\mu)}) : \; \mu = 1, 2, ..., N_{3r}\}, \quad S_{3l}(W') = \{\, (z_{3l}^{(\nu)}, \omega_{3l}^{(\nu)}) : \; \nu = 1, 2, ..., N_{3l}, \}$$

where $\{z_{3r}^{(\mu)}\}$ and $\{z_{3l}^{(\nu)}\}$ represent the possible (discretized) positions for splitting and merging, and their weights $\{\omega_{3r}\}, \{\omega_{3l}\}$ will be defined shortly. In other words, we can only split a region i into regions j and k along a contour z_{3r}^{μ} (i.e. z_{3r}^{μ} forms the new boundary). Similarly we can only merge regions j and k into region i by deleting a boundary contour z_{3l}^{μ}.

We now define the weights $\{\omega_{3r}\}, \{\omega_{3l}\}$. These weights will be used to determine probabilities for the splits and merges by

$$\mathbf{Q_{3r}}(\mathbf{W}'|\mathbf{W}:\mathbf{I}) = \frac{\omega_{\mathbf{3r}}(\mathbf{W}')}{\sum_{\mu=\mathbf{1}}^{\mathbf{N_{3r}}} \omega_{\mathbf{3r}}^{(\mu)}}, \;\; \mathbf{Q_{3l}}(\mathbf{W}|\mathbf{W}':\mathbf{I}) = \frac{\omega_{\mathbf{3l}}(\mathbf{W})}{\sum_{\nu=\mathbf{1}}^{\mathbf{N_{3l}}} \omega_{\mathbf{3l}}^{(\nu)}}.$$

Again, we would like ω_{3r}^{μ} and ω_{3l}^{ν} to approximate the ratios $\frac{p(W|\mathbf{I})}{p(W'|\mathbf{I})}$ and $\frac{p(W'|\mathbf{I})}{p(W|\mathbf{I})}$ respectively. $\frac{p(W'|\mathbf{I})}{p(W|\mathbf{I})}$ is given by

$$\frac{p(W'|\mathbf{I})}{p(W|\mathbf{I})} = \frac{p(\mathbf{I}_{R_i}|\zeta_i, L_i, \Theta_i)p(\mathbf{I}_{R_j}|\zeta_j, L_j, \Theta_j)}{p(\mathbf{I}_{R_k}|\zeta_k, L_k, \Theta_k)} \cdot \frac{p(\zeta_i, L_i, \Theta_i)p(\zeta_j, L_j, \Theta_j)}{p(\zeta_k, L_k, \Theta_k)} \cdot \frac{p(K+1)}{p(K)}.$$

This is expensive to compute, so we approximate $\frac{p(W'|\mathbf{I})}{p(W|\mathbf{I})}$ and $\frac{p(W|\mathbf{I})}{p(W'|\mathbf{I})}$ by

$$\omega_{3r}^{(\mu)} = \frac{q(R_i, R_j)}{p(\mathbf{I}_{R_k}|\zeta_k, L_k, \Theta_k)} \cdot \frac{[q(L_i)q(\zeta_i, \Theta_i)][q(L_j)q(\zeta_j, \Theta_j)]}{p(\zeta_k, L_k, \Theta_k)}, \tag{8.45}$$

$$\omega_{3l}^{(\nu)} = \frac{q(R_i, R_j)}{p(\mathbf{I}_{R_i}|\zeta_i, L_i, \Theta_i)p(\mathbf{I}_{R_j}|\zeta_j, L_j, \Theta_j)} \cdot \frac{q(L_k)q(\zeta_k, \Theta_k)}{p(\zeta_i, L_i, \Theta_i)p(\zeta_j, L_j, \Theta_j)}. \tag{8.46}$$

Here, $q(R_i, R_j)$ is an *affinity measure* [53] of the similarity of the two regions R_i and R_j (it is a weighted sum of the intensity difference $|\bar{I}_i - \bar{I}_j|$ and the chi-squared difference between the intensity histograms), $q(L_i)$ is given by the priors on the shape descriptors, and $q(\zeta_i, \Theta_i)$ is obtained by clustering in parameter space (see [59]).
Sub-kernel IV: Model Switching. These moves switch the attributes of a node i. This involves changing the region type ζ_i and the model parameters Θ_i. The move transitions between two states

$$W = ((\zeta_i, L_i, \Theta_i), W_-) \;\rightleftharpoons\; W' = ((\zeta_i', L_i', \Theta_i'), W_-).$$

The proposal, see equation (8.42), should approximate

$$\frac{p(W'|\mathbf{I})}{p(W|\mathbf{I})} = \frac{p(\mathbf{I}_{R_i}|\zeta_i', L_i', \Theta_i')p(\zeta_i', L_i', \Theta_i')}{p(\mathbf{I}_{R_i}|\zeta_i, L_i, \Theta_i)p(\zeta_i, L_i, \Theta_i)}.$$

We approximate this by a weight $\omega_4^{(\mu)}$ given by

$$\omega_4^{(\mu)} = \frac{q(L_i')q(\zeta_i', \Theta_i')}{p(\mathbf{I}_{R_i}|\zeta_i, L_i, \Theta_i)p(\zeta_i, L_i, \Theta_i)},$$

where $q(L_i')q(\zeta_i', \Theta_i')$ are the same functions used in the split and merge moves. The proposal probability is the weight normalized in the candidate set, $\mathbf{Q_4}(\mathbf{W}'|\mathbf{W}:\mathbf{I}) = \frac{\omega_4(\mathbf{W}')}{\sum_{\mu=1}^{\mathbf{N_4}} \omega_4^{(\mu)}}$.

8.10.5.3 AdaBoost for discriminative probabilities for face and text

This section describes how we use AdaBoost techniques to compute discriminative probabilities for detecting faces and text (strings of letters). We also describe the binarization algorithm used to detect the boundaries of text characters.
Computing discriminative probabilities by Adaboost. The standard AdaBoost algorithm used for example to distinguish faces from non-faces [63], learns a binary-valued strong classifier H_{Ada} by combining a set of n binary-valued weak classifiers or feature tests $\text{Tst}_{\text{Ada}}(\mathbf{I}) = (h_1(\mathbf{I}), ..., h_n(\mathbf{I}))$ using a set of weights $\alpha_{\text{Ada}} = (\alpha_1, ..., \alpha_n)$ [22] such that

$$H_{\text{Ada}}(\text{Tst}_{\text{Ada}}(\mathbf{I})) = \text{sign}(\sum_{i=1}^{n} \alpha_i h_i(\mathbf{I})) = \text{sign} < \alpha_{\text{Ada}}, \text{Tst}_{\text{Ada}}(\mathbf{I}) > . \tag{8.47}$$

The features are selected from a pre-designed dictionary Δ_{Ada}. The selection of features and the tuning of weights are posed as a supervised learning problem. Given a set of labeled examples, $\{(\mathbf{I}_i, \ell_i) : i = 1, 2, ..., M\}$ ($\ell_i = \pm 1$), AdaBoost learning can be formulated as greedily optimizing the following function [51]:

$$(\alpha_{\text{Ada}}^*, \text{Tst}_{\text{Ada}}^*) = \arg \min_{\text{Tst}_{\text{Ada}} \subset \Delta_{\text{Ada}}} \arg \min_{\alpha_{\text{Ada}}} \sum_{i=1}^{M} \exp^{-\ell_i < \alpha_{\text{Ada}}, \text{Tst}_{\text{Ada}}(\mathbf{I}_i) >}. \tag{8.48}$$

To obtain discriminative probabilities we use a theorem [23] which states that the features and tests learned by AdaBoost give (asymptotically) posterior probabilities for the object labels (e.g. face or non-face). The AdaBoost strong classifier can be rederived as the log posterior ratio test.

Theorem 8.3 (Friedman et al 2000) *With sufficient training samples M and features n, AdaBoost learning selects the weights* α_{Ada}^* *and tests* $\text{Tst}_{\text{Ada}}^*$ *to satisfy*

$$q(\ell = +1|\mathbf{I}) = \frac{e^{\ell < \alpha_{\text{Ada}}, \text{Tst}_{\text{Ada}}(\mathbf{I}_i) >}}{e^{< \alpha_{\text{Ada}}, \text{Tst}_{\text{Ada}}(\mathbf{I}_i) >} + e^{- < \alpha_{\text{Ada}}, \text{Tst}_{\text{Ada}}(\mathbf{I}_i) >}}.$$

Moreover, the strong classifier converges asymptotically to the posterior probability ratio test

$$H_{\text{Ada}}(\text{Tst}_{\text{Ada}}(\mathbf{I})) = \text{sign}(< \alpha_{\text{Ada}},\ \text{Tst}_{\text{Ada}}(\mathbf{I}) >) = \text{sign}(\frac{q(\ell = +1|\mathbf{I})}{q(\ell = -1|\mathbf{I})}).$$

In practice, the AdaBoost classifier is applied to windows in the image at different scales. Each window is evaluated as being face or non-face (or text versus non-text). For most images the posterior probabilities for faces or text are negligible for almost all parts of an image. So we use a cascade of tests [63, 65] which enables us to rapidly reject many windows by setting their marginal probabilities to be zero. Of course, AdaBoost will only converge to approximations to the true posterior probabilities $p(\ell|\mathbf{I})$ because only a limited number of tests can be used (and there is only a limited amount of training data). Note that AdaBoost is only one way to learn a posterior probability, (see Theorem (8.1)). It has been found to be quite effective for object patterns which have relatively rigid structures, such as faces and text (the shapes of letters are variable but the patterns of a sequence are fairly structured [11]).

AdaBoost Training. Standard AdaBoost training methods [22, 23] can be used and combined with the cascade approach using asymmetric weighting [63, 65]. The cascade enables the algorithm to rule out most of the image as face, or text, locations with a few tests and allows computational resources to be concentrated on the more challenging parts of the images.

Fig. 8.34: Some scenes from which the training text patches are extracted. © [2005] Springer. Reprinted, with permission, from ref. [57].

The AdaBoost for text was designed to detect text segments. The test data was extracted by hand from 162 images of San Francisco (see Figure 8.34) and contained 561 text images. More than half of the images were taken by blind volunteers (which reduces bias). We divided each text image into several overlapping text segments with fixed width-to-height ration 2:1 (typically containing between two and three letters). A total of 7,000 text segments were used as the positive training set. The negative examples were obtained by a bootstrap process similar to Drucker et al [20]. First the negative examples were selected by randomly sampling from windows in the image dataset. After training with these samples, the AdaBoost algorithm was applied at a range of scales to classify all windows in the training images. Those misclassified as text were then used as negative examples for the next stage of AdaBoost. The image regions most easily confused with text were vegetation, and repetitive structures such as railings or building facades. The features used for AdaBoost were image tests corresponding to the statistics of elementary filters. The features were chosen to detect properties of text segments that were relatively invariant to the shapes of the individual letters or digits. They included averaging the intensity within image windows, and statistics of the number of edges. We refer to [11] for more details.

The AdaBoost posteriors for faces was trained in a similar way. This time we used Haar basis functions [63] as elementary features. We used the FERET [50] database for our positive examples, and by allowing small rotation and translation transformation we had 5,000 positive examples. We used the same strategy as described above for text to obtain negative examples.

Object	False Positive	False Negative	Images	Subwindows
Face	65	26	162	355,960,040
Face	918	14	162	355,960,040
Face	7542	1	162	355,960,040
Text	118	27	35	20,183,316
Text	1879	5	35	20,183,316

Table 8.1: Performance of AdaBoost at different thresholds.

In both cases, we evaluated the log posterior ratio test on testing datasets using a number of different thresholds (see [63]). In agreement with work on faces [63], AdaBoost gave very high performance with very few false positives

and false negatives. But these low error rates are slightly misleading because of the enormous number of windows in each image (see Table (8.1)). A small false positive rate may imply a large number of false positives for any regular image. By varying the threshold, we can either eliminate the false positives or the false negatives but not both at the same time. We illustrate this by showing the face regions and text regions proposed by AdaBoost in Figure 8.35. If we attempt classification by putting a threshold then we can only correctly detect all the faces and the text at the expense of false positives.

Fig. 8.35: The boxes show faces and text as detected by the AdaBoost log posterior ratio test with fixed threshold. Observe the false positives due to vegetation, tree structure, and random image patterns. It is impossible to select a threshold which has no false positives and false negatives for this image. As it is shown in our experiments later, the generative models will remove the false positives and also recover the missing text. © [2005] Springer. Reprinted, with permission, from ref. [57].

When Adaboost is integrated with the generic region models in the image parser, the generic region proposals can remove false positives and find text that AdaBoost misses. For example, the '9' in the right panel of Figure 8.35 is not detected because our AdaBoost algorithm was trained on text segments. Instead it is detected as a generic shading region and later recognized as a letter '9', see Figure 8.37. Some false positive text and faces in figure 8.35 are removed in Figures 8.37 and 8.39.

Binarization Algorithm. The AdaBoost algorithm for text needs to be supplemented with a binarization algorithm, described below, to determine text character location. This is followed by applying shape contexts [4] and informative features [58] to the binarization results to make proposals for the presence of specific letters and digits. In many cases the results of binarization are so good that the letters and digits can be detected immeadiately (i.e. the proposals made by the binarization stage are automatically accepted), but this will not always be the case. We note that binarization gives far better results than alternatives such as edge detection [10].

Fig. 8.36: Example of binarization on the detected text. © [2005] Springer. Reprinted, with permission, from ref. [57].

The binarization algorithm is a variant of one proposed by Niblack [46]. We binarize the image intensity using an adaptive thresholding based on a adaptive window size. Adaptive methods are needed because image windows containing text often have shading, shadow, and occlusion. Our binarization method determines the threshold $T_b(v)$

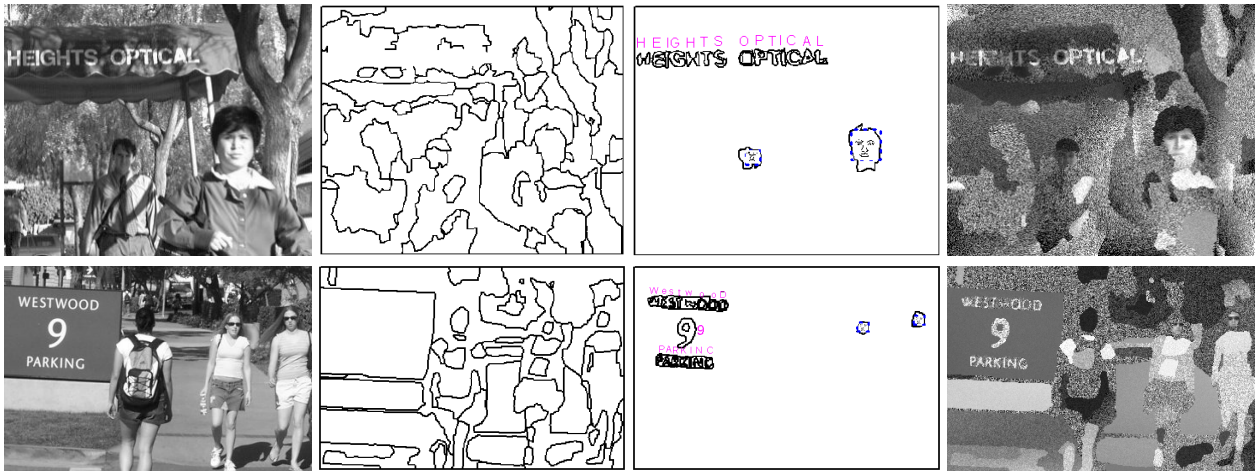
for each pixel v by the intensity distribution of its local window $r(v)$ (centered on v). This threshold is given by

$$T_b(v) = \mu(\mathbf{I}_{r(v)}) + k \cdot std(\mathbf{I}_{r(v)}),$$

where $\mu(\mathbf{I}_{r(v)})$ and $std(\mathbf{I}_{r(v)})$ are the intensity mean and standard deviation within the local window. The size of the local window is selected to be the smallest possibble window whose intensity variance is above a fixed threshold. The parameter $k = \pm 0.2$, where the $\pm$ allows for cases where the foreground is brighter or darker than the background.

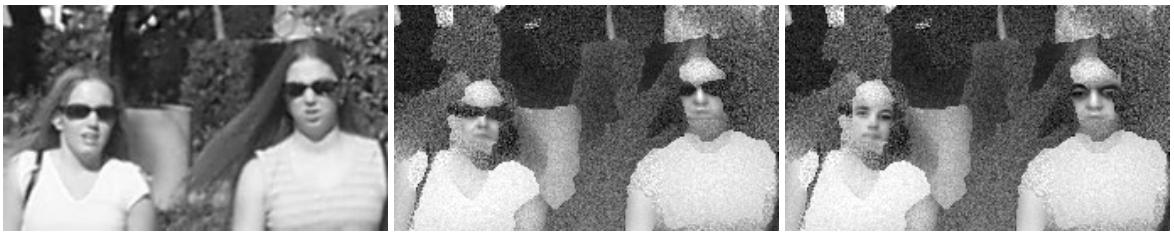
8.10.6 Image parsing experiments

The image parsing algorithm is applied to a number of outdoor/indoor images. The speed in PCs (Pentium IV) is comparable to segmentation methods such as normalized cuts [38] or the DDMCMC algorithm in [59]. It typically runs around 10-40 minutes. The main portion of the computing time is spent on segmenting generic regions and boundary diffusion [69].



a. Input image b. Segmentation c. Object recognition d. Synthesized image

Fig. 8.37: Results of segmentation and recognition on two images. The results are improved compared to the purely bottom-up (AdaBoost) results displayed in Figure 8.35. © [2005] Springer. Reprinted, with permission, from ref. [57].



a. Input image b. Synthesis 1 c. Synthesis 2

Fig. 8.38: A close-up look of an image in Figure 8.37. The dark glasses are explained by the generic shading model and so the face model does not have to fit this part of the data. Otherwise the face model would have difficulty because it would try to fit the glasses to eyes. Standard AdaBoost only correctly classifies these faces at the expense of false positives, see Figure 8.35. We show two examples of synthesized faces, one (Synthesis 1) with the dark glasses (modeled by shading regions) and the other (Synthesis 2) with the dark glasses removed (i.e. using the generative face model to sample parts of the face (e.g. eyes) obscured by the dark glasses. © [2005] Springer. Reprinted, with permission, from ref. [57].

Figures 8.37, 8.38, and 8.39 show some challenging examples which have heavy clutter and shading effects. We present the results in two parts. One shows the segmentation boundaries for generic regions and objects, and the other shows the text and faces detected with text symbols to indicate text recognition, i.e. the letters are correctly read by the algorithm. Then we synthesize images sampled from the likelihood model $p(\mathbf{I}|W^*)$ where W^* is the parse graph (the faces, text, regions parameters and boundaries) obtained by the parsing algorithm. The synthesized images are used to visualize the parsing graph W^*, i.e. the image content that the computer "understands".

In the experiments, we observed that the face and text models improved the image segmentation results by comparison to our previous work [59] which only used generic region models. Conversely, the generic region models improve object detection by removing some false positives and recovering objects which were not initially detected. We now discuss specific examples.

In Figure 8.35, we showed two images where the text and faces were detected purely bottom-up using AdaBoost. It is was impossible to select a threshold so that our AdaBoost algorithm had no false positives or false negatives. To ensure no false negatives, apart from the '9', we had to lower the threshold and admit false positives due to vegetation and heavy shadows (e.g. the shadow in the sign "HEIGHTS OPTICAL").

The letter '9' was not detected at any threshold. This is because our AdaBoost algorithm was trained to detect text segments, and so did not respond to a single digit.

By comparison, Figure 8.37 shows the image parsing results for these two images. We see that the false positives proposed by AdaBoost are removed because they are better explained by the generic region models. The generic shading models help object detection by explaining away the heavy shading on the text "HEIGHTS OPTICAL" and the dark glasses on the women, see Figure 8.38. Moreover, the missing digit '9' is now correctly detected. The algorithm first detected it as a generic shading region and then reclassified as a digit using the sub-kernel that switches node attributes.

The ability to synthesize the image from the parse graph W^* is an advantage of the Bayesian approach. The synthesis helps illustrate the successes, and sometimes the weaknesses, of the generative models. Moreover, the synthesized images show how much information about the image has been captured by the models. In Table (8.2) are given the number of variables used in our representation W^* and show that they are roughly proportional to the jpeg bytes. Most of the variables in W^* are used to represent points on the segmentation boundary, and at present they are counted independently. We could reduce the coding length of W^* substantially by encoding the boundary points effectively, for example, using spatial proximity. Image encoding is not the goal of our current work, however, and more sophisticated generative models would be needed to synthesize very realistic images.

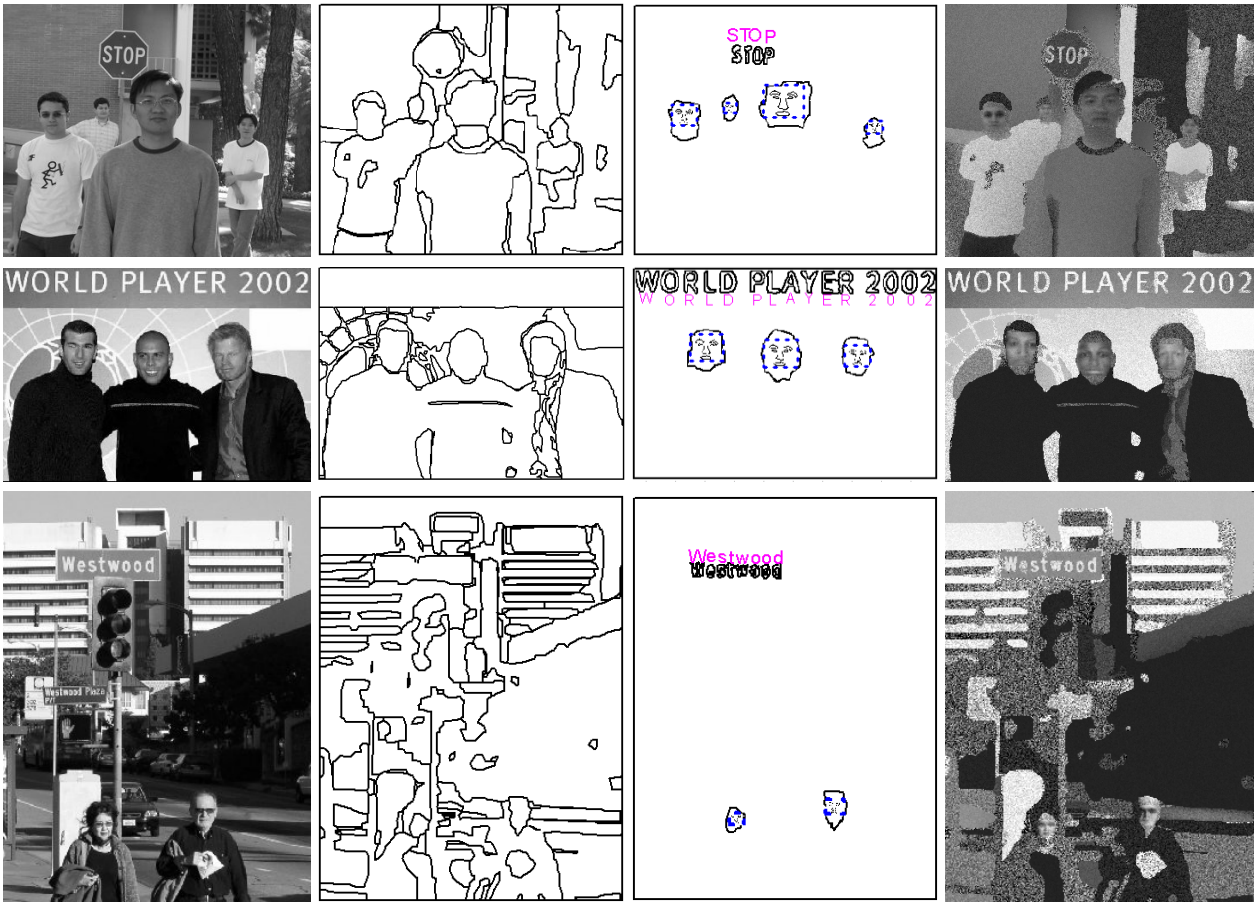
Image	Stop	Soccer	Parking	Street	Westwood
jpg bytes	23,998	19,563	23,311	26,170	27,790
$\|W^*\|$	4,886	3,971	5,013	6,346	9,687

Table 8.2: The number of variables in W^* for each image compared to the JPG bytes.

In this section, we describe two challenging technical problems for image parsing. First, the parse graph can be constructed in compositional and decompositional modes. The compositional mode proceeds by grouping small elements while the decompositional approach involves detecting an object as a whole and then locating its parts.

The compositional mode appears most effective for Figure 8.40(a). Detecting the cheetah by bottom-up tests, such as those learned by AdaBoost, seems difficult due to the large variability of shape and photometric properties of cheetahs. By contrast, it is quite practical to use Swendsen-Wang Cuts ([1] and Chapter 6) to segment the image and obtain the boundary of the cheetah using a bottom-up compositional approach and a parsing tree with multiple levels. The parse graph is constructed starting with the pixels as leaves (there are $46,256$ pixels in Figure 8.40 (a)). The next level of the graph is obtained using local image texture similarities to construct graph nodes (113 of them) corresponding to atomic regions of the image. The algorithm then constructs nodes (4 of them) for "texture regions" at the next level by grouping the atomic regions (i.e. each atomic region node will be the child of a texture region node). At each level we compute a discriminative (proposal) probability for how likely adjacent nodes (e.g. pixels or atomic regions) belong to the same object or pattern. We then apply a transition kernel implementing split and merge dynamics (using the proposals). We refer to Chapter 6 or [1] for more detailed discussion.

For objects with little variability, such as the faces shown in Figure 8.40 (b), we can use bottom-up proposals (e.g. AdaBoost) to activate a node that represents the entire face. The parse graph can then be constructed downwards (i.e. in the decompositional mode) by expanding the face node to create child nodes for the parts of the face. These child nodes could, in turn, be expanded to grandchild nodes representing finer scale parts. The amount of node expansion can be made adaptive to depend on the resolution of the image. For example, the largest face in Figure 8.40 (b) is



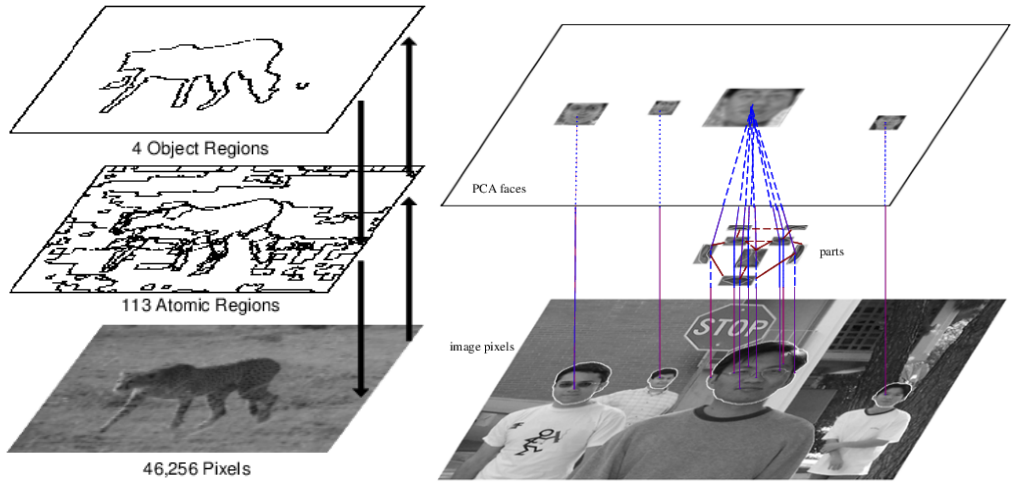
a. Input image b. Segmentation c. Object recognition d. Synthesized image

Fig. 8.39: Results of segmentation and recognition on outdoor images. Observe the ability to detect faces and text at multiple scale. © [2005] Springer. Reprinted, with permission, from ref. [57].

expanded into child nodes but there is not sufficient resolution to expand the face nodes corresponding to the three smaller faces. The major technical problem is to develop a mathematical criterion for which mode is most effective for which types of objects and patterns. This will enable the algorithm to adapt its search strategy accordingly.

The second challenge involves the optimal ordering. The control strategy of the image parsing algorithm does not select the tests and sub-kernels in an optimal way. At each time step, the choice of sub-kernel is independent of the current state W (though the choice of where in the graph to apply the sub-kernel will depend on W). Moreover, some bottom-up tests that are performed will never be used by the algorithm. It would be more efficient to have a control strategy that selects the sub-kernels and tests adaptively, provided the selection process requires low computational cost. We seek to find an optimal control strategy for selection which is effective for a large set of images and visual patterns. The selection criteria should select those tests and sub-kernels that maximize the gain in information.

We could use the two information criteria. The first is stated in Theorem 8.1 and measures the information gained for variable w in the parse graph by performing a new test Tst_+. The information gain is $\delta(w||\text{Tst}_+) = KL(p(w|\mathbf{I}) \,||\, q(w|\text{Tst}(\mathbf{I}))) - KL(p(w|\mathbf{I}) \,||\, q(w|\text{Tst}_t(\mathbf{I}), F_+))$, where $\text{Tst}(\mathbf{I})$ denotes the previous tests (and KL is the Kullback-Leibler divergence). The second is stated in Theorem 8.2. It measures the power of a sub-kernel $\mathcal{K}_a$ by the decrease of the KL-divergence $\delta(\mathcal{K}_a) = KL(p\,||\,\mu_t) - KL(p\,||\,\mu_t\mathcal{K}_a)$. The amount of decrease δ_a gives a measure of the power of the sub-kernel $\mathcal{K}_a$ when informed by $\text{Tst}_t(\mathbf{I})$. We need also take into account the computational cost of the selection procedures. See [6] for a case study for how to optimally select tests taking into account their computational costs.



(a) "composition" (b) "decomposition"

Fig. 8.40: Two mechanisms for constructing the parse graph. See text for explanation. © [2005] Springer. Reprinted, with permission, from ref. [57].

8.11 Exercises

Problem 1. Suppose we have a simple 1D range image (see Section 4.3.2) obtained as $y(x) = \alpha|x| + \varepsilon, x \in \{-100, .-99, ..., 100\}$, where α controls the signal strength and $\varepsilon \sim \mathcal{N}(0,1)$. Implement a simple DDMCMC algorithm for segmenting the image in up to three pieces. Use jump diffusion like in Problem 2 from Chapter 4, but use the Hough transform to generate proposals for the jumps. Try different signals strengths $\alpha \in \{0.01, 0.003, 0.1, 0.3, 1\}$ and obtain the average energy (4.8) vs computation time from 10 independent runs. Plot the average energy vs computation time and compare it with the corresponding plot from Problem 2 of Chapter 4.

Problem 2. Consider the same setup as in Problem 1 above, but using a genetic algorithm for optimization. The algorithm contains an initial generation 1000 solutions that are initialized randomly. Each solution from a generation produces 5 descendants through mutations (where the parameters suffer small perturbations) and 5 other descendants through cross-over (where some parameters are borrowed from another solution of the same generation). The descendants are then reduced back to 1000 by evaluating their energy functions (4.8) and keeping only the lowest energy solutions. On the same plot display the energy of the best solution vs. computation time for the genetic algorithm and the DDMCMC approach.

References

1. Adrian Barbu and Song-Chun Zhu. Multigrid and multi-level swendsen-wang cuts for hierarchic graph partition. In *CVPR*, volume 2, pages II–731, 2004.
2. Adrian Barbu and Song-Chun Zhu. Generalizing swendsen-wang to sampling arbitrary posterior probabilities. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(8):1239–1253, 2005.
3. Simon A Barker, Anil C Kokaram, and Peter JW Rayner. Unsupervised segmentation of images. In *SPIE's International Symposium on Optical Science, Engineering, and Instrumentation*, pages 200–211. International Society for Optics and Photonics, 1998.
4. Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(4):509–522, 2002.
5. Elie Bienenstock, Stuart Geman, and Daniel Potter. Compositionality, mdl priors, and object recognition. *NIPS*, pages 838–844, 1997.
6. Gilles Blanchard and Donald Geman. Hierarchical testing designs for pattern recognition. *Annals of Statistics*, pages 1155–1202, 2005.
7. Charles Bouman and Bede Liu. Multiple resolution segmentation of textured images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(2):99–113, 1991.
8. Kevin Bowyer, Christine Kranenburg, and Sean Dougherty. Edge detector evaluation using empirical roc curves. *Computer Vision and Image Understanding*, 84(1):77–103, 2001.

9. Pierre Bremaud. *Markov chains: Gibbs fields, Monte Carlo simulation, and queues*, volume 31. Springer, 1999.
10. John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 8(6):679–698, 1986.
11. Xiangrong Chen and Alan L Yuille. Detecting and reading text in natural scenes. In *CVPR*, volume 2, pages II–366. IEEE, 2004.
12. Yizong Cheng. Mean shift, mode seeking, and clustering. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(8):790–799, 1995.
13. Dorin Comaniciu and Peter Meer. Mean shift analysis and applications. In *ICCV*, volume 2, pages 1197–1203. IEEE, 1999.
14. Timothy F Cootes, Gareth J Edwards, and Christopher J Taylor. Active appearance models. *IEEE Transactions on pattern analysis and machine intelligence*, 23(6):681–685, 2001.
15. Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.
16. Peter Dayan, Geoffrey E Hinton, Radford M Neal, and Richard S Zemel. The helmholtz machine. *Neural computation*, 7(5):889–904, 1995.
17. Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.
18. Yining Deng, B Shin Manjunath, and Hyundoo Shin. Color image segmentation. In *CVPR*, volume 2, 1999.
19. Persi Diaconis and Phil Hanlon. Eigen-analysis for some examples of the metropolis algorithm. *Contemporary Mathematics*, 138:99–117, 1992.
20. Harris Drucker, Robert Schapire, and Patrice Simard. Boosting performance in neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(04):705–719, 1993.
21. David A Forsyth. Sampling, resampling and colour constancy. In *CVPR*, volume 1. IEEE, 1999.
22. Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *ICML*, volume 96, pages 148–156, 1996.
23. Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.
24. Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 6:721–741, 1984.
25. Stuart Geman and Chii-Ruey Hwang. Diffusions for global optimization. *SIAM Journal on Control and Optimization*, 24(5):1031–1043, 1986.
26. Peter J Green. Reversible jump markov chain monte carlo computation and bayesian model determination. *Biometrika*, 82(4):711–732, 1995.
27. Ulf Grenander and Michael I Miller. Representations of knowledge in complex systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 549–603, 1994.
28. Peter W Hallinan, Gaile G Gordon, Alan L Yuille, Peter Giblin, and David Mumford. *Two-and three-dimensional patterns of the face*. AK Peters, Ltd., 1999.
29. W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.
30. Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International journal of computer vision*, 1(4):321–331, 1988.
31. Dan Klein and Christopher D Manning. A generative constituent-context model for improved grammar induction. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 128–135. Association for Computational Linguistics, 2002.
32. Georges Koepfler, Christian Lopez, and Jean-Michel Morel. A multiscale algorithm for image segmentation by variational method. *SIAM journal on numerical analysis*, 31(1):282–299, 1994.
33. Scott Konishi, Alan L. Yuille, James M. Coughlan, and Song Chun Zhu. Statistical edge detection: Learning and evaluating edge cues. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(1):57–74, 2003.
34. Yvan G Leclerc. Constructing simple stable descriptions for image partitioning. *International journal of computer vision*, 3(1):73–102, 1989.
35. Hsien-Che Lee and David R Cok. Detecting boundaries in a vector field. *Signal Processing, IEEE Transactions on*, 39(5):1181–1194, 1991.
36. Fei Fei Li, Rufin VanRullen, Christof Koch, and Pietro Perona. Rapid natural scene categorization in the near absence of attention. *Proceedings of the National Academy of Sciences*, 99(14):9596–9601, 2002.
37. Jun S Liu. *Monte Carlo strategies in scientific computing*. springer, 2008.
38. Jitendra Malik, Serge Belongie, Thomas Leung, and Jianbo Shi. Contour and texture analysis for image segmentation. *International journal of computer vision*, 43(1):7–27, 2001.
39. Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.
40. D Marr. Vision, 1982. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*, 1982.
41. David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 2, pages 416–423, 2001.
42. Kerrie L Mengersen, Richard L Tweedie, et al. Rates of convergence of the hastings and metropolis algorithms. *The Annals of Statistics*, 24(1):101–121, 1996.
43. Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.
44. Baback Moghaddam and Alex Pentland. Probabilistic visual learning for object representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):696–710, 1997.

45. David Mumford. *Neuronal architectures for pattern-theoretic problems*. Large-Scale Theories of the Cortex. Cambridge, MA: MIT Press, 1994.
46. W Niblack. *An introduction to digital image processing*. Englewood Cliffs [etc.]: Prentice-Hall, 1986.
47. Shunichiro Oe. Texture segmentation method by using two-dimensional ar model and kullback information. *Pattern recognition*, 26(2):237–244, 1993.
48. Stanley Osher and James A Sethian. Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations. *Journal of computational physics*, 79(1):12–49, 1988.
49. Nikos Paragios and Rachid Deriche. Coupled geodesic active regions for image segmentation: A level set approach. In *ECCV*, pages 224–240. 2000.
50. P Jonathon Phillips, Harry Wechsler, Jeffery Huang, and Patrick J Rauss. The feret database and evaluation procedure for face-recognition algorithms. *Image and vision computing*, 16(5):295–306, 1998.
51. Robert E Schapire. The boosting approach to machine learning: An overview. In *Nonlinear estimation and classification*, pages 149–171. Springer, 2003.
52. Stan Sclaroff and John Isidoro. Active blobs. In *Computer Vision, 1998. Sixth International Conference on*, pages 1146–1153. IEEE, 1998.
53. Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.
54. Robert H Swendsen and Jian-Sheng Wang. Nonuniversal critical dynamics in monte carlo simulations. *Physical Review Letters*, 58(2):86–88, 1987.
55. Simon Thorpe, Denis Fize, Catherine Marlot, et al. Speed of processing in the human visual system. *Nature*, 381(6582):520–522, 1996.
56. Anne Treisman. Features and objects in visual processing. *Scientific American*, 255(5):114–125, 1986.
57. Zhuowen Tu, Xiangrong Chen, Alan L Yuille, and Song-Chun Zhu. Image parsing: Unifying segmentation, detection, and recognition. *International Journal of computer vision*, 63(2):113–140, 2005.
58. Zhuowen Tu and Alan L Yuille. Shape matching and recognition–using generative models and informative features. In *Computer Vision-ECCV 2004*, pages 195–209. Springer, 2004.
59. Zhuowen Tu and Song-Chun Zhu. Image segmentation by data-driven markov chain monte carlo. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(5):657–673, 2002.
60. Zhuowen Tu and Song-Chun Zhu. Parsing images into regions, curves, and curve groups. *International Journal of Computer Vision*, 69(2):223–249, 2006.
61. Shimon Ullman. Visual routines. *Cognition*, 18(1):97–159, 1984.
62. Shimon Ullman. Sequence seeking and counter streams: a computational model for bidirectional information flow in the visual cortex. *Cerebral cortex*, 5(1):1–11, 1995.
63. Paul Viola and Michael Jones. Fast and robust classification using asymmetric adaboost and a detector cascade. *Proc. of NIPS01*, 2001.
64. Jia-Ping Wang. Stochastic relaxation on partitions with connected components and its application to image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(6):619–636, 1998.
65. Jianxin Wu, James M Rehg, and Matthew D Mullin. Learning a rare event detection cascade by direct feature selection. In *NIPS*, page None, 2003.
66. Jonathan S Yedidia, William T Freeman, and Yair Weiss. Generalized belief propagation. In *NIPS*, pages 689–695, 2001.
67. Song Chun Zhu and Xiuwen Liu. Learning in gibbsian fields: How accurate and how fast can it be? *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):1001–1006, 2002.
68. Song Chun Zhu, Yingnian Wu, and David Mumford. Filters, random fields and maximum entropy (frame): Towards a unified theory for texture modeling. *International Journal of Computer Vision*, 27(2):107–126, 1998.
69. Song Chun Zhu and Alan Yuille. Region competition: Unifying snakes, region growing, and bayes/mdl for multiband image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 18(9):884–900, 1996.
70. Song-Chun Zhu, Rong Zhang, and Zhuowen Tu. Integrating bottom-up/top-down for object recognition by data driven markov chain monte carlo. In *CVPR*, volume 1, pages 738–745. IEEE, 2000.

Chapter 9
Hamiltonian and Langevin Monte Carlo

"Suppose we take a quantity of heat and change it into work. In doing so, we haven't destroyed the heat, we have only transferred it to another place or perhaps changed it into another energy form." - Isaac Asimov

Introduction

Hamiltonian Monte Carlo (HMC) is a powerful framework for sampling from high-dimensional continuous distributions. Langevin Monte Carlo (LMC) is a special case of HMC that is widely used in Deep Learning applications. Given an n-dimensional continuous density $P(X)$, the only requirement for implementing HMC is the differentiability of the energy $U(X) = -\log P(X)$. Like other MCMC methods (e.g. slice sampling, Swendsen-Wang cuts), HMC introduces auxiliary variables to facilitate movement in the original space. In HMC, the original variables represent *position*, and the auxiliary variables represent *momentum*. Each position dimension has a single corresponding momentum variable, so the joint space of the original and auxiliary variables has dimension $2n$, twice the size of the original space. Once the momentum variables are introduced, Hamilton's Equations are used to simulate the time evolution of a physical system with potential energy U. The properties of Hamilton's Equations ensure that movement in the joint space preserves the distribution of P in the original space.

9.1 Hamiltonian Mechanics

9.1.1 Hamilton's equations

The principles of Hamiltonian Mechanics are the foundation of the HMC sampling method. Hamiltonian Mechanics was originally developed as an alternative but equivalent formulation of Lagrangian Mechanics, and both are equivalent to Newtonian Mechanics. In Hamiltonian Mechanics, the states of a physical system are represented by a pair of n-dimensional variables q and p. The variable q represents *position* in the system, and p represents *momentum*. A joint state (q, p) provides a complete description of the physical system at a single instant in time. Position and momentum in the HMC framework can be interpreted as high-dimensional extensions of the familiar concepts of position and momentum in simple kinetic systems.

The evolution of a state (q, p) over time is governed by a scalar-valued function $H(q, p)$ representing the energy of the system, and a pair of partial differential equations known as Hamilton's Equations:

$$\frac{dq}{dt} = \frac{\partial H}{\partial p}, \tag{9.1}$$

$$\frac{dp}{dt} = -\frac{\partial H}{\partial q}. \tag{9.2}$$

$H(q,p)$ is often referred to as the *Hamiltonian* of the system. Updating q and p according to Hamilton's Equations ensures the conservation of many properties of properties of the system, including energy. In other words, $H(q,p)$ remains constant as (q,p) changes over time.

In many cases, including standard HMC and LMC, the Hamiltonian can be expressed in the *separable* form

$$H(q,p) = U(q) + K(p), \tag{9.3}$$

where $U(q)$ represents the potential energy of the system and $K(p)$ represents the kinetic energy. When the Hamiltonian is separable,

$$\frac{dq}{dt} = \frac{\partial H}{\partial p} = \frac{\partial K}{\partial p} \quad \text{and} \quad \frac{dp}{dt} = -\frac{\partial H}{\partial q} = -\frac{\partial U}{\partial q}, \tag{9.4}$$

and Hamilton's Equations have the simplified form

$$\frac{dq}{dt} = \frac{\partial K}{\partial p}, \tag{9.5}$$

$$\frac{dp}{dt} = -\frac{\partial U}{\partial q}. \tag{9.6}$$

9.1.2 A simple model of HMC

Consider a point mass moving on a frictionless 2D surface (Figure 9.1). Each instantaneous system state can be described by a pair (q,p), where q is a 2D variable giving the position of the point mass (latitude and longitude coordinates) and p is a 2D variable giving the momentum in each direction. The system Hamiltonian has the form $H(q,p) = U(q) + K(p)$, where $U(q)$ is the height of the point mass (relative to a fixed reference point) and the kinetic energy K has the form $K(p) = ||p||^2/(2m)$, where m is the mass.

The motion of the point mass is determined by Hamilton's Equations. Intuitively, Hamilton's Equations represent the tradeoff between kinetic and potential energy that occurs while traveling through the landscape. For example, a state with zero momentum on the side of a slope will be pulled downward, and its potential energy will be transferred into motion (kinetic energy) in the direction of steepest descent. On the other hand, if the point mass is moving forward along a flat plain and then encounters a barrier, kinetic energy is transferred to potential energy and the mass will slow or even reverse direction back towards the plain if the barrier is sufficiently steep.

This simple model is quite similar to the "physical system" of Hamiltonian Monte Carlo. In HMC, q represents a point in the n-dimensional space of the target density

$$P(q) = \frac{1}{Z}\exp\{-U(q)\}. \tag{9.7}$$

As in other MCMC methods, the intractable normalizing constant Z is not needed. HMC only requires that $U(q)$ is differentiable, because Hamilton's Equations involve the derivative of U. In practice, the momentum p follows a multivariate normal distribution, which has the energy function

$$K(p) = \frac{1}{2}\, p^\intercal \Sigma^{-1} p \tag{9.8}$$

for positive-definite covariance matrix Σ. The simplest choice is $\Sigma = \sigma^2 I_n$, which gives the standard kinetic energy equation $K(p) = ||p||^2/(2m)$ when $\sigma = \sqrt{m}$. HMC does not require a specific distribution for p, but the simplicity of sampling from a normal distribution and important connections to Information Geometry (see Section 9.5) make Gaussian momentum the natural choice.

At the beginning of each HMC iteration, a new momentum p is sampled. The joint state (q,p) is then updated using Hamilton's Equations to reach a proposal state (q^*, p^*). Unlike Random-Walk methods, where displacement scales with $\sqrt{t}$, displacement during an HMC trajectory can scale linearly in t. Because of this, HMC can make efficient global proposals in each iteration. In practice, Hamilton's Equations cannot be solved exactly and a Metropolis-Hastings

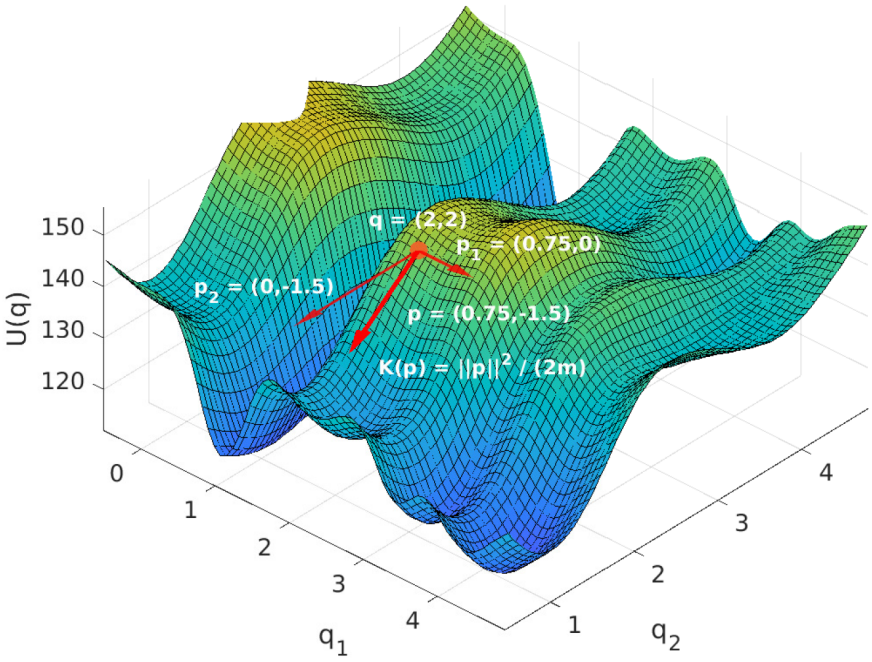


Fig. 9.1: Diagram of 2D Hamiltonian Mechanics at a single instant in time. The position q represents the original variables, and the momentum p is an auxiliary variable with the same dimension as q. The target energy $U(q)$ is a 2D surface in $\mathbb{R}^3$. The pair (q,p) completely describes the system state a single instant. The system has the Hamiltonian $H(q,p) = U(q) + K(p)$, and Hamilton's Equations (9.1) and (9.2) define how (q,p) changes over time. Introducing p allows for movement in q that preserves the distribution $P(q) = \frac{1}{Z}\exp\{-U(q)\}$.

acceptance step must be included as well. Crucially, the marginal distribution of q obtained from HMC has stationary distribution $P(q)$. The position variables q throughout the trajectory are kept as samples of $P(q)$ and the momenta p are discarded.

9.2 Properties of Hamiltonian Mechanics

Hamiltonian Mechanics has several important properties which ensure that HMC defines a valid sampling method. This section deals with the properties of the ideal, continuous-time solution to Hamilton's Equations. In practice, Hamilton's Equations cannot be solved explicitly and discrete numeric approximation must be used. Section 9.3 shows that the same properties can be preserved in a discrete-time implementation, *except* for conservation of energy.

9.2.1 Conservation of energy

Updating a physical system with Hamilton's Equations preserves the value of the Hamiltonian $H(q,p)$, so that the value of $H(q,p)$ should remain constant over time, even though q and p will vary. In other words, paths defined by Hamilton's Equations move along level curves of the Hamiltonian $H(q,p)$ (see Figure 9.2).

The proof of conservation of energy is straightforward:

$$\frac{dH}{dt} = \sum_{i=1}^{n}\left[\frac{\partial H}{\partial q_i}\frac{dq_i}{dt} + \frac{\partial H}{\partial p_i}\frac{dp_i}{dt}\right] = \sum_{i=1}^{n}\left[\frac{\partial H}{\partial q_i}\frac{\partial H}{\partial p_i} - \frac{\partial H}{\partial q_i}\frac{\partial H}{\partial p_i}\right] = 0.$$

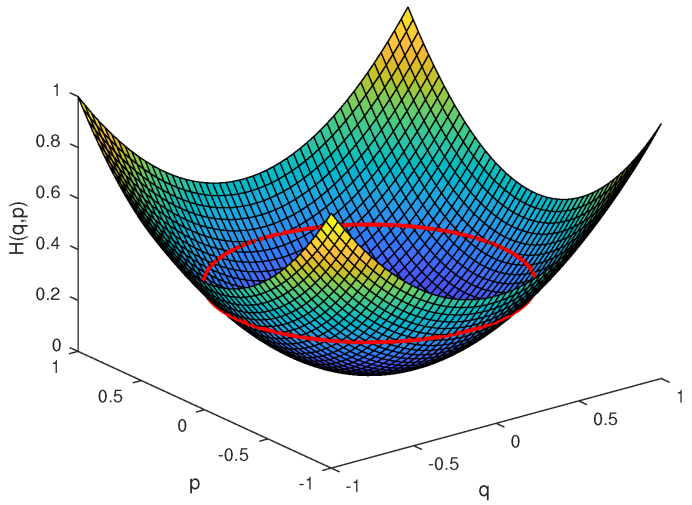


Fig. 9.2: Hamiltonian $H(q,p) = U(q) + K(p) = (q^2 + p^2)/2$ for $q \sim \mathrm{N}(0,1)$ and $p \sim \mathrm{N}(0,1)$. The path of a single HMC trajectory is restricted to a level curve of H. A possible level curve is shown in red.

This property is important in HMC because it ensures that $H(q,p) = H(q^*,p^*)$, where (q,p) is the previous state in the joint space and (q^*,p^*) is the proposed state. Combined with the other properties of Hamiltonian Mechanics, conservation of energy can be used to show that ideal HMC defines a Metropolis-Hastings proposal with an acceptance probability of 1. In practice, this property is only approximately true, since discrete numeric approximations must be used to solve Hamilton's Equations, so that $H(q,p)$ might differ from $H(q^*,p^*)$. If the approximation is accurate, this difference should be relatively small, and high acceptance probabilities can still be achieved.

9.2.2 Reversibility

The mapping from $(q(t),p(t))$ to $(q(t+s),p(t+s))$ defined by Hamiltonian Mechanics is unique and therefore invertible. If $H(q,p) = U(q) + K(p)$ and $K(p) = K(-p)$, which is true in standard HMC with Gaussian momentum, then the inverse mapping can be given explicitly by negating p at the end of the path, evolving the system for the same time s, and then negating p once more. In this case, the mapping $T : (p(t),q(t)) \mapsto (q(t+s),-p(t+s))$ is exactly reversible, since $T(q(t+s),-p(t+s)) = (q(t),p(t))$ (see Figure 9.3). Reversibility will be used to show that HMC satisfies detailed balance, which is the simplest way to prove that an MCMC method has the desired stationary distribution. Reversibility can be exactly preserved in a discrete implementation of Hamilton's Equations.

9.2.3 Symplectic structure and volume preservation

For any smooth function $H : \mathbb{R}^{2n} \to \mathbb{R}$, Hamilton's Equations define a special type of vector field and a symplectic structure on the manifold $\mathbb{R}^{2n}$. A symplectic manifold is a smooth manifold M (in practice, usually $\mathbb{R}^{2n}$) with a differential 2-form ω called the symplectic form. The standard symplectic form associated with Hamilton's Equations in $\mathbb{R}^{2n}$ is $\omega = \begin{pmatrix} 0 & I_n \\ -I_n & 0 \end{pmatrix}$, since

$$\frac{d}{dt}(q,p) = \omega \frac{dH}{d(q,p)}.$$

In general, ω only needs to be a closed and non-degenerate 2-form on M. The symplectic form can be intuitively understood as a way of generating a vector field from the differential 1-form dH of a Hamiltonian $H(q,p)$.

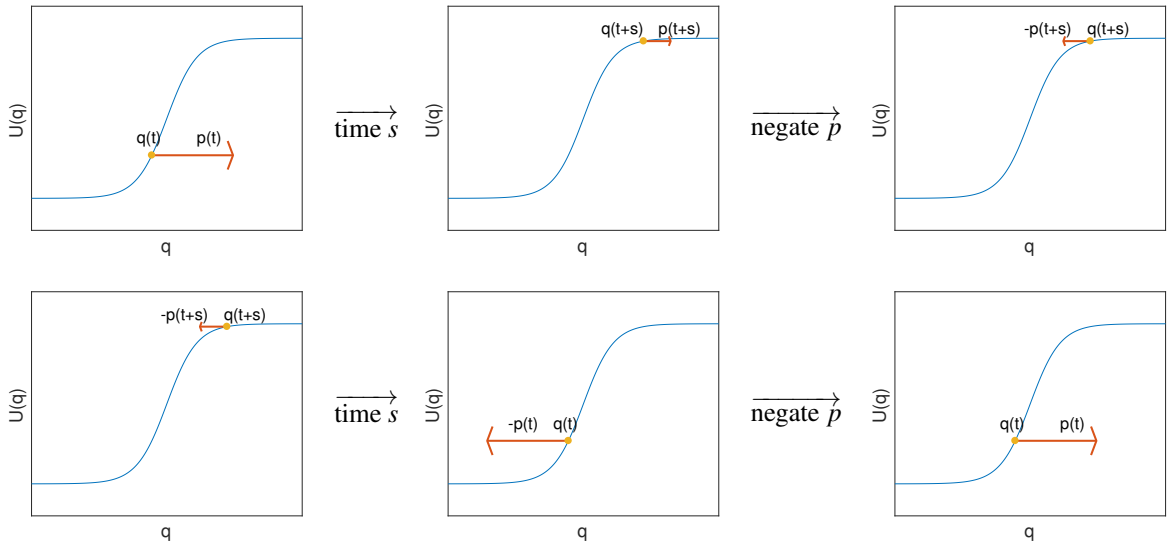


Fig. 9.3: Reversibility for 1D HMC with separable Hamiltonian $H(q,p) = U(q) + K(p)$ and $K(p) = K(-p)$. The coordinate change defined by updating a state (q,p) for time s according to Hamilton's Equations and negating p at the end of the trajectory (top row) can be exactly reversed by the same procedure (bottom row).

The solution obtained from integrating Hamilton's Equations (or equivalently flows on the vector field induced by the Hamiltonian H on the symplectic manifold M) has the important property of preserving the symplectic form ω. In other words, the mapping $(q(t), p(t)) \mapsto (q(t+s), p(t+s))$ over all $(q,p) \in M$ defines a diffemorphism from M to itself that respects the structure of ω. The invariance of ω under Hamiltonian flows is the mathematical foundation of the many conservation properties of Hamiltonian Mechanics, including conservation of energy.

An important consequence of the conservation of the 2-form ω is the conservation of volume under Hamilton's Equations, a result known as Louisville's Theorem. Using symplectic geometry, the proof of this theorem is quite simple. The non-degenerate 2-form ω on M can be raised to the n^{th} power to define a non-degenerate volume form ω^n (ω^n is a $2n$-form, since ω is a 2-form), and the conservation of ω under Hamiltonian flows implies the conservation of the volume form ω^n. This property is important for HMC because it ensures that the change of coordinates $(q,p) \mapsto (q^*,p^*)$ obtained from Hamilton's Equations has a Jacobian with a determinant of absolute value 1. Without volume preservation, the difficulty of calculating the determinant of the Jacobian to rescale the density after each proposal would make HMC infeasible in practice. Volume preservation can hold exactly even in discrete implementations of Hamilton's Equations if the correct updating scheme is used.

A simple proof of volume preservation can be given using only Hamilton's Equations without reference to symplectic geometry. Let $V = \left(\frac{dq}{dt}, \frac{dp}{dt}\right)$ be the vector field defined by Hamilton's Equations. Then the divergence of V is 0 everywhere, because

$$\text{div}(V) = \sum_{i=1}^{n} \left(\frac{\partial}{\partial q_i} \frac{dq_i}{dt} + \frac{\partial}{\partial p_i} \frac{dp_i}{dt} \right) = \sum_{i=1}^{n} \left(\frac{\partial}{\partial q_i} \frac{\partial H}{\partial p_i} - \frac{\partial}{\partial p_i} \frac{\partial H_i}{\partial q_i} \right) = \sum_{i=1}^{n} \left(\frac{\partial^2 H}{\partial q_i \partial p_i} - \frac{\partial^2 H}{\partial q_i \partial p_i} \right) = 0,$$

and a vector field with divergence 0 can be shown to preserve volume.

9.3 The Leapfrog Discretization of Hamilton's Equations

It is impossible to solve Hamilton's Equations exactly except in the simplest systems, so numeric implementations of Hamiltonian Mechanics must rely on a discrete approximation to the true continuous solution. Before discussing the most effective and widely used discretization, known as the Leapfrog Integrator, two less effective but instructive methods are introduced.

9.3.1 Euler's method

The most naive method of discretizing the time evolution of a Hamiltonian H under Hamilton's Equations is to update q and p simultaneously by some small step size ε as follows:

$$p(t+\varepsilon) = p(t) + \varepsilon \frac{dp}{dt}(q(t), p(t)) = p(t) - \varepsilon \frac{\partial H}{\partial q}(q(t), p(t)),$$
$$q(t+\varepsilon) = q(t) + \varepsilon \frac{dq}{dt}(q(t), p(t)) = q(t) + \varepsilon \frac{\partial H}{\partial p}(q(t), p(t)).$$

This discretization is known as Euler's Method. It does not preserve volume and can lead to inaccurate approximations after only a few steps.

9.3.2 Modified Euler's method

An improvement of Euler's Method is the Modified Euler's Method, which uses alternating ε-size updates of the q and p. When the energy function has the separable form $H(q,p) = U(q) + K(p)$, (9.5) and (9.6) hold, and updating q only depends on p (and vice versa). Based on this observation, the Modified Euler's Method consists of updating the current p from the current q with step size ε, and then updating the current q from the *updated* p with the same step size ε as follows:

$$p(t+\varepsilon) = p(t) - \varepsilon \frac{\partial U}{\partial q}(q(t)), \tag{9.9}$$
$$q(t+\varepsilon) = q(t) + \varepsilon \frac{\partial K}{\partial p}(p(t+\varepsilon)). \tag{9.10}$$

Reversing the update order of p and q is equally valid. The alternating updates are shear transformations, which preserve volume. By retaining the volume-preserving property of the true continuous solution to Hamilton's Equations, the Modified Euler Method provides much better discretization than the naive Euler Method. The Modified Euler's method only preserves volume when the Hamiltonian has the separable form $H(q,p) = U(q) + K(p)$.

Unlike the true solution to Hamilton's Equations, the Modified Euler's Method is *not* reversible because of the order of the updates. Suppose $K(p) = K(-p)$ and consider a proposal which starts from (q,p), updates p then q using Modified Euler to reach a new state (q^*, p^*), and finally negates momentum to reach $(q^*, -p^*)$. One could start a new chain at $(q^*, -p^*)$ and update q^* then $-p^*$ with Modified Euler to reach $(q, -p)$, and negate momentum to reach (q,p). On the other hand, starting at $(q^*, -p^*)$ and applying the original proposal with update order $-p^*$ then q^* would likely result in a state close but not equal to (q,p) due to discretization error. An ideal integrator should be able to exactly reverse all updates without changing the integrator itself (i.e. reversing the update order).

9.3.3 The Leapfrog integrator

The Leapfrog Integrator is a close relative of the Modified Euler's Method, and it is the standard discrete integration scheme used in HMC. When the Hamiltonian has the separable form $H(q,p) = U(q) + K(p)$, (9.5) and (9.6) hold and the Leapfrog Integrator satisfies both volume preservation and reversibility, which are desirable properties of the true continuous solution to Hamilton's Equations. A single step of the Leapfrog Integrator size is given below, where ε is a parameter for step size:

$$p(t+\varepsilon/2) = p(t) - (\varepsilon/2) \frac{\partial U}{\partial q}(q(t)), \tag{9.11}$$
$$q(t+\varepsilon) = q(t) + \varepsilon \frac{\partial K}{\partial p}(p(t+\varepsilon/2)), \tag{9.12}$$

$$p(t+\varepsilon) = p(t+\varepsilon/2) - (\varepsilon/2)\frac{\partial U}{\partial q}(q(t+\varepsilon)). \tag{9.13}$$

A Leapfrog update consists of a $(\varepsilon/2)$-size update of p with the old q, followed by a ε-size update of q with the new p, followed by a $(\varepsilon/2)$-size update of p with the new q. When performing multiple Leapfrog steps, the above scheme is equivalent to performing half-step updates of p only at the very beginning and very end of the trajectory, and alternating between full step updates of q and p in between, since the two $(\varepsilon/2)$-size updates of p and the end of an old step and beginning of a new step are equivalent to a single ε-size update of p.

The only difference between the Modified Euler's Method and the Leapfrog Method is the splitting of the initial ε-size update of p in a Modified Euler trajectory into two $(\varepsilon/2)$-step updates of p at the beginning and the end of a Leapfrog trajectory.

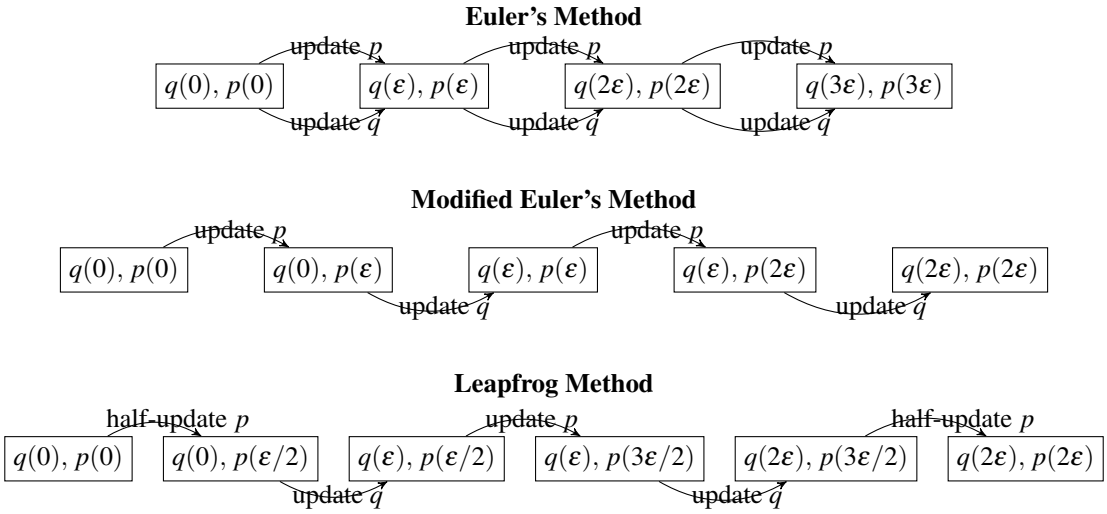


Fig. 9.4: Visualization of three methods of integrating Hamilton's Equations. *Top:* Euler's Method. Updating p and q simultaneously yields poor approximations. *Middle:* Modified Euler's Method. Alternating updates of p and q conserve volume when the Hamiltonian has separable form $H(q,p) = U(q) + K(p)$. Not reversible because of update order. *Bottom:* Leapfrog Method. The Leapfrog Method is identical to the Modified Euler's method except for the half-step p updates at the beginning and end of the trajectory. Half-updates needed for reversibility.

9.3.4 Properties of the Leapfrog integrator

The symmetry of the Leapfrog Integrator ensures reversibility, because a single Leapfrog step can be reversed by negating p, applying the Leapfrog Integrator, and negating p again. The negation of p at the end of a trajectory is needed for reversibility in the general case of HMC, but it can be ignored in practice when Gaussian auxiliary variables are used because $K(p) = K(-p)$.

The Leapfrog Integrator is volume-preserving for the same reason as the Modified Euler's Method: when the Hamiltonian has the separable form $H(q,p) = U(q) + K(p)$, the updates of q only depend on p and vice-versa. Therefore the change of coordinates $(q(t), p(t)) \mapsto (q(t+\varepsilon), p(t+\varepsilon))$ defined by Leapfrog equations (9.11), (9.12) and (9.13) is a composition of three shear transformations, each with a Jacobian of determinant 1. The composition defines a single coordinate change with Jacobian determinant 1, since the Jacobian determinant of the composition is the product of the Jacobian determinants of the individual coordinate changes.

An informal proof that the mapping $(q(t), p(t)) \mapsto (q(t), p(t+\varepsilon/2))$ defined by (9.11) is a shear transformation is given below. An almost identical proof can be used to show that (9.12) and (9.13), as well as (9.9) and (9.10), are also shear transformations.

Let $J_p = \begin{pmatrix} \frac{\partial q^*}{\partial q} & \frac{\partial q^*}{\partial p} \\ \frac{\partial p^*}{\partial q} & \frac{\partial p^*}{\partial p} \end{pmatrix}$ be the Jacobian of the coordinate change $(q,p) \mapsto (q^*,p^*)$ corresponding to $(q(t),p(t)) \mapsto (q(t),p(t+\varepsilon/2))$. Consider some initial state $(q(0),p(0))$ and its close neighbor $(q'(0),p'(0)) = (q(0)+\delta u_q, p(0)+\delta u_p)$ for some unit vector $u = (u_q,u_p)$ and some small $\delta > 0$. The first step of the Leapfrog update for these two states is given by

$$p(\varepsilon/2) = p(0) - (\varepsilon/2)\frac{\partial U}{\partial q}(q(0)),$$

$$p'(\varepsilon/2) = p'(0) - (\varepsilon/2)\frac{\partial U}{\partial q}(q'(0)) = p(0) + \delta u_p - (\varepsilon/2)\frac{\partial U}{\partial q}(q(0)+\delta u_q),$$

and $q(\varepsilon/2) = q(0)$, $q'(\varepsilon/2) = q'(0) = q(0)+\delta u_q$ since q is not updated during this step. Using a Taylor Expansion, $\frac{\partial U}{\partial q}(q(0)+\delta u_q) \approx \frac{\partial U}{\partial q}(q(0)) + \delta[\frac{\partial^2 U}{\partial q^2}(q(0))]u_q$ for small δ. Therefore

$$\begin{pmatrix} q'(\varepsilon/2) - q(\varepsilon/2) \\ p'(\varepsilon/2) - p(\varepsilon/2) \end{pmatrix} \approx \delta \begin{pmatrix} I_n & 0 \\ -(\varepsilon/2)\frac{\partial^2 U}{\partial q^2}(q(0)) & I_n \end{pmatrix} \begin{pmatrix} u_q \\ u_p \end{pmatrix}$$

and letting δ go to 0 implies

$$J_p = \begin{pmatrix} \frac{\partial q^*}{\partial q} & \frac{\partial q^*}{\partial p} \\ \frac{\partial p^*}{\partial q} & \frac{\partial p^*}{\partial p} \end{pmatrix} = \begin{pmatrix} I_n & 0 \\ -(\varepsilon/2)\frac{\partial^2 U}{\partial q^2} & I_n \end{pmatrix}$$

which is a shear matrix with determinant 1. Note that ε is arbitrary and fixed in this proof, and that the limit is taken only in the spatial perturbation δ. The Leapfrog Integrator exactly preserves volume for any ε. The other two steps of a Leapfrog update can be shown to be shear transformations by using a Taylor Expansion of the derivative $\frac{\partial U}{\partial q}$ or $\frac{\partial K}{\partial p}$ in the same way.

If a Gaussian auxiliary variable is used in HMC, the q-update given by Equation (9.12) has a Jacobian of the form

$$J_q = \begin{pmatrix} I_n & \varepsilon\Sigma^{-1} \\ 0 & I_n \end{pmatrix}$$

where Σ is the covariance matrix of p. Using a Gaussian proposal with $\Sigma \approx \frac{\partial^2 U}{\partial q^2}$ can dramatically improve movement through the q-space, especially when sampling from distributions with high ratios between the constrained width of the largest and smallest linear directions, i.e. a large ratio between the maximum and minimum eigenvalue of the local covariance. Unfortunately, if the energy function U does not have constant curvature, then the ideal choice of Σ varies with the position q, in which case the $H(q,p)$ is no longer separable, the Leapfrog Integrator does not preserve volume, and solving Hamilton's Equations becomes much more difficult. See Section 9.5 for a detailed discussion.

9.4 Hamiltonian Monte Carlo and Langevin Monte Carlo

This section introduces the HMC method, along with a special case of HMC known as Langevin Monte Carlo (LMC, also known as the Metropolis-Adjusted Langevin Algorithm or MALA). HMC tuning is discussed, and the section ends with a proof that HMC satisfies detailed balance.

9.4.1 Formulation of HMC

To be consistent with the notation used in the preceding sections, the target density to be sampled during HMC will be written as

$$P(q) = \frac{1}{Z}\exp\{-U(q)\} \tag{9.14}$$

for $q \in \mathbb{R}^n$ and a smooth potential energy function $U : \mathbb{R}^n \to \mathbb{R}$ with normalizing constant Z. For a discussion of handling constraints in HMC so that q can be restricted to a set $U \subset \mathbb{R}^n$, see [7]. In Bayesian inference, U is the negative log of the posterior distribution for a set of parameters q and dataset X with a prior π and log-likelihood l, i.e.

$$U(q) = -\log[\pi(q)] - l(X|q).$$

HMC is an auxiliary variable method, and the standard auxiliary variables are $p \sim \mathrm{N}(0, \Sigma)$ with negative log density

$$K(p) = \frac{1}{2} p^\intercal \Sigma^{-1} p \tag{9.15}$$

for some $n \times n$ positive-definite covariance matrix Σ. The pair $(q, p) \in \mathbb{R}^{2n}$ has the joint density

$$P(q,p) = \frac{1}{Z} \exp\{-H(q,p)\} = \frac{1}{Z} \exp\left\{-U(q) - \frac{1}{2} p^\intercal \Sigma^{-1} p\right\} \tag{9.16}$$

and the joint energy function

$$H(q,p) = U(q) + K(p) = U(q) + \frac{1}{2} p^\intercal \Sigma^{-1} p. \tag{9.17}$$

The joint density $P(q,p) = \frac{1}{Z} e^{-H(q,p)}$ has a marginal distribution $q \sim \frac{1}{Z} \exp^{-U(q)}$ because

$$\int_{\mathbb{R}^n} P(q,p) dp = \frac{1}{Z_q} e^{-U(q)} \int_{\mathbb{R}^n} \frac{1}{Z_p} e^{-\frac{1}{2} p^\intercal \Sigma^{-1} p} dp = \frac{1}{Z_q} e^{-U(q)}. \tag{9.18}$$

Therefore sampling from the joint density $P(q,p)$ will provide samples q that follow the target density $P(q)$.

Only normally distributed auxiliary variables will be discussed in this book, and normal auxiliary variables are the natural choice for several reasons. The standard kinetic energy function $K(p) = ||p||^2/m$ used in physics is equivalent to (9.15) with $\Sigma = mI_n$. Normal distributions can be simulated accurately and efficiently. More importantly, HMC with Gaussian momentum has two important connections with Information Geometry that shed light on the correct tuning of the momentum covariance Σ (see Section 9.5).

9.4.2 The HMC algorithm

Let U be the target energy function and let q be the current state. First, a normal auxiliary variable p is sampled from $N(0, \Sigma)$. Then, L Leapfrog updates of step size ε are performed on the state (q, p). Finally, a Metropolis-Hastings step is used to accept or reject the proposal (q^*, p^*) to correct for any discretization error from the Leapfrog Integrator. After the acceptance step, p^* is discarded and a new p is generated for the next HMC step. The standard HMC algorithm is given below.

Remark 9.1. To be fully correct, the proposed state at the end of Step 2 in the HMC algorithm should be $(q^*, p^*) = (q'_L, -p'_L)$ since a negation of the momentum at the end of the Leapfrog trajectory is needed to ensure reversibility and detailed balance for HMC, as will be shown in the next section. Since $K(p) = K(-p)$ for Gaussian distributions, the calculation in Step 3 does not change whether $p^* = p'_L$ or $p^* = -p'_L$, and the negation can be safely ignored.

Remark 9.2. A different covariance matrix Σ can be used to generate each p_i. However, the same Σ must be used for the duration of a single proposal. Changing Σ between Leapfrog iterations breaks the shear structure of the Leapfrog updates, and detailed balance can no longer be guaranteed. This is a major obstacle for RMHMC approaches which take local manifold structure into account by allowing a dependence $\Sigma(q)$ based on the current location in the landscape.

The Metropolis-Hastings acceptance probability in Step 3 corresponds to a ratio of the joint density of $P(q,p)$:

$$\alpha = \min\left(1, \frac{P(q^*, p^*)}{P(q_{i-1}, p_{i-1})}\right)$$

Algorithm 9 HMC Algorithm

Input: Differentiable energy function $U(q)$, initial state $q_0 \in \mathbb{R}^n$, $n \times n$ positive definite covariance matrix Σ, step size ε, number of Leapfrog steps L, number of iterations N

Output: Markov Chain sample $\{q_1, \ldots, q_N\}$ with stationary distribution U

for $i = 1 : N$ **do**

1. Generate momentum $p_{i-1} \sim \mathrm{N}(0, \Sigma)$.
2. Let $(q'_0, p'_0) = (q_{i-1}, p_{i-1})$. Perform L Leapfrog updates starting at (q'_0, p'_0) to reach a proposal state (q^*, p^*) as follows:

   a. Do the first half-step update for p,

   $$p'_{\frac{1}{2}} = p'_0 - (\varepsilon/2)\,\frac{\partial U}{\partial q}(q'_0). \tag{9.19}$$

   b. For $l = 1 : (L-1)$, perform alternating full-step updates of q and p:

   $$q'_l = q'_{l-1} + \varepsilon \Sigma^{-1} p'_{l-\frac{1}{2}}, \tag{9.20}$$

   $$p'_{l+\frac{1}{2}} = p'_{l-\frac{1}{2}} - \varepsilon\,\frac{\partial U}{\partial q}(q'_l). \tag{9.21}$$

   If $L = 1$, which is the LMC algorithm, skip this step.

   c. Compute the final full-step q-update and the final half-step p-update

   $$q'_L = q'_{L-1} + \varepsilon \Sigma^{-1} p'_{L-\frac{1}{2}}, \tag{9.22}$$

   $$p'_L = p'_{L-\frac{1}{2}} - (\varepsilon/2)\,\frac{\partial U}{\partial q}(q'_L). \tag{9.23}$$

   The proposed state is then $(q^*, p^*) = (q'_L, p'_L)$.
3. Accept the proposed state (q^*, p^*) according the Metropolis-Hastings acceptance probability

   $$\alpha = \min\left(1, \exp\left\{-\left(U(q^*) + \frac{1}{2}(p^*)^\intercal \Sigma^{-1} p^*\right) + \left(U(q_{i-1}) + \frac{1}{2} p_{i-1}^\intercal \Sigma^{-1} p_{i-1}\right)\right\}\right). \tag{9.24}$$

   If the proposal is accepted, then $q_i = q^*$. Otherwise, $q_i = q_{i-1}$. The momenta p_{i-1} and p^* can be discarded after the proposal.

end for

$$= \min\left(1, \frac{\exp\{-H(q^*, p^*)\}}{\exp\{-H(q_{i-1}, p_{i-1})\}}\right)$$
$$= \min\left(1, \frac{\exp\{-U(q^*) - K(p^*)\}}{\exp\{-U(q_{i-1}) - K(p_{i-1})\}}\right).$$

The Leapfrog Update is deterministic, volume preserving, and exactly reversible, so no transition probabilities $Q((q,p) \mapsto (q^*, p^*))$ appear in the Metropolis-Hastings ratio, only the density $P(q,p)$. The true continuous solution to Hamilton's Equations exactly satisfies $H(q,p) = H(q^*, p^*)$ for a proposal (q^*, p^*) generated according to Hamilton's Equations from an initial state (q,p). Therefore, the Metropolis-Hastings acceptance probability would always be equal to 1 if the exact solution for Hamilton's Equation was available.

Since the Leapfrog discretization must be used instead, the value of H is not exactly conserved, and a Metropolis-Hastings step is needed to correct for this error. It is necessary to tune the sampling variables Σ, ε, and L correctly in order to obtain an accurate approximation to Hamilton's Equations and a high acceptance probability. In theory, HMC has a stationary distribution $\frac{1}{Z} e^{-U(q)}$ for any parameter setting, but good tuning is needed for good mixing as with any MCMC method. See Section 9.4.4 for details about tuning HMC parameters.

Sampling a new momentum p_i in each iteration is needed to satisfy ergodicity in HMC. Recall that $H(q,p)$ remains (approximately) constant as (q,p) are updated. If $H(q,p) = U(q) + K(p)$ and $K(p) = (1/2)p^\intercal \Sigma^{-1} p$, then clearly $U(q) \leq h = H(q,p)$ for all q in the trajectory, which limits the possible states that can be visited (in particular, $\{q : U(q) > h\}$ cannot be reached). Each new momentum restricts exploration to a single level curve of the Hamiltonian $H(q,p)$, which might not cover the entire space of q. Refreshing momentum permits movement along different level curves of $H(q,p)$, and this is necessary for the ergodicity of HMC.

9.4.3 The LMC algorithm

Langevin Monte Carlo, or LMC, is simply the HMC algorithm where only $L = 1$ Leapfrog update is performed. LMC is equivalent to the Langevin Equation

$$q(t+\varepsilon) = q(t) - \frac{\varepsilon^2}{2}\Sigma^{-1}\frac{\partial U}{\partial q}(q(t)) + \varepsilon\sqrt{\Sigma^{-1}}z \tag{9.25}$$

(with $z \sim \mathrm{N}(0, I_n)$) used in optimization with an additional p-update and Metropolis-Hastings acceptance step which results in a sampling algorithm on $\frac{1}{Z}e^{-U(q)}$. The LMC algorithm could be implemented using the HMC algorithm from the previous section with $L = 1$, but is usually done in a slightly more compact way with only one q-update and one p-update, as written below.

Algorithm 10 LMC Algorithm

Input: Differentiable energy function $U(q)$, initial state $q_0 \in \mathbb{R}^n$, $n \times n$ positive definite covariance matrix Σ, step size ε, number of iterations N

Output: Markov Chain sample $\{q_1, \ldots, q_N\}$ with stationary distribution U

for $i = 1 : N$ **do**

1. Generate momentum $p_{i-1} \sim \mathrm{N}(0, \Sigma)$.
2. Let $(q_0', p_0') = (q_{i-1}, p_{i-1})$. Update q according the the Langevin Equation:

$$q_1' = q_0' - \frac{\varepsilon^2}{2}\Sigma^{-1}\frac{\partial U}{\partial q}(q_0') + \varepsilon\Sigma^{-1}p \tag{9.26}$$

and update p according to the Leapfrog Update

$$p_1' = p_0' - \frac{\varepsilon}{2}\frac{\partial U}{\partial q}(q_0') - \frac{\varepsilon}{2}\frac{\partial U}{\partial q}(q_1'). \tag{9.27}$$

The proposed state is then $(q^*, p^*) = (q_1', p_1')$.

3. Accept the proposed state (q^*, p^*) according the Metropolis-Hastings acceptance probability

$$\alpha = \min\left(1, \exp\left\{-\left(U(q^*) + \frac{1}{2}(p^*)^\intercal\Sigma^{-1}p^*\right) + \left(U(q_{i-1}) + \frac{1}{2}p_{i-1}^\intercal\Sigma^{-1}p_{i-1}\right)\right\}\right). \tag{9.28}$$

If the proposal is accepted, then $q_i = q^*$. Otherwise, $q_i = q_{i-1}$. The momentum p_{i-1} can be discarded after the proposal.

end for

The formulation of the q-update in (9.26) shows the two competing forces acting on the original space in LMC, and the same principles are at work in each Leapfrog update of HMC. The term $-\frac{\varepsilon^2}{2}\Sigma^{-1}\frac{\partial U}{\partial q}(q_0')$ is simply gradient descent rescaled by a positive definite matrix, corresponding roughly to "gravitational pull" in the energy landscape. When the momentum covariance is the Fisher Information $\Sigma(\theta) = \mathrm{E}_{X|\theta}\left[\frac{\partial^2 U}{\partial\theta^2}(X|\theta)\right]$ for a Bayesian inference problem given observations X, this term becomes the *natural gradient* $\Sigma^{-1}(\theta)\frac{\partial U}{\partial\theta}$ (Amari, [1]) which adapts to the local curvature of the parameter space. The Fisher Information is always positive-definite, and the natural gradient has better performance and invariance properties than the naive gradient $\frac{\partial U}{\partial\theta}$.

The term $\varepsilon\Sigma^{-1}p = \varepsilon\sqrt{\Sigma^{-1}}z$ for $z \sim \mathrm{N}(0, I_n)$ corresponds roughly to random "winds". The gravitational pull of the gradient term should overpower the random forces of the diffusion term along an energy slope, but the diffusion term becomes dominant once a local minimum has been reached and the gradient vanishes. An informed choice of Σ is needed to ensure that the random diffusion forces can make meaningful proposals once the chain has reached the bottom of an energy basin. If $\Sigma(q) \approx \frac{\partial^2 U}{\partial q^2}$, then $\sqrt{\Sigma(q)^{-1}}z \sim \mathrm{N}(0, \Sigma(q)^{-1})$ and the diffusion forces follow the local covariance structure $[\frac{\partial^2 U}{\partial q^2}]^{-1}$, so that the "wind" primarily blows along the local manifold. Imagine that the local landscape is a canyon (see Figure 9.5). If the winds blows perpendicular to the walls of the canyon, the steep edges of the canyon will prevent any meaningful movement. However, if the wind direction is parallel to the canyon valley, movement through the canyon becomes possible.

LMC has different properties than "full" HMC where a substantial number of Leapfrog updates L are used. Since the momenta are discarded after only a single step in LMC, successive proposals are not encouraged to move in the same direction and LMC explores the landscape $U(q)$ in a random walk. Unlike Random-Walk Metropolis-Hastings, which makes no use of local geometry, the gradient information used in the LMC update facilitates sampling in the optimization phase, but HMC has better theoretical properties in the diffusion phase because repeatedly updating the same momentum p can lead to longer trajectories in the joint space.

However, LMC can be more useful than full HMC in some situations. It is more practical and accurate to implement an approximation of the dynamics of a q-dependent momentum $p \sim \mathrm{N}(0, \Sigma(q))$ for LMC than HMC, as will be discussed in the RMHMC section. In complex landscapes, the benefits of HMC are limited by the instability of the

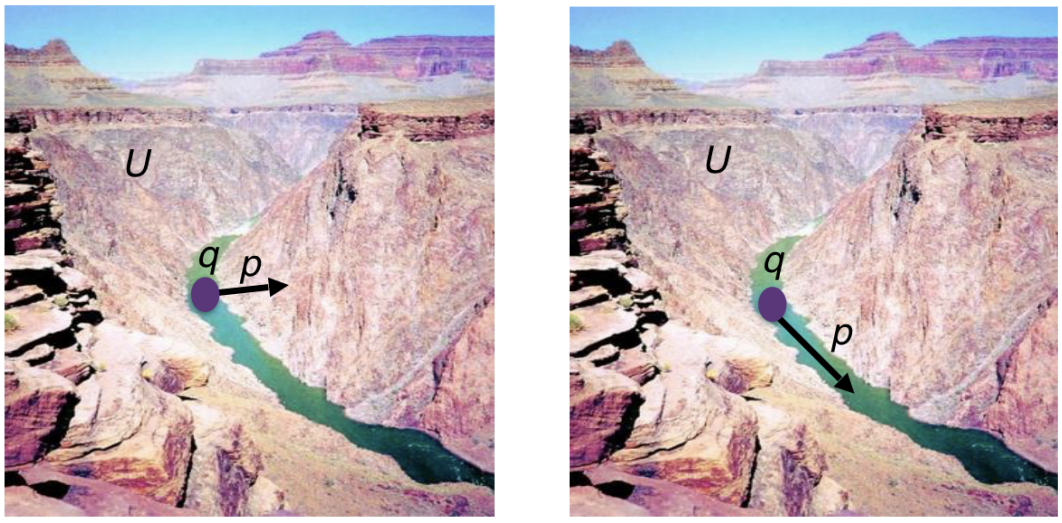


Fig. 9.5: Diffusion in the valley of a "canyon" potential U. *Left:* When the momentum p from HMC points towards the steep canyon wall, movement in q will quickly be reversed and the particle gets stuck. *Right:* When the momentum from HMC points along the canyon valley, free movement in q is possible because no barriers are encountered. Efficient sampling near the local minima of U requires a momentum covariance that reflects local scaling (see Section 9.5).

Leapfrog dynamics, and often the number of Leapfrog steps L must be kept small to achieve reasonable acceptance rates, in which case HMC and LMC are very similar.

9.4.4 Tuning HMC

This section discusses tuning the ε and L parameters in the standard HMC setting with a fixed Σ. It is sufficient to only consider $\Sigma = I_n$, because the results can naturally be extended to an arbitrary Σ by rescaling the q-space as in Lemma 9.1. Tuning Σ is the main topic of Section 9.5.

The step size ε must be small enough for the Leapfrog Integrator to accurately simulate Hamilton's Equations in order for H to stay approximately constant throughout the Leapfrog updates. Since the Metropolis-Hastings acceptance depends on the difference in H between the original state and the proposal, smaller step sizes tend to have higher acceptance rates because H will not change as much. On the other hand, if ε is too small then the chain will remain nearly stationary and effective sampling becomes impossible. A simple way to tune HMC in a new environment is to set $\Sigma = I_n$, $L = 1$, and vary ε until acceptance rates of 40% – 85% are obtained. This range provides a good balance between high acceptance and good movement, and more extreme ε in either direction are unlikely to substantially improve HMC performance. Different values of ε might be needed in different areas of the state space.

When $\Sigma = I_n$, the ideal step size ε^* should roughly equal the width of $U(q)$ in the most constrained linear direction of the local region of the energy landscape. If the landscape is Gaussian or approximately Gaussian, ε^* should be close to the square root of the smallest eigenvalue of the local covariance matrix. The q-update in (9.20) will lead to low acceptance rates when ε is substantially larger than the smallest marginal standard deviation of $U(q)$ because the spherical auxiliary variable will make unlikely proposals along the most constrained direction. On the other hand, proposals should be accepted with reasonably high probability when ε and the smallest standard deviation are about equal because local deviations in any direction will give states with about the same energy as the current state.

Since each Leapfrog update moves a distance of about ε in the q-space, ignoring the effect of the gradient, and ε is limited to at most the smallest marginal standard deviation of q, the number of leapfrog steps L^* needed to reach a nearly independent state in a single HMC step is $L^* \approx \sqrt{\lambda_{\max}}/\varepsilon^*$, where $\lambda_{\max}$ is the largest eigenvalue of the local covariance of the q-space. Remember that the Hamiltonian trajectory is *not* a random walk and tends to move in the same direction unless obstacles in the energy landscape are encountered, so displacement scales linearly with the number of steps L. In simple landscapes, L can be quite large (over 100) because the Leapfrog dynamics are very accurate, but in more complex landscapes the acceptance rates for HMC trajectories can drop very quickly when

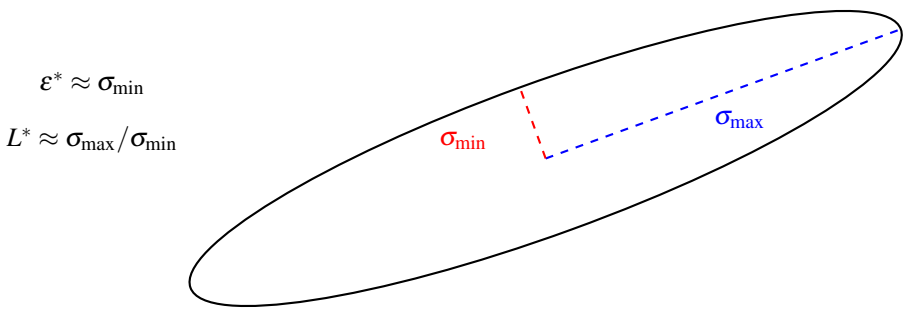


Fig. 9.6: Level curve (one standard deviation) for $q \sim \mathrm{N}(0, \Phi)$ with $\Sigma = U \begin{pmatrix} \sigma_{\min}^2 & 0 \\ 0 & \sigma_{\max}^2 \end{pmatrix} U^\intercal$. Let $p \sim \mathrm{N}(0, I_n)$. The optimal step size is $\varepsilon^* \approx \sigma_{\min}$, because larger proposals will lead to high-energy regions of U that will likely be rejected, while smaller proposals are less efficient. On the other hand, setting $L^* \approx \sigma_{\max}/\sigma_{\min}$ is needed to make global proposals in each HMC iteration. This can be problematic when $\sigma_{\max}/\sigma_{\min}$ is large, because Leapfrog dynamics usually become unstable after many updates. The same principles apply to momentum covariance $\Sigma \neq I_n$ after the re-scaling in Section 9.5.1.

L becomes too large. If εL is not on the order of the largest standard deviation, HMC will exhibit auto-correlation and will not be able to move through the space effectively. See Section 9.6 for experimental demonstrations of these principles.

The local correlation could vary drastically throughout the state space, and a parameter setting that is effective in one mode of the landscape might perform badly in another mode. However, using RMHMC (Section 9.5) alleviates these problems because ε becomes a "dimensionless" quantity and small numbers of Leapfrog steps (even $L = 1$) can still provide good movement through the space.

9.4.5 Proof of detailed balance for HMC

The simplest way show that an MCMC sampling method preserves a distribution P is to show that the method satisfies the detailed balance equation

$$P(x)T(x \mapsto x^*) = P(x^*)T(x^* \mapsto x) \tag{9.29}$$

for the proposal density T defined by the MCMC method. A proof that HMC satisfies detailed balance is given below. It can also be shown that HMC is ergodic and guaranteed to explore the entire q-space, provided that ε is selected from a small random interval at the beginning of each HMC update. This random selection of ε is needed to ensure ergodicity in theory, because there is the possibility for exactly or nearly periodic orbits to occur during HMC, but this phenomenon only exists for a narrow band of ε. See [5] for more details.

Theorem 9.1 *The HMC Algorithm satisfies detailed balance and has a stationary distribution* $P(q) = \frac{1}{Z}e^{-U(q)}$.

Proof. It is sufficient to show that an HMC step satisfies detailed balance for the joint distribution $P(q,p) = \frac{1}{Z}e^{-U(q)-K(p)}$ in order to show that the stationary distribution of q in the HMC process follows $\frac{1}{Z}e^{-U(q)}$, as shown in (9.18). For this proof, $p \sim \frac{1}{Z}e^{-K(p)}$ for a smooth energy function K on $\mathbb{R}^n$ with $K(p) = K(-p)$ (not necessarily Gaussian).

Let $q \sim \frac{1}{Z}e^{-U(q)}$. After generating $p \sim \frac{1}{Z}e^{-K(p)}$ in Step 1 in the HMC algorithm, it is clear that $(q,p) \sim P(q,p)$ because of the independence of q and p implied by the factorizable form of the joint density. Let the proposal (q^*, p^*) be the state reached after performing L Leapfrog steps of size ε from the state (q,p) and negating p at the end of the trajectory. As shown in Section 9.3.4, each Leapfrog step is a change of coordinates with determinant 1, and the negation of p at the end of the trajectory is a change of coordinates with Jacobian determinant of absolute value 1. Therefore $(q,p) \mapsto (q^*, p^*)$ is a change of coordinates with a Jacobian determinant of absolute value 1, since the determinant of the composition of coordinate changes is the product of the determinants of each change. By the change of coordinate rule for probability densities

$$g(y) = f(x)\left|\det\left(\frac{dx}{dy}\right)\right|$$

where $f(x)$ is the original density and $g(y)$ is the new density for the mapping $x \mapsto y$, it follows that (q^*, p^*) has the same density function as (q, p) because $|\det(dx/dy)| = 1$. The proposal is also exactly reversible, since applying L Leapfrog steps of size ε to (q^*, p^*) and negating p^* at the end of the trajectory will give the original state (q, p).

Since the mapping $(q, p) \mapsto (q^*, p^*)$ is deterministic and reversible, the proposal density T defined by the HMC Algorithm is

$$T((q,p) \mapsto (q^*,p^*)) = \min\left(1, \exp\left\{-\left(U(q^*)+K(p^*)\right)+\left(U(q)+K(p)\right)\right\}\right),$$
$$T((q,p) \mapsto (q',p')) = 0 \quad \text{if} \quad (q',p') \neq (q^*,p^*).$$

Similarly, the transition density starting from (q^*, p^*) is non-zero only for the proposal (q, p) and has the form

$$T((q^*,p^*) \mapsto (q,p)) = \min\left(1, \exp\left\{-\left(U(q)+K(p)\right)+\left(U(q^*)+K(p^*)\right)\right\}\right).$$

The the detailed balance equation (9.29) for HMC is

$$\frac{1}{Z}e^{-U(q)-K(p)}\min\left(1, \frac{\exp\{-U(q^*)-K(p^*)\}}{\exp\{-U(q)-K(p)\}}\right) = \frac{1}{Z}e^{-U(q^*)-K(p^*)}\min\left(1, \frac{\exp\{-U(q)-K(p)\}}{\exp\{-U(q^*)-K(p^*)\}}\right)$$

which is clearly true. Therefore HMC satisfies detailed balance and preserves the joint distribution $\frac{1}{Z}e^{-U(q)-K(p)}$.□

9.5 Riemann Manifold HMC

Riemann Manifold HMC (or RMHMC) extends the standard HMC method by allowing the covariance matrix for the auxiliary momentum variables p to have a dependency $\Sigma(q)$ on the current location q in the energy landscape. This can vastly improve the sampling properties of HMC, especially in situations where the distribution of q is concentrated along a low-dimensional manifold in the state space.

Traditional HMC with $\Sigma = I_n$ is ineffective in these situations because the step size needed for acceptance must be on the order of the smallest standard deviation of q, which will be orders of magnitude smaller than the standard deviations along the primary manifold dimensions. Using a large number of Leapfrog steps L can only partially compensate for this discrepancy, and in complex landscapes the trajectory can become unstable when L is too large.

RMHMC, on the other hand, uses local geometry to make proposals in meaningful directions along the local manifold, leading to better sampling with only a few Leapfrog steps. The dependency $\Sigma(q)$ complicates the dynamics of RMHMC and requires additional computational considerations, some of which are very problematic. While an exact RMHMC implementation is infeasible in many practical situations, approximate implementations could provide many of the benefits of RMHMC in a flexible and generalizable framework.

9.5.1 Linear transformations in HMC

The lemma below states an important invariance property of HMC under a certain type of linear transformation, elucidating the role of the proposal covariance Σ in HMC.

Lemma 9.1. *Let $U(q)$ be a smooth energy function and let $p \sim N(0, \Sigma)$ be the distribution of the HMC auxiliary variables for positive definite matrix Σ, and let A be an invertible matrix. The HMC dynamics of (q, p) initialized at (q_0, p_0) are equivalent to the HMC dynamics of $(q', p') = (Aq, (A^\intercal)^{-1}p)$ initialized at $(q'_0, p'_0) = (Aq_0, (A^\intercal)^{-1}p_0)$ because $(Aq_t, (A^\intercal)^{-1}p_t) = (q'_t, p'_t)$ for any Leapfrog step $t \geq 1$.*

Proof. Let $(q', p') = (Aq, (A^\intercal)^{-1}p)$. By the change of variables formula for probability densities, $P'(q') = P(q)/|\det(A)|$, and since A is constant, the new denominator is absorbed into the normalizing constant, so the energy functions of q and q' differ only by an additive constant: $U'(q') = U(A^{-1}q') + c$. Using the chain rule,

$$\frac{\partial U'}{\partial q}(q^*) = (A^\intercal)^{-1}\frac{\partial U}{\partial q}(A^{-1}q^*)$$

for any vector q^*. The transformed momenta have a distribution $p' \sim \mathrm{N}(0, (A^\intercal)^{-1}\Sigma A^{-1})$ and energy function $K'(p') = A\Sigma^{-1}A^\intercal p'$. One Leapfrog update of $(q_0', p_0') = (Aq_0, (A^\intercal)^{-1}p_0)$ is given by

$$p'_{1/2} = p'_0 - \frac{\varepsilon}{2}\frac{\partial U'}{\partial q}(q'_0) = (A^\intercal)^{-1}p_0 - \frac{\varepsilon}{2}(A^\intercal)^{-1}\frac{\partial U}{\partial q}(q_0) \tag{9.30}$$

$$q'_1 = q'_0 + \varepsilon A\Sigma^{-1}A^\intercal p'_{1/2} = Aq_0 - \frac{\varepsilon^2}{2}A\Sigma^{-1}\frac{\partial U}{\partial q}(q_0) + \varepsilon A\Sigma^{-1}p_0 \tag{9.31}$$

$$p'_1 = p'_{1/2} - \frac{\varepsilon}{2}\frac{\partial U'}{\partial q'}(q'_1) = (A^\intercal)^{-1}p_0 - \frac{\varepsilon}{2}(A^\intercal)^{-1}\frac{\partial U}{\partial q}(q_0) - \frac{\varepsilon}{2}(A^\intercal)^{-1}\frac{\partial U}{\partial q}(A^{-1}q'_1). \tag{9.32}$$

Multiplying (9.30) and (9.32) by $A^\intercal$ and multiplying (9.31) by A^{-1} gives the Leapfrog update for the original pair (q_0, p_0), and it is clear that $(q'_1, p'_1) = (Aq_1, (A^\intercal)^{-1}p_1)$. By induction, this relation must hold for any number of Leapfrog steps. □

Remark 9.3. In practice, the equivalence in Lemma 9.1 is not exact because of computational inaccuracies arising from the matrix operations performed with A. However, if A is well-conditioned, numeric implementations of the two chains should give very similar results.

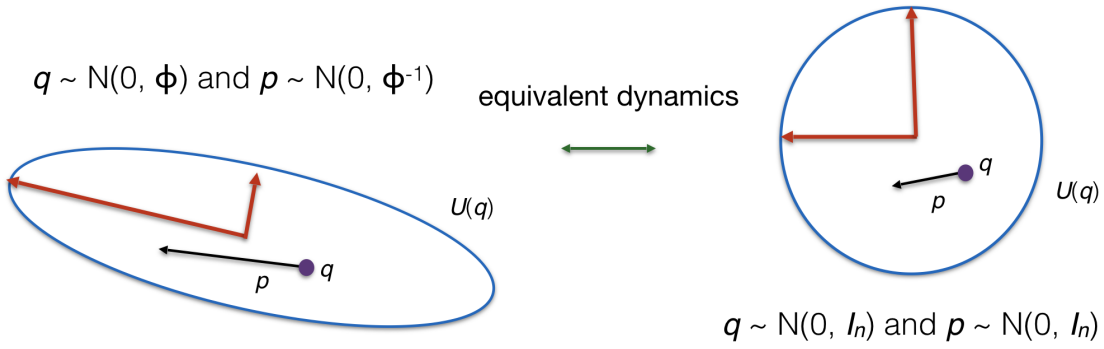


Fig. 9.7: Visualization of Lemma 9.1. When q has covariance Φ, the HMC dynamics with momentum $p \sim \mathrm{N}(0, \Phi^{-1})$ are equivalent to the ideal HMC dynamics where both q and p both have isotropic covariance. This shows that the ideal momentum covariance in HMC is the inverse of the local covariance of U at the current position q.

This lemma provides a key insight about tuning HMC. Suppose that the distribution of q is approximately Gaussian in some region around a point q^* with positive definite covariance Σ_{q^*}. The covariance can be decomposed as $\Sigma_{q^*} = AA^\intercal$ by several possible methods such as Choelsky or eigenvalue decomposition. Consider the HMC dynamics of the chain (q, p) with $p \sim \mathrm{N}(0, \Sigma_{q^*}^{-1})$. By Lemma 9.1, the dynamics of (q, p) are equivalent to the dynamics of $(A^{-1}q, A^\intercal p)$. Now, $A^\intercal p \sim \mathrm{N}(0, I_n)$, and $\mathrm{Var}(A^{-1}q) = I_n$ in a region near q^*, so the transformed position and momentum variables are approximately independent with variance 1 in each dimension. Since the transformed space is easy to sample, the HMC dynamics of $(A^{-1}q, A^\intercal p)$ should lead to nearly independent states in a small number number of Leapfrog steps (even $L = 1$), and the same sampling properties hold for the equivalent dynamics of the original (q, p).

This observation is one of several motivations for using local curvature information to improve the performance of HMC dynamics, as is done in RMHMC. Let q^* be a position in the energy landscape, and suppose $\frac{\partial U^2}{\partial^2 q}(q^*)$ is positive definite, so that $\Sigma_{q^*} = [\frac{\partial U^2}{\partial^2 q}(q^*)]^{-1}$ gives the local correlation and scaling structure of U in a neighborhood around q^*. In general, $\frac{\partial U^2}{\partial^2 q}(q^*)$ may not be positive definite, but a positive definite relative Σ'_{q^*} obtained by thresholding the eigenvalues of $\frac{\partial U^2}{\partial^2 q}(q^*)$ and inverting could provide the same benefits.

By the discussion above, using the momentum $p \sim \mathrm{N}(0, \Sigma_{q^*}^{-1})$ should lead to a nearly independent proposal in the q-space in a small number of Leapfrog steps, so $\Sigma_{q^*}^{-1}$ is the ideal proposal covariance at the point q^*. As discussed in Section 10, using $\Sigma_{q^*}^{-1}$ as the covariance for the momenta promotes movement along the local manifold and allows the chain to travel along level curves near the bottom of energy basins, which is impossible using gradient information alone. In order for HMC to be an effective *sampling* method, instead of just an optimization method, it is necessary to use an informative momentum covariance. If estimates s_i of the marginal standard deviation can be obtained, then a diagonal covariance Λ where $\lambda_i = 1/s_i^2$ can account for differences in scale between the variables. However, the s_i might vary throughout the state space, and a diagonal covariance Λ cannot account for correlations between the dimensions, which are usually strong in real world problems.

The following lemma gives three equivalent ways of implementing the HMC dynamics of a chain (q, Cp) for an invertible matrix C, and where the original momentum distribution is $p \sim \mathrm{N}(0, \Sigma)$. In RMHMC, $\Sigma = I_n$ and $C = \sqrt{\partial U^2/\partial^2 q}$, assuming that the curvature is positive definite. Although equivalent, the computational cost of these implementations can vary depending on the matrix decompositions and inversions required. When working in high dimensions with large matrices, the cost of matrix operations quickly balloons, and care is needed to make sure that the chain can be updated in a reasonable amount of time.

Lemma 9.2. *Let $U(q)$ be a smooth energy function, Σ a positive definite matrix, and C an invertible matrix. The dynamics of the following HMC chains are equivalent:*

1. *The momentum is sampled from $p \sim N(0, C^\intercal \Sigma C)$ and the chain is updated according to the standard HMC dynamics of (q, p).*
2. *The momentum is sampled from $p \sim N(0, \Sigma)$ and the chain is updated according to the standard HMC dynamics of $(q, C^\intercal p)$, i.e.*
3. *The momentum is sampled from $p \sim N(0, \Sigma)$ and the chain is updated according to the altered HMC dynamics defined by*

$$\frac{dq}{dt} = C^{-1} \frac{\partial K}{\partial p}, \tag{9.33}$$

$$\frac{dp}{dt} = -[C^{-1}]^\intercal \frac{\partial U}{\partial q}. \tag{9.34}$$

Moreover, (2) and (3) can both be implemented using the altered Leapfrog update

$$p_{t+1/2} = p_t - \frac{\varepsilon}{2}[C^{-1}]^\intercal \frac{\partial U}{\partial q}(q_t) \tag{9.35}$$

$$q_{t+1} = q_t + \varepsilon C^{-1} \Sigma^{-1} p_{t+1/2} \tag{9.36}$$

$$p_{t+1} = p_{t+1/2} - \frac{\varepsilon}{2}[C^{-1}]^\intercal \frac{\partial U}{\partial q}(q_{t+1}). \tag{9.37}$$

Proof. Consider the dynamics of (2). Sample $p_0 \sim \mathrm{N}(0, \Sigma)$, and let $p_0' = C^\intercal p_0$, which means p_0' is distributed $\mathrm{N}(0, C^\intercal \Sigma C)$. The Leapfrog updates of $(q, p') = (q, C^\intercal p)$ are given by

$$p'_{t+1/2} = p'_t - \frac{\varepsilon}{2} \frac{\partial U}{\partial q}(q_t) \tag{9.38}$$

$$q_{t+1} = q_t + \varepsilon C^{-1} \Sigma^{-1} [C^{-1}]^\intercal p'_{t+1/2} \tag{9.39}$$

$$p'_{t+1} = p'_{t+1/2} - \frac{\varepsilon}{2} \frac{\partial U}{\partial q}(q_{t+1}) \tag{9.40}$$

which are identical to the standard Leapfrog updates of (1), proving the equivalence between (1) and (2). On the other hand, multiplying (9.38) and (9.40) by $[C^{-1}]^\intercal$ gives the same updates as (9.35) through (9.37), because $p_t = [C^{-1}]^\intercal p'_t$ at each step t. The updates (9.35) through (9.37) are easily identified as the Leapfrog dynamics of the altered Hamiltonian Equations (9.33) and (9.34), showing the equivalence of (2) and (3). □

The above lemma is important for two reasons. First, it shows that HMC dynamics with $p \sim \mathrm{N}(0, M)$ with $M = C^\intercal C$ can be interpreted as the HMC dynamics resulting from a momentum $p \sim \mathrm{N}(0, I_n)$ with the altered form of Hamilton's

Equations in (9.33) and (9.34). This provides an important link between RMHMC and other "skew-symmetric" HMC methods that alter Hamilton's Equations in a similar way, most importantly stochastic gradient HMC.

Second, the lemma provides an alternative way to implement the dynamics of $p \sim \mathrm{N}(0, M)$ that only requires the calculation of $\sqrt{M^{-1}}$, not M itself. This follows from letting $\Sigma = I_n$, $C = \sqrt{M}$, and observing that the updates in (9.35) through (9.37) require only C^{-1}. The ideal momentum covariance is $\frac{\partial U^2}{\partial^2 q}$, and in convex regions $\sqrt{[\frac{\partial U^2}{\partial^2 q}]^{-1}}$ can be approximated from a sample of local positions using a variant of the LBFGS algorithm. The calculation requires no matrix inversion or decomposition, and provides a computationally efficient way to implement an approximate RMHMC algorithm, as is discussed later. Still, obtaining an accurate estimate of the root inverse Hessian in complex landscapes is a substantial obstacle for any RMHMC implementation.

9.5.2 RMHMC dynamics

Linear transformation that adapt to the local curvature have clear theoretical benefits, and allow for nearly independent sampling in only a few Leapfrog steps. However, HMC dynamics that include local curvature information are much more difficult to discretize than standard HMC dynamics, and costly computational methods are needed.

In standard HMC, the same matrix Σ is used as the covariance of the momenta throughout a single proposal. In RMHMC, there is a dependence of the momentum covariance $\Sigma(q)$ on the current position q in the energy landscape. For now, $\Sigma(q)$ is any smooth matrix function of q that returns a positive definite symmetric matrix, but in practice this matrix should reflect the local curvature near a position q in the energy landscape. A discussion of the choice of $\Sigma(q)$ in practice can be found later in this section. The RMHMC momentum distribution is $p \sim \mathrm{N}(0, \Sigma(q))$, with energy function

$$K(q,p) = \frac{1}{2}\log((2\pi)^n|\Sigma(q)|) + \frac{1}{2}p^\intercal\Sigma(q)^{-1}p, \tag{9.41}$$

and the joint Hamiltonian is

$$H(q,p) = U(q) + \frac{1}{2}\log((2\pi)^n|\Sigma(q)|) + \frac{1}{2}p^\intercal\Sigma(q)^{-1}p. \tag{9.42}$$

The momentum energy function must include an extra term $\frac{1}{2}\log((2\pi)^n|\Sigma(q)|)$ not found in standard HMC, and evaluating derivatives of this term is a source of computational difficulty. Observe that

$$\int_{\mathbb{R}^n}\frac{1}{Z}e^{-H(q,p)}dp = \frac{1}{Z}e^{-U(q)}\int_{\mathbb{R}^n}\frac{1}{\sqrt{(2\pi)^n|\Sigma(q)|}}e^{-p^\intercal\Sigma(q)^{-1}p}dp = \frac{1}{Z}e^{-U(q)}, \tag{9.43}$$

so the marginal distribution of q is the target distribution, and the q-sample obtained from updating (q,p) jointly will follow the correct distribution, just as in standard HMC. The Hamiltonian Equations governing RMHMC are

$$\frac{dq}{dt} = \frac{\partial H}{\partial p} = \Sigma(q)^{-1}p, \tag{9.44}$$

$$\frac{dp}{dt} = -\frac{\partial H}{\partial q} = -\frac{\partial U}{\partial q} - \frac{1}{2}\mathrm{Tr}\left[\Sigma(q)^{-1}\frac{\partial\Sigma(q)}{\partial q}\right] + \frac{1}{2}p^\intercal\Sigma(q)^{-1}\frac{\partial\Sigma(q)}{\partial q}\Sigma(q)^{-1}p. \tag{9.45}$$

The updates of q and p and no longer separable, since the update of p depends on both q and the current p. Therefore, if the Leapfrog integrator were used, the coordinate changes would no longer have a shear structure, so there is no guarantee that the Leapfrog coordinate change has determinant 1 for RMHMC dynamics. This upsets the detailed balance of HMC, and naively implementing the Leapfrog integrator in an RMHMC setting does not preserve the distribution of $\frac{1}{Z}e^{-U(q)}$.

To overcome the non-separability of the RMHMC update equations, the update values are defined by an implicit set of equations that must be solved using fixed point iteration. One iteration of the Generalized Leapfrog Integrator for discretizing the dynamics of a non-separable joint Hamiltonian H is given by

$$p_{t+1/2} = p_t - \frac{\varepsilon}{2}\frac{\partial H}{\partial q}(q_t, p_{t+1/2}), \tag{9.46}$$

$$q_{t+1/2} = q_t + \frac{\varepsilon}{2}\left[\frac{\partial H}{\partial p}(q_t, p_{t+1/2}) + \frac{\partial H}{\partial p}(q_{t+1}, p_{t+1/2})\right], \tag{9.47}$$

$$p_{t+1} = p_{t+1/2} - \frac{\varepsilon}{2}\frac{\partial H}{\partial q}(q_t, p_{t+1/2}). \tag{9.48}$$

The update in the first two steps is implicitly defined, allowing for the simulation of the dynamics of a non-separable H. In the case of standard HMC, $H(q,p) = U(q) + K(p)$ and the Generalized Leapfrog updates are identical to the standard Leapfrog scheme. When H is non-separable, fixed point iteration must be used to solve (9.46) and (9.47). The details of the fixed point updates are included with the RMHMC Algorithm later in the section.

It can be shown that the Generalized Leapfrog updates are volume-preserving, and that RMHMC maintains detailed balance and preserves the target distribution. The proof is similar to the proof of detailed balance for standard HMC, with appropriate adjustments made for the proofs of volume preservation and reversibility based on the Generalized Leapfrog updates.

9.5.3 RMHMC algorithm and variants

There are several variants of the RMHMC algorithm. First the full RMHMC algorithm is presented, which requires fixed point iteration and the calculation of the derivative of $\Sigma(q)$. Since $\Sigma(q)$ in practice is the local curvature, full RMHMC requires calculation of the *third* derivatives of the target energy U, which is a large computational burden and is impossible in many practical situations. There is an RMHMC variant for $L = 1$ Leapfrog updates which requires the calculations of third derivatives of U at the original and proposed state but involves no fixed point iteration. The details of this algorithm are slightly different than full RMHMC, and the interested reader should refer to [3].

Algorithm 11 Full RMHMC Algorithm

Input: Differentiable energy function $U(q)$, initial state $q_0 \in \mathbb{R}^n$, $n \times n$ differentiable positive definite covariance function $\Sigma(q)$, step size ε, number of iterations N, number of Leapfrog steps L, number of fixed point steps K
Output: Markov Chain sample $\{q_1, \ldots, q_N\}$ with stationary distribution U
for $i = 1 : N$ **do**

1. Generate momentum $p_{i-1} \sim \mathrm{N}(0, \Sigma(q_{i-1}))$.
2. Let $(q'_0, p'_0) = (q_{i-1}, p_{i-1})$. For $l = 1 : L$, update (q'_{l-1}, p'_{l-1}) using the Generalized Leapfrog integrator to reach a proposal state $(q^*, p^*) = (q'_L, p'_L)$ as follows:

   a. Let $\hat{p}_0 = p'_{l-1}$. For $k = 1 : K$, update $\hat{p}_{k-1}$ according to the fixed point equation

   $$\hat{p}_k = \hat{p}_{k-1} - \frac{\varepsilon}{2} \Sigma(q'_{l-1})^{-1} \hat{p}_{k-1} \tag{9.49}$$

   to obtain the half-step momentum update $p'_{l-1/2} = \hat{p}_K$.

   b. Let $\hat{q}_0 = q'_{l-1}$. For $k = 1 : K$, update $\hat{q}_{k-1}$ according to the fixed point equation

   $$\hat{p}_k = \hat{p}_{k-1} - \frac{\varepsilon}{2} \Sigma(q'_{l-1})^{-1} \hat{p}_{k-1} \tag{9.50}$$

   where $\partial H / \partial p$ is given in (9.45), to obtain the full step position update $q'_l = \hat{q}_K$.

   c. Update $p'_{l-1/2}$ according to the explicit equation

   $$p'_l = p'_{l-1/2} - \frac{\varepsilon}{2} \Sigma(q'_l)^{-1} p'_{l-1/2} \tag{9.51}$$

   to obtain the full step momentum update p'_l.

3. Accept the proposed state (q^*, p^*) according the Metropolis-Hastings acceptance probability

   $$\alpha = \min\left(1, \exp\{-H(q^*, p^*) + H(q_{i-1}, p_{i-1})\}\right) \tag{9.52}$$

   where $H(q, p)$ is the joint Hamiltonian of (9.42). If the proposal is accepted, then $q_i = q^*$. Otherwise, $q_i = q_{i-1}$. The momentum p_{i-1} can be discarded after the proposal.

end for

Remark 9.4. The step size ε is a "dimensionless" quantity, because the RMHMC dynamics should locally correspond to the trivial HMC distribution where both q and p are $\mathrm{N}(0, I_n)$. The scale of RMHMC is implicitly the standard normal scale, and therefore setting ε to a value slightly less than 1, the smallest (and largest) rescaled standard deviation, should yield good results for any RMHMC algorithm.

To alleviate the difficulties of full RMHMC, one can use an approximate RMHMC algorithm where the covariance $\Sigma(q_{t-J+1}, q_{t-J+2}, \ldots, q_t)$ can depend on the previous states of the chain before the momentum is sampled, but is fixed throughout the Leapfrog updates. This variant is essentially the standard HMC algorithm with a principled way of changing Σ between proposals.

Changing Σ between proposals does not violate the preservation of the target distribution, because if q has the correct distribution after an update with covariance Σ_0, q will still follow the correct distribution after an HMC update with any covariance Σ_1, not necessarily equal to Σ_0. At first it might appear that using the previous states to obtain $\Sigma(q_{t-J+1}, q_{t-J+2}, \ldots, q_t)$ might violate the Markovian structure of HMC, but this is not the case, because detailed balance holds for any $\Sigma(x_1, \ldots, x_J)$, and in particular it is not required that $(x_1, \ldots, x_J)$ have the target distribution. Therefore there is no *distributional* dependence except of the current state q_t, and the same proofs of detailed balance still hold.

Although the simplified RMHMC algorithm cannot capture the full dependence implied by RMHMC dynamics, since $\Sigma(q)$ is not updated through the Leapfrog iterations, it is computationally identical to standard HMC and provides an efficient way to incorporate curvature information by using a quasi-Newton estimate of the inverse Hessian. Even if this information is approximate, it can still substantially improve the movement of the chain in the energy landscape.

Algorithm 12 Simplified RMHMC Algorithm

Input: Differentiable energy function $U(q)$, initial state $q_0 \in \mathbb{R}^n$, $n \times n$ positive definite covariance function $\Sigma(q_1, \ldots, q_J)$, step size ε, number of iterations N, number of previous states in memory J
Output: Markov Chain sample $\{q_1, \ldots, q_N\}$ with stationary distribution U
for $i = 1 : N$ **do**
1. Calculate the current covariance matrix $\Sigma^* = \Sigma(q_{i-1-J}, q_{i-J}, \ldots, q_{i-1})$.
2. Update (q_{i-1}, p_{i-1}) according to standard HMC dynamics using the proposal covariance Σ^*.

end for

Remark 9.5. Since Σ^* is fixed throughout updates, only a small number of Leapfrog steps (usually $L = 1$) are used in the simplified RMHMC algorithm, because the local curvature varies with each update.

9.5.4 Covariance functions in RMHMC

The RMHMC algorithm and its simplification preserve the target for any differentiable positive definite covariance function $\Sigma(q)$, but to actually improve sampling $\Sigma(q)$ must reflect the local curvature of the space. In general, $\partial^2 U / \partial q^2$ is not necessarily positive definite, so the naive choice $\Sigma(q) = \frac{\partial U^2}{\partial q^2}(q)$ is not always possible in practice.

The original authors of RMHMC restrict their attention to sampling from a posterior probability $p(\theta|X)$ for a family of probability models $p(X|\theta)$. In this case, there is a natural choice for the proposal covariance given by the Fisher Information

$$\Sigma(\theta) = -\mathrm{E}_{X|\theta}\left[\frac{\partial^2}{\partial \theta^2} \log p(X|\theta)\right] \tag{9.53}$$

which is guaranteed to be positive definite. In simple cases the Fisher Information can be obtained analytically. If this is not possible, it can be estimated by taking the expectation of the curvature over the observed data and thresholding the eigenvalues of the resulting matrix for stability. The positive definite structure of the Fisher Information is a nice property in theory but in practice the matrix must be estimated and very small or even negative eigenvalues can still be encountered. When the Fisher Information is used, the gradient term $\Sigma(\theta)^{-1} \frac{\partial U}{\partial \theta}(\theta)$ that appears in the Langevin equation corresponds to the natural gradient of Amari et al [1].

The Fisher Information is not a complete solution because it can only be used when sampling a parameter θ for a family of distributions with a set of observed data X. When sampling from just a probability distribution $P(q) = \frac{1}{Z} e^{-U(q)}$, there is no way to make the curvature positive definite by taking an expectation. However, the largest eigenvalues of $\frac{\partial U}{\partial q}$ should be the most important for the dynamics of HMC. This is because of largest eigenvalues of the curvature represent the most constrained linear dimensions of the distribution. When near a local minima, negative eigenvalues or eigenvalues near 0 are not problematic, because movement in these directions leave H approximately constant or decrease H. Thresholding the eigenvalues of $\frac{\partial U}{\partial q}$ can give a positive definite covariance that preserves the most important local geometric information when the curvature itself is not positive definite.

Another option is to estimate the local curvature $\Sigma(q)$ as is done in quasi-Newton methods. This type of method of method uses a sequence of past states $q_{t+1-J}, q_{t+2-J}, \ldots, q_t$ to estimate the inverse Hessian at the current state q_t. As demonstrated in Lemma 9.2, only the root inverse Hessian is needed to simulate the HMC dynamics of $\Sigma(q) = \frac{\partial U}{\partial q}$, and there is a variant of the LBFGS algorithm that estimates the root inverse Hessian directly. See [2] and [9] for details.

9.6 HMC in Practice

In this section, two applications of HMC and related algorithms are presented. The first application is a toy experiment with Gaussian distributions that are highly constrained in all but a few directions. This experiment is very useful for understanding the basic principles behind tuning the HMC parameters. Next, posterior sampling with HMC and variants in a logistic regression model is examined. This setting is one of the few cases where the full RMHMC algorithm can be implemented in practice, because the Fisher Information is available in closed form. Logistic regression is a good setting for direct comparison between the different HMC models. Finally, the Alternating Backwards Propagation algorithm is presented, which uses LMC as a key step when sampling from distributions defined over high-dimensional parameters and images.

9.6.1 Simulated experiments on constrained normal distributions

In this section, HMC and variants are used to sample from normal distributions that are highly constrained in all but a few directions. Such distributions are challenging for any MCMC method that uses local updates based on the current position in the energy landscape, because it is difficult to efficiently sample the unconstrained dimensions while still remaining in the tightly constrained region of the landscape.

Two different distributions are considered: $\mathrm{N}(0,\Sigma_1)$ and $\mathrm{N}(0,\Sigma_2)$. Both Σ_1 and Σ_2 are 100×100 diagonal matrices. The first 15 entries of Σ_1 and Σ_2 are 1, representing the unconstrained sampling direction. The last 85 entries of Σ_1 are 0.01^2, and the last 85 entries of Σ_2 are 0.0001^2. Σ_1 is an easier sampling case, because the ratio between the largest and smallest standard deviations is around 100, so about 100 Leapfrog steps should be needed for effective sampling with standard HMC. Σ_2 is a more difficult scenario, because the ratio between the largest and smallest standard deviations is 10,000. When the local covariance exhibits such an extreme difference in scales, HMC is no longer an effective sampling method because the Leapfrog approximation becomes very unstable when very large number of steps are used, as can be seen from the Australian Credit Data in Section 9.6.2. This can be addressed by including approximate curvature information with a quasi-Newton HMC variant. All experiments use 5000 burn-in iterations and 10,000 sampling iterations.

The energy functions of the two target distributions have the form $U_1(q) = \frac{1}{2}q^\intercal\Sigma_1^{-1}p$ and $U_2 = \frac{1}{2}q^\intercal\Sigma_2^{-1}q$. Like all normal distributions, the target distributions have constant curvature Σ_1^{-1} and Σ_2^{-1} respectively. The full RMHMC algorithm can be implemented by simply letting $p\sim\mathrm{N}(0,\Sigma_1^{-1})$ or $p\sim\mathrm{N}(0,\Sigma_2^{-1})$ for each update, and the standard dynamics can be used because the curvature $\Sigma(q)$ is constant throughout the state space, so the derivative of $\Sigma(q)$ is 0 and the Generalized Leapfrog update becomes identical to the standard Leapfrog update. In this experiment, the RMHMC algorithm with only one Leapfrog update can obtain virtually independent samples from the target distribution in every iteration.

Consider the target distribution $\mathrm{N}(0,\Sigma_1)$. Three different methods are used to sample this distribution: Random-Walk Metropolis Hastings, LMC, and HMC with $L=150$ Leapfrog updates. The momentum covariance was set to I_{100} for both HMC and LMC. A step size of 0.04 was used for RW Metropolis, and a step size of 0.008 was used for LMC and HMC, since 0.008 is slightly less than the smallest marginal standard deviation of 0.01. The first two methods cannot effectively sample the target distribution, because both methods will be forced to explore the unconstrained directions of the distribution with a random walk with a small step size. LMC has difficulty in this situation because the momenta are refreshed after a single update, whereas HMC uses the same momentum for a large number of updates, so HMC does *not* explore the distribution via random-walk like the other two methods. Since $\varepsilon L = 1.5$ for HMC, and the largest marginal standard deviation of the target distribution is 1, HMC obtains nearly independent samples in every iteration. For fair comparison between the methods, 150 RW Metropolis and LMC updates are counted as a single iteration in the figures below.

Next, consider the target distribution $\mathrm{N}(0,\Sigma_2)$. The ratio between the largest and smallest standard deviations for this distribution is 10,000, so about 10,000 Leapfrog steps would be needed in standard HMC with an identity covariance to obtain independent sample with each HMC update. Even in fairly regular landscapes such as the logistic regression landscapes from Section 9.6.2, the accuracy of the Leapfrog approximation degrades after more than a few hundred Leapfrog steps, and in practical problems, it is simple not possible to compensate for the differences in scale in the target distribution by using $L = 10,000$ Leapfrog updates with standard HMC. To sample effectively, it is necessary to take second order information into account.

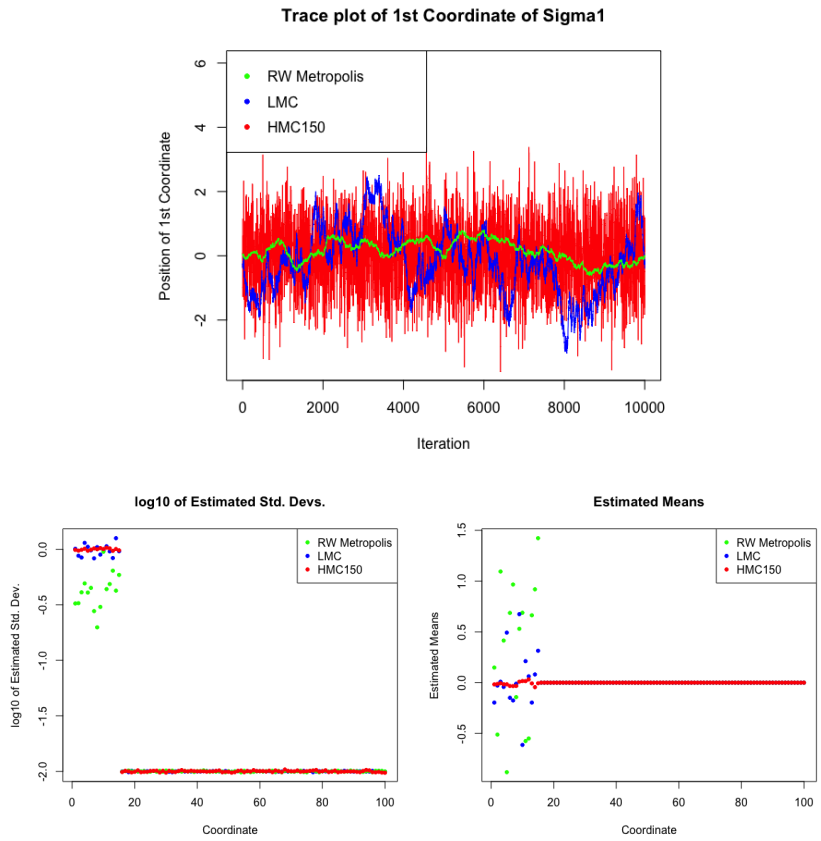


Fig. 9.8: Simulation study using Σ_1. The performance of the samplers is similar across the constrained dimensions, but RW Metropolis and LMC have difficulty effectively sampling the unconstrained dimensions, even though 150 iterations of each of these methods was counted as a single update in the figures. On the other hand, the capability of HMC to move across the sample space by reusing the same momentum for a large number of leapfrog iterations allows for very effective sampling.

Assume that the true position covariance Σ_2 is unknown and impossible to calculate directly, which is true in most practical situations. In this case, it is still possible to estimate Σ_2 from the previously sampled positions, and this approximate information can still facilitate fairly effective sampling. First, 40 positions were sampled using standard HMC with $\varepsilon = 0.000075$ and momentum covariance I_{100}. After obtaining some initial points, the simplified RMHMC algorithm was implemented using a LBFGS estimate of Σ_2. The LBFGS recursion was started from an initial matrix $H_0 = \gamma I_{100}$ using the past 40 sampled positions.

Unfortunately, the raw LBFGS estimate cannot improve sampling significantly, since the true matrix Σ_2 is simply too large to be accurately estimated using only 40 points. However, after decomposition and adjustment, a useful relative can be obtained. When looking at the eigenvalues from the LBFGS estimate, one observes that the estimate for the smallest eigenvalue is very close to the true value of 0.0001^2, and that the LBFGS estimate can identify several unconstrained directions with large eigenvalues. The estimates for the largest eigenvalue tended to be very inaccurate and to depend heavily on the value of γ selected. Between the largest and smallest eigenvalues, the majority of the remaining eigenvalues are unchanged by LBFGS and remain γ, which is unsurprising because a very small amount of data is being used to estimate a very large matrix.

Although the true covariance Σ_2 is unknown, in some situations it is reasonable to assume that only the first few largest eigenvalues of Σ_2 matter, and that most of the other eigenvalues are close to 0. This situation occurs when sampling a low-dimensional manifold in a high-dimensional space, which is common when sampling distributions defined over images or other complex data structures. The largest eigenvalues correspond to the relatively small number of un-

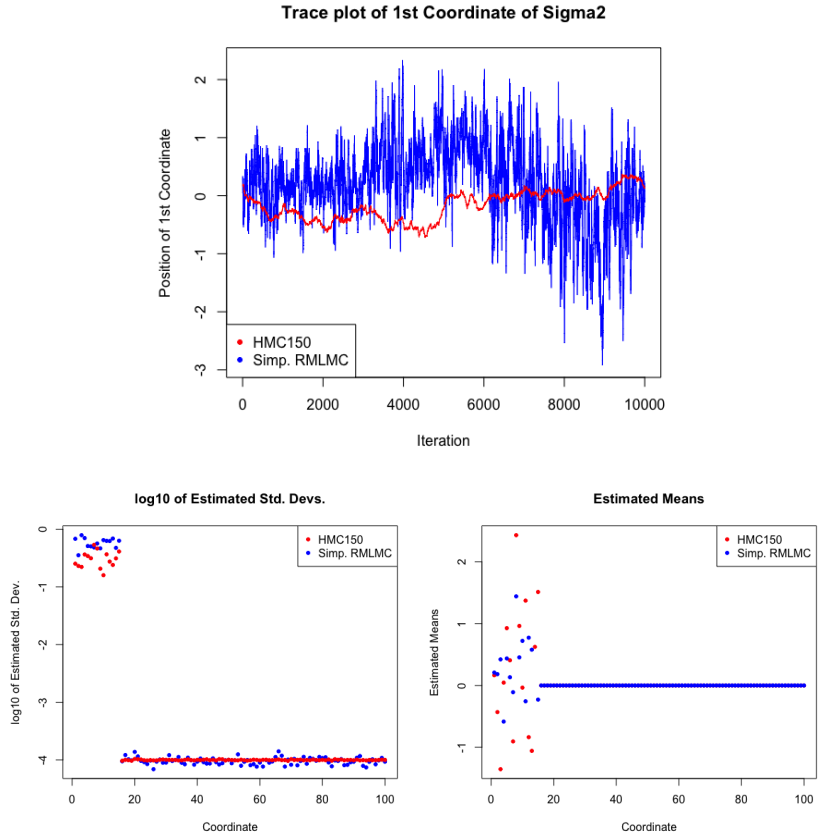


Fig. 9.9: Simulation study using Σ_2. HMC with $L = 150$ Leapfrog updates can no longer effectively sample in the unconstrained dimensions. However, using the simplified RMHMC algorithm with $L = 1$ and a covariance calculated as described above can sample much more effectively.

constrained dimensions in the target distribution. Given some knowledge about the local structure of the eigenvalues, the raw LBFGS estimate can be adjusted to provide a more useful covariance.

Let H^* be the raw LBFGS estimate obtained from the past $J = 40$ HMC samples. Let $U\Lambda U^\intercal$ be the symmetric eigenvalue decomposition of H^*, where Λ is a diagonal matrix with eigenvalues $\lambda_1, ..., \lambda_{100}$ sorted in decreasing order. Let $\lambda_i^* = \lambda_{100}$ for $i = K+1, \ldots, 100$ for some parameter K, which is the estimated number of unconstrained directions in the target distribution. The true value of K is 15, but in the experiment the conservative estimate $K = 10$ is used. The first K of the λ_i^* are equal to the original values. The momentum covariance is then given by $\Sigma^* = U\Lambda^* U^\intercal$. In theory, ε should be set to slightly less than 1 for any RMHMC method. However, due to the inaccuracy of the estimate of the largest standard deviation, which tends to be too large, ε should be set so that $\varepsilon\lambda_1 \approx 1$. The value of γ did not have too much effect on sampling, and values ranging from 0.000001 to 1 gave about the same results, as long as ε was set accordingly. Only $L = 1$ Leapfrog step was needed to achieve good results. The third method of Lemma 9.2 with $\sqrt{C}^{-1} = U(\Lambda^*)^{1/2}U^\intercal$ was used during implementation.

Two methods are presented in Figure 9.9 for sampling Σ_2: HMC with $L = 150$, and the simplified RMHMC algorithm described above. The simplified RMHMC algorithm using quasi-Newton information outperforms standard HMC with only $L = 1$ Leapfrog step. The eigenvalue decomposition required at each iteration of the simplified RMHMC algorithm is computationally expensive, but it is necessary for good results, since the LBFGS estimate simply cannot estimate all of the true eigenvalues of Σ_2 with such a limited amount of data, so some adjustment of the eigenvalues is needed.

9.6.2 Sampling logistic regression coefficients with RMHMC

Sampling coefficients from distributions defined by logistic regression is an ideal scenario for applying the full RMHMC method, because the Fisher Information at any point in the state space can be given in closed form. In many practical situations (for example, L_1-regularized regression) this is not the case. Given an $N \times P$ matrix of observations X (each row gives a single case) and a binary 0 or 1 response Y and the regularization coefficient λ (treated as a given constant), the energy of a P-length vector of coefficients β is given by

$$U(\beta) = -\log[L(X,Y|\beta,\lambda)p(\beta|\lambda)] = -\beta^\intercal X^\intercal Y + \sum_{j=1}^{N} \log(1+e^{\beta^\intercal X_n^\intercal}) + \frac{\lambda}{2}\beta^\intercal\beta \tag{9.54}$$

where X_n is row n of the matrix X. The derivative of the energy function is

$$\frac{dU}{d\beta}(\beta) = -X^\intercal Y + X^\intercal S + \lambda\beta \tag{9.55}$$

where S is a length P vector with $S_n = \sigma(\beta^\intercal X_n^\intercal)$ where $\sigma(*)$ is the sigmoid function, and has Fisher Information

$$I(\beta) = \mathrm{E}_{Y|X,\beta,\lambda}\left[\frac{d^2U}{d\beta^2}(\beta)\right] = X^\intercal\Lambda X + \lambda I \tag{9.56}$$

where Λ is an $N \times N$ diagonal matrix with elements $\Lambda_{n,n} = \sigma(\beta^\intercal X_n^\intercal)(1-\sigma(\beta^\intercal X_n^\intercal))$. Full RMHMC also requires the derivatives of $I(\beta)$, which are given by

$$\frac{dI(\beta)}{d\beta_i} = X^\intercal\Lambda V_i X \tag{9.57}$$

where V_i is diagonal matrix with elements $V_{i,(n,n)} = (1-2\sigma(\beta^\intercal X_n^\intercal))X_{n,i}$.

Presented below are the results of a study by Giorlami and Calderhead in [3], comparing the performance of RMHMC, traditional HMC, and other common methods when sampling regression coefficients. The authors used 6 different datasets with a binary response and features matrices of various sizes, and the results for 4 datasets are presented here. We give the results for 6 of the sampling algorithms studied by the authors: component-wise Metropolis-Hastings, LMC, HMC, RMHMC, RMLMC, and simplified RMLMC.

For LMC and HMC samplers, the momentum covariance $\Sigma = I_n$ was used. For all samplers, the step size ε was set so that acceptance rates were around 70%. The step size L for RMHMC and HMC was set so that $\varepsilon L \approx 3$, slightly larger than the largest marginal standard deviation, so that approximately independent samples should be obtained with each HMC iteration. The Hamiltonian dynamics defined by logistic regression are relatively well-behaved so L can be set to be quite large (several hundred) without a significant drop in acceptance rates. The number of leapfrog steps L is usually relatively small for RMHMC because nearly independent points can be obtained with each leapfrog update. For HMC, a large number of leapfrog updates is needed to compensate for the difference in scale between the smallest and largest marginal standard deviations. Simplified RMLMC corresponds to the simplified RMHMC algorithm where $\Sigma(\beta_t) = I(\beta_t)$, the Fisher information at the current point ($J = 1$) and $L = 1$ Leapfrog update. RMLMC is a slight variant of the RMHMC algorithm with $L = 1$. See [3] for details.

All regression models included an intercept, so the number of coefficients P is one more than the number of columns of the data matrix. Each sampling run consisted of 5000 burn-in iterations and 5000 sampling iterations, and 10 trials were run for each sampling method.

A variety of useful instructive observations can be drawn from the results of these experiment. Following the original authors, the speed and relative speed of the algorithms are given based on the *minimum* ESS out of the 10 trials.

First, consider the performance of HMC and LMC across the 4 datasets, shown in Table 9.1. As mentioned earlier, the number of leapfrog steps L in HMC was tuned so that εL was greater than the largest observed standard deviation in the dataset, so HMC should in theory provide nearly independent samples, provided that the dynamics can be simulated accurately. Although slow, HMC does manage to achieve an ESS that is a significant proportion of the ideal

Table 9.1: Evaluation of six sampling algorithms on four datasets.

Pima Indian Dataset, $N = 532$, $P = 8$				
Method	Time (sec)	ESS(Min,Med,Max)	s/ESS(Min)	Rel. Speed
Metropolis	4.1	(14, 37, 201)	0.29	$\times$ 1.9
LMC	1.63	(3, 10, 39)	0.54	$\times$ 1
HMC	1499.1	(3149,3657,3941)	0.48	$\times$ 1.1
RMLMC	4.4	(1124,1266,1409)	0.0039	$\times$ 138
Simp. RMLMC	1.9	(1022,1185,1312)	0.0019	$\times$ 284
RMHMC	50.9	(5000,5000,5000)	0.01	$\times$ 54
Australian Credit Dataset, $N = 690$, $P = 14$				
Method	Time (sec)	ESS(Min,Med,Max)	s/ESS(Min)	Rel. Speed
Metropolis	9.1	(15, 208, 691)	0.61	$\times$ 1
LMC	No Conv.	-	-	-
HMC	No Conv.	-	-	-
RMLMC	11.8	(730, 872, 1033)	0.0162	$\times$ 37
Simp. RMLMC	2.6	(459, 598, 726)	0.0057	$\times$ 107
RMHMC	145.8	(4940,5000,5000)	0.023	$\times$ 26
German Credit Dataset, $N = 1000$, $P = 24$				
Method	Time (sec)	ESS(Min,Med,Max)	s/ESS(Min)	Rel. Speed
Metropolis	20.9	(10, 82, 601)	2.09	$\times$ 1
LMC	2.7	(3, 5, 130)	0.9	$\times$ 2.6
HMC	3161.6	(2707, 4201, 5000)	1.17	$\times$ 2
RMLMC	36.2	(616, 769, 911)	0.059	$\times$ 39.6
Simp. RMLMC	4.1	(463, 611, 740)	0.0009	$\times$ 260
RMHMC	287.9	(4791, 5000,5000)	0.06	$\times$ 39
Caravan Dataset, $N = 5822$, $P = 86$				
Method	Time (sec)	ESS(Min,Med,Max)	s/ESS(Min)	Rel. Speed
Metropolis	388.7	(3.8, 23.9, 804)	101.9	$\times$ 3.7
LMC	17.4	(2.8, 5.3, 17.2)	6.2	$\times$ 59
HMC	12,519	(33.8, 4032, 5000)	369.7	$\times$ 1
RMLMC	305.3	(7.5, 21.1, 50.7)	305.3	$\times$ 1.2
Simp. RMLMC	48.9	(7.5, 18.4, 44)	6.5	$\times$ 56
RMHMC	45,760	(877, 1554, 2053)	52.1	$\times$ 7.1

Table 9.2: Ratio of Largest to Smallest Marginal Standard Deviations for the four datasets.

Dataset	Pima	Australian	German	Caravan
Ratio	225	6404	303	236

ESS of 5000 in the Pima, German, and Caravan datasets. However, the authors found that neither HMC nor LMC converged to the stationary distribution in the Australian dataset.

This can be understood by referring to the table giving the ratios of largest to smallest marginal standard deviations (Table 9.2). Recall that the ratio between the largest and smallest marginal standard deviations is the minimum number of Leapfrog steps that would be needed to reach an independent state in a single HMC update. In the Pima, German, and Caravan datasets, this ratio was about 200 to 300, meaning that between 200 to 300 leapfrog steps would be needed to reach an independent state using a trivial covariance matrix $\Sigma = I_n$. However, the Australian dataset has a ratio of over 6000 between the length of its largest and smallest constrained directions, so several thousand leapfrog steps are needed for each HMC update to reach an independent state. The Leapfrog discretization was not accurate enough to provide high acceptance rates for such large L, and the updates from the LMC sampler were too small to allow for effective sampling. The logistic regression landscape is relatively well-behaved; in more complex landscapes, it might not be possible to use even 200 to 300 Leapfrog steps.

It is interesting to note that in general, LMC performs significantly worse than all of the other methods, *except* in the high-dimensional Caravan dataset, where it outperforms all other methods. LMC has been found to be an effective

method for sampling very high-dimensional generative models of image patterns. LMC is an attractive choice in such situations because it scales well and it is quick to implement.

Now, consider the RMHMC methods. RMHMC and RMLMC have similar performance across the datasets, indicating that after local curvature is taken into account, the extra sampling power of RMHMC is evenly balanced by the faster speed of RMLMC. Simplified RMLMC outperforms the full RMHMC implementations across all datasets, providing evidence that simply taking into account local curvature information with no change in the HMC dynamics can be an effective way to implement a RMHMC approximation.

For the smaller datasets, the full RMHMC methods outperform standard HMC methods, but the reverse was true for the Caravan dataset. Calculating the derivative of the Fisher Information in full RMHMC requires evaluating N^2 expressions along the diagonal in 9.56, which explains the poor scaling. In special cases, such as sampling from Gaussian processes, the Fisher Information can have a sparse structure, but it general it is unavailable or very expensive to compute, and full RMHMC is not a practical solution to sampling in complex high-dimensional landscapes.

9.6.3 Sampling image densities with LMC: FRAME, GRADE and DeepFRAME

Stochastic models are a powerful and unified way of representing and understanding image data, and the capabilities of stochastic image models have greatly increased in recent years. By learning a density $P(I)$ from a set of training images $\{I_k\}_{k=1}^K$, one can synthesize novel realistic images and even explore the structure of the image space. However, sampling from an image density $P(I)$ is often problematic. Even relatively small images are high-dimensional data, because image dimension scales with the *square* of image width. Moreover, strong correlations exist between image dimensions, especially among nearby pixels. Methods such as Metropolis-Hastings and Gibbs sampling are too slow for practical use in this situation. LMC is a popular way of overcoming these difficulties.

Given an image density of the form

$$P(I) = \frac{1}{Z}\exp\{-U(I)\},$$

where I is an $n \times n$ image with continuous pixel intensity (typically each pixel intensity lies within a bounded interval such as [0, 1], [-1 -1], or [0, 255]), LMC can be used to obtain MCMC samples from the density P. The image updates have the standard Langevin form

$$I_{t+1} = I_t - \frac{\varepsilon^2}{2}\frac{dU}{dI}(I_t) + \varepsilon Z_t \tag{9.58}$$

where $Z_t \sim \mathrm{N}(0, \sigma^2 I_{n\times n})$. The gradient term $\frac{dU}{dI}(I_t)$ promotes faster convergence than random-walk Metropolis-Hastings or Gibbs sampling. LMC is often used instead of HMC with $L > 1$ Leapfrog steps due to the instability of the Leapfrog integrator when applied to a complex energy U.

LMC provides a powerful method for sampling from a learned energy, but the form of the energy $U(I)$ itself is a central issue of image modeling. The function U must be expressive enough to capture both the common structure and idiosyncratic variation that exists in real-world image patterns. In particular, U must capture the complex structure of the correlation between pixel intensities found in observed images to successfully synthesize realistic images. This section introduces two central formulations of image models. Methods for learning a parametric image density $P(I;\theta)$ are discussed in Chapter 10.

Example 9.1. **FRAME (Filters, Random Fields, and Maximum Entropy) Model. (Zhu et al., 1998 [12])** The FRAME model is a pioneering formulation of a family of image densities capable of synthesizing realistic textures. The density of a FRAME model has the Gibbs form

$$P(I;\mathscr{F},\Lambda) = \frac{1}{Z(\mathscr{F},\Lambda)}\exp\{-U(I;\mathscr{F},\Lambda)\}. \tag{9.59}$$

for a set of hyperparameters $(\mathscr{F},\Lambda)$. The hyperparameter $\mathscr{F}$ is a set of pre-defined filters $\mathscr{F} = \{F^{(1)},\ldots,F^{(K)}\}$, which are typically Gabor filters and Laplacian of Gaussian filters of different sizes/orientations, that measure convolutional response at each spatial location in I. The hyperparameter Λ is a set of potential functions $\Lambda = \{\lambda^{(1)}(\cdot),\ldots,\lambda^{(K)}(\cdot)\}$ where each $\lambda^{(i)}(\cdot)$ encourages the response of filter $F^{(i)}$ for synthesized images to match the observed response of

$F^{(i)}$ for training images. Typically, the filter set $\mathscr{F}$ is pre-selected and the potentials Λ are learned through Stochastic Gradient methods discussed in Chapter 10.

A FRAME energy has the form

$$U(I;\mathscr{F},\Lambda) = \sum_{i=1}^{K} \sum_{(x,y)\in\mathscr{L}} \lambda^{(i)}(F^{(i)} * I_{(x,y)}) \tag{9.60}$$

where $\mathscr{L}$ is the pixel lattice. The notation $F^{(i)} * I_{(x,y)}$ refers to the convolution of filter $F^{(i)}$ and image I centered at position (x,y) in the image lattice. To simplify the learning process, the responses $F^{(i)} * I_{(x,y)}$ for each $(x,y) \in \mathscr{L}$ are placed into discrete bins $\{B_1^{(i)}, \ldots, B_L^{(i)}\}$ and the potential functions $\lambda^{(i)}(\cdot)$ are replaced by potential vectors $\lambda^{(i)} = (\lambda_1^{(i)}, \ldots, \lambda_L^{(i)})$. This leads to the more tractable FRAME energy

$$U(I;\mathscr{F},\Lambda) = \sum_{i=1}^{K} \langle \lambda^{(i)}, H^{(i)}(I) \rangle = \langle \Lambda, H(I) \rangle \tag{9.61}$$

where $H^{(i)}(I) = (H_1^i(I), \ldots, H_L^{(i)}(I))$ is the frequency histogram of responses of image I to filter $F^{(i)}$ across all spatial locations, and $H(I) = (H^{(1)}(I), \ldots, H^{(K)}(I))$. In the rest of this section, we discuss the FRAME energy of the form (9.60), and the same properties hold for (9.61).

Inspection of the learned potentials $\{\lambda^{(i)}(\cdot)\}$ in a FRAME energy $U(I;\mathscr{F},\Lambda)$ trained to model real-world textures reveals two competing forces at work in the gradient term $\frac{\partial U}{\partial I}$ of the Langevin Equation (9.58). The learned potentials $\lambda^{(i)}$ fall into two generic families of potentials:

$$\phi(\xi) = a\left(1 - \frac{1}{1+(|\xi-\xi_0|/b)^\gamma}\right) \quad \text{with } a > 0, \tag{9.62}$$

or

$$\psi(\xi) = a\left(1 - \frac{1}{1+(|\xi-\xi_0|/b)^\gamma}\right) \quad \text{with } a < 0, \tag{9.63}$$

where ξ_0 is a translation constant, b is a scaling constant, and $|a|$ is the weight of the contribution from filter $F^{(i)}$. The two families of potential are visualized in Figure 9.10.

The dynamics induced by these two families of potentials play opposite roles in the process of pattern formation. The *diffusion* potential $\phi(\xi)$ assigns lowest energy (or highest probability) to filter responses close to the translation constant ξ. This type of potential is the primary object of study in early image models, and it induces anistropic diffusion where pixel intensity diffuses among neighbors in a process comparable to the classic heat equation. However, the $\phi(\xi)$ alone are incapable of synthesizing realistic image patterns. An MCMC process governed by $\phi(\xi)$ alone will eventually degenerate into a constant image, just as a concentration of heat in a closed system will eventually spread out and reach thermal equilibrium.

The *reaction* potential $\psi(\xi)$, an inversion of the diffusion potential, is a significant departure from all earlier image models. Low energy at tail ends of the reaction potential promotes high-magnitude response to the filters in $\mathscr{F}$. MCMC samples from a process governed by $\psi(\xi)$ do not degenerate into a constant image, because the high probability associated with high magnitude filter response leads to the active formation of pattern features such as edges and textures.

Two groups of filters $\mathscr{F}_d = \{F_d^{(1)}, \ldots, F_d^{(K_d)}\}$ and $\mathscr{F}_r = \{F_r^{(1)}, \ldots, F_r^{(K_r)}\}$ naturally emerge from the two families of potentials $\phi(\xi)$ and $\psi(\xi)$, with $\mathscr{F} = \mathscr{F}_d \cup \mathscr{F}_r$. Similarly, the FRAME energy (9.60) can be rewritten by separating the diffusion and reaction potentials:

$$U(I;\mathscr{F},\Lambda) = \sum_{i=1}^{K_d} \sum_{(x,y)\in\mathscr{L}} \phi^{(i)}(F_d^{(i)} * I_{(x,y)}) + \sum_{i=1}^{K_r} \sum_{(x,y)\in\mathscr{L}} \psi^{(i)}(F_r^{(i)} * I_{(x,y)}). \tag{9.64}$$

Filters in $\mathscr{F}_d$ tend to be gradient or Laplacian of Gaussian filters, since the features captured by these types of filters relate to the smoothness of an image. The potentials of smoothing filters usually encourage uniformity among nearby

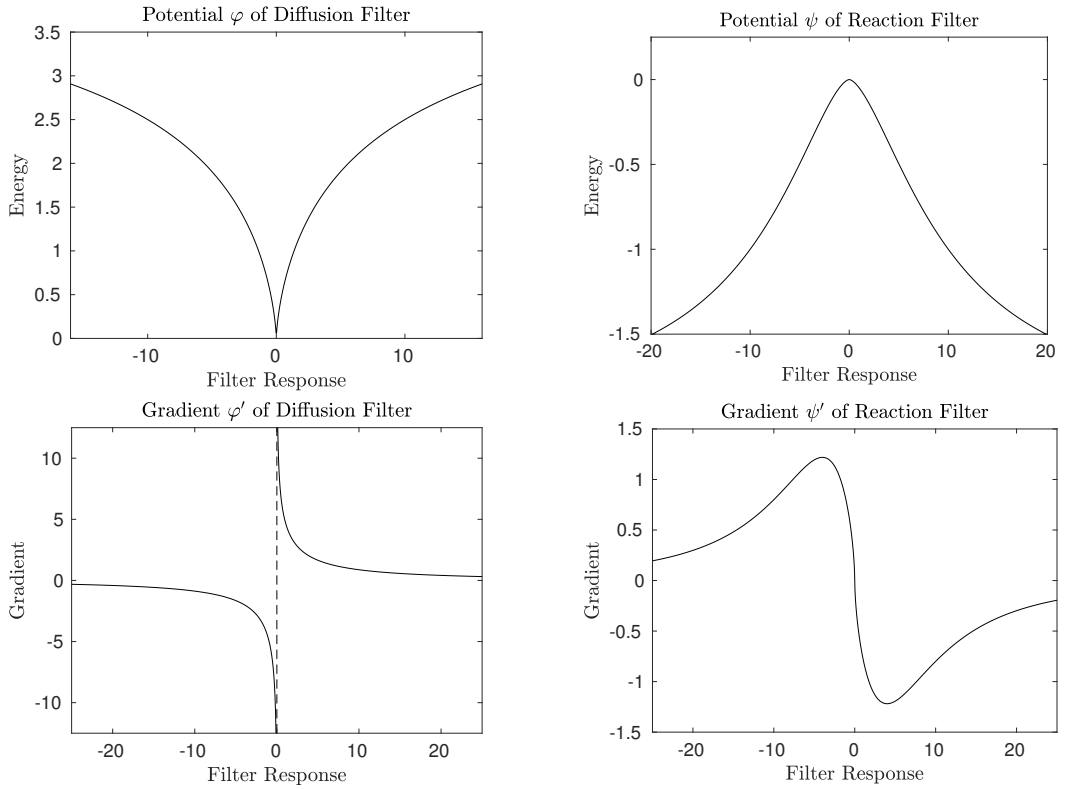


Fig. 9.10: *Left:* Potential and gradient of diffusion filter as a function of filter response. The potential parameters are $(a = 5, , b = 10, \gamma = 0.7, \xi_0 = 0)$. Diffusion filters have low energy when the filter response is close to $\xi_0 = 0$. Laplacian of Gaussian and gradient filters typically act as diffusion filters because they control the smoothness of an image. Since $\gamma < 1$, a non-differentiable cusp forms at $\xi = \xi_0$. *Right:* Energy and gradient of reaction filter as a function of filter response. The potential parameters are $(a = -2, , b = 10, \gamma = 1.6, \xi_0 = 0)$. Reaction filters have low energy when filter response is close to the extremes. Gabor filters typically act as reaction filters because they encode the formation of prominent pattern features such as bars and stripes. There is no cusp at $\xi = \xi_0$ because $\gamma > 1$.

groups of pixels. On the other hand, the filters in $\mathscr{F}_r$ are typically Gabor filters, which characterize salient features such as edges at different orientations.

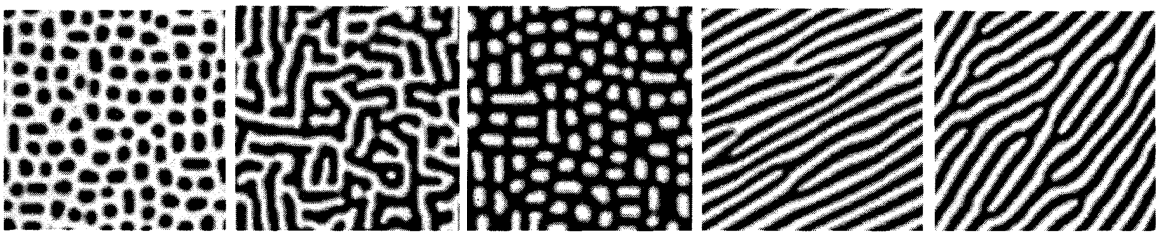
Minimizing (9.64) by gradient descent yields the partial differential equation for image $I(x,y,t)$:

$$\frac{\partial I}{\partial t} = \sum_{i=1}^{K_d} F_d^{(i)'} * \phi'(F_d^{(i)} * I) + \sum_{i=1}^{K_r} F_r^{(i)'} * \psi'(F_r^{(i)} * I), \tag{9.65}$$

where $F^{(i)'}(x,y) = -F^{(i)}(-x,-y)$. The first term reduces the response gradients of the diffusion filters $\mathscr{F}_d$, encouraging smoothness and uniformity, while the second term increases the gradients of the reaction filters $\mathscr{F}_r$, encouraging the formation of pattern features. (9.65) is known as the Gibbs Reaction And Diffusion Equation (GRADE) [10, 11]. To sample from $U(I;\mathscr{F},\Lambda)$ as opposed to simply minimizing $U(I;\mathscr{F},\Lambda)$, one can add isotropic noise to obtain the Langevin Equation

$$I_{s+1} = I_s + \frac{\varepsilon^2}{2}\frac{\partial I}{\partial t}(I_s) + \varepsilon Z_s \tag{9.66}$$

for MCMC iteration s, where $Z_s \sim \mathrm{N}(0, \sigma^2 I_{n\times n})$. The starting image I_0 is arbitrary. See Figure 9.11 for examples of images synthesized by Langevin dynamics applied to a FRAME potential.



a) b) c) d) e)

Fig. 9.11: Examples of images synthesized using Langevin Dynamics on a FRAME density parameterized by GRADE potentials. *a)-c):* Images synthesized using one Laplacian of Gaussian diffusion filter and one Laplacian of Gaussian reaction filter. The reaction filter for b) has parameter $\xi_0 = 0$, while $\xi_0 < 0$ for a) and $\xi_0 > 0$ for c). In this case, ξ_0 controls the blob color. *d)-e):* Images synthesized using one Laplacian of Gaussian diffusion filter and one or more Gabor reaction filters. The image d) uses a single cosine Gabor filter at angle 30°, while the image e) uses two cosine Gabor filters at angles 30°and 60°.

Example 9.2. **DeepFRAME Model. [8]** In the FRAME model, filters are selected from a pre-defined filter bank, which limits the kinds of patterns that can be represented. There is no guarantee that the filter bank is capable of effectively representing the training images, and synthesis results are poor when the filters fail to capture important image features. Recent trends in the neural network community have shown that learning the filters themselves during training can lead to flexible and realistic image models. Including multiple layers of filter convolution can also lead to significantly better representations of complex data. The DeepFRAME model [4, 8] extends the FRAME model to incorporate these new features. A DeepFRAME density has the form

$$p(I;W) = \frac{1}{Z}\exp\{F(I;W)\}q(I) \tag{9.67}$$

where q is the prior distribution $\mathrm{N}(0,\sigma^2 I_N)$ of Gaussian white noise, and the scoring function $F(\cdot;W)$ is defined by a ConvNet with weights W, which must be learned. The associated energy function has the form

$$U(I;W) = -F(I;W) + \frac{1}{2\sigma^2}||I||_2^2. \tag{9.68}$$

We may interpret $p(I;W)$ as an exponential tilting of q which has the effect of mean shifting. The non-linearity induced by the activation functions between network layers is essential for successful representation of real images.

When the the activation functions are rectified linear units (ReLU), $F(I;W)$ is piecewise linear in I, and the borders between linear regions are governed by the activations in the network [6]. Let $\Omega_{\delta,W} = \{I : \sigma_k(I;W) = \delta_k, 1 \le k \le K\}$, where W gives the network weights, K is the number of activation functions in the entire network, $\sigma_k(I;W) \in \{0,1\}$ indicates whether activation function k turns on for image I, and $\delta = (\delta_1,\ldots,\delta_K) \in \{0,1\}^K$. Since $F(I;W)$ is linear on $\Omega_{\delta,W}$ for all δ, the energy can be written as

$$U(I;W) = -(\langle I, B_{\delta,W}\rangle + a_{\delta,W}) + \frac{1}{2\sigma^2}||I||_2^2 \tag{9.69}$$

for some constants $a_{\delta,W}$ and $B_{\delta,W}$, which shows that $I \sim \mathrm{N}(\sigma^2 B_{\delta,W}, \sigma^2 I_N)$ on $\Omega_{\delta,W}$ and that $p(I;W)$ is piecewise Gaussian over the image space. This analysis also characterizes the local minima of $U(I;W)$, which are simply $\{\sigma^2 B_{\delta,W} : \sigma^2 B_{\delta,W} \in \Omega_{\delta,W}\}$. However, there is no guarantee that the Gaussian piece $\Omega_{\delta,W}$ contains its mode $\sigma^2 B_{\delta,W}$, and the number of Gaussian pieces is extremely large, so a direct enumeration of local minima is not viable.

As in virtually all Deep Learning applications, the gradient of $F(I;W)$ with respect to both the weights W and image I can be computed efficiently by backpropagation. Given a set of learned weights W, it is easy to apply the Langevin Dynamics in (9.58) to sample from an image density $P(I;W)$. See Figure 9.12 for examples of images synthesized using Langevin dynamics on a trained DeepFRAME density.

Fig. 9.12: *Left:* Langevin samples from a DeepFRAME model trained on an unaligned texture image. The left image is the training image, and the right two images are synthesized samples. The synthesized images reproduce the main texture features (the flower heads and grassy regions). *Right:* Langevin samples from a DeepFRAME model trained on aligned images. The training images are in the top row and example images synthesized from the DeepFRAME density are in the bottom row. © [2016] Xie et al. Reprinted, with permission, from ref. [8].

9.7 Exercises

Problem 1. Sampling from a Non-Isotropic Gaussian. Consider the target distribution $(X,Y) \sim N(0,\Phi)$ where $\Phi = \begin{pmatrix} 1 & 0.9998 \\ 0.9998 & 1 \end{pmatrix}$

a) What is the energy function of the target distribution? Make a contour plot of the energy function.
b) Suppose you are sampling from the target distribution with the identity matrix I_2 as the momentum covariance. What is the max step size ε^* and minimum number of Leapfrog steps L^* needed to obtain an independent sample from the target distribution in a single HMC iteration?
c) What is the ideal choice of the momentum covariance matrix Σ_{ideal}? What is the max step size ε^*_{ideal} and minimum number of Leapfrog steps L^*_{ideal} needed to obtain independent samples when using the ideal momentum covariance?
d) For each of the methods listed below, start a chain from the state $(X,Y) = (0,-10)$, run $1,000 \cdot K$ burn-in iterations, and $10,000 \cdot K$ sampling iterations from the target distribution (K is needed for fair comparison between methods and is given). For each method, visualize the burn-in path, plot the values of the X and Y coordinates over iterations of the sampling phase, and calculate the ESS of the final X and Y samples. For the plots and ESS calculation, use one sample point per K iterations. Comment on differences between the results. ε^* and L^* refer to the answers from b).

   1. Direct sampling from $N(0,\Phi), K = 1$.
   2. Metropolis-Hastings with Gaussian Proposal $N(0,(\varepsilon^*)^2), K = L^*$.
   3. Random walk: HMC with $p \sim N(0,I_2), \varepsilon = \varepsilon^*, L = L^*, K = 1$.
   4. HMC with $p \sim N(0,I_2), \varepsilon = \varepsilon^*, L = L^*/2, K = 1$.
   5. LMC with $p \sim N(0,I_2), \varepsilon = \varepsilon^*, K = L^*$.
   6. HMC with $p \sim N(0,\Sigma_{ideal}), \varepsilon = \varepsilon^*_{ideal}, L = L^*_{ideal}, K = 1$. (using the answers from part c) above)
   7. LMC with $p \sim N(0,\Sigma_{ideal}), \varepsilon = \varepsilon^*_{ideal}, K = L^*_{ideal}$.

Problem 2. Sampling from a "Banana" Distribution. Consider the posterior distribution of $\theta = (\theta_1,\theta_2)$ with prior $\theta \sim N(0,I_2)$ and $Y|\theta \sim N(\theta_1 + \theta_2^2, 2)$.

a) What is the energy function of the posterior density $P(\theta|Y)$? Make a contour plot of the energy function.
b) In new settings, the step size ε^* can be found by tuning the acceptance rate. For a grid of θ, run 2000 LMC iterations with momentum covariance I_2 starting from the origin. Choose a step size with rejection rates between 10%-35%, and report the value.
c) For each of the methods listed below, start a chain from the state $(\theta_1,\theta_2) = (0,0)$, run $1,000 \cdot K$ burn-in iterations, and $10,000 \cdot K$ sampling iterations from the target distribution (K is needed for fair comparison between methods and is given). For each method, plot the values of the θ_1 and θ_2 coordinates over iterations of the sampling phase, and calculate the ESS of the final θ_1 and θ_2 samples. For the plots and ESS calculation, use one sample point per K iterations. For the HMC methods 4 and 5, visualize the Leapfrog steps of an accepted path and a rejected path. Comment on differences between the results. ε^* refers to the value from b) above).

1. Metropolis-Hastings with Gaussian Proposal $N(0,(\epsilon^*)^2), K = 25$.
2. Random walk: LMC with $p \sim N(0, I_2), \epsilon = \epsilon^*, K = 1$.
3. LMC with $p \sim N(0, I_2), \epsilon = \epsilon^*, K = 25$.
4. HMC with $p \sim N(0, I_2), \epsilon = \epsilon^*, L = 5, K = 1$.
5. HMC with $p \sim N(0, I_2), \epsilon = \epsilon^*, L = 25, K = 1$.

References

1. S. Amari and H. Nagaoka. *Methods of Information Geometry*. Oxford Univ Press, 2000.
2. K. Brodlie, A. Gourlay, and J. Greenstadt. Rank-one and rank-two corrections to positive definite matrices expressed in product form. *IMA Journal of Applied Mathematics*, 11(1):73–82, 1973.
3. Mark Girolami and Ben Calderhead. Riemann manifold langevin and hamiltonian monte carlo. *Journal ofthe Royal Statistical Society: B*, 73(2):123–214, 2011.
4. Yang Lu, Song Chun Zhu, and Ying Nian Wu. Learning frame models using cnn filters. *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.
5. P.B. Mackenzie. An improved hybrid monte carlo method. *Physics Letters B*, 226:369–371, 1989.
6. Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2924–2932, 2014.
7. Radford M Neal. Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo Chapter 5*, 2011.
8. Jianwen Xie, Wenze Hu, Song Chun Zhu, and Ying Nian Wu. A theory of generative convnet. *International Conference on Machine Learning*, 2016.
9. Y. Zhang and C. Sutton. Quasi-newton methods for markov chain monte carlo. *Advances in Neural Information Processing Systems*, pages 2393–2401, 2011.
10. Song Chun Zhu and David Mumford. Prior learning and gibbs reaction-diffusion. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(11):1236–1250, 1997.
11. Song Chun Zhu and David Mumford. Grade: Gibbs reaction and diffusion equations. *ICCV*, pages 847–854, 1998.
12. Song Chun Zhu, Yingnian Wu, and David Mumford. Filters, random fields and maximum entropy (frame): Towards a unified theory for texture modeling. *International Journal of Computer Vision*, 27(2):107–126, 1998.

Chapter 10
Learning with Stochastic Gradient

"The starting point of a mathematical study of online learning must be a mathematical statement for our subjective understanding of what a learning system is." - Léon Bottou

Introduction

Statistical learning often involves minimizing an objective function to find a suitable value of the model parameter. A simple and ubiquitous method for minimizing a differentiable objective function is Gradient Descent, which uses iterative parameter updates in the direction of steepest descent to minimize the objective. However, there are situations where the calculation of the objective function gradient is either analytically intractable or computationally infeasible. Two important examples are parameter estimation for Gibbs models and weight optimization in deep neural networks. Stochastic Gradient methods, which use a random but unbiased estimate of the full gradient, can be a useful tool for overcoming situations where the full gradient is unavailable. In the first half of the chapter, several theorems concerning the approximation of the true gradient from per-observation gradients are presented, and an important connection between Stochastic Gradient and Langevin Dynamics is discussed. The second section covers parameter estimation for Markov Random Field models, and the final section presents MCMC learning methods for deep image models.

10.1 Stochastic Gradient: Motivation and Properties

Minimization of an objective function f is the most common framework for statistical learning. In discriminative modeling, the objective is a loss function $f(w)$ that measures the predictive error of the model with parameters w, such as the cross-entropy between the true and predicted classes. In generative modeling, the objective is a measure $f(\theta) = D(q, p_\theta)$ of the difference between the true data distribution q and a distribution p_θ from a family $\mathscr{P} = \{p_\theta : \theta \in \Theta\}$. A popular measure of separation between probability distributions is the KL Divergence.

When the parameters x of a statistical model are continuous and the objective function $f(x)$ is differentiable, a simple but effective method for minimizing f is *Gradient Descent* [2]. Starting from an initial point x_0, the parameters are updated according to the rule

$$x_{t+1} = x_t - \gamma_t \nabla f(x_t), \tag{10.1}$$

where $\gamma_t > 0$ is the step size of iteration t, until a local minimum is found. When the true gradient $\nabla f(x)$ is unavailable for either analytical or computational reasons, one can instead use a stochastic approximation $\widetilde{\nabla} f(x)$ of the true gradient satisfying $\mathrm{E}\left[\widetilde{\nabla} f(x)\right] = \nabla f(x)$. The update rule

$$x_{t+1} = x_t - \gamma_t \widetilde{\nabla} f(x_t), \tag{10.2}$$

is known as *Stochastic Gradient Descent* (SGD). Two typical cases are discussed below.

10.1.1 Motivating cases

Example 10.1. **Maximum Likelihood Parameter Estimation for a Gibbs Model.** Stochastic gradients are useful when the gradient of f has the form of an analytically intractable expectation

$$\nabla f(\theta) = E_{p_\theta}[g(X;\theta)] \tag{10.3}$$

for a family of distributions $\mathscr{P} = \{p_\theta\}_{\theta\in\Theta}$ and random variable $X \in \mathscr{X}$. This situation is encountered when Maximum Likelihood is used to estimate the parameters of a Gibbs model

$$p_\theta(X) = \frac{1}{Z(\theta)} \exp\{-U(X;\theta)\} \tag{10.4}$$

with partition function $Z(\theta) = \int_{\mathscr{X}} \exp\{-U(X;\theta)\}dX$. The log-likelihood of p_θ given i.i.d. observed data $\{X_i\}_{i=1}^n$ is

$$l(\theta) = \frac{1}{n}\sum_{i=1}^{n} \log p_\theta(X_i) = -\log Z(\theta) - \frac{1}{n}\sum_{i=1}^{n} U(X_i;\theta), \tag{10.5}$$

and maximizing $l(\theta)$ yields the Maximum Likelihood Estimate (MLE) θ^* of the model parameters.

Maximizing the log-likelihood (10.5) to find the MLE is equivalent to finding the value of θ that minimizes the KL Divergence between p_θ and the true data distribution q. Observe that

$$\begin{aligned} KL(q||p_\theta) &= \mathrm{E}_q\left[\log \frac{q(X)}{p_\theta(X)}\right] \\ &= \mathrm{E}_q[\log q(X)] - \mathrm{E}_q[\log p_\theta(X)], \end{aligned}$$

and that $\mathrm{E}_q[\log q(X)]$ does not depend on θ. Minimizing $KL(q||p_\theta)$ requires minimizing only $-\mathrm{E}_q[\log p_\theta(X)]$. Given a dataset $\{X_i\}_{i=1}^n$ that follows q, we can use the Law of Large Numbers to obtain the approximation $\mathrm{E}_q[\log p_\theta(X)] \approx \frac{1}{n}\sum_{i=1}^n \log p_\theta(X_i)$. Therefore minimizing $KL(q||p_\theta)$ is equivalent to maximizing (10.5), and p_{θ^*} can be interpreted as the closest approximation to q in the family $\mathscr{P}$.

The intractable partition function $Z(\theta)$ is a major obstacle when evaluating $\nabla l(\theta)$. Fortunately, the gradient of $\log Z(\theta)$ can be expressed in closed form:

$$\frac{d}{d\theta}\log Z(\theta) = -\mathrm{E}_{p_\theta}\left[\frac{\partial}{\partial\theta}U(X;\theta)\right]. \tag{10.6}$$

This can be derived as follows:

$$\begin{aligned} \frac{d}{d\theta}\log Z(\theta) &= \frac{1}{Z(\theta)}\left[\frac{d}{d\theta}Z(\theta)\right] \\ &= \frac{1}{Z(\theta)}\left[\frac{d}{d\theta}\int_{\mathscr{X}} \exp\{-U(X;\theta)\}dX\right] \\ &= \frac{1}{Z(\theta)}\int_{\mathscr{X}} \frac{\partial}{\partial\theta}\exp\{-U(X;\theta)\}dX \\ &= -\frac{1}{Z(\theta)}\int_{\mathscr{X}} \exp\{-U(X;\theta)\}\left[\frac{\partial}{\partial\theta}U(X;\theta)\right]dX \\ &= -\int_{\mathscr{X}} p_\theta(X)\left[\frac{\partial}{\partial\theta}U(X;\theta)\right]dX = -\mathrm{E}_{p_\theta}\left[\frac{\partial}{\partial\theta}U(X;\theta)\right]. \end{aligned}$$

The interchange of the integral and derivative operations in the third line is justified under mild regularity conditions. This analysis shows that the likelihood gradient can be written as

$$\nabla l(\theta) = \mathrm{E}_{p_\theta}\left[\frac{\partial}{\partial\theta}U(X;\theta)\right] - \frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial\theta}U(X_i;\theta), \tag{10.7}$$

which is a special case of the form (10.3) where

$$g(X;\theta) = \frac{\partial}{\partial\theta}U(X;\theta) - \frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial\theta}U(X_i;\theta).$$

In all but the simplest cases, the expectation $\mathrm{E}_{p_\theta}\left[\frac{\partial}{\partial\theta}U(X;\theta)\right]$ cannot be computed exactly. However, by obtaining MCMC samples $\{Y_i\}_{i=1}^m$ from the distribution p_θ, we can use the Law of Large Numbers approximation $\mathrm{E}_{p_\theta}\left[\frac{\partial}{\partial\theta}U(X;\theta)\right] \approx \frac{1}{m}\sum_{i=1}^m \frac{\partial}{\partial\theta}U(Y_i;\theta)$ and calculate a stochastic approximation $\widetilde{\nabla} l(\theta)$ of the true gradient $\nabla l(\theta)$:

$$\widetilde{\nabla} l(\theta) = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial\theta}U(Y_i;\theta) - \frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial\theta}U(X_i;\theta). \tag{10.8}$$

The MCMC samples $\{Y_i\}_{i=1}^m$ from p_θ are sometimes called *negative samples*, as opposed to the "positive" samples $\{X_i\}_{i=1}^n$ that follow the true distribution q. After the approximate gradient is computed, one can apply the Stochastic Gradient Descent update (10.2) to iteratively solve for the MLE $\hat{\theta}$, with $\widetilde{\nabla} f(\theta) = -\widetilde{\nabla} l(\theta)$ since we are *maximizing* $l(\theta)$. Intuitively, the gradient in (10.8) encourages the MCMC samples of p_θ to match the training data $\{X_i\}_{i=1}^n$ in the features encoded by $U(X;\theta)$.

Example 10.2. **Large Scale Learning.** Stochastic Gradient Descent is also useful when the objective is the sum of a large number of differentiable sub-functions:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n} L_i(w). \tag{10.9}$$

The derivative of $L(w)$ has the simple form

$$\nabla L(w) = \frac{1}{n}\sum_{i=1}^{n}\nabla L_i(w). \tag{10.10}$$

The objective function form (10.9) is encountered frequently in practice. An additive objective can appear when a per-observation loss $L(X;w)$ is applied to a set of observed data $\{X_i\}_{i=1}^n$, where $L_i(w) = L(X_i;w)$. In supervised learning, the observations come as pairs $X_i = (Z_i, Y_i)$ and the individual loss terms are $L(X_i;w) = L(Z_i,Y_i;w)$. Generally, the different observations X_i are treated as i.i.d. samples. When $L(X;w)$ is the negative log-likelihood $-\log p(X;w)$ in unsupervised learning or the conditional negative log-likelihood $-\log p(Z|Y;w)$ in supervised learning, independence between observations causes the loss of the entire dataset to split into the sum of the individual loss terms:

$$-\log p(\{X_i\}_{i=1}^n;w) = -\log\left[\prod_{i=1}^{n} p(X_i;w)\right] = \sum_{i=1}^{n} -\log p(X_i;w) = \sum_{i=1}^{n} L(X_i;w) \propto L(w).$$

When modeling complex data, the individual gradients $\frac{\partial}{\partial w}L(X_i;w)$ can be costly to compute. If the dataset is very large, calculation of the full gradient $\nabla L(w)$ can quickly become prohibitively expensive. Costly gradient calculation and large datasets are both typical in deep learning applications. On the other hand, the per-observation gradients $\frac{\partial}{\partial w}L(X_i;w)$ can be considered noisy versions of the true gradient $\nabla L(w)$, and the predictive power of a small number of gradients remains constant in the limit of large sample size. This motivates the use of *mini-batches* $\{X_i : i \in B\}$ for a random sample $B \subset \{1,\ldots,n\}$ of size $|B| = n_B$. In each iteration, gradients are calculated for mini-batch observations only, yielding the stochastic gradient

$$\nabla_B L(w) = \frac{1}{n_B} \sum_{i \in B} \frac{\partial}{\partial w} \nabla L(X_i; w). \tag{10.11}$$

It is straightforward to show that $\mathrm{E}[\nabla_B L(w)] = \nabla L(w)$, and further properties of (10.11) are discussed in Section 10.1.3. A special case of (10.11) is *online learning*, which uses a single observation in each batch (i.e. $n_B = 1$).

In deep learning applications that use MCMC (see Section 10.3), stochastic gradients from both Example 10.1 and Example 10.2 are needed. When learning the MLE parameters of deep network functions with the gradient (10.8), MCMC samples are needed to obtain an approximate gradient in the first term, while mini-batches of observed data are used in the second term to reduce computational cost.

10.1.2 Robbins-Monro theorem

The convergence analysis of the SGD algorithm (10.2) is limited to the 1-dimensional case $d = 1$.

Theorem 10.1 (Robbins-Monro) *The sequence w_n converges in L^2 (hence in probability) to θ if the following conditions are met:*

1. *$G(w)$ is uniformly bounded in the sense that there exists $C < \infty$ such that $P(|G(w)| \leq C) = 1$.*
2. *$F(w)$ is non-decreasing. differentiable and $F'(\theta) > 0$.*
3. *The sequence γ_n satisfies $\sum_{n=0}^{\infty} \gamma_n = \infty$ and $\sum_{\gamma=0}^{\infty} \gamma_n^2 < \infty$.*

Multi-dimensional versions of Theorem 10.1 are limited by many restrictive assumptions. A recent explicit version version is [18], which considers the case when the noisy observations $G(\theta_n)$ are gradients of convex differentiable functions $f_n : \mathbb{R}^d \to \mathbb{R}$, thus $G(\theta_n) = \nabla f_n(\theta_n)$.

Theorem 10.2 (Moulines, 2011) *Assume the following conditions are met*

H1) There exists a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ such that

$$E[\nabla f_n(\theta)] = \nabla f(\theta), \quad \forall n \geq 1 \quad \textit{with probability } 1$$

H2) For all $n \geq 1$ f_n is almost surely convex, differentiable and

$$\forall \theta_1, \theta_2 \in \mathbb{R}^d, E(\|\nabla f_n(\theta_1) - \nabla f_n(\theta_2)\|^2) \leq L^2 \|\theta_1 - \theta_2\|^2 \quad \textit{with probability } 1$$

H3) The function f is strongly convex with respect to the norm $\|\cdot\|$ with some constant $\mu > 0$:

$$\forall \theta_1, \theta_2 \in \mathbb{R}^d, f(\theta_1) \geq f(\theta_2) + (\theta_1 - \theta_2) \nabla f(\theta_2) + \frac{\mu}{2} \|\theta_1 - \theta_2\|^2$$

H4) There exists $\sigma > 0$ such that $\forall n \geq 1$, $E(\|\nabla f_n(\theta)\|^2) \leq \sigma$.

Let $\delta_0 = \|\theta_0 - \theta\|^2$, $\varphi_\beta(t) = \frac{t^\beta - 1}{\beta}$ and $\gamma_n = Cn^{-\alpha}$ for some $\alpha \in [0,1]$. Then the sequence from eq. (10.2) *satisfies*

$$E\|\theta_n - \theta\|^2 \leq 2 \exp\left[4L^2C^2\varphi_{1-2\alpha}(n) - \frac{\mu C}{2} n^{1-\alpha}\right] \left(\delta_0 + \frac{\sigma^2}{L^2}\right) + \frac{4C\sigma^2}{\mu n^\alpha}$$

when $\alpha < 1$. If $\alpha = 1$ then:

$$E\|\theta_n - \theta\|^2 \leq \frac{\exp(2L^2C^2)}{n^{\mu C}} \left(\delta_0 + \frac{\sigma^2}{L^2}\right) + 2\frac{C^2\sigma^2}{n^{\mu C/2}} \varphi_{\mu C/2 - 1}(n).$$

Convergence time is proportional to the condition number $\kappa = \lambda_{max}/\lambda_{min}$ of the Hessian matrix $H_{ij} = \frac{\partial^2 L(\theta)}{\partial w_i \partial w_j}$ at the minimum [15]. This behavior is comparable to the HMC properties discussed in Section 9.5. The full gradient

iteration (with batch size equal to n) has *linear* convergence in the sense that $L(\theta^k) - L(\theta) = O(\rho^k)$ where $\rho < 1$ depends on the condition number κ (from [19], Theorem 2.1.15). The online version has a sub-linear convergence $E[L(\theta^k)] - L(\theta) = O(1/k)$ [20] but each iteration is n times faster.

10.1.3 Stochastic gradient descent and the Langevin equation

Consider the stochastic mini-batch gradient from (10.11). The sampling process of mini-batch selection defines a multivariate distribution of gradients. One can analyze SGD using the first and second moments of the gradients to define a continuous-time analog of SGD in the form of a Langevin Equation (see Section 9.4.3). Surprisingly, the diffusion matrix that appears in the SGD Langevin Equation is the *inverse* of matrix used in traditional Newtonian optimization. The connections between SGD and the Langevin equation shed light on important properties of SGD, including the role of batch size, the steady-state distribution and generalization.

Consider the minimization of the additive loss $f(x) = \sum_{i=1}^{n} f_i(x)$ using mini-batch gradients $\nabla_B f(x) = \frac{1}{n_B} \sum_{i \in B} \nabla f_i(x)$. Following Hu et al. [13], we begin by finding the expectation and variance of the stochastic gradient $\nabla_B f(x)$. The matrix

$$D(x) = \left(\frac{1}{N} \sum_{i=1}^{N} \nabla f_i(x) \nabla f_i(x)^\mathsf{T} \right) - \nabla f(x) \nabla f(x)^\mathsf{T} \tag{10.12}$$

will play an important role in the analysis, and we will refer to it as the *diffusion matrix* for reasons that will become clear. Let $B = \{i_1, \ldots, i_{n_B}\}$ be the indices of mini-batch members. Assume for ease of analysis that B is a simple random sample with replacement of $\{1, \ldots, n\}$. The single-sample expectation and variance are

$$\mathrm{E}[\nabla f_{i_j}(x)] = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(x) = \nabla f(x),$$

$$\mathrm{Var}[\nabla f_{i_j}(x)] = \mathrm{E}\left[\nabla f_{i_j}(x) \nabla f_{i_j}(x)^\mathsf{T}\right] - \mathrm{E}\left[\nabla f_{i_j}(x)\right] \mathrm{E}\left[\nabla f_{i_j}(x)\right]^\mathsf{T} = D(x).$$

Standard results about the mean and variance of a simple random sample with replacement show:

$$\mathrm{E}[\nabla_B f(x)] = \frac{1}{n_B} \sum_{j=1}^{n_B} \mathrm{E}[\nabla f_{i_j}(x)] = \frac{1}{n_B} \sum_{j=1}^{n_B} \nabla f(x) = \nabla f(x) \tag{10.13}$$

$$\mathrm{Var}[\nabla_B f(x)] = \mathrm{Var}\left[\frac{1}{n_B} \sum_{j=1}^{n_B} \nabla f_{i_j}(x) \right] = \frac{1}{n_B^2} \sum_{j=1}^{n_B} \mathrm{Var}\left[\nabla f_{i_j}(x)\right] = \frac{D(x)}{n_B} \tag{10.14}$$

where the second equality in the variance calculation uses the fact that the mini-batch gradients are independent random variables.

We can separate the mean from the mini-batch gradient and write the SGD update (10.2) as

$$X_{t+1} = X_t - \eta \nabla_B f(X_t) = X_t - \eta \nabla f(X_t) + \sqrt{\eta}\, V_t$$

where $V_t = \sqrt{\eta}(\nabla f(X_t) - \nabla_B f(X_t))$. The SGD diffusion term V_t clearly has mean $\mathrm{E}[V_t] = 0$ and variance $\mathrm{Var}[V_t] = \frac{\eta}{|B|} D(X_t)$. Li et al. [16] analyze V_t to prove the following connection between SGD and Langevin Dynamics. See Hu et al. [13] for additional analysis.

Theorem 10.3 (Li et al., 2017 [16]) *Assume f and $\{f_i\}_{i=1}^n$ are Lipschitz continuous, have at most linear asymptotic growth, and have sufficiently high derivatives belonging to the set of functions with polynomial growth. Then*

$$dX_t = -\nabla f(X_t) dt + \sqrt{\frac{\eta}{n_B} D(X_t)}\, dW_t, \tag{10.15}$$

where dW_t is Brownian motion, is a first-order stochastic approximation of the SGD process in (10.2) with minibatch gradients (10.11).

The equivalence between SGD and the Langevin Dynamics in (10.15) reveals several important properties of SGD. One observation is that SGD batch size plays the role of inverse temperature, since the magnitude of the noise $\frac{\eta}{n_B}D(X_t)$ varies inversely with batch size n_B. Using a smaller batch size can be intuitively understood as exploring the loss landscape at a higher temperature, since more variation exists in the sampling process when smaller batches are used. SGD with large batches is similar to full-gradient optimization, just like low-temperature sampling. Interestingly, recent work [14] has shown that deep network parameters found using SGD with *small* batches have better generalization properties than parameters found using SGD with large batches. This is evidence that "high-temperature" dynamics resulting from small-batch SGD are less prone to becoming trapped in local minima than full-gradient methods. While SGD was initially adopted for reasons of computational efficiency, it also possesses properties that are naturally suited to finding parameters with good generalization in highly overparameterized settings.

Zhang et al. [25] observe that the structure of the diffusion matrix $D(x)$ is also related to the generalization properties of SGD. Consider an SGD process near a local minimum. In this case, $\nabla f(x) \approx 0$ and the diffusion matrix $D(x)$ is approximately equal to the empirical Fisher information:

$$D(x) \approx \frac{1}{N}\sum_{i=1} \nabla f_i(x)\nabla f_i(x)^\intercal \approx \mathrm{E}[\nabla f_i(x)\nabla f_i(x)]^\intercal. \tag{10.16}$$

Eigenvectors of the Fisher Information with large eigenvalues have large curvature, meaning that small perturbations along these directions can cause large changes in the output of $f(x)$. In contrast, eigenvectors of the Fisher Information with small eigenvalues correspond to "flat" directions that are resistant to small perturbations of the input. Intuitively, "flat" minima have better generalization properties, because robustness against input perturbation indicates that the model does not suffer from overfitting.

Comparing the SGD Langevin Dynamics in (10.15) with Riemann Manifold Langevin Dynamics (see Section 9.5) reveals the surprising fact the the noise matrix $D(x)$ in (10.16) is the *inverse* of the noise matrix that appears in RMLMC. Recall that the RMLMC noise covariance is the *inverse* Fisher Information, which rescales the local landscape geometry to have uncorrelated unit variance in each direction. In other words, the RMLMC dynamics adjust the isotropic noise added to the gradient in LMC so that smaller steps are taken along more constrained directions of the local landscape (directions with larger eigenvalue), while larger steps are taken along flatter directions of the local landscape (directions with smaller eigenvalue). The RMLMC dynamics are consistent with the philosophy of traditional second-order optimization techniques.

On the other hand, the noise matrix $D(x)$ in SGD is an approximation of the *uninverted* Fisher Information. Unlike traditional second-order optimization, SGD takes larger steps in directions with high eigenvalues and smaller steps in directions with low eigenvalues. The diffusion dynamics of SGD actively probe the most constrained directions of the local covariance. The constrained directions can be intuitively understood as "untrustworthy" directions, because small changes in the model along these directions results in large change in model performance. Interestingly, the diffusion term in SGD appears to focus on the untrustworthy directions. By seeking out the most constrained directions at each iteration, SGD eventually locates "flat" local minima where nearly all directions are unconstrained. Thus, SGD can escape narrow basins early in training and avoid overfitting. SGD dynamics are an important departure from classical optimization techniques. Tightly constrained directions in the local geometry need not be respected – instead, active perturbation of the stiff directions can help find flat minima with better generalization. Optimization of highly complex functions in deep learning benefits not only from the computational efficiency of SGD, but also from properties of SGD that naturally adapt to the geometry of the loss landscape.

Given that SGD can be approximated by a continuous-time Langevin Equation, it is natural to analyze the steady state distribution of the Langevin dynamics. Naive intuition might suggest that the steady state is $\frac{1}{Z}\exp\{-f(x)/T\}$, a Gibbs distribution of the objective function f. However, the true steady state of the SGD approximation is more complex. For some intuition, consider the dynamics of (10.15) when $D(x)$ is a constant matrix D. In this case, the dynamics (10.15) can be written as

$$dX_t = -\frac{\eta}{2n_B} D\left[\frac{2n_B}{\eta}D^{-1}\nabla f(X_t)\right]dt + \sqrt{2\left(\frac{\eta}{2n_B}\right)D}\,dW_t. \tag{10.17}$$

Discussion of the Langevin Equation from Section 9.4.3 shows that, if there is a function g that satisfies $\nabla g(x) = D^{-1}\nabla f(x)$, then $\frac{1}{Z}\exp\{-\frac{2n_B}{\eta}\, g(x)\}$ must be the steady state of (10.17). It is easy to see that the steady state of (10.17) has the form $\frac{1}{Z}\exp\{-f(x)/T\}$ when $D = c\,\mathrm{Id}$, a scalar times the identity matrix.

More generally, Chaudhari and Soatto [3] show under simplifying assumptions on the structure of $\nabla f(x)$ that the steady state of the SGD Langevin (10.15) is equal to $\frac{1}{Z}\exp\{-f(x)/T\}$ if and only if $D(x) = c\,\mathrm{Id}$. In practice, the diffusion matrix observed in an SGD trajectory is highly non-isotropic. This is unsurprising given preceding observations that the structure of $D(x)$ is related to the non-Euclidean geometry of f. Therefore the authors in [3] conclude that SGD does *not* follow a Gibbs distribution of the objective function. On the other hand, the appearance of the natural gradient $D^{-1}\nabla f(X)$ and the diffusion matrix D in (10.17) suggest that the dynamics of SGD adapt naturally to the energy landscape of f. A further understanding of the steady state and geometric properties of SGD could continue to yield valuable insight into the complex behavior of large scale optimization and deep learning.

10.2 Parameter Estimation for Markov Random Field (MRF) Models

This section covers methods for learning the parameters of a Markov Random Field (MRF) model. This family of densities has the form

$$p_\beta(x) = \frac{1}{Z(\beta)}\exp\left\{-\sum_{k=1}^{K}\langle\beta^{(k)}, U_k(x)\rangle\right\} = \frac{1}{Z(\beta)}\exp\{-\langle\beta, U(x)\rangle\} \tag{10.18}$$

where $\beta = (\beta^{(1)}, \ldots, \beta^{(K)})$ with $\beta^{(k)} \in \mathbb{R}^{d_k}$ contains the model parameters to be learned, $x \in \Omega \subset \mathbb{R}^d$ is an observation in the MRF space, $U_k(x) : \Omega \to \mathbb{R}^{d_k}$ are sufficient statistics, and $Z(\beta)$ is the partition function $Z(\beta) = \int_\Omega \exp(-\langle\beta, U(x)\rangle)dx$. The MRF density is a special case of the Gibbs density (10.4) where the potential energy has the linear decomposition $U(x;\beta) = \langle\beta, U(x)\rangle$.

A MRF distribution can be derived as the *Maximum Entropy* distribution satisfying a given set of expectations. Consider sufficient statistics $U_k(x) : \Omega \to \mathbb{R}^{d_k}$ for $k = 1, \ldots, K$, and suppose one wants to find a probability distribution p with a specific expected value $\mathrm{E}_p[U_k(X)] = a_k$ for each U_k. In order to avoid specifying the model beyond the enforced expectations, one should look for the most general distribution that meets only the requirements on the sufficient statistics. Using the entropy $-\mathrm{E}_p[\log p(X)]$ as a measure for the generality of a distribution yields the constrained optimization problem

$$p(x) = \mathrm{argmax}_p\left\{-\int_\Omega p(x)\log p(x)\,dx\right\} \quad \text{subject to} \quad \mathrm{E}_p[U_k(X)] = a_k \quad \text{for } k = 1, \ldots, K. \tag{10.19}$$

Using the method of Lagrange Multipliers, one can show that the solution of the constrained maximization problem (10.20) is the MRF distribution (10.18), where $\beta = (\beta^{(1)}, \ldots, \beta^{(K)})$ is the Lagrange multiplier.

While the form of the maximum entropy density for a given set of expectations is always an MRF density, learning an MRF model still requires finding the hyperparameter β. The Lagrange multipliers cannot be calculated explicitly and iterative estimation by gradient descent must be used instead. As mentioned in Example 10.1, finding the MLE β^* by maximizing the log-likelihood of i.i.d. observed data is equivalent to minimizing the KL Divergence between the MRF model p_β and the true data distribution, which further is equivalent to minimizing the entropy of p_β under the true distribution. Therefore MLE learning for MRF models can be interpreted as *Minimax Entropy* learning: the MRF density gives the form of the maximum entropy model, while MLE learning finds the minimum-entropy model in the MRF family.

Given a set of i.i.d. observations $\{X_i\}_{i=1}^n$, one can follow Example 10.1 to find the MLE β^* of the log-likelihood (10.5) by optimizing β according to the gradient in (10.8). Since the potential energy of the MRF density is linear in β, the score function is $\frac{\partial}{\partial\beta}\nabla\log p_\beta(x) = -\nabla\log Z(\beta) - U(x)$ and the gradient of the log-likelihood $l(\beta)$ has the simpler form

$$\nabla l(\beta) = \mathrm{E}_{p_\beta}[U(X)] - \frac{1}{n}\sum_{i=1}^{n} U(X_i) \approx \frac{1}{m}\sum_{i=1}^{m} U(Y_i) - \frac{1}{n}\sum_{i=1}^{n} U(X_i) \tag{10.20}$$

where $\{Y_i\}_{i=1}^m$ are samples of p_β obtained from MCMC. The Monte Carlo simulations are needed to estimate the intractable expectation $\mathrm{E}_{p_\beta}[U(X)]$ to obtain an approximate gradient of β. Due to the linearity of potential $-\langle \beta, U(x)\rangle$, the log-likelihood $l(\beta)$ is convex and the MLE β^* is unique for MRF models. Observe that the stochastic gradient (10.20) encourages samples from the current distribution p_β to match the sufficient statistics of samples from the true data distribution.

10.2.1 Learning a FRAME model with stochastic gradient

As a concrete example of stochastic gradient learning for MRF models, we discuss the FRAME model [28] introduced in Section 9.6.3 Example 9.1. Let $\mathbf{I}_\Lambda$ be an image defined on a lattice Λ and $\mathbf{I}_{\partial\Lambda}$ be the fixed boundary conditions for a neighborhood $\partial\Lambda$ of the active lattice Λ. Let $\mathbf{h}(\mathbf{I}_\Lambda|\mathbf{I}_{\partial\Lambda})$ be the feature statistics of $\mathbf{I}_\Lambda$ under boundary conditions $\mathbf{I}_{\partial\Lambda}$. Typically $\mathbf{h}(\cdot)$ is a set of histograms $\{\mathbf{h}_k(\cdot)\}_{k=1}^K$ that measure response to convolutional filters $\{F_k\}_{k=1}^K$ on $\mathbf{I}_\Lambda|\mathbf{I}_{\partial\Lambda}$. The FRAME density has the form (see [27])

$$p(\mathbf{I}_\Lambda|\mathbf{I}_{\partial\Lambda};\beta) = \frac{1}{Z(\mathbf{I}_{\partial\Lambda},\beta)}\exp\left\{-\sum_{k=1}^K \langle \beta^{(k)}, \mathbf{h}_k(\mathbf{I}_\Lambda|\mathbf{I}_{\partial\Lambda})\rangle\right\} = \frac{1}{Z(\mathbf{I}_{\partial\Lambda},\beta)}\exp\{-\langle \beta, \mathbf{h}(\mathbf{I}_\Lambda|\mathbf{I}_{\partial\Lambda})\rangle\}. \tag{10.21}$$

In the original FRAME applications, a single texture image $\mathbf{I}^{\text{obs}}$ is used as training data for the model. Assuming that the distribution of a texture image is spatially invariant, different patches of $\mathbf{I}^{\text{obs}}$ (local regions where filter convolution is non-zero) can be treated as i.i.d. observations. Thus, random patches from a single large texture image have the same statistics $\mathbf{h}(\mathbf{I}_\Lambda|\mathbf{I}_{\partial\Lambda})$ as independent patches. More care must be taken when modeling aligned data such as objects.

Maximizing the FRAME log-likelihood

$$\mathcal{G}(\beta) = \log p(\mathbf{I}_\Lambda^{\text{obs}}|\mathbf{I}_{\partial\Lambda}^{\text{obs}};\beta) \tag{10.22}$$

using the gradient (10.20) in the form

$$\widetilde{\nabla}\mathcal{G}(\beta) = \mathbf{h}(\mathbf{I}_\Lambda^{\text{syn}}|\mathbf{I}_{\partial\Lambda}^{\text{obs}}) - \mathbf{h}(\mathbf{I}_\Lambda^{\text{obs}}|\mathbf{I}_{\partial\Lambda}^{\text{obs}}) \tag{10.23}$$

can be used to iteratively solve for the unique MLE β^*. The synthesized image $\mathbf{I}^{\text{syn}}$ is generated by MCMC samples from the current model $p(\mathbf{I}_\Lambda|\mathbf{I}_{\partial\Lambda};\beta)$. A sketch of the FRAME learning algorithm is given below. See Section 10.2.4 for examples of images synthesized by FRAME.

Algorithm 13 FRAME Algorithm

Input: Observed texture image $\mathbf{I}^{\text{obs}}$, filter bank $\mathscr{F} = \{F_1, \ldots, F_K\}$, number of Gibbs sweeps S, step size $\delta > 0$, convergence error $\varepsilon > 0$.
Output: MLE $\beta^* = ((\beta^{(1)})^*, \ldots, (\beta^{(K)})^*)$ and synthesized image $\mathbf{I}^{\text{syn}}$.

Compute $\mathbf{h}(\mathbf{I}_\Lambda^{\text{obs}}|\mathbf{I}_{\partial\Lambda}^{\text{obs}}) = \{\mathbf{h}_k(\mathbf{I}_\Lambda^{\text{obs}}|\mathbf{I}_{\partial\Lambda}^{\text{obs}})\}_{k=1}^K$ by filter convolution with $\{F_k\}_{k=1}^K$ at each location $(x,y) \in \Lambda$.
Initialize $\beta_0^{(k)} = 0$ for $k = 1, \ldots, K$ and $\mathbf{I}^{\text{syn}}$ as uniform white noise.
repeat
a. Compute $\mathbf{h}(\mathbf{I}_\Lambda^{\text{syn}}|\mathbf{I}_{\partial\Lambda}^{\text{obs}}) = \{\mathbf{h}_k(\mathbf{I}_\Lambda^{\text{syn}}|\mathbf{I}_{\partial\Lambda}^{\text{obs}})\}_{k=1}^K$.
b. Update β according to

$$\beta_t = \beta_{t-1} + \delta\widetilde{\nabla}\mathcal{G}(\beta_{t-1})$$

where $\widetilde{\nabla}\mathcal{G}(\beta)$ is the stochastic gradient in (10.23).
c. Apply a Gibbs sampler to update $\mathbf{I}^{\text{syn}}$ for S sweeps using the density $p(\mathbf{I}_\Lambda|\mathbf{I}_{\partial\Lambda};\beta_t)$ in (10.21).
until $\frac{1}{2}\sum_{k=1}^K ||\mathbf{h}_k(\mathbf{I}_\Lambda^{\text{obs}}|\mathbf{I}_{\partial\Lambda}^{\text{obs}}) - \mathbf{h}_k(\mathbf{I}_\Lambda^{\text{syn}}|\mathbf{I}_{\partial\Lambda}^{\text{obs}})||_1 < \varepsilon$.

10.2.2 Alternate methods of learning for FRAME

A major obstacle when training a FRAME model is the computational cost of the MCMC steps used to synthesize images after each update of β. Since the histogram features are not differentiable in $\mathbf{I}$, HMC is not a viable option and single-site Gibbs sampling must be used instead. This method of generating negative samples for stochastic gradient learning can be very slow. Variants of the original FRAME algorithm can be used to increase training speed by reducing the computational burden of evaluating $\mathrm{E}_{p(I_\Lambda|I_{\partial\Lambda};\beta)}[\mathbf{h}(\mathbf{I}_\Lambda|\mathbf{I}_{\partial\Lambda})]$.

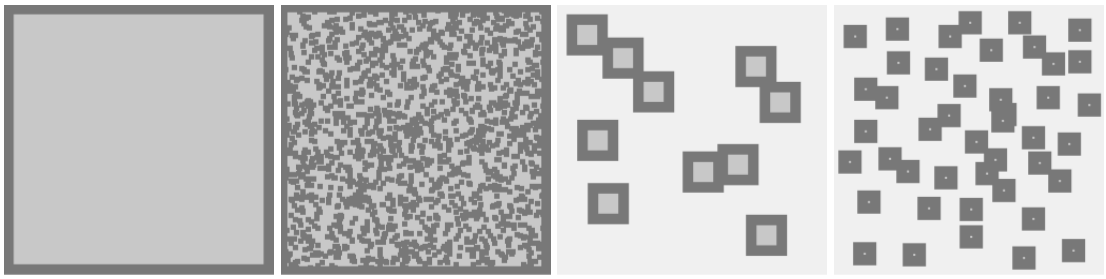
MCMC sampling is used in the original FRAME algorithm to bypass calculation of the intractable partition function $Z(\mathbf{I}_{\partial\Lambda},\beta)$. Variants of the FRAME algorithm exploit local structure of texture images and the linear structure of the FRAME potential $-\langle\beta,\mathbf{h}(\mathbf{I}_\Lambda|\mathbf{I}_{\partial\Lambda})\rangle$ to find alternate methods for approximating $Z(\mathbf{I}_{\partial\Lambda},\beta)$. This section introduces two principles for designing alternatives to the FRAME algorithm, and the next section presents four variant algorithms.

Design Principle I: The number, sizes, and shapes of foreground patches.

The first design principle is to break the full image lattice Λ into a set a smaller, possibly overlapping sub-lattices $\{\Lambda_i\}_{i=1}^M$. Size and shape of the sub-lattices can be arbitrary. The joint log-likelihood is then defined as the sum of patch log-likelihoods:

$$\mathcal{G}_{\text{patch}}(\beta)=\sum_{j=1}^{M}\log p(\mathbf{I}^{\text{obs}}_{\Lambda_i}|\mathbf{I}^{\text{obs}}_{\partial\Lambda_i};\beta). \tag{10.24}$$

Dividing the image lattice into smaller pieces can help increase sampling efficiency during training. Fixing background pixels can dramatically increase the mixing rate of MCMC samples without much loss in the accuracy of synthesized images.



a b c d

Fig. 10.1: Various choices of Λ_i, $i=1,2,...,M$. The lighter pixels are in foreground Λ_i are surrounded by darker background pixels in $\partial\Lambda_i$. The width of $\partial\Lambda_i$ depends on the size of the largest convolutional filter. Examples correspond to: a) likelihood (original FRAME Algorithm), b) partial likelihood (Algorithm I), c) patch likelihood (Algorithm II) or satellite likelihood (Algorithm III), and d) pseudolikelihood (Algorithm IV). © [2002] IEEE. Reprinted, with permission, from ref. [26].

Figure 10.1 displays four typical choices for $\{\Lambda_i\}_{i=1}^M$. The lighter pixels are in the *foreground* Λ_i, which are surrounded by darker pixels in the *background* $\partial\Lambda_i$. In cases (a), (c), and (d), Λ_i are square patches with $m\times m$ pixels. In one extreme, Figure 10.1(a) chooses one largest patch denoted by Λ_1, i.e. $M=1$ and $m=N-2w$ where w is the width of the boundary. In this situation $\mathcal{G}_{\text{patch}}$ corresponds to the standard *log-likelihood*, and it is adopted by the stochastic gradient MLE approach ([24, 27] [4–6]) discussed in Example 10.1. In the other extreme, Figure 10.1(d) chooses the minimum patch size $m=1$. Here $\mathcal{G}_{\text{patch}}$ is *log-pseudolikelihood* from Maximum Pseudo Likelihood Estimation (MPLE) [1]. In MPLE, the joint density of the image is assumed to be the product of the conditional density of each univariate dimension given the rest of the input, so each pixel defines an independent patch. Figure 10.1(c) is an example between the extremes (a) and (d) where $\mathcal{G}_{\text{patch}}$ is *log-patch-likelihood* or *log-satellite-likelihood*. In the fourth case,

Figure 10.1(b) chooses only one ($M = 1$) irregular-shaped patch Λ_1 that is set of randomly selected pixels. The rest of the pixels are in the background $\partial\Lambda_1$. In this case $\mathcal{G}_{\text{patch}}$ is called the *log-partial-likelihood*. In Figures 10.1(c) and (d), a foreground pixel may serve as background in different patches. It is straightforward to prove that maximizing $\mathcal{G}(\beta)$ leads to a consistent estimator for all four choices [7].

Design Principle II: The reference model used for estimating the partition functions.

Instead of sampling from the updated model in each step to obtain a stochastic gradient, one could instead use a fixed reference model and evaluate the expectation $\mathrm{E}_{p(I_\Lambda|I_{\partial\Lambda};\beta)}[\mathbf{h}(\mathbf{I}_\Lambda|\mathbf{I}_{\partial\Lambda})]$ using importance sampling. The linear structure of the FRAME potential facilitates this approach. Let β_o be a reference value of the FRAME parameter β, so $p(\mathbf{I}_{\Lambda_i}|\mathbf{I}_{\partial\Lambda_i};\beta_o)$ is the reference model. Suppose $\{\Lambda_i\}_{i=1}^M$ are the foreground patches and that $\mathbf{I}_{ij}^{\text{syn}}, j = 1,2,...,L$ are *typical* samples from the reference model $p(\mathbf{I}_{\Lambda_i}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}};\beta_o)$ for each patch Λ_i. Using the linear structure of the FRAME potential, we can approximate the partition function $Z(\mathbf{I}_{\partial\Lambda_i}^{\text{obs}},\beta)$ by Monte Carlo integration with a reference model β_o as follows:

$$\begin{aligned} Z(\mathbf{I}_{\partial\Lambda_i}^{\text{obs}},\beta) &= \int \exp\{-\langle\beta,\mathbf{h}(\mathbf{I}_{\Lambda_i}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}})\rangle\}d\mathbf{I}_{\Lambda_i} \\ &= Z(\mathbf{I}_{\partial\Lambda_i},\beta_o)\int \exp\{-\langle\beta-\beta_o,\mathbf{h}(\mathbf{I}_{\Lambda_i}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}})\rangle\}p(\mathbf{I}_{\Lambda_i}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}};\beta_o)d\mathbf{I}_{\Lambda_i} \\ &\approx \frac{Z(\mathbf{I}_{\partial\Lambda_i}^{\text{obs}},\beta_o)}{L}\sum_{j=1}^{L}\exp\{-\langle\beta-\beta_o,\mathbf{h}(\mathbf{I}_{ij}^{\text{syn}}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}})\rangle\}. \end{aligned} \tag{10.25}$$

We can use this to obtain an alternate approximation of the histogram expectation:

$$\begin{aligned} \mathrm{E}_{p(\mathbf{I}_{\Lambda_i}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}};\beta)}\left[\mathbf{h}(\mathbf{I}_{\Lambda_i}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}})\right] &= \mathrm{E}_{p(\mathbf{I}_{\Lambda_i}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}};\beta_o)}\left[\mathbf{h}(\mathbf{I}_{\Lambda_i}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}})\frac{p(\mathbf{I}_{\Lambda_i}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}};\beta)}{p(\mathbf{I}_{\Lambda_i}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}};\beta_o)}\right] \\ &= \mathrm{E}_{p(\mathbf{I}_{\Lambda_i}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}};\beta_o)}\left[\mathbf{h}(\mathbf{I}_{\Lambda_i}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}})\exp\{-\langle\beta-\beta_o,\mathbf{h}(\mathbf{I}_{\Lambda_i}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}})\rangle\}\frac{Z(\mathbf{I}_{\partial\Lambda_i}^{\text{obs}},\beta_o)}{Z(\mathbf{I}_{\partial\Lambda_i}^{\text{obs}},\beta)}\right] \\ &\approx \mathrm{E}_{p(\mathbf{I}_{\Lambda_i}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}};\beta_o)}\left[L\,\omega_{ij}\,\mathbf{h}(\mathbf{I}_{\Lambda_i}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}})\right] \\ &\approx \sum_{j=1}^{L}\omega_{ij}\,\mathbf{h}(\mathbf{I}_{ij}^{\text{syn}}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}}) \end{aligned}$$

where ω_{ij} is the weight for sample $\mathbf{I}_{ij}^{\text{syn}}$:

$$\omega_{ij} = \frac{\exp\{-\langle\beta-\beta_o,\mathbf{h}(\mathbf{I}_{ij}^{\text{syn}}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}})\rangle\}}{\sum_{j'=1}^{L}\exp\{-\langle\beta-\beta_o,\mathbf{h}(\mathbf{I}_{ij'}^{\text{syn}}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}})\rangle\}}.$$

Then optimizing (10.24) can be accomplished with the stochastic gradient

$$\widetilde{\nabla}\mathcal{G}_{\text{patch}}(\beta) = \sum_{i=1}^{M}\left\{\sum_{j=1}^{L}\omega_{ij}\mathbf{h}(\mathbf{I}_{ij}^{\text{syn}}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}}) - \mathbf{h}(\mathbf{I}_{\Lambda_i}^{\text{obs}}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}})\right\}. \tag{10.26}$$

The selection of the reference model $p(\mathbf{I}_{\Lambda_i}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}};\beta_o)$ depends on the sizes of the patches Λ_i. In general, importance sampling is only valid when the two distributions $p(\mathbf{I}_{\Lambda_i}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}};\beta_o)$ and $p(\mathbf{I}_{\Lambda_i}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}};\beta)$ heavily overlap. In one extreme case $m = 1$, the MPLE method [1] selects $\beta_o = 0$ and sets $p(\mathbf{I}_{\Lambda_i}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}};\beta_o)$ as a uniform distribution. In this case $Z(\mathbf{I}_{\partial\Lambda_i}^{\text{obs}},\beta)$ can be computed exactly. In the other extreme case for a large foreground $m = N - 2w$, the stochastic gradient MLE method has to choose $\beta_o = \beta$ in order to obtain sensible approximations. Thus, both methods must sample $p(\mathbf{I};\beta)$ iteratively starting from $\beta_0 = 0$.

10.2.3 Four variants of the FRAME algorithm

Algorithm I: Maximizing partial likelihood.

We choose a lattice shown in Figure 10.1.d by choosing at random a certain percentage (say 30%) of pixels as foreground Λ_1 and the rest are treated as background Λ/Λ_1. We define a *log-partial-likelihood*

$$\mathcal{G}_1(\beta) = \log p(\mathbf{I}^{\text{obs}}_{\Lambda_1} | \mathbf{I}^{\text{obs}}_{\Lambda/\Lambda_1}; \beta).$$

Maximizing $\mathcal{G}_1(\beta)$ by gradient descent, we update β iteratively:

$$\nabla \mathcal{G}_1(\beta) = \mathrm{E}_{p(\mathbf{I}_{\Lambda_1} | \mathbf{I}^{\text{obs}}_{\Lambda/\Lambda_1}; \beta)}[\mathbf{h}(\mathbf{I}_{\Lambda_1} | \mathbf{I}^{\text{obs}}_{\Lambda/\Lambda_1})] - \mathbf{h}(\mathbf{I}^{\text{obs}}_{\Lambda_1} | \mathbf{I}^{\text{obs}}_{\Lambda/\Lambda_1}) \approx \mathbf{h}(\mathbf{I}^{\text{syn}}_{\Lambda_1} | \mathbf{I}^{\text{obs}}_{\Lambda/\Lambda_1}) - \mathbf{h}(\mathbf{I}^{\text{obs}}_{\Lambda_1} | \mathbf{I}^{\text{obs}}_{\Lambda/\Lambda_1}), \tag{10.27}$$

where $\mathbf{I}^{\text{syn}}$ is an MCMC sample from $p(\mathbf{I}_{\Lambda_1} | \mathbf{I}^{\text{obs}}_{\Lambda/\Lambda_1}; \beta)$.

This algorithm follows the same procedure as the original method in FRAME [27]. It trades off between accuracy and speed in a better way than the original algorithm in FRAME [27]. The log-partial-likelihood has lower Fisher information than the log-likelihood, however our experiments demonstrate that it is about 25 times faster than the original minimax learning method without losing much accuracy. We observed that the reason for this speedup is that the original sampling method [27] spends a major portion of its time synthesizing $\mathbf{I}^{\text{syn}}_{\Lambda_1}$ under "non-typical" boundary conditions starting with white noise images. In contrast, the new algorithm works on typical boundary condition $\mathbf{I}^{\text{obs}}_{\Lambda/\Lambda_1}$ where the probability mass of the Gibbs model $p(\mathbf{I}; \beta)$ is focused on. The speed appears to be decided by the diameter of the foreground lattice measured by the maximum circle that can fit in the foreground lattice.

Algorithm II: Maximizing patch likelihood.

Algorithm II chooses a set of M overlapping patches from $\mathbf{I}^{\text{obs}}_{\Lambda}$ and "digs" a hole Λ_i on each patch as Figure 10.1.b shows. Thus we define a *patch log-likelihood*

$$\mathcal{G}_2(\beta) = \sum_{i=1}^{M} \log p(\mathbf{I}^{\text{obs}}_{\Lambda_i} | \mathbf{I}^{\text{obs}}_{\partial\Lambda_i}; \beta).$$

Maximizing $\mathcal{G}_2(\beta)$ by gradient descent, we update β iteratively as Algorithm I does:

$$\nabla \mathcal{G}_2(\beta) = \sum_{i=1}^{M} \mathrm{E}_{p(\mathbf{I}_{\Lambda_i} | \mathbf{I}^{\text{obs}}_{\Lambda/\Lambda_i}; \beta)}[\mathbf{h}(\mathbf{I}^{\text{syn}}_{\Lambda_i} | \mathbf{I}^{\text{obs}}_{\Lambda/\Lambda_i})] - \sum_{i=1}^{M} \mathbf{h}(\mathbf{I}^{\text{obs}}_{\Lambda_i} | \mathbf{I}^{\text{obs}}_{\Lambda/\Lambda_i}) \approx \sum_{i=1}^{M} \mathbf{h}(\mathbf{I}^{\text{syn}}_{\Lambda_i} | \mathbf{I}^{\text{obs}}_{\Lambda/\Lambda_i}) - \sum_{i=1}^{M} \mathbf{h}(\mathbf{I}^{\text{obs}}_{\Lambda_i} | \mathbf{I}^{\text{obs}}_{\Lambda/\Lambda_i}). \tag{10.28}$$

In comparison with algorithm I, the diameters of the lattices are evenly controlled. Algorithm II has similar performance compared to Algorithm I.

Algorithm III: Maximizing satellite likelihood.

Both Algorithms I and II still need to synthesize images for each parameter update, which is a computationally intensive task. Now we propose an third algorithm which can approximately compute β in a few seconds without synthesizing images online. We select a set of reference models $\mathscr{R}$ in the exponential family Ω to which the Gibbs model $p(\mathbf{I}; \beta)$ belongs:

$$\mathscr{R} = \{p(\mathbf{I}; \beta_j) \,:\, \beta_j \in \Omega, j = 1, 2, ..., s\}.$$

We use MCMC to sample (or synthesize) one large typical image $\mathbf{I}^{\text{syn}}_j \sim p(\mathbf{I}; \beta_j)$ for each reference model offline. These reference models estimate β from different "viewing angles" in Ω. By analogy to the Global Positioning System, we call the reference models the "satellites". The reference models are known systems that allow estimation of an arbitrary system, just like the known location of satellites can be used for the estimation of an arbitrary location.

The *log-satellite-likelihood*

$$\mathcal{G}_3(\beta) = \sum_{i=1}^{M} \log p(\mathbf{I}^{\text{obs}}_{\Lambda_i} | \mathbf{I}^{\text{obs}}_{\partial\Lambda_i}; \beta).$$

is the same as the log-patch-likelihood, but we will use importance sampling on the reference models $p(\mathbf{I}_{\Lambda_i} | \mathbf{I}^{\text{obs}}_{\partial\Lambda_i}; \beta_j)$ to evaluate the expectation $\mathrm{E}_{p(\mathbf{I}_{\Lambda_i} | \mathbf{I}^{\text{obs}}_{\partial\Lambda_i}; \beta)} \left[\mathbf{h}(\mathbf{I}_{\Lambda_i} | \mathbf{I}^{\text{obs}}_{\partial\Lambda_i}) \right]$. As discussed in Section 10.2.2, we can use the approximation:

$$\mathrm{E}_{p(\mathbf{I}_{\Lambda_i} | \mathbf{I}^{\text{obs}}_{\partial\Lambda_i}; \beta)} \left[\mathbf{h}(\mathbf{I}_{\Lambda_i} | \mathbf{I}^{\text{obs}}_{\partial\Lambda_i}) \right] \approx \sum_{\ell=1}^{L} \frac{\exp\{-\langle \beta - \beta_o, \mathbf{h}(\mathbf{I}^{\text{syn}}_{ij\ell} | \mathbf{I}^{\text{obs}}_{\partial\Lambda_i}) \rangle\}}{\sum_{\ell'=1}^{L} \exp\{-\langle \beta - \beta_o, \mathbf{h}(\mathbf{I}^{\text{syn}}_{ij\ell'} | \mathbf{I}^{\text{obs}}_{\partial\Lambda_i}) \rangle\}} \mathbf{h}(\mathbf{I}^{\text{syn}}_{ij\ell} | \mathbf{I}^{\text{obs}}_{\partial\Lambda_i})$$

where $\mathbf{I}^{\text{syn}}_{ij\ell}$ is an MCMC sample from $p(\mathbf{I}_{\Lambda_i} | \mathbf{I}^{\text{obs}}_{\partial\Lambda_i}; \beta_j)$ and ω_{ij} is the weight for sample $\mathbf{I}^{\text{syn}}_{ij\ell}$:

$$\omega_{ij\ell} = \frac{\exp\{-\langle \beta - \beta_j, \mathbf{h}(\mathbf{I}^{\text{syn}}_{ij\ell} | \mathbf{I}^{\text{obs}}_{\partial\Lambda_i}) \rangle\}}{\sum_{\ell'=1}^{L} \exp\{-\langle \beta - \beta_j, \mathbf{h}(\mathbf{I}^{\text{syn}}_{ij\ell'} | \mathbf{I}^{\text{obs}}_{\partial\Lambda_i}) \rangle\}}.$$

Following the approach in Section 10.2.2, we maximize $\mathcal{G}_3(\beta)$ with the stochastic gradient

$$\widetilde{\nabla} \mathcal{G}_3(\beta) = \sum_{j=1}^{s} \left\{ \sum_{i=1}^{M} \left[\sum_{\ell=1}^{L} \omega_{ij} \mathbf{h}(\mathbf{I}^{\text{syn}}_{ij\ell} | \mathbf{I}^{\text{obs}}_{\partial\Lambda_i}) \; - \; \mathbf{h}(\mathbf{I}^{\text{obs}}_{\Lambda_i} | \mathbf{I}^{\text{obs}}_{\partial\Lambda_i}) \right] \right\}. \tag{10.29}$$

For every foreground patch Λ_i and for every reference model $p(\mathbf{I}; \beta_j)$, we need to generate a set of L synthesized patches $\mathscr{I}^{\text{syn}}_{ij} = \{\mathbf{I}^{\text{syn}}_{ij\ell}; \ell = 1, 2, ..., L, \forall i, j\}$ to fill Λ_i for the calculation of histogram frequencies. There are two ways to generate $\mathscr{I}^{\text{syn}}_{ij}$:

1. Sampling from the conditional distribution $\mathbf{I}^{\text{syn}}_{ij\ell} \sim p(\mathbf{I}_{\Lambda_i} | \mathbf{I}^{\text{obs}}_{\partial\Lambda_i}; \beta_j)$. This is expensive and has to be computed online.
2. Sampling from the marginal distribution $\mathbf{I}^{\text{syn}}_{ij\ell} \sim p(\mathbf{I}_{\Lambda_i}; \beta_j)$. In practice, this is just to fill the holes with randomly selected patches from the synthesized image $\mathbf{I}^{\text{syn}}_{j}$ computed offline.

In our experiments, we tried both cases and we found that differences are very little for middle-sized $m \times m$ lattices, say $4 \leq m \leq 13$. The second case leads to a useful training algorithm, because it allows us to bypass MCMC sampling while learning β. Equation (10.29) converges in a matter of seconds for an average texture model when pre-synthesized images are for parameter updates.

However, we should be aware of the risk that the log-satellite-likelihood $\mathcal{G}_3(\beta)$ may not be upper bounded. This case occurs when $\mathbf{h}(\mathbf{I}^{\text{obs}}_{\Lambda_i} | \mathbf{I}^{\text{obs}}_{\partial\Lambda_i})$ cannot be described by a linear combination of the statistics of the sampled patches $\sum_{\ell=1}^{L} \omega_{ij} \mathbf{h}(\mathbf{I}^{\text{syn}}_{ij\ell} | \mathbf{I}^{\text{obs}}_{\partial\Lambda_i})$. When this happens, β does not converge. We can handle this problem by including the observed patch $\mathbf{I}^{\text{obs}}_{\Lambda_i}$ in H^{syn}_{ij}, which ensures that the satellite likelihood is always upper bounded. Intuitively, let $\mathbf{I}^{\text{syn}}_{ij1} = \mathbf{I}^{\text{obs}}_{\Lambda_i}$. Then β is learned so $\omega_{ij1} \to 1$ and $\omega_{ij\ell} \to 0$, $\forall \ell \neq 1$. Since L is often relatively large, say $L = 64$, adding one extra sample will not contaminate the sample set.

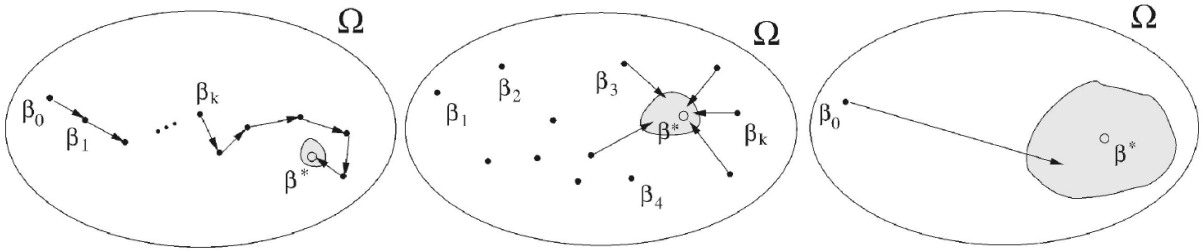
Algorithm IV: Maximizing pseudolikelihood.

Maximum Pseudo Likelihood Estimation (MPLE) assumes that the target density $p(X; \theta)$ can be factored as

$$p(X; \theta) = \prod_{i=1}^{N} p(X_i | X_{\partial\{i\}}; \theta)$$

where $\partial\{i\}$ is the border of the singleton $\{i\}$. Thus the FRAME log-pseudolikelihood has the form

$$\mathcal{G}_4(\beta) = \sum_{(x,y) \in \Lambda} \log p(\mathbf{I}^{\text{obs}}_{(x,y)} | \mathbf{I}^{\text{obs}}_{\partial\{(x,y)\}}; \beta)$$



Group 1 Group 2 Group 3

Fig. 10.2: The shadow areas around β^* illustrate the variance of the estimated β or the efficiency of the log-likelihood functions. *Group 1:* The original FRAME algorithm and Algorithms I and II generate a sequence of satellites online to approach β closely. The patch size can be small or large. *Group 2:* The maximum satellite likelihood estimator uses a general set of satellites computed off-line, and can be updated incrementally. This can be used for small patches. *Group 3:* MPLE uses a single satellite: $\beta_o = 0$. © [2002] IEEE. Reprinted, with permission, from ref. [26].

for a foreground lattice Λ. In other words, the patches are each a single pixel $\Lambda_i = (x_i, y_i)$.

Even after the MPLE factorization, the term $\log p\left(\mathbf{I}^{\text{obs}}_{(x,y)} | \mathbf{I}^{\text{obs}}_{\partial\{(x,y)\}}; \beta\right)$ is difficult to evaluate for an arbitrary β, and a reference distribution is required as in Algorithm III. A simple choice for MRF models is the trivial reference model $\beta_o = 0$. Clearly, the density $p_{\beta_o}(I)$ of the reference model is uniform over the image space, so generating samples of the reference distribution is trivial. One can use the gradient (10.29) with $s = 1$ and $\beta_1 = 0$ to maximize the pseudolikelihood $\mathcal{G}_4(\beta)$. MPLE bypasses the burden of online image synthesis by using a trivial reference distribution, but the computational gains often come at the cost of reduced realism in the sampled images.

To summarize, we compare different algorithms for estimating $\beta^* \in \Omega$ for a MRF model, and divide them into three groups. Figure 10.2 illustrates the comparison. The ellipse stands for the space Ω and each Gibbs model is represented by a single point.

Group 1 represents the maximum likelihood estimator (original FRAME algorithm) and the maximum partial/patch likelihood estimators. As Figure 10.2(a) illustrates, the Group 1 methods generate and sample a sequence of "satellites" $\beta_0, \beta_1, ..., \beta_k$ online. These satellites get closer and closer to β^* (the supposed truth). The shadow area around β^* represents the uncertainty in computing β, whose size can be measured by the Fisher information.

Group 2 is the maximum satellite likelihood estimator. This estimator uses a general set of satellites which are pre-computed and sampled offline, as shown in Figure 10.2(b). To save time, one may select a small subset of satellites for computing a given model. One can choose satellites based on the differences $\mathbf{h}(\mathbf{I}^{\text{syn}}_j)$ and $\mathbf{h}(\mathbf{I}^{\text{obs}})$. The smaller the differences are, the closer the satellite is to the estimated model, and the better the approximation. The satellites should be distributed evenly around β^* to obtain a good estimation. Group 3 is the maximum pseudolikelihood method, which is a special case of Group 2. As Figure 10.2(c) shows, the pseudolikelihood uses a uniform model $\beta_o = 0$ as a "satellite" to estimate any model, and thus has large variance.

10.2.4 Experiments

In this section, we evaluate the performance of various algorithms in the context of learning FRAME models for textures. We use 12 filters including an intensity filter, two gradient filters, three Laplacian of Gaussian filters and six Gabor filters at a fixed scale and different orientations. Thus $\mathbf{h}(\mathbf{I})$ includes 12 histograms of filter responses and each histogram has 12 bins, so β has 12×11 free parameters. We choose 15 natural texture images. The stochastic gradient MLE estimates β^*_j from the original FRAME algorithm (see Section 10.2.1 and [27]) are treated as ground truth for comparison. After applying the FRAME algorithm to each texture, we obtain 15 synthesized images $\mathbf{I}^{\text{syn}}_j$ that can be used as offline satellite images in Algorithm III.

Experiment I: Comparison of five algorithms.

In the first experiment, we compare the performance of the five algorithms in texture synthesis. Figure 10.3 displays 3 texture patterns of 128×128 pixels. For each row, the first column is the "ground-truth" synthesized image from the stochastic gradient MLE method used in the original FRAME model. The other four images are synthesized using the methods in Section 10.2.3. For Algorithms I to III, we fixed the total number of foreground pixels to $5,000$. The patch size is fixed to 5×5 pixels for patch likelihoods and satellite likelihoods. We select 5 satellites out of 14 available pre-computed models for each texture.

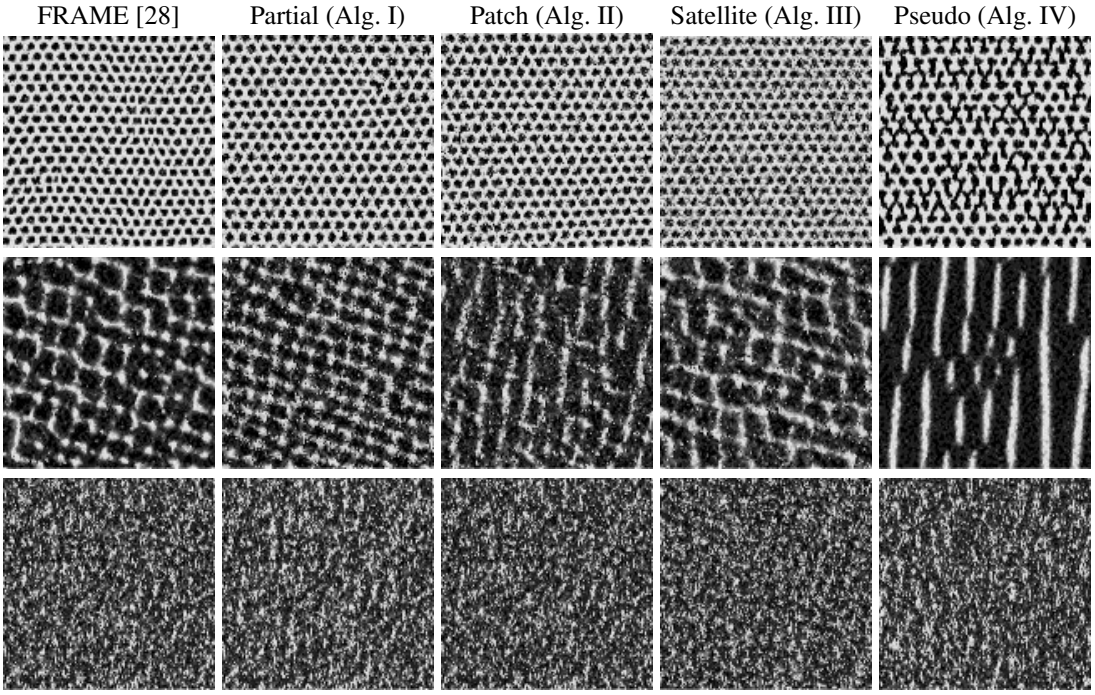


Fig. 10.3: Synthesized texture images using β learned from various algorithms. For each column from left to right. 1. Stochastic gradient algorithm using full likelihood as the ground truth, 2. pseudolikelihood, 3. satellite likelihood, 4. patch likelihood, 5. partial likelihood. © [2002] IEEE. Reprinted, with permission, from ref. [26].

For different textures, the model $p(\mathbf{I};\beta)$ may be more sensitive to some elements of β (such as tail bins) than to the other parameters, and the β vectors are highly correlated between components. Therefore, it is not very meaningful to compare the accuracy of the learned β using an error measure $|\beta - \beta^*|$. Instead, we sample each learned model $\mathbf{I}^{\text{syn}} \sim p(\mathbf{I};\beta)$ and compare the histogram errors of the synthesized image against the observed. The loss measure is $||\mathbf{h}(\mathbf{I}^{\text{syn}}) - \mathbf{h}(\mathbf{I}^{\text{obs}})||_1$, a summation over 12 pairs of normalized histograms. The table below shows the errors for each algorithms for the synthesized images in Figure 10.3. The numbers are subject to some computational fluctuations from the sampling process.

Fig.4.	FRAME	Pseudo	Satellite	Patch	Partial
Fig.4.a	0.449	2.078	1.704	1.219	1.559
Fig.4.b	0.639	2.555	1.075	1.470	1.790
Fig.4.c	0.225	0.283	0.325	0.291	0.378

The experimental results show that the four algorithms work reasonably well. The satellite method is often close to the patch and partial likelihood methods, though it sometimes may yield slightly better results than other methods depending on the similarity between the reference models and the model to be learned. The pseudolikelihood method can also capture some large image features. In particular, it works well for textures of stochastic nature, such as the texture in Figure 10.3(c).

In terms of computational complexity, the satellite algorithm is the fastest, and it computes β in the order of 10 seconds on a HP-workstation. The second fastest is the pseudolikelihood, which takes a few minutes. However, the pseudolikelihood method consumes a large memory, as it needs to remember all the k histograms for the g grey levels in $N \times N$ pixels. The space complexity is $O(N^2 \times g \times k \times B)$ with B the number of bins, and more than one Gigabyte of memory is often needed. The partial likelihood and patch likelihood algorithms are very similar to the stochastic gradient algorithm with the full MLE. Since the initial boundary condition is typical, these two estimators take only in general $1/10^{\text{th}}$ of the number of sweeps to convergence. In addition, only a portion the pixel lattice needs to be synthesized, which can further save computation. The synthesis times of the patch and partial likelihood algorithms are only about $1/20^{\text{th}}$ of the full likelihood algorithm.

Experiment II: Analysis of the maximum satellite likelihood estimator.

In the second experiment, we study how the performance of the satellite algorithm is influenced by 1) the boundary condition, and 2) the patch size $m \times m$.

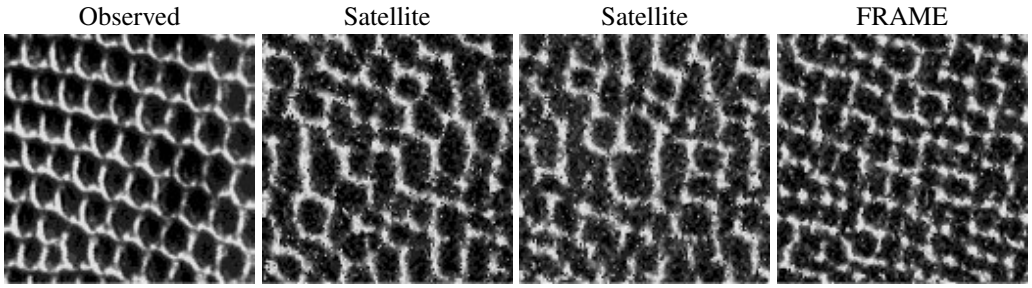


Fig. 10.4: Performance evaluation of the satellite algorithm. (a) Observed texture image. (b) Synthesized image using β learned without boundary conditions. (c) Synthesized image using β learned with boundary conditions. (d) Synthesized image using β learned by stochastic gradient. © [2002] IEEE. Reprinted, with permission, from ref. [26].

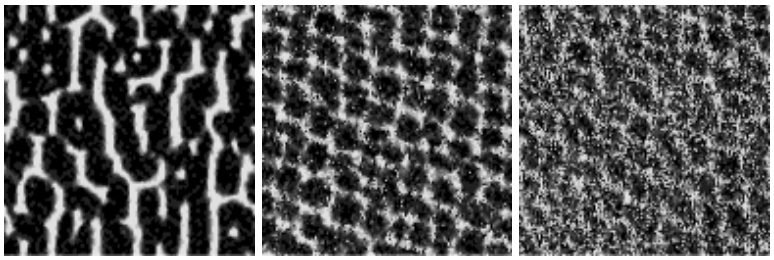
1) Influence of boundary conditions.

Figure 10.4.a displays a texture image as $\mathbf{I}^{\text{obs}}$. We run three algorithms for comparison. Figure 10.4(d) is a result from the FRAME (stochastic gradient method). Figure 10.4(b) and (c) are results using two version of the satellite algorithm: online and offline. As described in Section 10.2.3 Algorithm III, the algorithm that generated Figure 10.4(c) uses the observed boundary condition for each patch and does online sampling, while the algorithm that generated Figure 10.4(b) ignores the boundary condition and synthesizes images offline. The results are very similar, which is evidence that the loss in accuracy for the offline version of Algorithm III is negligible compared to the computational gains of offline sampling.

2) Influences of the patch size $m \times m$.

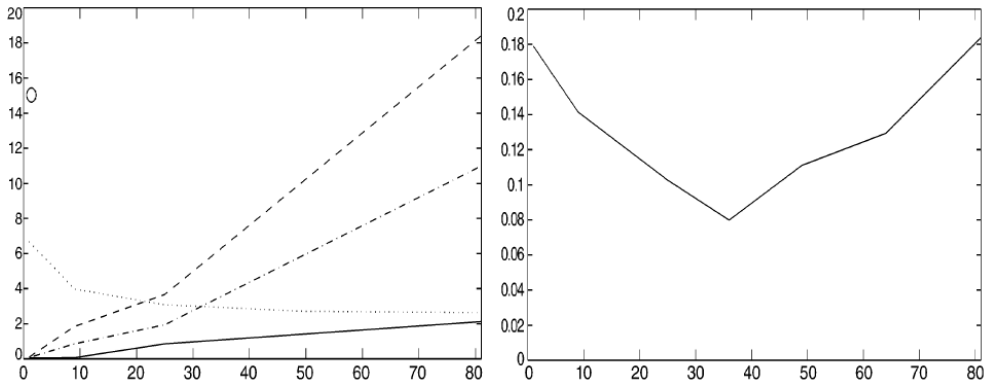
We fix the total number of foreground pixels $\sum_i |\Lambda_i|$ and study the performance of the satellite algorithm with different patch sizes m. Figures 10.5(a)-(c) show three synthesized images using β learned by the satellite algorithm with different patch sizes $m = 2, 6, 9$ respectively. It is clear from Figures 10.5(a)-(c) that the 6×6 pixel patch size gives better results.

To explain why the patch size of $m = 6$ gives better satellite approximation, we compute the two key factors that determine performance, as displayed in Figure 10.6(a). When the patch size is small, the partition function can be estimated accurately, as shown by the small $E_p[(\hat{Z} - Z)^2]$ in solid, dash-dotted, and dashed curves in Figure 10.6. However, the variance $E_f[(\hat{\beta} - \beta^*)^2]$ is large for small patches, which is shown by the dotted curve in figure 10.6(a). The optimal choice of the patch size is thus approximately the intersection of the two curves. Since the reference models that we used are close to the dash-dotted line shown in figure 10.6(a), we predict optimal patch size is between 5×5 and 6×6. Figure 10.6(b) shows the average error between the statistics of synthesized image $\mathbf{I}^{\text{syn}} \sim p(\mathbf{I}; \beta)$ and



a b c

Fig. 10.5: Synthesized images using β learned by the satellite method with different patch sizes. (a) $m = 2$. (b) $m = 6$. (c) $m = 9$. © [2002] IEEE. Reprinted, with permission, from ref. [26].



a b

Fig. 10.6: The x-axes are the patch size m^2. a). Dotted curve is $E_f[(\hat{\beta} - \beta^*)^2]$ plotted against the patch size m^2. The solid, dash-dotted, and dashed curves are $E_p[(\hat{Z} - Z)^2]$ for three different reference models. b). Average synthesis error per filter with respect to the patch size m^2. © [2002] IEEE. Reprinted, with permission, from ref. [26].

the observed statistics $\text{err} = \frac{1}{12}|\mathbf{h}(\mathbf{I}^{\text{obs}}) - \mathbf{h}(\mathbf{I}^{\text{syn}})|$, where β is learned using the satellite method for $m = 1, 2, ..., 9$. Here the patch size of 6×6 pixels gives the best result.

10.3 Learning Image Models with Neural Networks

This section covers stochastic gradient methods for learning the weights of deep network models of image data. The common thread among all methods presented is the use of MCMC to evaluate intractable gradients needed for maximum likelihood estimation. Since deep network functions are almost always differentiable, Langevin Dynamics (see Section 9.4.3) is often the preferred method of generating MCMC samples. In the first part of the section, we introduce Contrastive Divergence and Persistent Contrastive Divergence, two important techniques for accelerating the convergence of Monte Carlo samples from sequentially updated models. We then cover learning techniques for potential energy image models, image generator networks, and cooperative systems of energy and generator models.

10.3.1 Contrastive divergence and persistent contrastive divergence

Stochastic gradient MLE methods (see Example 10.1) use a sample $\{Y_i\}_{i=1}^m$ from the current distribution $p_\theta(x)$ to update θ with the log-likelihood gradient in Equation (10.8). Since $p_\theta(x)$ changes every time that θ is updated, new MCMC samples are needed for each new gradient calculation. Generating these samples can be very time-consuming.

MCMC samples often exhibit high autocorrelation and have long mixing times, and in most cases it is infeasible to generate truly independent MCMC samples from $p_\theta(x)$ for every gradient update. On the other hand, MCMC samples initialized close to the target distribution can converge quickly, so a sensible initialization scheme for stochastic gradient learning can lead to dramatic computational savings.

Contrastive Divergence (CD) [11] and Persistent Contrastive Divergence (PCD) [21] are two popular methods for initializing MCMC samples. Because they use a "warm-start" initialization, CD and PCD need only a few MCMC iterations to reach approximate convergence for each gradient calculation. Even before CD and PCD were formally introduced into the machine learning literature, stochastic gradient methods such as the FRAME algorithm [27] had already used PCD without giving the technique an explicit name.

In CD learning, the observed data $\{X_i\}_{i=1}^n$ (or a mini-batch of the observations) are used as the initial points $\{Y_i^{(0)}\}_{i=1}^n$ of the negative MCMC samples. In PCD learning, the final negative samples $\{\tilde{Y}_i\}_{i=1}^m$ of the *previous* parameter update are used as the initial points $\{Y_i^{(0)}\}_{i=1}^m$ for the current parameter update. Both CD and PCD use only a small number of MCMC updates k (even $k=1$) on the initial points to obtain a sample $\{Y_i^{(k)}\}_{i=1}^m$ for gradient computation in Equation (10.8). From a theoretical standpoint, a very large number of Markov transitions should be used to obtain a reliable steady-state sample $\{Y_i^{(\infty)}\}_{i=1}^m$. From a computational standpoint, a small number of transitions from a meaningful initialization can still provide accurate gradient information for parameter estimation. See Figure 10.7 for a visual comparison of full stochastic gradient MLE, CD, and PCD.

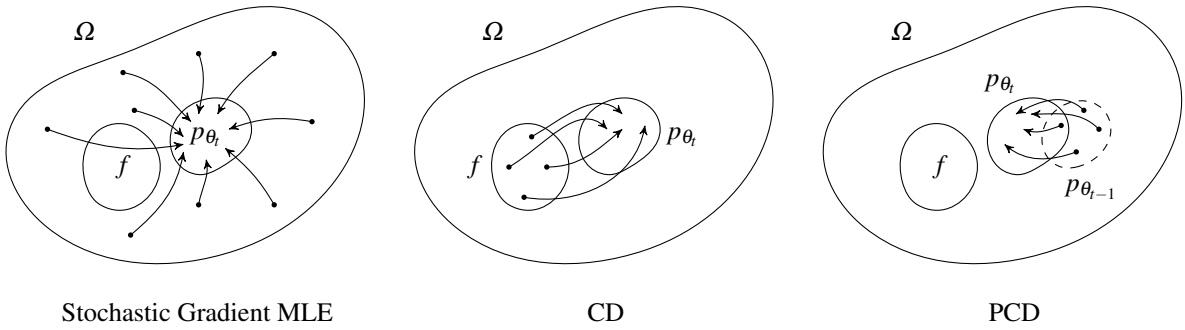
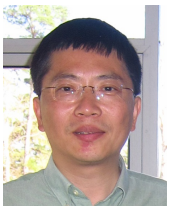


Fig. 10.7: In all diagrams, Ω is the state space of the data x, p_{θ_t} is the learned distribution at training step t, f is the true data distribution, and dots are the initial points for MCMC samples. *Left:* Full Stochastic Gradient MLE. MCMC samples are initialized from random points in Ω and many Markov updates are used to obtain a reliable steady state sample. *Middle:* Approximate MLE with CD. MCMC samples are initialized from the training data, and only a few updates are needed for convergence when p_{θ_t} is close to the true distribution f. *Right:* Approximate MLE with PCD. MCMC samples are initialized from samples of $p_{\theta_{t-1}}$ from the previous learning iteration. If the difference between θ_t and θ_{t-1} is small, the MCMC samples should converge after only a few Markov updates.

Since the initial images for CD come from the true distribution, the use of only a small number of MCMC updates should be justified near the end of training, when the learned distribution is close to the true distribution. On the other hand, the PCD initial images come from the distribution of the previous learned model. As long as the parameter updates are small, the previous model should be close to the current model, and the PCD updating assumption should be justified throughout the entire training process. PCD appears to possess superior properties to CD in some cases, since the initial samples are closer to the current model and because the initial samples vary throughout the training process. Use of either CD or PCD is practically unavoidable in stochastic gradient learning, but both methods introduce additional errors into the training process that are difficult to analyze.

10.3.2 Learning a potential energy for images with deep networks: DeepFRAME


Ying Nian Wu

The DeepFRAME model (see Section 9.6.3 Example 9.2 and [17, 22]) extends the FRAME model to incorporate insights from deep learning. There are two key differences between the models. First, the DeepFRAME potential is a non-linear composition of linear layers and activation functions. Deep network potentials are significantly more expressive than the linear potentials used in the original FRAME. Second, the filters of the DeepFRAME model are learned during training, in contrast to the fixed hand-selected filters used in the FRAME model. Learning the filters themselves during training is essential for flexible image representation, because it is very difficult to hand-design filters that capture relevant features of complex images.

A DeepFRAME density with weight parameter w has the form

$$p_w(I) = \frac{1}{Z(w)} \exp\{F(I;w)\} q(I), \tag{10.30}$$

where q is a Gaussian prior distribution $\mathrm{N}(0, \sigma^2 \mathrm{Id})$. The DeepFRAME potential energy function is

$$U(I;w) = -F(I;w) + \frac{||I||^2}{2\sigma^2}. \tag{10.31}$$

Learning the weights w of a DeepFRAME density can be accomplished by following the stochastic gradient MLE approach outlined in Example 10.1. In fact, learning a DeepFRAME model is nearly identical to learning a FRAME model. Both methods are typically trained using PCD, which means that MCMC samples from the previous parameter update are used as the initialization for MCMC samples for the current parameter update. The original FRAME model uses Gibbs sampling on discrete images for the MCMC updates. When training a DeepFRAME model, Langevin Dynamics is used to update the MCMC image samples in a continuous space. The image updates have the form

$$I_{t+1} = I_t + \frac{\varepsilon^2}{2}\left(\frac{\partial}{\partial I} F(I;w) - \frac{I}{\sigma^2}\right) + \varepsilon Z_t \tag{10.32}$$

where $Z_t \sim \mathrm{N}(0, \tau^2 \mathrm{Id})$ is the momentum. The weights w can be updated using Equation (10.8):

$$\widetilde{\nabla} l(w) = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial}{\partial w} F(X_i;w) - \frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial w} F(Y_i;w) \tag{10.33}$$

where $\{X_i\}_{i=1}^n$ is the set of training images and $\{Y_i\}_{i=1}^m$ are negative samples of $p_w(I)$. An outline of the DeepFRAME learning algorithm is given below.

Algorithm 14 DeepFRAME Algorithm

Input: Observed images $\{X_i\}_{i=1}^n$, number of Langevin Iterations S, step size $\delta > 0$, number of learning iterations T.

Output: MLE w^*, synthesized images $\{Y_i\}_{i=1}^m$.

Initialize weight w_0 and negative samples $\{Y_i\}_{i=1}^m$ as white noise.

for $t = 1 : T$ **do**

a. Using Equation (10.32), apply S Langevin updates to the persistent negative samples $\{Y_i\}_{i=1}^m$ under the current model $p_{w_{t-1}}(I)$.

b. Update w according to

$$w_t = w_{t-1} + \delta \widetilde{\nabla} l(w_{t-1})$$

where $\widetilde{\nabla} l(w)$ is the stochastic gradient in (10.33).

end for

Fig. 10.8: DeepFRAME Experiment 1. In each group of three, the image on the left is the training texture and the middle and right images are synthesized from a DeepFRAME potential.

Four experiments that show the capabilities of the DeepFRAME model are presented below.

DeepFRAME Experiment 1: Synthesizing Texture Images (Figure 10.8)

In the first experiment, new texture images are synthesized from a single training texture image. Filters in the lower levels of the network learn to synthesize local texture features, while higher level filters determine the composition of local features observed in the training texture. The non-linear structure of the DeepFRAME potential improves the synthesis capabilities of the original FRAME model.

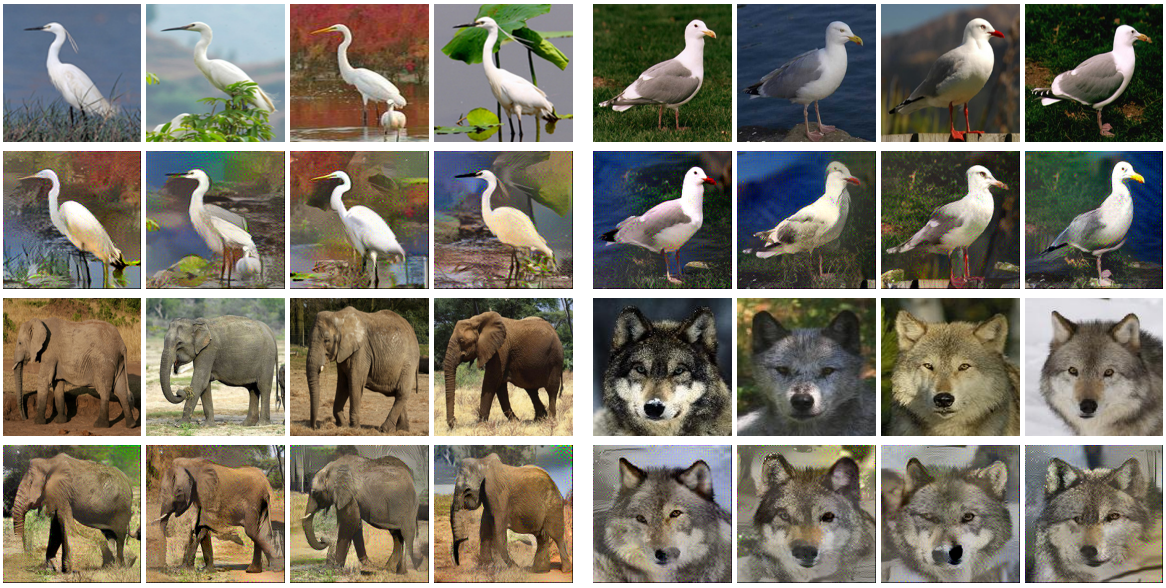


Fig. 10.9: DeepFRAME Experiment 2. In each image group, the top row shows training images and the bottom row shows synthesized images.

DeepFRAME Experiment 2: Synthesizing Object Images (Figure 10.9)

The DeepFRAME model can learn to synthesize object images as long as the training images have a consistent alignment. The final layer of the ConvNet scoring function $F(I;w)$ should be fully connected to enforce geometric consistency in the energy function. The synthesized object images are consistent with the appearance and alignment of the training data.

DeepFRAME Experiment 3: Synthesizing Hybrid Images (Figure 10.10)

When aligned images from different categories are used as the training data, the DeepFRAME model learns to synthesize hybrid images that combine different local features of the training categories. While the overall image shape is consistent with the alignment of the training data, novel feature combinations unseen in the training images appear in the synthesized images.

Fig. 10.10: DeepFRAME Experiment 3. In each image group, the top row shows training images and the bottom row shows synthesized images. © [2016] AAAI. Reprinted, with permission, from ref. [17].

DeepFRAME Experiment 4: Image Reconstruction (Figure 10.11)

Since the energy function of a DeepFRAME function is an approximation of the true image density, unseen images from the true distribution should be close to local modes of the learned density. A potential energy over the image space can be interpreted as *memory*, as originally described by Hopfield [12]. The evolution of an image under a MCMC process on a DeepFRAME potential will be attracted towards the nearest local mode. If the initial image resembles the training data, the MCMC process should converge to a mode that looks similar to the initial image. On the other hand, an MCMC sample initialized from an image that differs from the training data will change greatly during sample until it resembles a training image.

Fig. 10.11: DeepFRAME Experiment 4. The top row are the real images that are not used during training, and the bottom row is the reconstructed image after a single Langevin update. © [2016] Xie et al. Reprinted, with permission, from ref. [22].

10.3.3 Generator networks and alternating backward propagation

Potential energy functions such as DeepFRAME are one of several families of image models. *Generator networks* are another important type of image model. These networks generalize the classical factor analysis model and can be used to generate realistic images from a simple signal. In contrast to potential energy models, which need MCMC for image synthesis, generator networks can synthesize images directly from a latent input.

Let $Z \in \mathbb{R}^d$ be latent factor with a trivial distribution (e.g. uniform or $\mathrm{N}(0, I_d)$). In practice, the dimension d of Z is typically 100 or less. A generator network $G(Z;w)$ with weights w defines a transformation from the latent space of Z to the high-dimensional image space $\mathbb{R}^D$. One can learn the weights w of a generator network so that the synthesized images $G(Z;w) \in \mathbb{R}^D$ match the appearance of a training dataset across the latent distribution of Z.

The classical factor analysis model uses a linear generator $G(Z;w) = WZ + \varepsilon$. A deep generator network introduces non-linearities in G by including an activation function between linear layers. A generator network can be understood as a recursive factor analysis model with the relation

$$Z_{l-1} = f_l(W_l Z_l + b_l) \tag{10.34}$$

where f_l is a non-linear activation (usually ReLU), (W_l, b_l) are the weights and bias from layer l, $w = \{(W_l, b_l) : l = 1, \ldots, L\}$, $Z_L = Z$, and $Z_0 = G(Z;w)$. The hidden factor layer Z_{l-1} is a linear combination of the columns of W_l with coefficients Z_l, plus a shift and activation. If the ReLu activation is used, then $G(Z;w)$ is a piecewise linear function, and the boundaries between linear regions correspond to activation boundaries of f_l. The non-linear structure of G is essential for generating realistic images.

The Alternating Backward Propagation (ABP) Algorithm [10] is one method for learning the weights w of a generator network. As the name suggests, the ABP Algorithm has two phases. In the first phase, the latent factors of the set of training images are inferred using Langevin dynamics. In the second phase, the weights that transform the latent factors into images are updated based on the new latent factors. Since the latent factors are inferred during the training process, the ABP algorithm performs unsupervised learning. The ABP algorithm is closely related to the EM algorithm: the first phase corresponds to the E step of the EM algorithm, where expected values are evaluated based on the current parameters, and the second phase corresponds to the M step, where the parameters are adjusted to explain the expected factors.

In ABP learning, the conditional distribution of a D-dimensional image X given a d-dimensional latent factor Z is defined as

$$X|Z \sim \mathrm{N}(G(Z;w), \sigma^2 \mathrm{Id}_D)$$

for a generator network $G(Z;w)$ with weights w. Since Z and $X|Z$ are both multivariate normal, their joint energy function has the form

$$U(X,Z;W) = \frac{1}{2\sigma^2}||X - G(Z;w)||^2 + \frac{1}{2}||Z||^2$$

which is simply the sum of the Gaussian energy functions of Z and $X|Z$. The energy function of the conditional variable $Z|X$ is $U_{Z|X=x;w}(z) = U(z,x;w)$, since the posterior distribution $Z|X$ is proportional to the joint distribution of X and Z. Given a set of complete observations $\{X_i, Z_i\}_{i=1}^n$, one could estimate the weights w by maximizing the log-likelihood

$$l(w, \{Z_i\}) = \frac{1}{n}\sum_{i=1}^{n} \log p(X_i, Z_i; w) = -\frac{1}{n}\sum_{i=1}^{n} U(X_i, Z_i) + \text{constant}$$

through gradient ascent. Since the normalizing constant does not depend on w, stochastic gradients are not needed for the full data log-likelihood. However, as in the EM algorithm, the latent factors $\{Z\}_{i=1}^n$ are unknown, and w must be learned by maximizing the observed data log-likelihood, which corresponds to maximizing the function

$$l(w) = \sum_{i=1}^{n} \log p(X_i; w) = \sum_{i=1}^{n} \log \int p(X_i, Z; w) dZ$$

that integrates the latent factors out of the joint distribution. This loss cannot be computed directly, but the gradient of the loglikelihood can be rewritten as

$$\begin{aligned}
\frac{\partial}{\partial W}\log p(X;w) &= \frac{1}{p(X;w)}\frac{\partial}{\partial w}p(X;w) \\
&= \frac{1}{p(X;w)}\frac{\partial}{\partial w}\int p(X,Z;w)dZ \\
&= \int\left(\frac{1}{p(X;w)}\frac{\partial}{\partial w}p(X;w)\right)p(Z|X;w)dZ \\
&= \int\left(\frac{\partial}{\partial w}\log p(X;w)\right)p(Z|X;w)dZ \\
&= -\mathrm{E}_{Z|X;w}\left[\frac{\partial}{\partial w}U(X,Z;w)\right],
\end{aligned}$$

so the log-likelihood gradient can be estimated by drawing MCMC samples of $Z|X$, the latent factors conditioned on the observed data, using the current weight w. Langevin Dynamics can be used to sample from $Z|X_i;w$, and the Langevin update equation is

$$Z_{t+1} = Z_t + \frac{\varepsilon^2}{2}\left(\frac{1}{\sigma^2}(X_i - G(Z_t;w))\frac{\partial}{\partial Z}G(Z_t;w) - Z_t\right) + \varepsilon U_t \tag{10.35}$$

for $U_t \sim \mathrm{N}(0,I_d)$ and step size ε, for $t = 1,\ldots,T$ iterations. One Z_i is inferred for each observed image X_i. PCD is used during training, so MCMC sampling in each new inference phase is started from the Z_i of the previous inference phase. Once the Z_i have been sampled from $Z|X_i;w$, the weights w can be updated with

$$\widetilde{\nabla} l(w) = -\frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial w}U(X_i,Z_i;w) = -\frac{1}{n}\sum_{i=1}^{n}\frac{1}{\sigma^2}(X_i - G(Z_i;w))\frac{\partial}{\partial w}G(Z_i;w) \tag{10.36}$$

in the second phase of the algorithm. The inference phase uses a back-propagation gradient $\frac{\partial}{\partial Z}G(Z;w)$, while the learning phase uses a back-propagation gradient $\frac{\partial}{\partial w}G(Z;w)$. The calculations required to obtain $\frac{\partial}{\partial Z}G(Z;w)$ are needed as part of the calculation of $\frac{\partial}{\partial w}G(Z;w)$, so both phases can be implemented in a similar way. An algorithm sketch is given below.

Algorithm 15 Alternating Backward Propagation Algorithm

Input: Observed images $\{X_i\}_{i=1}^n$, number of Langevin Iterations S, step size $\delta > 0$, number of learning steps T.
Output: MLE w^* of generator network $G(Z;w)$.

Initialize weights w_0 as white noise and sample inferred latent factors $\{Z_i\}_{i=1}^m$ from the latent distribution.
for $t = 1 : T$ **do**
a. Using Equation (10.35), apply S Langevin updates to the persistent latent factors $\{Z_i\}_{i=1}^n$ using the current weights w_{t-1}.
b. Update w according to

$$w_t = w_{t-1} + \delta\widetilde{\nabla} l(w_{t-1})$$

where $\widetilde{\nabla} l(w)$ is the stochastic gradient in (10.36).
end for

Three different experiments are presented to show the capabilities of the ABP algorithm.

ABP Experiment 1: Generating Texture Patterns (Figure 10.12)

Let the input Z be a $\sqrt{d}\times\sqrt{d}$ dimensional image with each pixel following a standard normal distribution. The weights at each layer are given by convolutional filters with an upsampling factor of 2 at each layer. Once the filters are learned, it is straightforward to expand the network and generate larger texture patterns simply by increasing the size of Z and

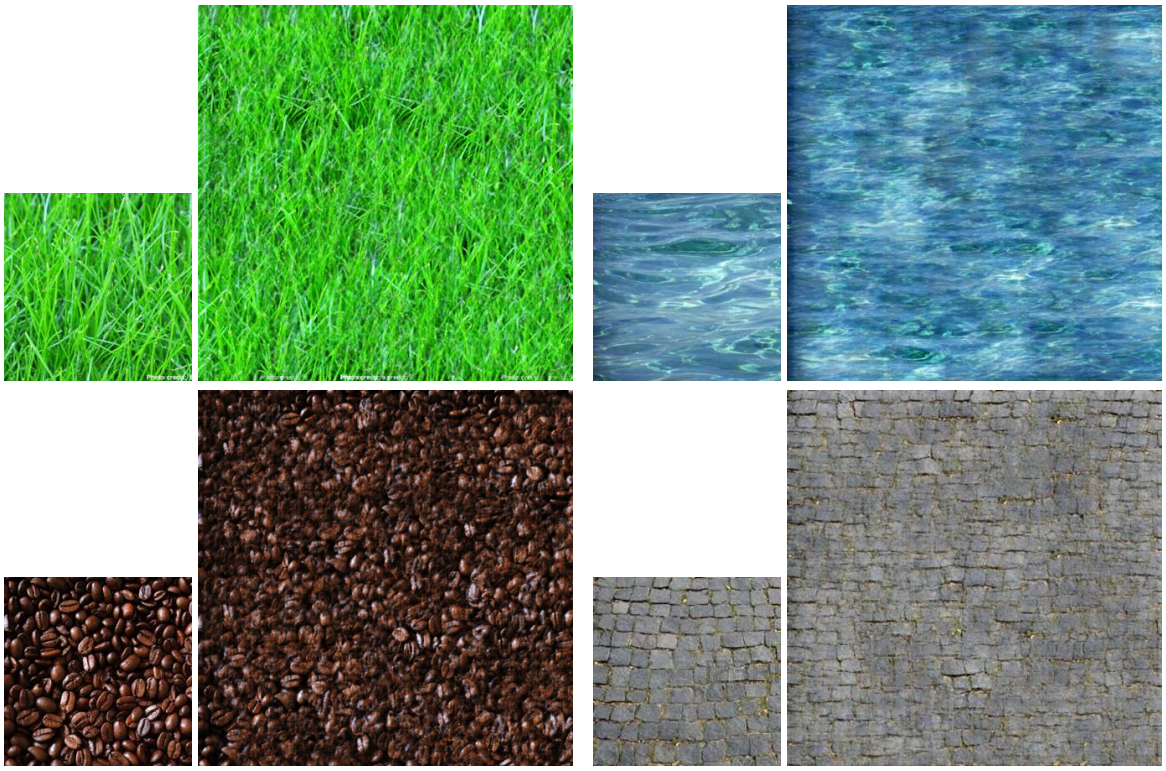


Fig. 10.12: ABP Experiment 1. Original images sized 224×224 next to synthesized images sized 448×448.

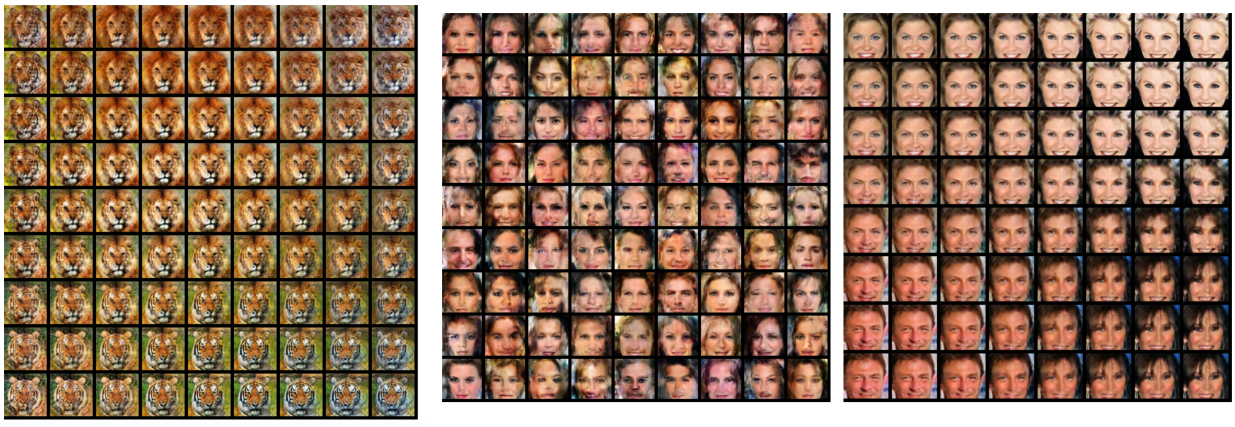


Fig. 10.13: ABP Experiment 2. *Left:* A 9×9 discretization of the 2-dimensional latent space of a generative model for lion/tiger faces. The latent space has identifiable regions separating lions and tigers, and the interpolations between these regions smoothly transform a lion face to a tiger face. *Right:* Synthesized human faces from a generative model with 100 latent factors. The left images show 81 faces sampled from the learned model, and the right images show linear interpolations in the latent space between the faces on the four corners of the image. © [2017] AAAI. Reprinted, with permission, from ref. [10].

running the filter convolutions over the larger input. In the example below, Z was a 7×7 image recreating a 224×224 image during training, while during test Z was expanded to a 14×14 image generating a 448×448 image using exactly the same weights.

ABP Experiment 2: Generating Object Patterns (Figure 10.13)

Generating object patterns is similar to generating texture patterns, except that the latent factor layer must be fully connected, so the input Z is a d-dimensional vector instead of a $\sqrt{d} \times \sqrt{d}$ matrix. The figures below show two different object patterns generated by the ABP algorithm: lion/tiger faces, and human faces. Interpolating between points in the latent space of the learned network gives non-linear interpolations in the image space along the manifold of the training data.

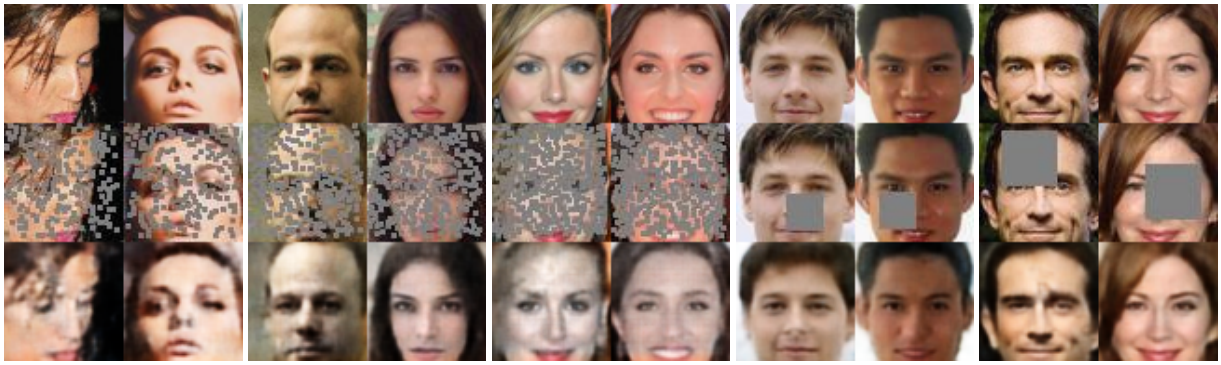


Fig. 10.14: ABP Experiment 3. *Top:* Original images. *Middle:* occluded training images. *Bottom:* Reconstructed training images.

ABP Experiment 3: Learning from Incomplete Data (Figure 10.14)

In some situations, corruption or occlusion can cause training images to have missing pixels. The ABP algorithm can learn a generator model for the complete image given a training set where certain pixels are labeled as missing. The only adjustment that needs to be made is to define the energy as a sum over only the observed pixels Λ^{obs} of X:

$$U_{\text{obs}}(X,Z;w) = \frac{1}{2\sigma^2} \sum_{(x,y)\in\Lambda^{\text{obs}}} \big(X_{(x,y)} - g(Z;w)_{(x,y)}\big)^2 + \frac{1}{2}||Z||^2. \tag{10.37}$$

Then $Z|X \sim \frac{1}{Z(w)} \exp\{-U_{\text{obs}}(X,Z;w)\}$ and the latent vector of an occluded image can be inferred using Langevin Dynamics on $Z|X$. The learned model can then accomplish three tasks: (1) Recover the missing pixels from the training images; (2) Recover the missing pixels from the testing images; (3) Synthesize new images from the model.

10.3.4 Cooperative energy and generator models

The DeepFRAME model from Section 10.3.2 and the ABP model from Section 10.3.3 can be integrated into a cooperative learning scheme where weights w_F of the energy function $-F(I;w_F)$ and the weights w_G of a generator network $G(Z;w_G)$ are learned jointly. The equations used to train a DeepFRAME model and an ABP model cooperatively are identical to the equations used to train the models separately. The innovation of cooperative learning is the way in which MCMC samples are initialized when training the networks in tandem. The synthesized images created by each model can be used to jump-start the sampling phase of the partner model in the next learning iteration. Cooperative learning [23] is an alternative to CD and PCD when training DeepFRAME and ABP models.

The ABP network (called the *generator* network) plays the role of the student in cooperative learning, while the DeepFRAME network (called the *descriptor* network) plays the role of the teacher. Since the generator model can efficiently synthesize images that are close to the true distribution, samples from the generator can be used as the initial images for Langevin updates when training a descriptor network. On the other hand, MCMC updates from the descriptor energy can be treated as samples from the "true" distribution when training the generator network. Since the descriptor revisions are similar to the original images created by the generator network, Langevin samples initialized from the original latent vectors should converge quickly to give a good latent approximation of the descriptor revision. The generator network receives guidance from the descriptor network, while the descriptor network compares the work of the generator with real data to determine what the generator needs to learn. Fusing the learning algorithms provides

a natural initialization scheme that allows both networks to see a greater diversity of initial samples. Other than MCMC initialization, the cooperative training procedure for each network is identical to solo training. An algorithm sketch is given below.

Algorithm 16 Cooperative Training Algorithm

Input: Observed images $\{X_i\}_{i=1}^n$, number of latent samples m, number of Langevin Iterations S, descriptor step size $\delta_F > 0$, generator step size $\delta_G > 0$, number of learning steps T.
Output: Weights w_F^* of descriptor network F and weights w_G^* of generator network G.

Initialize weights $w_{F,0}$ and $w_{G,0}$.
for $t = 1 : T$ **do**
a. Draw i.i.d. samples $\{Z_i\}_{i=1}^m$ from the latent distribution $\mathrm{N}(0, I_d)$ of the generator network $G(Z; w_{G,t-1})$. Compute images $\{Y_i\}_{i=1}^m$, where $Y_i = G(Z_i; w_{G,t-1})$.
b. Using Equation (10.32), apply S Langevin updates to the images $\{Y_i\}_{i=1}^m$ with the current energy $F(X; w_{F,t-1})$.
c. Using Equation (10.35), apply S Langevin updates to the latent factors $\{Z_i\}_{i=1}^m$ with the current weights $w_{G,t-1}$, where the revised Y_i from the previous step is the conditional image for each Z_i.
d. Sample a mini-batch $\{X_i\}_{i=1}^m$ of training data and update w_F according to

$$w_{F,t} = w_{F,t-1} + \delta_F \widetilde{\nabla} l_F(w_{F,t-1})$$

where $\widetilde{\nabla} l_F(w)$ is the stochastic gradient in (10.33).
e. Use revised latent factors $\{Z_i\}_{i=1}^m$ and revised images $\{Y_i\}_{i=1}^m$ to update w_G according to

$$w_{G,t} = w_{G,t-1} + \delta_G \widetilde{\nabla} l_G(w_{G,t-1})$$

with $\widetilde{\nabla} l_G(w)$ is the stochastic gradient in (10.36).
end for

The weakness of each network is made up for by the capabilities of the partner network. The descriptor can easily revise images that are close to the true data distribution to make them look more realistic. However, it is very difficult for the descriptor to generate realistic images starting from white noise, because many MCMC iterations are needed for convergence. On the other hand, it is easy for the generator to synthesize images that are close to the true data distribution with a simple forward pass through the network. However, it is very difficult for the generator to infer the latent vector for a new image, because this again requires a long process of MCMC inference. Cooperative learning addresses both of these shortcomings. The generator provides initial MCMC samples to the descriptor that are close to the true distribution, so the descriptor revisions converge quickly. Since the original generator images are similar to the descriptor revisions, the original latent factors are good points of initialization for the latent factor inference performed by the generator. The cooperative pipeline naturally facilitates both phases of MCMC sampling.

Cooperative Networks are related to Generative Adversarial Networks (GANs) [8], another method for training a generator network in tandem with a partner network. As suggested by the name, the two networks of a GAN model are trained in a competitive rather than cooperative scheme. Specifically, the descriptor $F(X; w)$ of a GAN (usually called a discriminator) is trained to distinguish samples of the generator network from samples of real data. When training a GAN, the generator network learns to fool the discriminator network, while the discriminator network tries to correctly separate generated images from real images. The objective function for a GAN is

$$\min_{w_G} \max_{w_D} L(w_G, w_F) = \mathrm{E}_q[\log F(X; w_F)] + \mathrm{E}_{\mathrm{N}(0,I_d)}[\log(1 - F(G(Z; w_G); w_F))],$$

where q is the true data distribution and $F(X; w_F) \in (0, 1)$ is a discriminative classifier. One can iteratively solve for w_G and w_F with alternating updates of gradient ascent with respect to w_F and gradient descent with respect to w_G. The first expectation can be approximated by taking a mini-batch of observed data, while the second expectation can be approximated by drawing a sample of Z from the latent normal distribution.

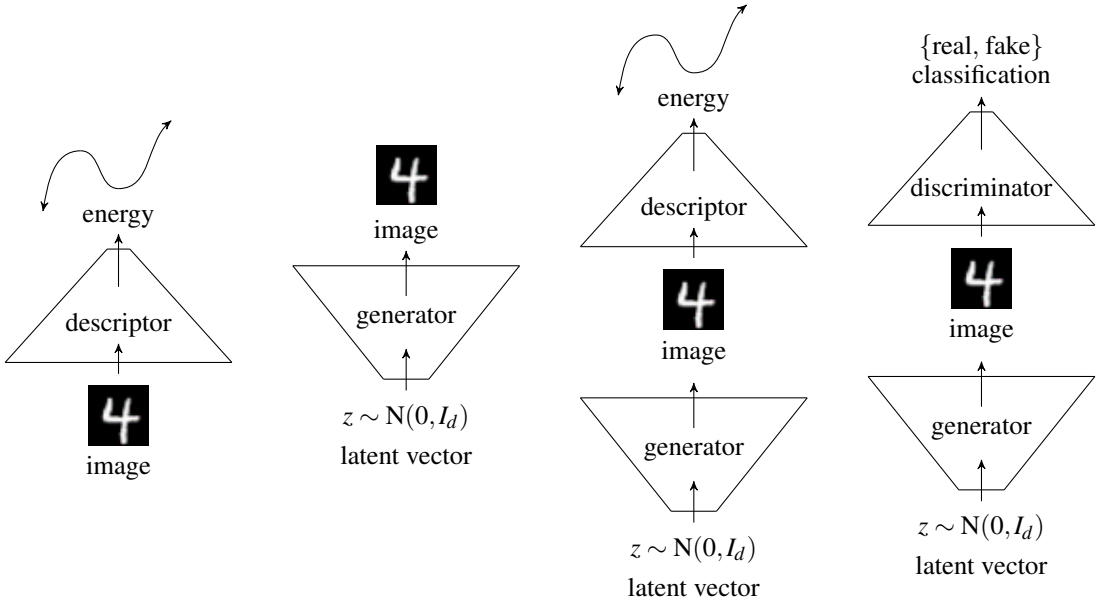


Fig. 10.15: Comparison of four different image models. The DeepFRAME model takes an image as input and returns a scalar energy value. The ABP model transforms a low-dimensional latent signal into a realistic image. Cooperative Learning fuses the DeepFRAME and ABP model together to train the descriptor and generator networks jointly. The GAN model is similar to the Cooperative model except that the descriptor is replaced by a discriminator, which classifies whether images are real or fake instead of returning an unnormalized density. The energy function from a Cooperative model is more useful that the discriminator of a GAN because it can be used to map the structure of the image space as described in Chapter 11.

In cooperative learning, the descriptor network learns a potential energy over the training data which guides the synthesis of the generator to match features of the training data. The energy function from cooperative learning is therefore more meaningful than the discriminator learned from GAN, because it is an unnormalized density of the training data rather than a classifier. We can use the descriptor energy to map the latent space of cooperative models with the energy landscape mapping techniques in Chapter 11. The same techniques cannot be applied to the GAN model because the discriminator network is not useful after training. Figure 10.15 gives a visual comparison of the DeepFRAME, ABP, Cooperative, and GAN models.
Three experiments testing the capabilities of Cooperative Learning are presented below.

Cooperative Learning Experiment 1: Synthesizing Images (Figure 10.16)

Cooperative Networks can synthesize images from either the descriptor energy via MCMC or from the generator network with a direct forward pass. Usually the second method is preferred since it is much more efficient. The cooperative networks can be trained to synthesize both texture images and aligned object images.

Cooperative Learning Experiment 2: Latent Space Interpolation (Figure 10.17)

As in the ABP model, linear interpolation in the latent space of the generator network corresponds to non-linear but visually intuitive interpolations between images in the image space. The generator network defines a low-dimensional manifold in the image space that approximates the manifold of the true data distribution. Moving through the generator space rather than the raw image space makes it easier to map the structure of the descriptor energy. See Chapter 11 for more details.

Cooperative Learning Experiment 3: Image Completion (Figure 10.18)

Fig. 10.16: Cooperative Learning Experiment 1. *Left:* Generator network synthesis after training on landscape images. *Right:* In each image group, the leftmost image is the training texture, and the other images are synthesized from the generator network. © [2018] AAAI. Reprinted, with permission, from ref. [23].

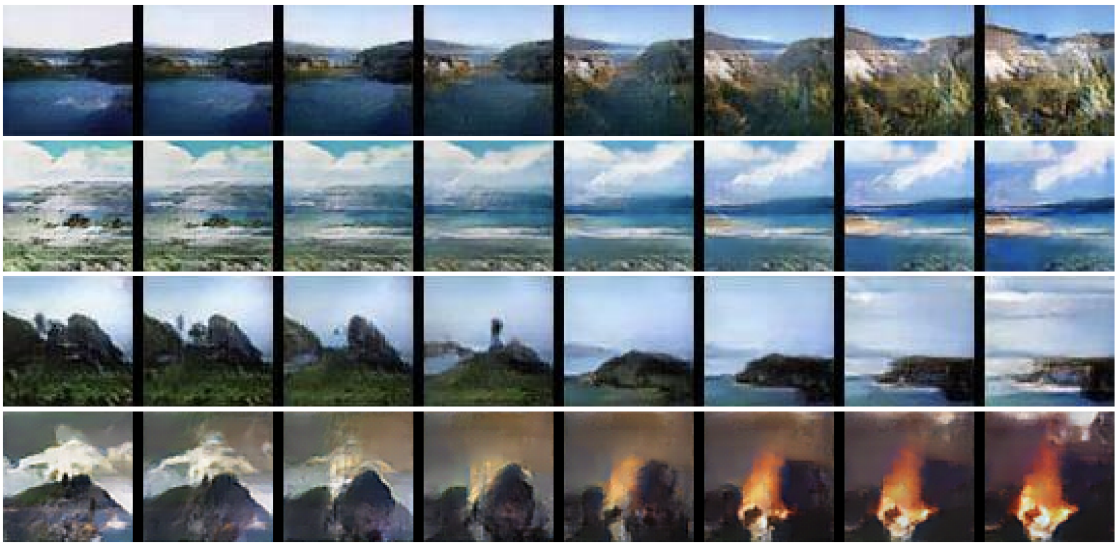


Fig. 10.17: Cooperative Learning Experiment 2. Linear interpolation in the latent space corresponds to intuitive non-linear interpolation in the image space. © [2018] AAAI. Reprinted, with permission, from ref. [23].

The generator network can be used to reconstruct images with occlusions or missing pixels. The occluded image X_0 is used as the conditional image for the distribution $Z|X = X_0$ and the latent vector Z that gives the best reconstruction of X_0 can be inferred by the Langevin Dynamics with the potential in Equation (10.37).

10.4 Exercises

Problem 1. Implement the DeepFRAME algorithm using a Convolutional Neural Network (CNN) and perform the three DeepFRAME experiments shown in Figures 10.8, 10.9 and 10.10:

Fig. 10.18: Cooperative Learning Experiment 3. In each group, the top row shows the original images, the middle row shows the occluded observations of the original image, and the bottom row shows the reconstruction of the occluded image from the generator latent space. © [2018] AAAI. Reprinted, with permission, from ref. [23].

a) Use a texture image such as one of those from Figure 10.8 to learn a DeepFRAME model and obtain MCMC samples from the learned model. Experiment with different CNN architectures (number of layers, number of filters, filter sizes) to see what works best.
b) Use aligned object images to learn a non-homogeneous DeepFRAME model and obtain samples from the model. Make sure the last layer of the CNN is fully connected.
c) Use multiple types of aligned objects (e.g. tigers and lions) to learn hybrid models and obtain samples from the learned model.

Problem 2. Implement the Alternating Back-Propagation algorithm using a Convolutional Neural Network (CNN) and perform the three ABP experiments shown in Figures 10.12, 10.13 and 10.14:

a) Use a texture image such as one of those from Figure 10.12 to learn a generator network with 7×7 input and 224×224 output and obtain samples from the learned model.
b) Use aligned face images to learn a non-homogeneous generator network with a 100 dimensional input and obtain samples from the model.
c) Use Eq. (10.37) to train a generator network on aligned face images with missing pixels (where for each image we know which pixels are missing). Use the obtained model to fill-in the missing pixels for other images that have not been used for training.

References

1. J Besag. Efficiency of pseudo-likelihood estimation for simple gaussian fields. *Biometrika*, 64:616–618, 1977.
2. Augustin Louis Cauchy. Methode generale pour la resolution des systemes d'equations simultanees. *C. R. Acad. Sci. Paris*, 25:536–538, 1847.
3. Pratik Chaudhari and Soatto Stefano. Stochastic gradient descent performs variational inference, converges to limit cycles for deep networks. *ICLR*, 2018.
4. Xavier Descombes, Robin Morris, Josiane Zerubia, and Marc Berthod. Maximum likelihood estimation of markov random field parameters using markov chain monte carlo algorithms. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 133–148. Springer, 1997.
5. Charles J Geyer. On the convergence of monte carlo maximum likelihood calculations. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 261–274, 1994.
6. Charles J Geyer and Elizabeth A Thompson. Constrained monte carlo maximum likelihood for dependent data. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 657–699, 1992.
7. Basilis Gidas. Consistency of maximum likelihood and pseudo-likelihood estimators for gibbs distributions. In *Stochastic differential systems, stochastic control theory and applications*, pages 129–145. 1988.
8. Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherijl Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.
9. Tian Han, Yang Lu, Song-Chun Zhu, and Ying Nian Wu. Alternating back-propagation for generator network. In *AAAI*, volume 3, page 13, 2017.
10. Tian Han, Yang Lu, Song-Chun Zhu, and Ying Nian Wu. Alternating back-propagation for generator network. In *AAAI*, volume 3, page 13, 2017.
11. Geoffrey Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.
12. John J Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences*, 81(10):3088–3092, 1984.

13. Wenqing Hu, Chris J. Li, Lei Li, and Jian-Guo Liu. On the diffusion approximation of nonconvex stochastic gradient descent. *arXiv preprint arXiv:1705.07562*, 2017.
14. Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *ICLR*, 2017.
15. Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.
16. Qianxiao Li, Cheng Tai, and Weinan E. Stochastic modified equations and adaptive stochastic gradient algorithms. *ICML*, pages 2101–2110, 2017.
17. Yang Lu, Song Chun Zhu, and Ying Nian Wu. Learning frame models using cnn filters. *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.
18. Eric Moulines and Francis R Bach. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *Advances in Neural Information Processing Systems*, pages 451–459, 2011.
19. Yurii Nesterov. *Introductory lectures on convex optimization*, volume 87. Springer Science & Business Media, 2004.
20. Nicolas L Roux, Mark Schmidt, and Francis R Bach. A stochastic gradient method with an exponential convergence _rate for finite training sets. In *Advances in Neural Information Processing Systems*, pages 2663–2671, 2012.
21. Tijmen Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. *ICML*, 2008.
22. Jianwen Xie, Wenze Hu, Song Chun Zhu, and Ying Nian Wu. A theory of generative convnet. *International Conference on Machine Learning*, 2016.
23. Jianwen Xie, Yang Lu, and Ying Nian Wu. Cooperative learning of energy-based model and latent variable model via mcmc teaching. *AAAI*, 2018.
24. Laurent Younes. Estimation and annealing for gibbsian fields. *Ann. Inst. H. Poincaré Probab. Statist*, 24:269–294, 1988.
25. Yao Zhang, Andrew M. Saxe, Madhu S. Advani, and Alpha A. Lee. Entropy-energy competition and the effectiveness of stochastic gradient descent in machine learning. *arXiv preprint arXiv:1803.0192*, 2018.
26. Song Chun Zhu and Xiuwen Liu. Learning in gibbsian fields: How accurate and how fast can it be? *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):1001–1006, 2002.
27. Song Chun Zhu, Ying Nian Wu, and David Mumford. Minimax entropy principle and its application to texture modeling. *Neural Computation*, 9(8):1627–1660, 1997.
28. Song Chun Zhu, Yingnian Wu, and David Mumford. Filters, random fields and maximum entropy (frame): Towards a unified theory for texture modeling. *International Journal of Computer Vision*, 27(2):107–126, 1998.

Chapter 11
Mapping the Energy Landscape

"By visualizing information we turn it into a landscape that you can explore with your eyes: a sort of information map. And when you're lost in information, an information map is kind of useful." - David McCandless

Introduction

In many statistical learning problems, optimization is performed on a target function that is highly non-convex. A large body of research has been devoted to either approximating the target function by a related convex function, such as replacing the L_0 norm with the L_1 norm in regression models, or designing algorithms to find a good local optimum, such as the Expectation-Maximization algorithm for clustering. The task of analyzing the non-convex structure of a target function has received much less attention. In this chapter, inspired by successful visualization of landscapes for molecular systems [2] and spin-glass models [40], we compute *Energy Landscape Maps* (ELMs) in the high-dimensional spaces. The first half of the chapter explores and visualizes the model space (i.e. the hypothesis spaces in the machine learning literature) for clustering, bi-clustering, and grammar learning. The second half of the chapter introduces a novel MCMC method for identifying macroscopic structures in locally noisy energy landscapes. The technique is applied to explore the formation of stable concepts in deep network models of images.

11.1 Landscape Examples, Structures, and Tasks

An *energy function* over a state space $\Omega \subset \mathbb{R}^n$ is the exponential term $U(x)$ of a Gibbs density

$$p(x) = \frac{1}{Z} \exp\{-U(x)\}.$$

The normalizing constant Z is almost always intractable and is treated as an unknown. By relaxing the full definition of a probability distribution and leaving the density unnormalized, it is possible to extend the probabilistic treatment to a wide variety of functions. The only requirement for treating an arbitrary function $U : \Omega \to \mathbb{R}$ as an energy function is the existence of the integral $\int_\Omega \exp\{-U(x)\}\,dx$. The normalizing integral exists for many relevant cases, notably when U is bounded below and has sufficient asymptotic growth or when Ω is bounded.

The energy function $U(x)$ characterizes the *relative* probability of two states x_1 and x_2. States with lower energy are more stable and appear with greater probability compared to higher energy states. The log probability ratio

$$\log \frac{p(x_1)}{p(x_2)} = U(x_2) - U(x_1)$$

depends only on the potential energy difference between the states. MCMC methods typically require the log probability ratio rather than the full density, so only the energy function is needed for implementation of sampling. Since the

normalizing constant is unknown, the energy function cannot provide any information about the *absolute* probability of a single state x. Indeed, $U^*(x) = U(x) + c$ defines the same distribution as $U(x)$ for any constant c. Therefore the values $U_1(x)$ and $U_2(x)$ of two different energy functions U_1 and U_2 at the same point x *cannot* be directly compared to determine whether x is more likely in the system of U_1 or U_2.

Many complex systems can be represented by defining an energy function over the possible states of the system. The state spaces associated with complex systems are far too large for enumeration and brute force analysis of the system energy function is not possible. However, not all system states are relevant because the energy function often takes low energy only in concentrated regions of the state space. Most *possible* states have virtually zero probability. The local minima of the energy function correspond to locally *stable* or *optimal* system states. Local minima and nearby low-energy states are the most relevant system states because they appear with the highest probability, and the system as a whole can be analyzed by examining the structure of the local minima. The relatively "tiny" portion of the state space occupied by the most relevant low-energy regions is often still vast and impossible to analyze with brute force computation.

The manifold of the energy surface defines a non-Euclidean geometry over the state space. The lowest energy barrier between two minima is a geodesic distance that measures the similarity of the states in terms of the system potential. Therefore, two states which are distant in Euclidean space might be nearby according to the geodesic measure along the energy surface, and vice-versa. The non-Euclidean geometry of an energy manifold reveals important structures of the system which the energy function represents. Intuitively, local minima that are separated by low energy barriers are similar in terms of system properties, while local minima separated by high energy barriers represent distinct subsystems. Examples of complex systems that can be represented by an energy function and its minima structure include:

System States	**Energy**	**Local Minima**
Model Parameters	Loss	Locally Optimal Parameters
Physical States	Physical Energy	Stable States
Images	Memory	Reliable Memories
Biological/Social Traits	Utility	Adaptations/Social Norms

In the literature, Becker and Karplus [2] present the first work for visualizing multidimensional energy landscapes. They map the local minima of molecular systems whose energy structure is known through physical experimentation. The landscapes are visualized using a Disconnectivity Graph, or DG [2], which is a tree structure where each leaf node represents a local minimum and each non-leaf node represents the barrier between adjacent energy basins. A simple 1D landscape and its associated DG are shown in Figure 11.1. The DG characterizes the energy landscape with the following information: (1) the number of local minima and their energy levels, (2) the energy barriers between adjacent local minima, and (3) the probability mass and volume of each local minimum (see Section 11.1.2 and Figure 11.6).

Wales et al. [34, 35] develop computational methods to map the potential energy surface of quantitative molecular models and use the DG visualization from [2] to display the mappings. Statisticians also developed a series of MCMC methods for improving the efficiency of the sampling algorithms that traverse complex state spaces. Most notably, [21] generalizes the Wang-Landau algorithm [36] for random walks in the state space. Zhou [40] uses the generalized Wang-Landau algorithm to plot the disconnectivity graph for Ising model with hundreds of local minima and proposes an effective way for estimating the energy barriers. Furthermore, experiments in [41] construct the energy landscape for Bayesian inference of DNA sequence segmentation by clustering Monte Carlo samples.

Computationally identifying the structure of the energy landscape is useful for the following tasks:

1. Analyzing the intrinsic difficulty (or complexity) of the optimization problems, for either inference or learning tasks. For example, in bi-clustering, we divide the problem into the *easy*, *hard*, and *impossible* regimes under different conditions.
2. Analyzing the effects of various conditions on the ELM complexity. Examples include: separability in clustering, the number of training examples, the level of supervision (i.e. what percent of the examples are labeled), and the strength of regularization (i.e. prior model).
3. Analyzing the behavior of various algorithms by showing their frequencies of visiting the various minima. For example, in the Gaussian mixture clustering problem, we find that when the Gaussian components are highly separable, K-means clustering works better than the EM algorithm [10], and the opposite is true when the components are less separable. In contrast to the frequent visits of local minima by K-means and EM, the Swendsen-Wang cut method [1] converges to the global minimum in all separability conditions.
4. Analyzing protein folding, where the energy landscape has a funnel-like appearance [25]. Locally, the energy landscape of different folding configurations is very noisy with astronomically many minima. The global structure,

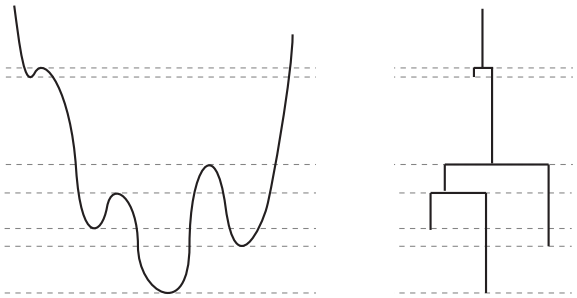


Fig. 11.1: An energy function and the corresponding Energy Landscape Map (ELM). The y-axis of the ELM is the energy level, each leaf node is a local minimum and the leaf nodes are connected at the ridges of their energy basins. © [2019] Brown University. Reprinted, with permission, from ref. [16].

on the other hand, is essentially unimodal and convex because the vast majority of local minima are shallow features along the sides of the energy funnel. This structure efficiently guides an unfolded protein to its native state, because environmental perturbations are strong enough to overcome shallow intermediate minima while the folding is in progress. Once folding is complete, the depth of the energy funnel allows the protein to maintain a stable shape and perform its function. See Figure 11.2 for an illustration. In pathological cases multiple energy funnels might exist, only one of which leads to the native state (global optimum) while the others give rise to stable mis-folded proteins. Large quantities of mis-folded proteins are related to neurodegenerative diseases such as Alzheimer's disease, Parkinson's, mad cow disease, etc.

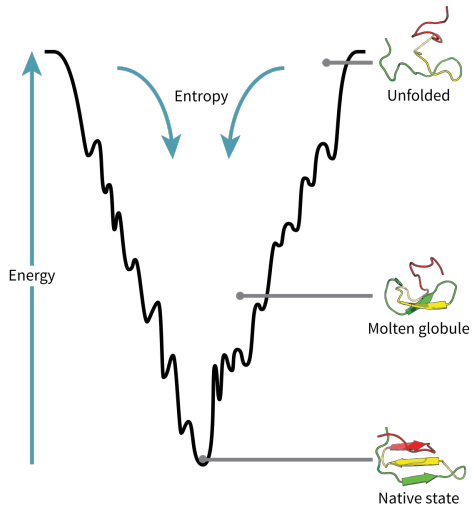


Fig. 11.2: Diagram of potential energy for protein folding. The unfolded protein is guided to its native state by the funnel structure of the landscape. The landscape has a huge number of local minima, but most are quite shallow. The macroscopic structure of the landscape has a single global basin.

5. Analyzing the concepts/memories learned by probabilistic image models. After training an image model to learn a potential energy function over the image space (see Chapter 10), one can map the local minima of the image energy to identify stable regions of the image space. Since the image potentials are trained to approximate the negative log density of the training data, regions of the image space with low energy correspond to images with an appearance similar to the training data, while high energy images have an appearance that differs significantly from training images. Images with a similar appearance should be separated by low energy barriers, since it is

possible to morph between similar images without violating energy constraints. On the other hand, images that differ significantly should be separated by high energy barriers, since it is not possible to smoothly change between the images without encountering an improbable intermediate image. Thus, one can map the local minima of the image space to identify different memory concepts learned during training. See Section 11.6 for more information.

11.1.1 Energy-based partitions of the state space

In this chapter, "mapping" the landscape of a non-convex target function means partitioning the state space in a way that reflects the evolution of each point under gradient or diffusion flow on the target landscape. Let Ω be the state space over which a probability distribution $p(x)$ and energy $U(x)$ are defined. The goal of energy landscape mapping is to partition Ω into J disjoint energy basins D_j:

$$\Omega = \cup_{j=1}^{J} D_j, \quad \cap_{j=1}^{J} D_j = \emptyset. \tag{11.1}$$

This chapter will use two different definitions of the energy basins D_j to address different mapping situations. The first definition is discussed in this section, and the other definition is introduced with the macroscopic mapping techniques in Section 11.4.1.

The simplest way of defining a energy-based partition $\mathscr{D} = \{D_j\}_{j=1}^{J}$ is to associate each distinct local minimum with the set of points which converge to that minimum under gradient descent. Formally, let $\{x_j^*\}_{j=1}^{J}$ be the set of local minima of U. Let $\varphi(x)$ be a function which maps a point x to its target local minimum under steepest descent. Then x_j^* defines a basin $D_j = \{x : \varphi(x) = x_j^*\}$, as shown by the vertical dashed lines in Figure 11.3.

This partitioning method can be used for mapping as long as U has a finite number J of local minima. Landscapes should have finitely many minima if U is sufficiently regular or if Ω is discrete and finite. If U has infinitely many local minima then we cannot use this definition of $\mathscr{D}$ for mapping because discovery of new minima would continue indefinitely. Defining a separate basin for each distinct minimum can be still problematic when there are a finite but astronomical number of local minima. The alternate partitioning in Section 11.4.1 addresses this issue.

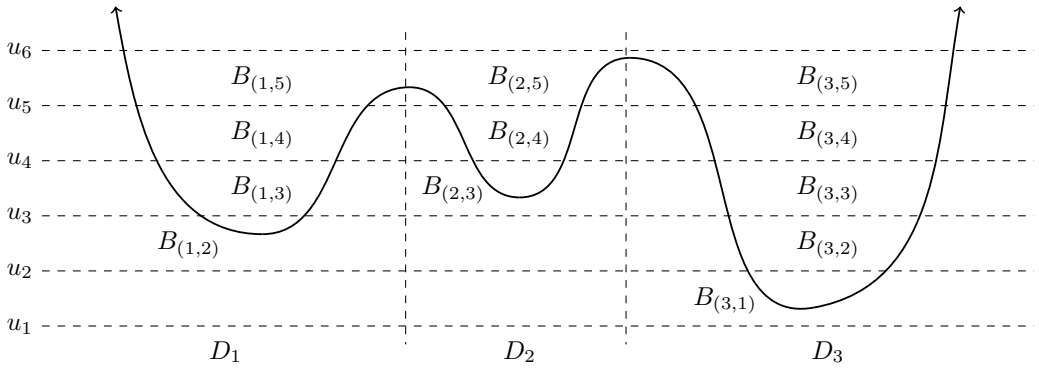


Fig. 11.3: 1D state space $\Omega = \mathbb{R}$ partitioned into energy basins D_j (along the x-axis), and the energy $\mathbb{R}$ (the y-axis) is partitioned into uniform intervals $[u_k, u_{k+1})$. The intersection of these partitions creates energy bins B_{jk}. Non-empty bins are labeled in the figure.

Basins D_j from a partition $\mathscr{D}$ can be further partitioned into intervals $\{[u_1,u_2), [u_2,u_3), \ldots, [u_K,u_{K+1})\}$ across the energy spectrum. Figure 11.3 illustrates such a division. Thus we obtain a set of bins $\mathscr{B} = \{B_{jk}\}_{1\le j\le J, 1\le k\le K}$ where

$$B_{jk} = \{x : x \in D_j,\ U(x) \in [u_k, u_{k+1})\}. \tag{11.2}$$

The partition $\mathscr{B}$ plays a central role in the Generalized Wang-Landau Algorithm covered in the first half of the chapter. The choice of intervals depends on the energy function and careful tuning of the number of intervals K and the width of the intervals is necessary for good results.

11.1.2 Constructing a disconnectivity graph

A Disconnectivity Graph (DG) can be used to visualize the structure of a landscape partition $\mathscr{D} = \{D_j\}_{j=1}^J$ once the basins D_j have been identified by a mapping algorithm. Recall that the energy barrier between two system states gives a non-Euclidean measure of distance between the states that indicates similarity of the states in terms of system properties. Therefore the minimum energy barrier between two basins can be used to identify hierarchical relations between basins. In a DG, basins are displayed as separate in the spectrum below the energy barrier between basins and are merged in the spectrum above the barrier. The merging process continues hierarchically until all basins have merged in the DG.

A DG displays the basins and their minimum energy in the terminal nodes of the tree, and the minimum energy barrier at which basins or groups or basins merge in the branching nodes. One can also display the probability mass and volume of each basin (see Figure 11.6 and Section 11.2.2). DG construction requires an estimate of the minimum energy barrier between each pair of basins, which can be stored as a symmetric matrix. Given the symmetric barrier matrix, the DG can be created one basin at at time using the algorithm below. Figure 11.4 illustrates the process. After construction, one can remove spurious basins of depth less than a constant ε. Circles indicating probability mass and volume can be added to each leaf node after construction.

Algorithm 17 Disconnectivity Graph (DG) Construction

Input: Symmetric $N \times N$ matrix M (each row/column gives lowest barrier between single basin and others, diagonal gives the minimum energy within each basin)

Output: Tree Graph G with label and y-value for each node (giving either basin minimum energy or energy of merge)

Initialize the leaf nodes of G as the N basins. The y-value for each leaf node is the minimum energy of each basin. Let $M_1 = M$ and let $L_1 = \{1, 2, \ldots, N\}$ be an ordered set representing the initial nodes.

for $n = 1 : (N-1)$ **do**

1. Find $(i_n^*, j_n^*) = \text{argmin}_{i \neq j} [M_n]_{(i,j)}$. Let $a_n = L_n(i_n^*)$ and $b_n = L_n(j_n^*)$, where $L_n(k)$ refers to the k^{th} element of the ordered set L_n.
2. Add node with label $(N+n)$, y-value $[M_n]_{(i_n^*, j_n^*)}$, and child nodes a_n and b_n to G.
3. Let $L_{n+1} = (L_n \setminus \{a_n, b_n\}) \cup \{(N+n)\}$ represent the nodes which have yet to be merged.
4. Define an updated $(N-n) \times (N-n)$ symmetric matrix M_{n+1} as

$$[M_{n+1}]_{(i,j)} = \begin{cases} [M_n]_{(\varphi_n(i), \varphi_n(j))} & \text{if } L_{n+1}(i) \in L_n \text{ and } L_{n+1}(j) \in L_n \\ \min\left([M_n]_{(\varphi_n(i), i_n^*)}, [M_n]_{(\varphi_n(i), j_n^*)}\right) & \text{if } L_{n+1}(i) \in L_n \text{ and } L_{n+1}(j) \notin L_n \\ \min\left([M_n]_{(i_n^*, \varphi_n(j))}, [M_n]_{(j_n^*, \varphi_n(j))}\right) & \text{if } L_{n+1}(i) \notin L_n \text{ and } L_{n+1}(j) \in L_n \\ [M_n]_{(i_n^*, j_n^*)} & \text{if } L_{n+1}(i) \notin L_n \text{ and } L_{n+1}(j) \notin L_n \end{cases}.$$

where $\varphi_n(i) = L_n^{-1}(L_{n+1}(i))$.

end for

11.1.3 ELM example in 2D

We present a simple illustrative example of landscape mapping in Figures 11.5 and 11.6. Suppose the underlying probability distribution is a 4-component Gaussian mixture model (GMM) in 1D space with well-separated components. The model space is 11-dimensional with parameters $\{(\mu_i, \sigma_i, \alpha_i) : i = 1,2,3,4\}$, where μ_i denotes the mean, σ_i the variance, and α_i the weight for each component. We sample 70 data points $\{X_i\}_{i=1}^{70}$ from the ground-truth GMM and construct the ELM in the model space using the negative log likelihood of the GMM sample as the energy function.

For this example, we will restrict the 11-dimensional model space to 2 dimensions so that it is possible to fully visualize the raw energy function and easily compare the structure of the energy function with the structure of the

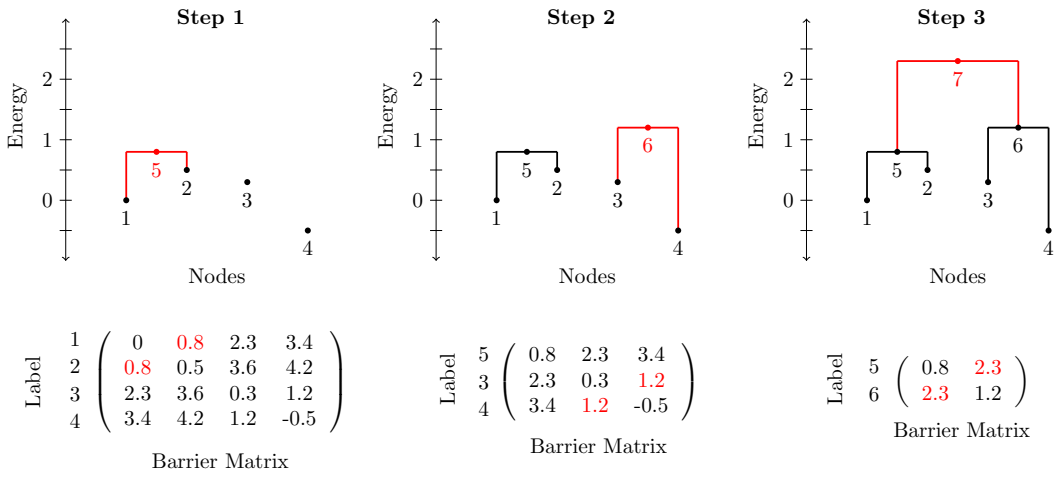
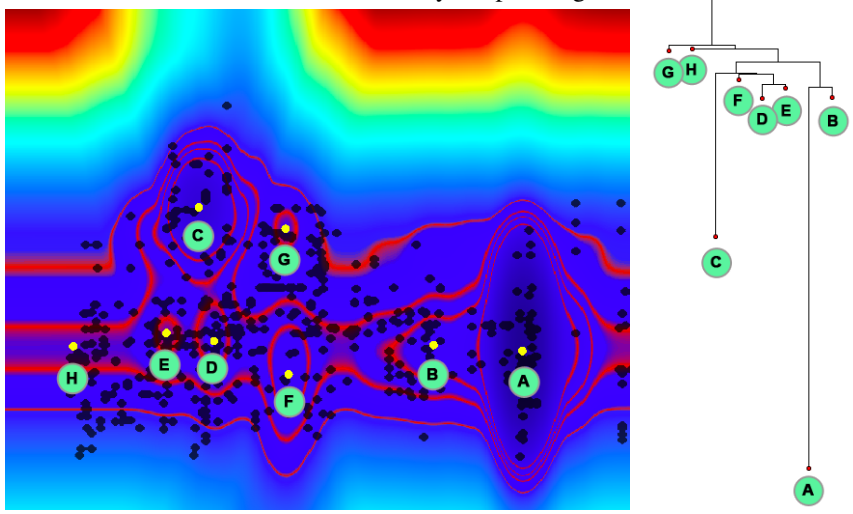


Fig. 11.4: Iterative construction of a Disconnectivity Graph using the energy barriers between basins.



(a) (b)

Fig. 11.5: (a) Energy Landscape for a 4-component 1D GMM with all parameters fixed except two means. Level sets are highlighted in red. The local minima are shown in yellow dots and the first 200 MCMC samples are shown in black dots. (b) The resulting ELM and the correspondence between the leaves and the local minima from the energy landscape. © [2014] Maria Pavlovskaia. Reprinted, with permission, from ref. [26].

1

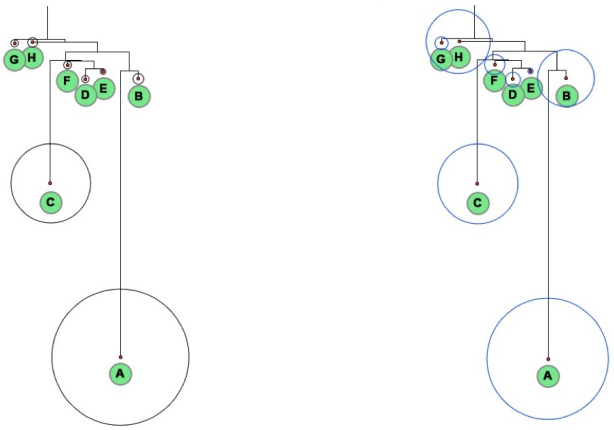
ELM. To do this, we set all parameters to the ground-truth value except for μ_1 and μ_2, which are unknown. The energy function is then

$$U_{\mu_3,\mu_4,\{\sigma_j\}_{j=1}^4,\{\alpha_j\}_{j=1}^4}(\mu_1,\mu_2) = \sum_{i=1}^{70}\sum_{j=1}^{4}\frac{\alpha_j}{\sigma_j}\varphi\left(\frac{X_i-\mu_j}{\sigma_j}\right)$$

where φ is the standard normal density. Figure 11.5 (a) shows the energy map on a range of $0 \le \mu_1, \mu_2 \le 5$. The asymmetry in the landscape is caused by the fact that the true model has different weights between the first and second component. Some shallow local minima, like E, F, G, and H, are little "dents" caused by the finite sample size.

Figure 11.5 (a) shows that all the local minima are identified. Additionally, the black dots in the figure show the first 200 MCMC samples that were accepted by the Generalized Wang-Landau algorithm that we will discuss in Section 11.2. The samples are clustered around the local minima, and cover all energy basins. They are not present in the high energy areas away from the local minima, as would be desired. Figure 11.5 (b) shows the resulting ELM and

the correspondence between the DG nodes and the local minima in the energy landscape. Furthermore, Figures 11.6 (a) and (b) show the probability mass and the volume of these energy basins.



(a) probability mass (b) volume

Fig. 11.6: The probability mass and volume of the energy basins for the 2-d landscape shown in Figure 11.5. © [2014] Maria Pavlovskaia. Reprinted, with permission, from ref. [26].

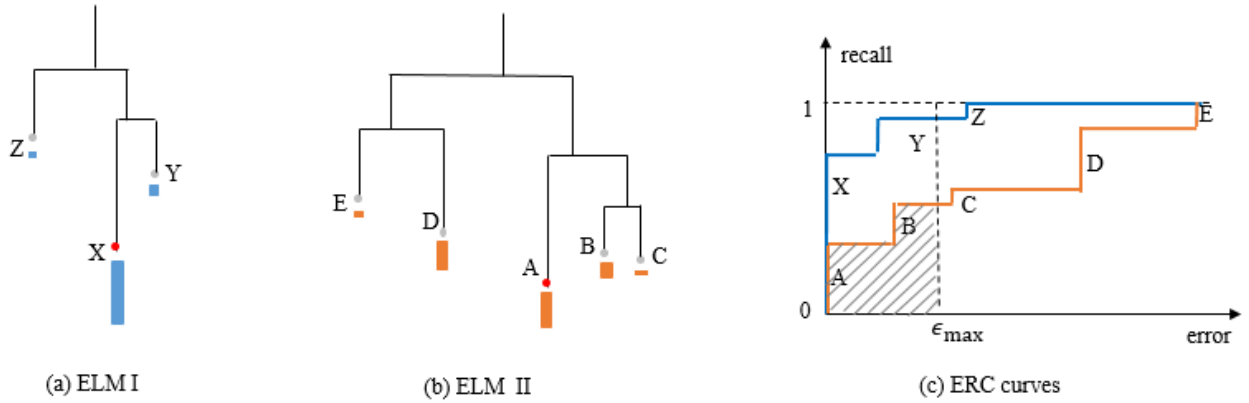
The spurious local minima in this small example foreshadow a major difficulty encountered when trying to map energy landscapes in real applications. Most non-convex landscapes of interest have an astronomical number of local minima and full enumeration of all minima as presented in this section is not a realistic approach. Energy basins tend to have wide and flat bottoms that contain a rich variety of related but distinct minima. Efficient and scalable mapping techniques must overcome this difficulty. See Section 11.4 for more information.

11.1.4 Characterizing the difficulty (or complexity) of learning tasks

It is often desirable to measure the difficulty of the learning task. Mapping the structure of a non-convex objective function can shed light on the complexity of optimization-based learning. For example, we can compare two ELMs in Figure 11.7 and might conclude that learning in the landscape of ELM I looks easier than that of ELM II. However, the difficulty also depends on the learning algorithms used for optimization. Thus we can run a learning algorithm many times and record the frequency that it converges to each basin or minimum. The frequency is shown by the lengths of the colored bars under the leaf nodes.

Suppose that Θ^* is the true model to be learned. In Figure 11.7, Θ^* corresponds to node X in ELM I and node A in ELM II. In general, Θ^* may not be the global minimum or not even a minimum. We then measure the distance (or error) between Θ^* and any other local minimum. As the error increases, we accumulate the frequency to plot a curve. We call this the Error-Recall Curve (ERC), as the horizontal axis is the error and the vertical axis is the frequency of recall for the solutions. This curve is analogous to the ROC (Receiver Operating Characteristic) curves in Bayesian decision theory, pattern recognition and machine learning. By sliding the threshold $\varepsilon_{\max}$ which is the maximum error tolerable, the curve characterizes the difficulty of the ELM with respect to the learning algorithm.

One quantitative measure that characterizes learning difficulty is the Area Under the Curve (AUC) for a given ERC and $\varepsilon_{\max}$. This measure is illustrated by the shadowed area in 11.7(c) for ELM II. When the AUC is close to 1, the task is easy, and when the AUC is close to 0, learning is impossible. In a learning problem, we can set different conditions that correspond to a range of landscape structures. The difficulty measures of learning under these conditions can be visualized in the space of the parameters as a difficulty map.



(a) ELM I (b) ELM II (c) ERC curves

Fig. 11.7: Characterizing the difficulty of learning in the ELM. For two learning tasks with ELM I and ELM II, the colored bar show the frequency that a learning algorithm converges to the basins, from which two Error-recall curves are plotted. The difficulty of the learning task, with respect to this algorithm, can be measured by the area under the curve within an acceptable maximum error. © [2014] Maria Pavlovskaia. Reprinted, with permission, from ref. [26].

11.2 Generalized Wang-Landau Algorithm

The objective of the Generalized Wang-Landau (GWL) algorithm is to simulate a Markov chain that visits all the bins $\{B_{jk}\}_{1\le j\le J,1\le k\le K}$ with equal probability, and thus effectively reveals the structure of the landscape. Let $\phi : \Omega \to \{1,\ldots,J\}\times\{1,...,K\}$ be the mapping between the model space and bin indices: $\phi(x) = (j,k)$ if $x \in B_{jk}$. Given any x, we can use gradient descent or its variants to find and record the basin D_j to which x belongs and compute its energy level $U(x)$ to find the index $\phi(x)$.

We define $\beta(j,k)$ to be the probability mass of a bin

$$\beta(j,k) = \int_{B_{jk}} p(x)\,dx. \tag{11.3}$$

Then, we can define a new probability distribution that has equal probability among all the bins,

$$p'(x) = \frac{1}{Z}p(x)/\beta(\phi(x)), \tag{11.4}$$

with Z being a scaling constant.

To sample from $p'(x)$, one can estimate $\beta(j,k)$ by a variable γ_{jk}. We define the probability function $p_\gamma : \Omega \to \mathbb{R}$ to be

$$p_\gamma(x) \propto \frac{p(x)}{\gamma_{\phi(x)}} = \sum_{j,k}\frac{p(x)}{\gamma_{jk}}\mathbb{1}(x\in B_{jk}) \;\text{ st. }\int_\Omega p_\gamma(x)dx = 1.$$

We start with an initial γ^0, and update $\gamma^t = \{\gamma^t_{jk},\forall j,k\}$ iteratively using stochastic approximation [22]. Suppose x_t is the MCMC state at time t, then γ^t is updated in an exponential rate,

$$\log\gamma^{t+1}_{jk} = \log\gamma^t_{jk} + \eta_t\,\mathbb{1}(x_t\in B_{jk}), \quad \forall j,k. \tag{11.5}$$

η_t is the step size at time t. The step size is decreased over time and the decreasing schedule is either pre-determined as in [22] or determined adaptively as in [39].

Each iteration with given γ^t uses a Metropolis step. Let $Q(x,y)$ be the proposal probability for moving from x to y, then the acceptance probability is

$$\alpha(x,y) = \min\left(1, \frac{Q(y,x)p_\gamma(y)}{Q(x,y)p_\gamma(x)}\right) = \min\left(1, \frac{Q(y,x)}{Q(x,y)}\frac{p(y)}{p(x)}\frac{\gamma'_{\phi(x)}}{\gamma'_{\phi(y)}}\right). \tag{11.6}$$

Intuitively, if $\gamma'_{\phi(x)} < \gamma'_{\phi(y)}$, then the probability of visiting y is reduced. For the purpose of exploring the energy landscape, the GWL algorithm improves upon conventional methods, such as the simulated annealing [14] and tempering [23] process. The latter samples from $p(x)^{\frac{1}{T}}$ and does not visit the bins with equal probability even at high temperature.

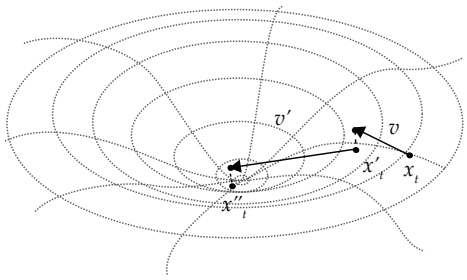


Fig. 11.8: First two steps of projected gradient descent. The algorithm is initialized with MCMC sample x_t. v is the gradient of $E(x)$ at the point x_t. Armijo line search is used to determine the step size α along the vector v. x'_t is the projection $T(x_t + \alpha v)$ onto the subspace Γ. Then x''_t is the projection $T(x_t + \alpha' v')$, and so on. © [2014] Maria Pavlovskaia. Reprinted, with permission, from ref. [26].

In performing gradient descent, we employ Armijo line search to determine the step size; if the model space Ω is a manifold in $\mathbb{R}^n$, we perform projected gradient descent, as shown in Figure 11.8. To avoid erroneously identifying multiple local minima within the same basin (especially when there are large flat regions), we merge two local minima identified by gradient descent based on the following criteria: (1) the distance between the local minima is smaller than a constant ε; or (2) there is no barrier along the straight line between the two local minima.

11.2.1 Barrier estimation for GWL mapping

After the GWL mapping has been implemented, it is necessary to estimate the energy barriers between the basins identified by GWL to construct a DG and visualize the landscape. Suppose we have collected a chain of samples $x_1, \ldots, x_N$ from the GWL algorithm. Since each state in the chain is obtained by a local MCMC proposal from the previous state, the GWL path can be used to pinpoint transitions between energy basins by identifying consecutive samples that belong to different basins. Figure 11.9 illustrates a sequence of Markov chain states $x_t, ..., x_{t+9}$ over two energy basins. The dotted curves are the level sets of the energy function. A transition from one basin to another occurs between x_{t+3} and x_{t+4}.

The energy of consecutive GWL states that belong to different basins is an upper bound for the minimum energy barrier between the basins. This conservative estimate of the minimum energy barrier can be refined using a ridge descent algorithm. We collect all consecutive MCMC states that move across two basins D_k and D_ℓ,

$$X_{kl} = \{(x_t, x_{t+1}) : x_t \in D_k, x_{t+1} \in D_\ell\} \tag{11.7}$$

and we choose $(a_0, b_0) \in X_{kl}$ with the lowest energy

$$(a_0, b_0) = \text{argmin}_{(a,b)\in\Omega_{kl}} \left[\min(U(a), U(b))\right].$$

Next we iterate the following step as Figure 11.10 illustrates

$$a_i = \text{argmin}_a \{U(a) : a \in \text{Neighborhood}(b_{i-1}) \cap D_k\}$$
$$b_i = \text{argmin}_b \{U(b) : b \in \text{Neighborhood}(a_i) \cap D_\ell\}$$

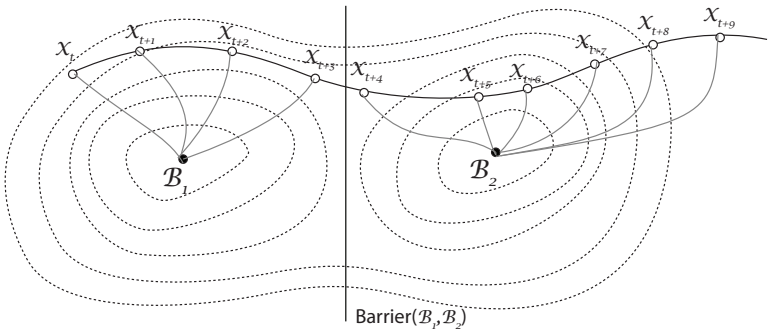


Fig. 11.9: Sequential MCMC samples $x_t, x_{t+1}, \ldots, x_{t+9}$. For each sample, we perform gradient descent to determine which energy basin the sample belongs to. If two sequential samples fall into different basins (x_{t+3} and x_{t+4} in this example), we estimate or update the upper-bound of the energy barrier between their respective basins (B_1 and B_2 in this example). © [2015] Springer. Reprinted, with permission, from ref. [27].

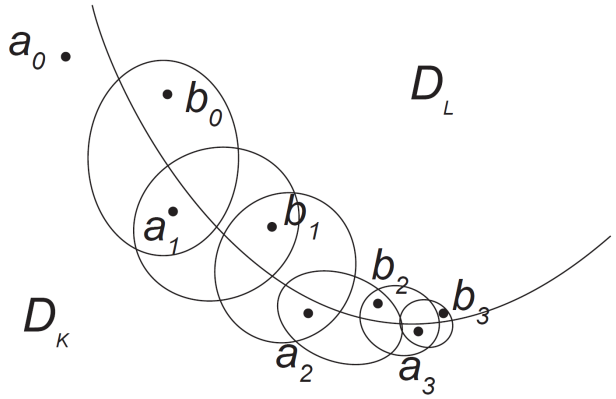


Fig. 11.10: The ridge descent algorithm is used for estimating the energy barrier between basins D_k and D_ℓ initialized at consecutive MCMC samples $a_0 = x_t, b_0 = x_{t+1}$ where $a_0 \in D_k$ and $b_0 \in D_\ell$. © [2014] Maria Pavlovskaia. Reprinted, with permission, from ref. [26].

until $b_{i-1} = b_i$. The neighborhood is defined by an adaptive radius. Then b_i is the energy barrier and $U(b_i)$ is the energy level of the barrier. A discrete version of this ridge descent method was used in [40].

11.2.2 Volume estimation with GWL

We can use the information obtained from a GWL mapping to estimate the probability mass and the volume of each energy basin. When the algorithm converges, the normalized value of γ_{jk} approaches the probability mass of bin B_{jk}:

$$\hat{P}(B_{jk}) = \frac{\gamma_{jk}}{\sum_{i,\ell} \gamma_{i\ell}} \to \beta(j,k), \quad \text{almost surely.}$$

Therefore the probability mass of a basin D_j can be estimated by

$$\hat{P}(D_j) = \sum_k \hat{P}(B_{jk}) = \frac{\sum_k \gamma_{jk}}{\sum_{i,\ell} \gamma_{i\ell}} \tag{11.8}$$

Suppose the energy $U(x)$ is partitioned into sufficiently small intervals of size du. Based on the probability mass, we can then estimate the size of the bins and basins in the model space Ω. A bin B_{jk} with energy interval $[u_k, u_k + du)$

can be seen as having energy u_k and probability density αe^{-u_k} (α is a normalization factor). The size of bin B_{jk} can be estimated by

$$\hat{A}(B_{jk}) = \frac{\hat{P}(B_{jk})}{\alpha e^{-u_k}} = \frac{\gamma_{jk}}{\alpha e^{-u_k} \sum_{i,\ell} \gamma_{i\ell}}$$

The size of basin D_j can be estimated by

$$\hat{A}(D_j) = \sum_k \hat{A}(B_{jk}) = \frac{1}{\sum_{i,\ell} \gamma_{i\ell}} \sum_k \frac{\gamma_{jk}}{\alpha e^{-u_k}} \tag{11.9}$$

Further, we can estimate the volume of a basin in the energy landscape which is defined as the amount of space contained in a basin in the space of $\Omega \times \mathbb{R}$.

$$\hat{V}(D_j) = \sum_k \sum_{i:u_i \le u_k} \hat{A}(B_{ji}) \times du = \frac{du}{\sum_{\ell,m} \gamma_{\ell m}} \sum_k \sum_{i:u_i \le u_k} \frac{\gamma_{ji}}{\alpha e^{-u_i}} \tag{11.10}$$

where the range of k depends on the definition of the basin. In a restricted definition, the basin only includes the volume under the closest barrier, as Figure 11.11 illustrates. The volume above the basins 1 and 2 is shared by the two basins, and is between the two energy barriers C and D. Thus we define the volume for a non-leaf node in the ELM to be the sum of its childen plus the volume between the barriers. For example, node C has volume $V(A)+V(B)+V(AB)$.

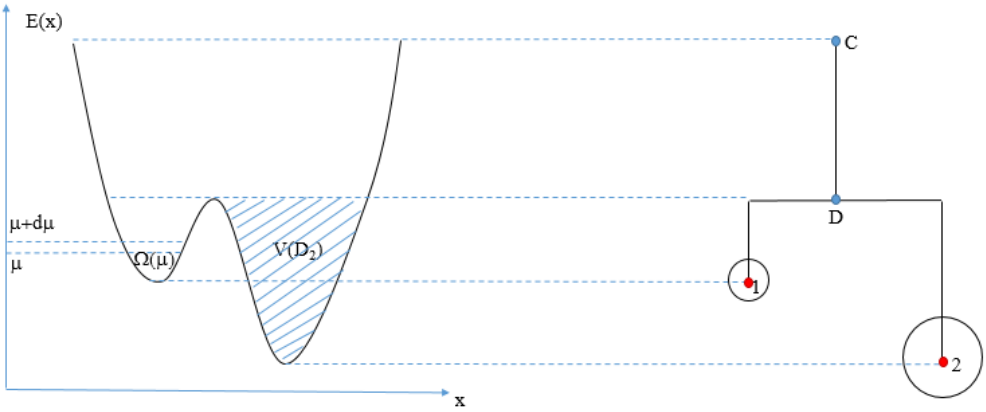


Fig. 11.11: The volume of basins. Assuming that du is sufficiently small, the volume of an energy basin can be approximated by the summation of the estimated volume at each energy interval.

If our goal is to develop a scale-space representation of the ELM by repeatedly smoothing the landscape, then basins A and B will be merged into one basin at certain scale, and the volume above the two basins will be also added to this new merged basin. Note that the partition of the space into bins, rather than basins, facilitates the computation of energy barriers, the mass and the volume of the basins.

11.2.3 GWL convergence analysis

As shown in Figure 11.12, the Disconnectivity Graphs constructed from two different mappings may have minor variations due to the precision of the energy barrier estimation. In experiments, we can monitor the convergence of GWL in the model space. We run multiple MCMC initialized with random starting values. After a burn-in period, we collect samples and project them to a 2-3 dimensional space using multi-dimensional scaling. We check whether the chains have converged to a stationary distribution using the multivariate extension of the Gelman and Rubin criterion [6, 13].

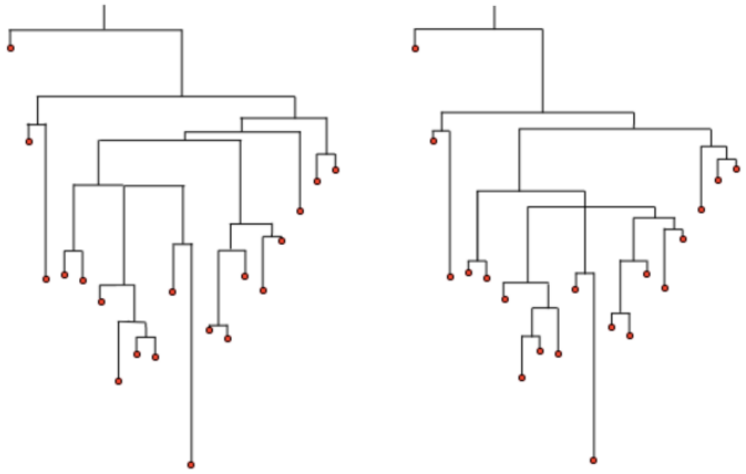
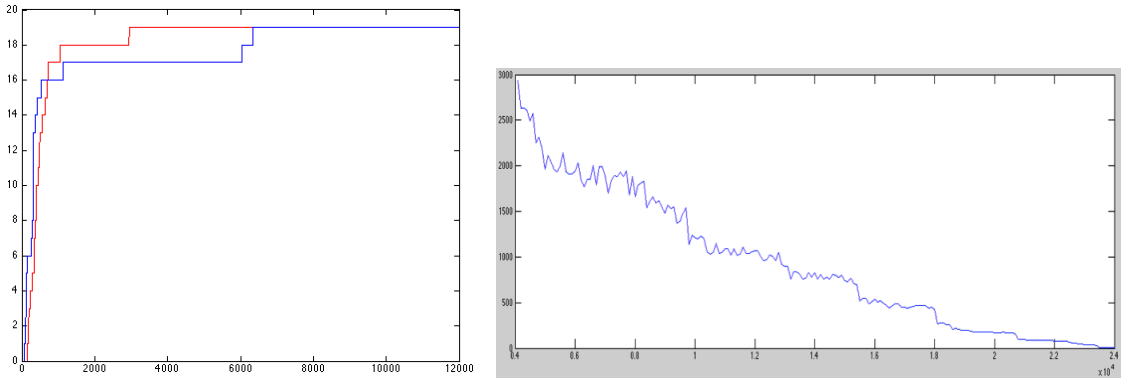


Fig. 11.12: Two ELMs generated from two MCMC chains C_1 and C_2 initialized at different starting points after convergence in $24,000$ iterations. © [2014] Maria Pavlovskaia. Reprinted, with permission, from ref. [26].

Once the GWL is believed to have converged, we can monitor the convergence of the ELM by checking the convergence of the following two sets over time t:

1. The set of leaf notes of the tree S_L^t in which each point x is a local minimum with energy $U(x)$. As t increase, S_L^t grows monotonically until no more local minima are found, as shown in Figure 11.13.(a).
2. The set of internal nodes of the tree S_N^t in which each point y is an energy barrier at level $U(y)$. As t increases, we may find a lower barrier as the Markov chain crosses different ridge between the basins. Thus $E(y)$ decreases monotonically until no barrier in S_N^t is updated during a certain time period.



(a) (b)

Fig. 11.13: Monitoring the convergence of ELMs generated from two MCMC chains C_1 and C_2 initialized at different starting points. (a) The number of local minima found vs number of iterations for C_1 and C_2. (b) the distance between the two ELMs vs. number of iterations. © [2014] Maria Pavlovskaia. Reprinted, with permission, from ref. [26].

We can calculate a distance measure between two ELMs constructed by two GWL implementations with different initializations. To do so, we compute a best node matching between the two trees and then the distance is defined on the differences of the matched leaf nodes and barriers, and penalties on unmatched nodes. Figure 11.13.(b) shows how the distance decreases as more samples are generated.

11.3 GWL Experiments

11.3.1 GWL mappings of Gaussian mixture models

In this section, we compute the ELMs for learning Gaussian mixture models for two purposes: (i) to study the influences of different conditions, such as separability and level of supervision, and (ii) to compare the behaviors and performances of popular algorithms including K-mean clustering, EM (Expectation-Maximization), two-step EM, and Swendsen-Wang cuts. We use both synthetic data and real data in the experiments.

11.3.1.1 GMM Energy and Gradient Computations

A Gaussian mixture model Θ with n components in d dimensions has weights $\{\alpha_i\}$, means $\{\mu_i\}$ and covariance matrices $\{\Sigma_i\}$ for $i = 1,\ldots,n$ as model parameters. Given a set of observed data points $\{z_i, i = 1,...,m\}$, we write the energy function as

$$U(\Theta) = -\log p(z_i : i = 1 \ldots m|\Theta) - \log p(\Theta) \tag{11.11}$$

$$= -\sum_{i=1}^{m} \log f(z_i|\Theta) - \log p(\Theta). \tag{11.12}$$

$p(\Theta)$ is the product of a Dirichlet prior and a NIW prior. Its partial derivatives are trivial to compute. $f(z_i|\Theta) = \sum_{j=1}^{n} \alpha_j G(z_i;\mu_j,\Sigma_j)$ is the likelihood for data z_i, where $G(z_i;\mu_j,\Sigma_j) = \frac{1}{\sqrt{\det(2\pi\Sigma_j)}} \exp\left[-\frac{1}{2}(z_i-\mu_j)^T \Sigma_j^{-1}(z_i-\mu_j)\right]$ is a Gaussian model. In case a data point is labeled (i.e. it is known from which component it was sampled), the likelihood is simply $G(z_i;\mu_j,\Sigma_j)$.

During gradient descent, we need to restrict the Σ_j matrices so that each inverse Σ_j^{-1} exists in order to have a defined gradient. Each Σ_j is semi-positive definite, so each eigenvalue is greater than or equal to zero. Consequently we only need the minor restriction that for each eigenvalue λ_i of Σ_j, $\lambda_i > \varepsilon$ for some $\varepsilon > 0$. However, it is possible that after one gradient descent step, the new GMM parameters will be outside of the valid GMM space, i.e. the new Σ_j^{t+1} matrices at step $t+1$ will not be symmetric positive definite. Therefore, we need to project each Σ_j^{t+1} into the symmetric positive definite space with the projection

$$P_{\text{pos}}(P_{\text{symm}}(\Sigma_j^{t+1})).$$

The function $P_{\text{symm}}(\Sigma)$ projects the matrix into the space of symmetric matrices by

$$P_{\text{symm}}(\Sigma) = \frac{1}{2}(\Sigma + \Sigma^T).$$

Assuming that Σ is symmetric, the function $P_{\text{pos}}(\Sigma)$ projects Σ into the space of symmetric matrices with eigenvalues greater than ε. Because Σ is symmetric, it can be decomposed into $\Sigma = Q\Lambda Q^T$ where Λ is the diagonal eigenvalue matrix $\Lambda = \text{diag}\{\lambda_1,\ldots,\lambda_n\}$, and Q is an orthonormal eigenvector matrix. Then the function

$$P_{\text{pos}}(\Sigma) = Q \begin{pmatrix} \max(\lambda_1,\varepsilon) & 0 & \ldots & 0 \\ 0 & \max(\lambda_2,\varepsilon) & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \max(\lambda_n,\varepsilon) \end{pmatrix} Q^T$$

ensures that $P_{\text{pos}}(\Sigma)$ is symmetric positive definite.

11.3.1.2 Experiments on Synthetic GMM Data

We start with synthetic data with a $n = 3$ component GMM in 2 dimensional space, draw m samples and run our algorithm to plot the ELM under different settings.

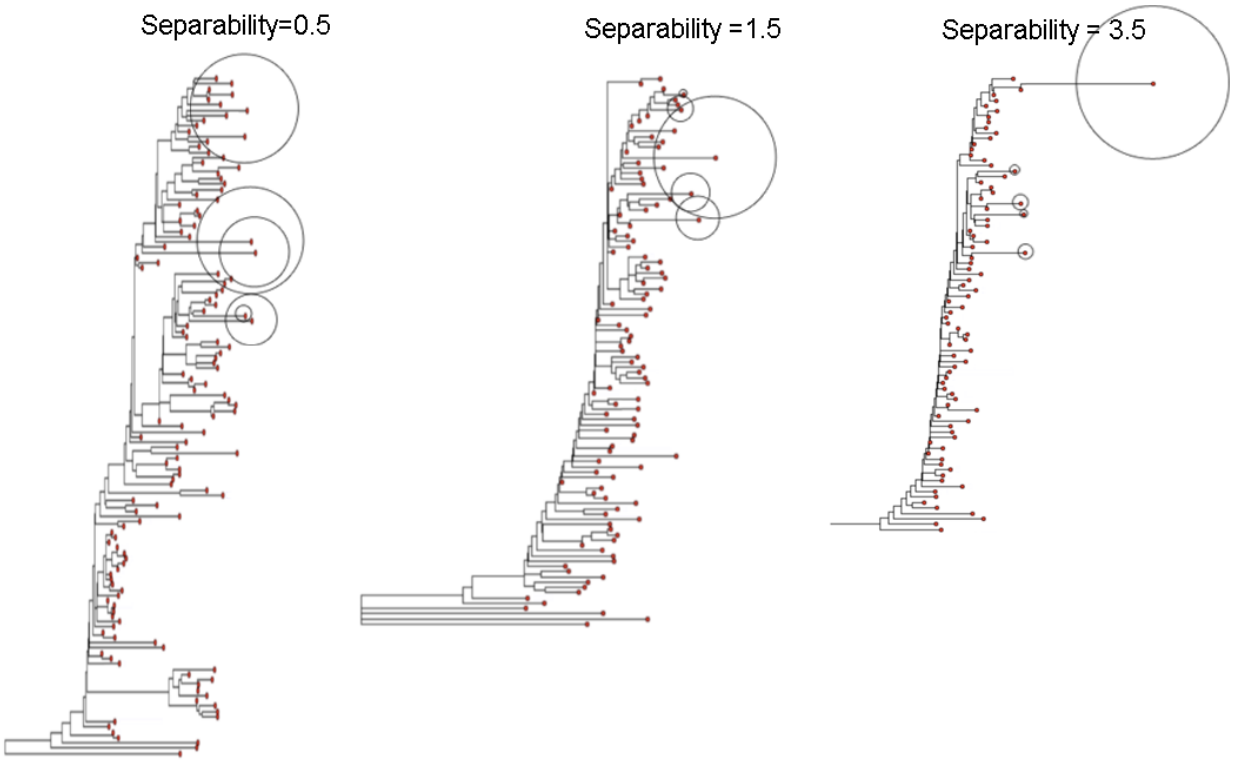


Fig. 11.14: ELMs for $m = 100$ samples drawn from GMMs with low, medium and high separability $c = 0.5, 1.5, 3.5$ respectively. The circles represent the probability mass of the basins. © [2015] Springer. Reprinted, with permission, from ref. [27].

1) The effects of separability. The separability of the GMM represents the overlap between components of the model and is defined as $c = \min\left(\frac{||\mu_i - \mu_j||}{\sqrt{n}\max(\sigma_1, \sigma_2)}\right)$. This is often used in the literature to measure the difficulty of learning the true GMM model.

Figure 11.14 shows three representative ELMs with the separability $c = 0.5, 1.5, 3.5$ respectively for $m = 100$ data points. This clearly shows that at $c = 0.5$, the model is hardly identifiable with many local minima reaching similar energy levels. The energy landscape becomes increasingly simple as the separability increases. When $c = 3.5$, the prominent global minimum dominates the landscape.

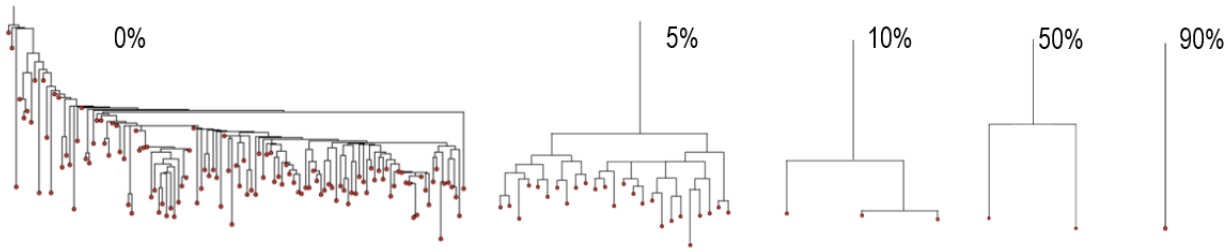


Fig. 11.15: ELMs with of synthesized GMMs (separability $c = 1.0$, nSamples = 100) with $\{0\%, 5\%, 10\%, 50\%, 90\%\}$ labeled data points. © [2015] Springer. Reprinted, with permission, from ref. [27].

2) The effects of partial supervision. We assign ground truth labels to a portion of the m data points. For z_i, its label ℓ_i indicates which component it belongs to. We set $m = 100$, separability $c = 1.0$. Figure 11.15 shows the ELMs with $0\%, 5\%, 10\%, 50\%, 90\%$ data points labels. While unsupervised learning (0%) is very challenging, it becomes much simpler when 5% or 10% data are labeled. When 90% data are labeled, the ELM has only one minimum. Figure 11.16 shows the number of local minima in the ELM when labeling $1, \ldots, 100$ samples. This shows a significant decrease in landscape complexity for the first 10% labels, and diminishing returns from supervised input after the initial 10%.

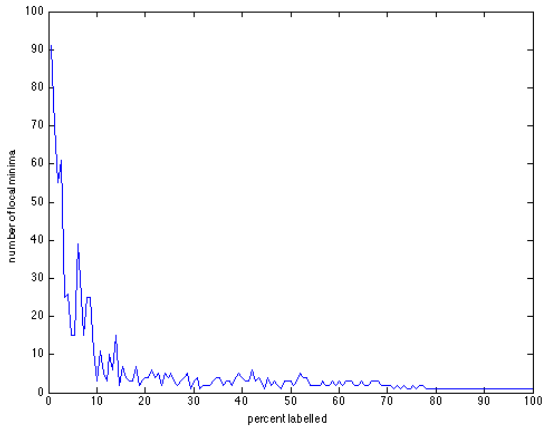


Fig. 11.16: Number of local minima versus the percentage of labeled data points for a GMM with separability $c = 1.0$. © [2015] Springer. Reprinted, with permission, from ref. [27].

3) Behavior of Learning Algorithms. We compare the behaviors of the following algorithms under different separability conditions.

- Expectation-maximization (EM) is the most popular algorithms for learning GMM in statistics.
- K-means clustering is a popular algorithm in machine learning and pattern recognition.
- Two-step EM is a variant of EM proposed in [9] who have proved a performance guarantee under certain separability conditions. It starts with an excessive number of components and then prunes them.
- The Swedsen-Wang Cut (SW-cut) algorithm proposed in [1]. This generalizes the SW method [31] from Ising/Potts models to arbitrary probabilities.

We modified EM, two-step EM and SW-cut in our experiments so that they minimize the energy function defined in Equation 11.11. K-means does not optimize our energy function, but it is frequently used as an approximate algorithm for learning GMM and therefore we include it in our comparison.

For each synthetic dataset in the experiment, we first construct the ELM, and then ran each of the algorithms for 200 times and record which of the energy basins the algorithm lands to. Hence we obtain the visiting frequency of the basins by each algorithm, which are shown as bars of varying length at the leaf nodes in Figures 11.17 and 11.18.

Figure 11.17 shows a comparison between the K-means, EM and two-step EM algorithms for $n = 10$ samples drawn from a low ($c = 0.5$) separability GMM. The results are scattered across different local minima regardless of the algorithm. This illustrates the difficulty in learning a model from a landscape with many local minima separated by large energy barriers.

Figure 11.18 show a comparison of the EM, k-means, and SW-cut algorithms for $m = 100$ samples drawn from low ($c = 0.5$) and high ($c = 3.5$) separability GMMs. The SW-cut algorithm performs best in each situation, always converging to the global optimal solution. In the low separability case, the k-means algorithm is quite random, while the EM algorithm almost always finds the global minimum and thus outperforms k-means. However, in the high separability case, the k-means algorithm converges to the true model the majority of the time, while the EM almost always converges to a local minimum with higher energy than the true model. This result confirms a recent theoretical result showing that the objective function of hard-EM (with k-means as a special case) contains an inductive bias in favor of high-separability models [29, 33]. Specifically, one can show that the actual energy function of hard-EM is:

$$U(\Theta) = -\log p(\Theta|Z) + \min_q \left(\mathbf{KL}(q(L)||p(L|Z,\Theta)) + H_q(L)\right)$$

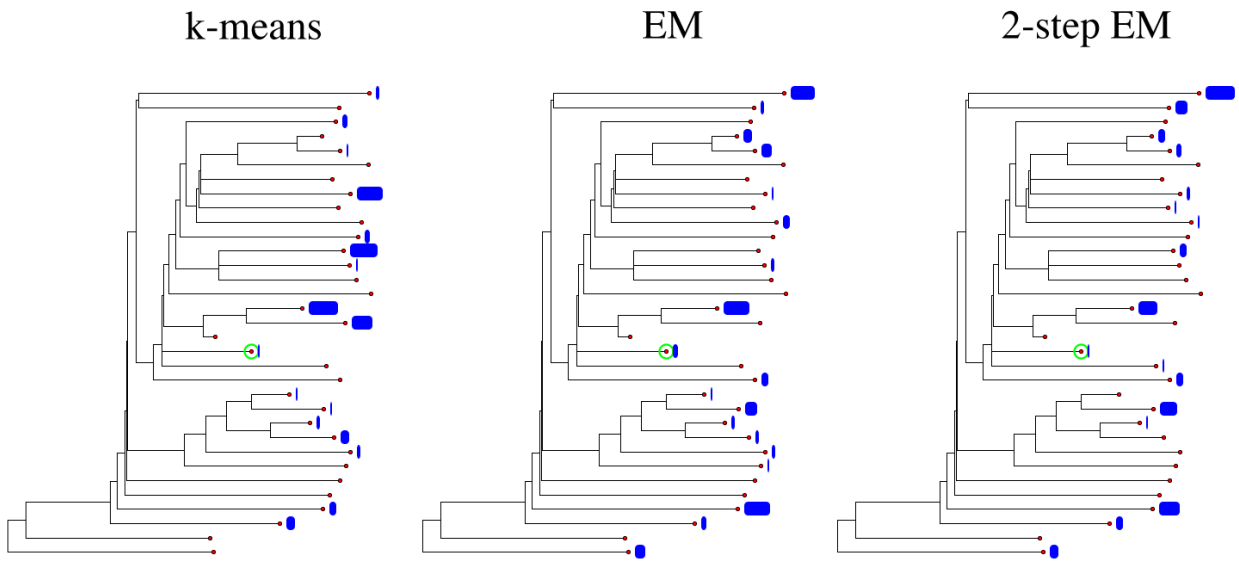


Fig. 11.17: The performance of the k-means, EM and 2-step EM algorithms on the ELMs with 10 samples drawn from a GMM with low separability ($c = 0.5$). © [2014] Maria Pavlovskaia. Reprinted, with permission, from ref. [26].

where Θ contains the model parameters, $Z = \{z_1, \ldots, z_m\}$ is the set of observable data points, L is the set of latent variables (the data point labels in a GMM), q is an auxiliary distribution of L, and H_q is the entropy of L measured with $q(L)$. The first term in the above formula is the standard energy function of clustering with GMM. The second term is called a posterior regularization term [12], which essentially encourages the distribution $P(L|Z,\Theta)$ to have a low entropy. In the case of GMM, it is easy to see that a low entropy in $P(L|Z,\Theta)$ implies high separability between Gaussian components.

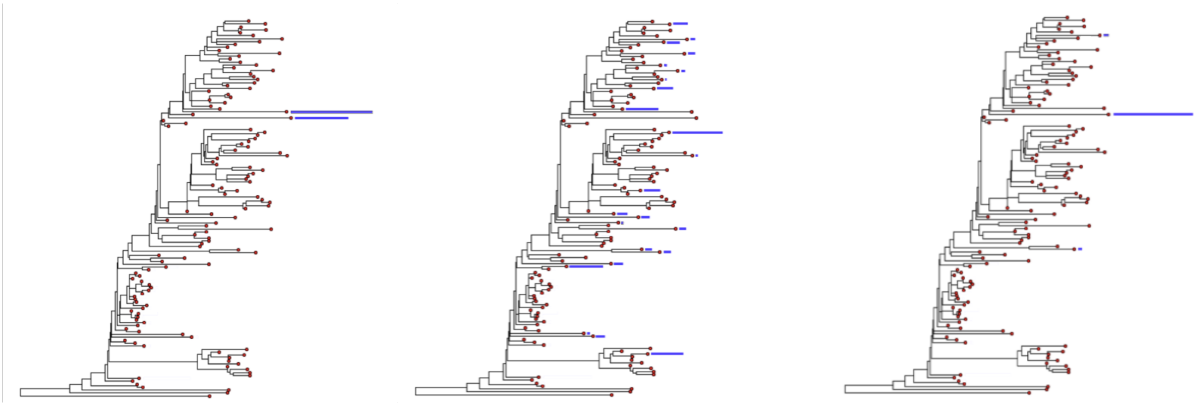
11.3.1.3 Experiments on GMMs of Real Data

We ran our algorithm to plot the ELM for the well-known Iris dataset from the UCI repository [4]. The Iris dataset contains 150 points in 4 dimensions and can be modeled by a 3-component GMM. Each of the three components represents a type of iris plant and the true component labels are known. The points corresponding to the first component are linearly separable from the others, but the points corresponding to the remaining two components are not linearly separable.

Figure 11.19 shows the ELM of the Iris dataset. We visualize the local minima by plotting the ellipsoids of the covariance matrices centered at the means of each component in 2 of the 4 dimensions.

The 6 lowest energy local minima are shown on the right and the 6 highest energy local minima are shown on the left. The high energy local minima are less accurate models than the low energy local minima. The local minima (E) (B) and (D) have the first component split into two and the remaining two (non-separable) components merged into one. The local minima (A) and (F) have significant overlap between the 2nd and 3rd components and (C) has the components overlapping completely. The low-energy local minima (G-L) all have the same 1st component and slightly different positions of the 2nd and 3rd components.

We ran the algorithm with $0, 5, 10, 50, 90, 100$ percent of the points with the ground truth labels assigned. Figure 11.20 shows the global minimum of the energy landscape for these cases.



(a) EM (b) k-means (c) SW-cut

(d) EM (e) k-means (f) SW-cut

Fig. 11.18: The performance of the EM, k-means, and SW-cut algorithm on the ELM. (a-c) Low separability $c = 0.5$. (d-f) High separability $c = 3.5$. © [2015] Springer. Reprinted, with permission, from ref. [27].

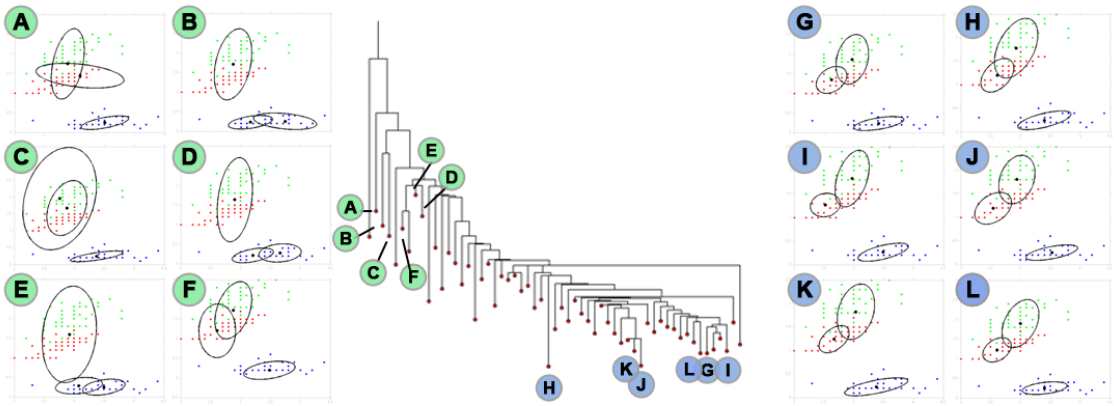


Fig. 11.19: ELM and some of the local minima of the Iris dataset. © [2015] Springer. Reprinted, with permission, from ref. [27].

11.3.2 GWL mapping of grammar models

11.3.2.1 Learning Dependency Grammars

A dependency grammar models the syntactic structure of a sentence via a set of dependency relations between the words of the sentence (Figure 11.21). Dependency grammars have been widely used in natural language syntactic parsing, especially for languages with free word order [8, 20, 24]. A dependency grammar contains a special root node and a set of n other nodes that represent the words of a language. The grammar contains the following parameters: 1.

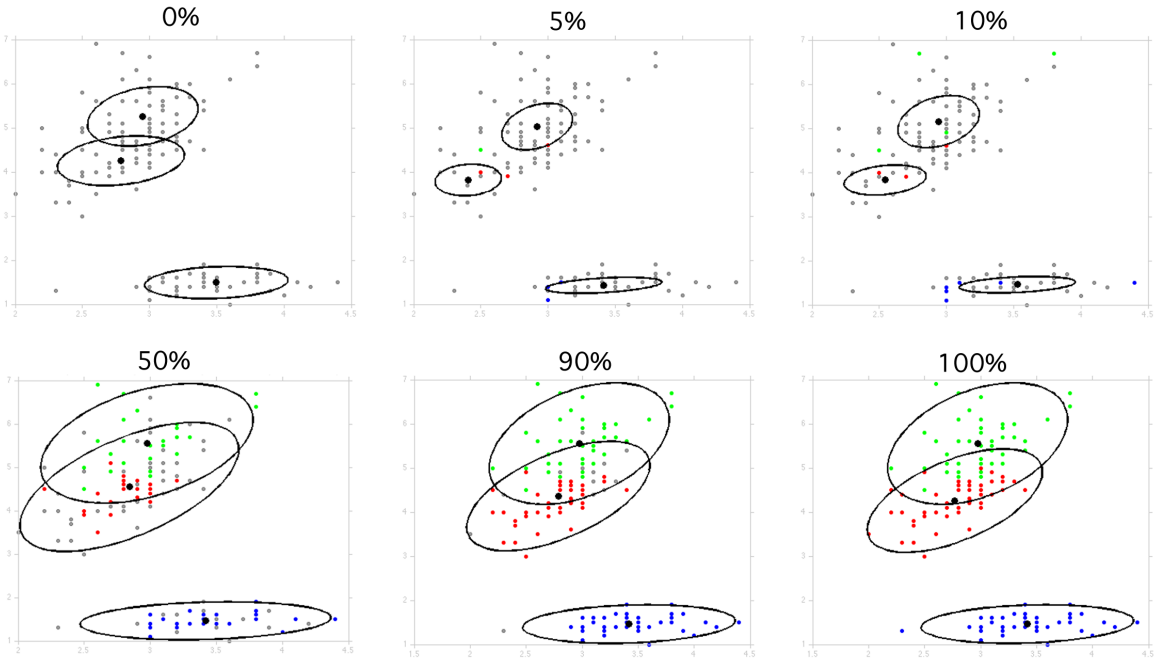


Fig. 11.20: Global minima for learning from the Iris dataset with 0, 5, 10, 50, 90, and 100% of the data labeled with the ground truth values. Unlabeled points are drawn in grey and labelled points are colorized in red, green or blue. © [2014] Maria Pavlovskaia. Reprinted, with permission, from ref. [26].

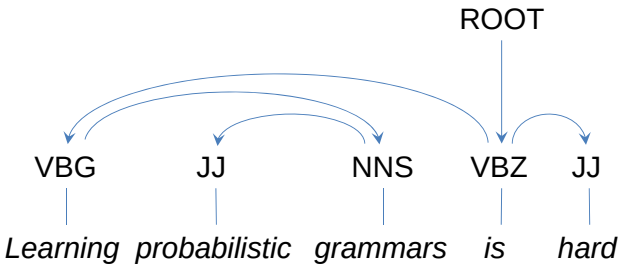


Fig. 11.21: The grammatical structure generated by a dependency grammar. © [2014] Maria Pavlovskaia. Reprinted, with permission, from ref. [26].

the vector of transition probabilities from the root node to the word nodes; 2. the transition probability matrix between the word nodes; and 3. the probabilities of each node continuing or stopping the generation of child nodes in the left and right directions. Hence the space of dependency grammars with n nodes has $n^2+n+2\cdot 2\cdot n$ dimensions. Since each probability vector is constrained to sum up to 1, the valid dependency grammars form a subspace of dimensionality n^2+2n-1. To generate a sentence using the dependency grammar, one starts with the root node and recursively generates child nodes from each node; the child node generation process at each node is controlled by the continuing/stopping probabilities (whether to generate a new child node or not) as well as the transition probabilities (what child node to generate). The generation process can be represented by a parse tree, such as the one shown in Figure 11.21. The probability of a parse tree is the product of the probabilities of all the choices during generation. The probability of a sentence is the sum of the probabilities of all possible parse trees of the sentence.

There has been increasing interest in learning dependency grammars from data, in either a supervised way (e.g. [7, 8]) or an unsupervised way (e.g., [15, 19]). The learning problem is typically nonconvex, especially in the unsupervised case in which the dependency parse of a training sentence is latent. Most of the learning algorithms try to identify a local optimum and there is little theoretical analysis as to the quality of such local optima.

Most of the existing automatic approaches to learning dependency grammars start with all the sentences of a training corpus and try to learn the whole grammar. In contrast, humans learn the grammar of their native language in a very different manner: they are exposed to very simple sentences as infants and then to increasingly more complex sentences as they grow up. Such a learning strategy has been termed *curriculum learning* [3]. Earlier research into curriculum

learning of grammars produced both positive [11] and negative results [28]. More recently, [30] empirically showed that curricula are helpful in unsupervised dependency grammar learning.

To explain the benefits of curricula, [32] suggest that an ideal curriculum gradually emphasizes data samples that help the learner to successively discover new grammar rules of the target grammar, which facilitates the learning. Another explanation that is possibly compatible with the previous one is given by [3], who hypothesize that a good curriculum corresponds to learning starting with a smoothed objective function and gradually reducing the degree of smoothing over the curriculum stages, thus guiding the learner to better local minima of the energy function.

11.3.2.2 Energy Function of Dependency Grammar

The energy function for unsupervised learning of dependency grammars is $U(\theta) = -\log p(\theta|D)$ where θ is the parameter vector of the grammar and D is the set of training sentences. $\log p(\theta|D)$ is the logarithmic posterior probability of the grammar, which is defined as:

$$\log p(\theta|D) = \sum_{x \in D} \log p(x|\theta) + \log p(\theta)$$

where $p(x|\theta)$ is the probability of sentence x as defined in the previous section and $p(\theta)$ is the Dirichlet prior.

11.3.2.3 Discretization of the Dependency Grammar Hypothesis Space

From our experiments, we found that the continuous hypothesis space of dependency grammars cannot be efficiently traversed by the WL algorithm even when the number of nodes n is small, because

- The number of local minima in the space is too large (number of local minima still growing linearly after 100,000 iterations);
- Gradient computation is slow, especially for long sentences, and the gradient is typically computed over 100 times per iteration;
- The rejection rate is over 90%, so less than 10 percent of proposed MCMC moves are accepted.

To solve or alleviate these problems (especially the first two), we propose to discretize the parameter space. The discretization reduces the number of local minima and replaces gradient descent with steepest descent which is much more computationally efficient. The discretized ELM is an approximation of the original ELM that still conveys useful information about the landscape.

We discretize the parameter space in the following way: let Ω_r be the discretized parameter space with discretization resolution $r > 4$:

$$\Omega_r = \{\vec{\theta} = [\theta_1, \ldots, \theta_{n^2+n+4n}] \mid \theta_i \in \{0, \frac{1}{r}, \frac{2}{r}, \ldots, \frac{r-1}{r}, 1\} \text{ and } \sum_{j \in I_k} \theta_j = 1\}.$$

where index set I_k ranges over all the probability vectors in $\vec{\theta}$.

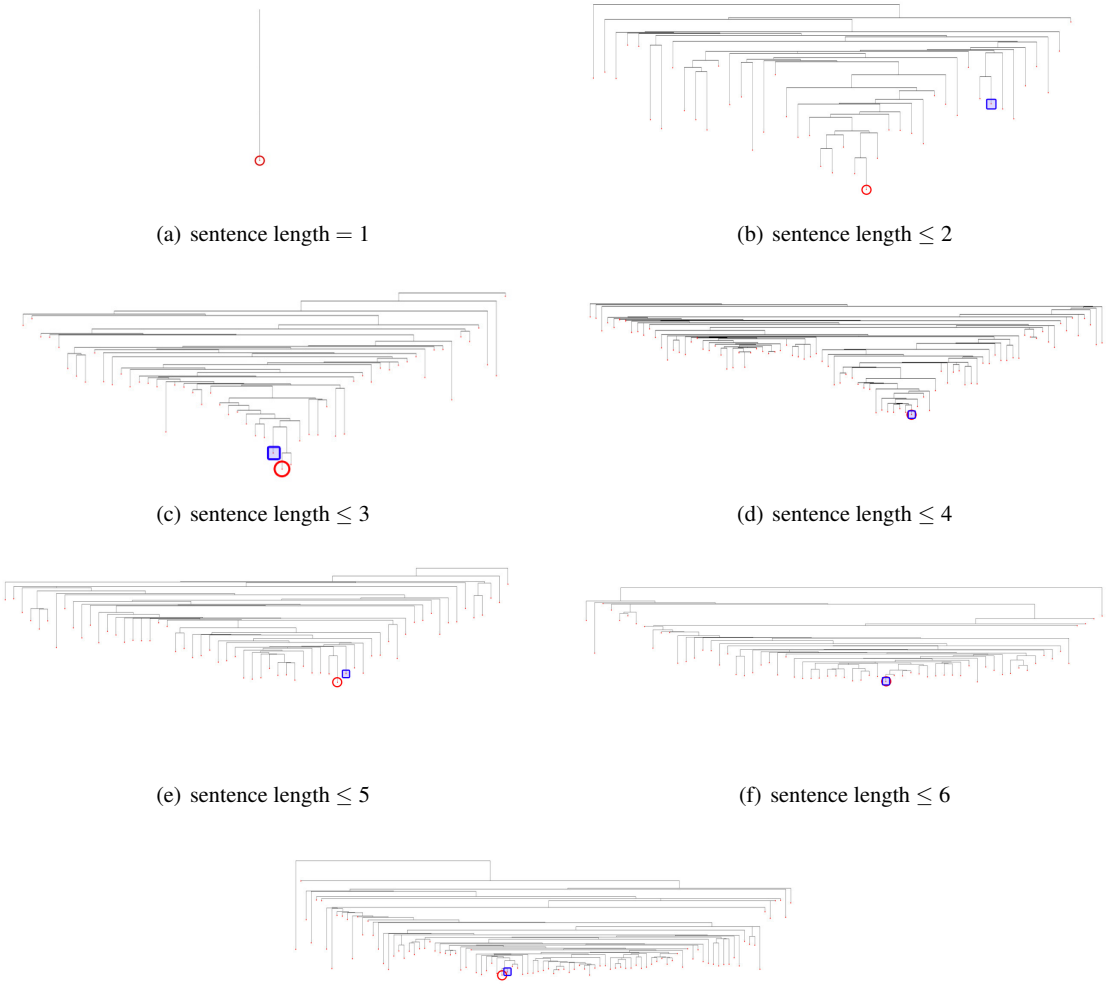
In the discrete space, we perform steepest descent (in lieu of gradient descent) to find the local minima. Given $\theta_t = [\theta_1, \ldots, \theta_{n^2+n+4n}] \in \Omega_r$, let $\theta_t^{(i,j)} = [\theta_1, \ldots, \theta_i - \frac{1}{r}, \ldots, \theta_j + \frac{1}{r}, \ldots, \theta_{n^2+n+4n}]$ for every ordered pair (i, j) that index probabilities in the same probability vector $i, j \in I_k$. One steepest descent step is given by

$$\theta_{t+1} = \text{argmin}_{(i,j)} \left(E\left(\theta_t^{(i,j)}\right) \mid i, j \in I_k \text{ for some } k \right).$$

The steepest descent algorithm terminates when $\theta_t \leq \theta_{t+1}$ for some t, indicating that θ_t is a local minimum in the discrete space.

For the proposal distribution $Q(\theta_t, \theta')$ in the generalized Wang-Landau algorithm, we use a uniform distribution over the space of all θ' generated by selecting two probabilities from the same probability vector in θ_t, adding $\frac{1}{r}$ to the first, and subtract $\frac{1}{r}$ from the second.

When we attempt to run the naive implementation of the discretized algorithm, two issues emerge: (1) there are multiple discrete local minima found that belong to the same energy basin in the continuous space, (2) the energies of a local minimum in the discrete space may be poor approximations to the energy of the corresponding local minimum in the continuous space if the gradient is steep at the discrete local minimum. Therefore, we employ a hybrid discrete-continuous approach. We run the main algorithm loop in the discrete space, and after each sample θ_t is accepted, we: (1) perform steepest descent in the discretized space initialized with θ_t to find the discrete local minimum θ_t^*; (2) perform gradient descent in the continuous space initialized with θ_t^* to find the more accurate local minimum θ_t'. The use of the discrete space limits the number of local minima and the number of gradient descent computations and the subsequent use of the continuous space merges the discrete local minima belonging to the same continuous energy basin. To improve the energy boundary estimations, we repeat the following two steps until convergence: run ridge descent on the discrete mesh and refine the discretization by a factor of 2.



(a) sentence length $= 1$

(b) sentence length ≤ 2

(c) sentence length ≤ 3

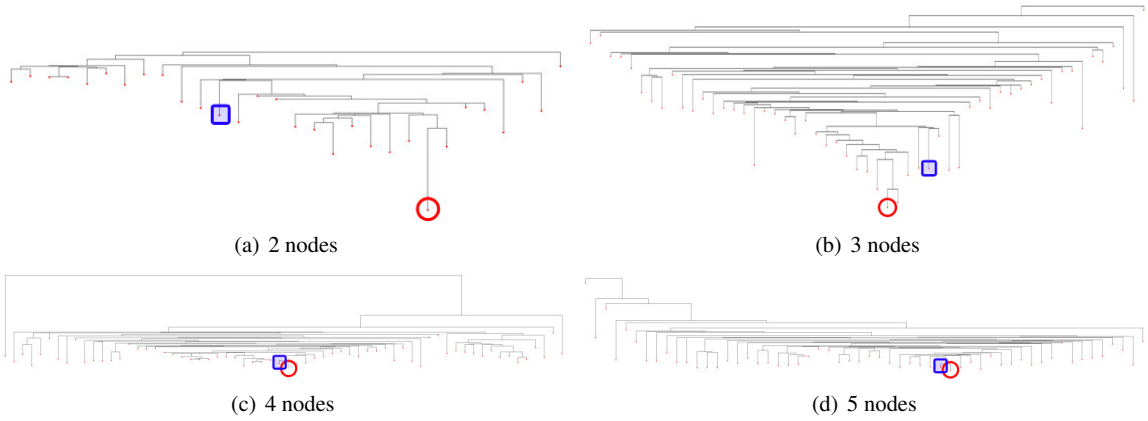
(d) sentence length ≤ 4

(e) sentence length ≤ 5

(f) sentence length ≤ 6

(g) sentence length ≤ 7

Fig. 11.22: Curriculum based on training sample sentence length. © [2014] Maria Pavlovskaia. Reprinted, with permission, from ref. [26].



(a) 2 nodes (b) 3 nodes

(c) 4 nodes (d) 5 nodes

Fig. 11.23: Curriculum based on number of nodes. (a) 2 nodes. (b) 3 nodes. (c) 4 nodes. (d) 5 nodes

11.3.2.4 GWL Dependency Grammar Experiments and Curriculum Learning

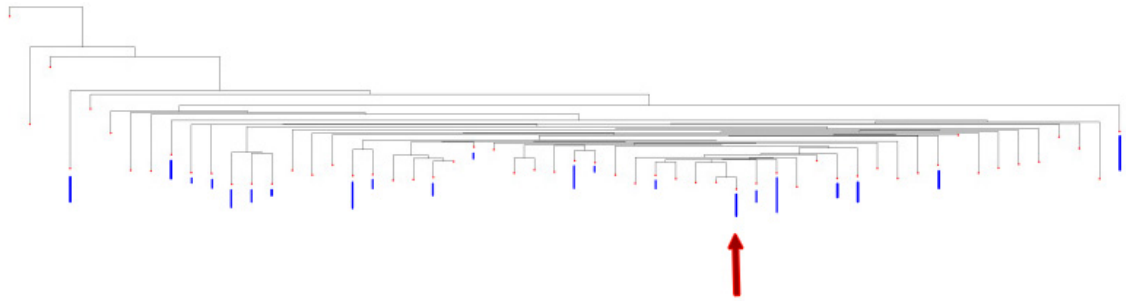
We constructed several dependency grammars of simple English syntax by simplifying a treebank grammar learned from the WSJ corpus of the Penn Treebank. Each node in the dependency grammar represents a part-of-speech tag such as Noun, Verb, Adjective and Adverb. The simplification was done by removing the nodes that appear with the least frequency in the WSJ corpus.

We first explored a curriculum based on sample sentence length. We used a 3-node dependency grammar and discretized the hypothesis space using discretization factor $r = 10$. Denote this grammar by θ_e. Next we sampled $m = 200$ sentences $D = \{x_j | j = 1, \ldots, 200\}$ from θ_e. We define $D_i \subset D$ to be the set of all sentences x_j containing i words or less. Let $w(x_j)$ be the word count of the sentence x_j, then $D_i = \{x_j | w(x_j) \leq i\}$. The sets D_i are nested ($D_i \subseteq D_{i+1}$) and $\cup_i^\infty D_i = D$. In the curriculum learning process, the i-th stage employs the train set of D_i. Figures 11.22 (a-g) show the Energy Landscape maps of the curriculum stages 1 through 7.

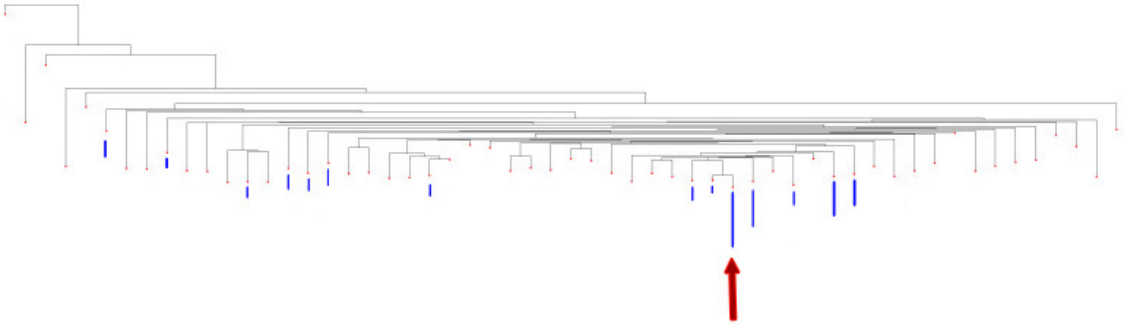
Next, we explored a curriculum based on the number of nodes n in the grammar. We used a 5-node dependency grammar and its simplifications to $n = 4, 3, 2, 1$ nodes with discretization factor $r = 10$. We sampled $m = 200$ sentences $D_i = \{x_j | j = 1, \ldots, 200\}$ from each grammar $\theta_i, i = 1, \ldots, 5$. Again, the i-th stage of curriculum learning employs the train set of D_i. Figures 11.23 (a-d) show the Energy Landscape maps of the curriculum stages 2 through 5. The ELM for Stage 1 is omitted because it is the same as the ELM in Figure 11.22 (a) due to the convexity of the landscape.

For both curricula (based on the sentence length and based on the number of nodes in the grammar), we observe that the ELM becomes more complex in the later stages of the curricula; the landscapes in the later stages are flatter and have more local minima. In each ELM shown in the figures, the global minimum is highlighted in red and the closest local minimum to the global minimum of the previous curriculum stage is highlighted in blue. It is evident that for stages 3-7 of the curriculum based on sentence lengths (Figures 11.22 c-g) and stages 3-5 of the curriculum based on the number of nodes (Figures and 11.23 b-d), the global minimum from curriculum stage i is close to the global minimum of stage $i+1$. This provides an explanation for the performance benefit of curriculum learning: early stages (which can be learned more easily) provide a good starting guess for later stages, which allows later stages to converge to a better local minimum, which also results in less computation time overall.

Finally, we ran the expectation-maximization learning algorithm on the training data to confirm the advantage of curriculum learning. The second curriculum (based on the number of nodes) is used in the experiments. To speed up curriculum learning, we allot $18,000$ seconds total running time for each run and assign each successive stage twice as much time as the previous stage. The exponentially increasing time in the curriculum design is chosen because the complexity of the later stages requires more time to converge. We ran the learning algorithm for $1,000$ times and found the energy basins of the ELM that the learned models belong to. Hence we obtained a histogram of the learned models on the leaf nodes of the ELM as shown in Figure 11.24 (b). For comparison, Figure 11.24 (a) shows the histogram of the models learned without using a curriculum. The utilization of the curriculum results in more frequent convergence to the global minimum as well as energy basins near the global minimum.



(a) no curriculum



(b) exponentially decreasing curriculum

Fig. 11.24: Distribution of learned grammars (a) without a learning curriculum (b) with the time-constrained curriculum. The blue bars histogram the number of learned grammars belonging to each energy basin, the red arrow indicates the energy basin of the ground truth solution. © [2014] Maria Pavlovskaia. Reprinted, with permission, from ref. [26].

11.4 Mapping the Landscape with Attraction-Diffusion

A major obstacle for ELM applications is the astronomical number of local minima that exist in many of the non-convex landscapes encountered in practice. Mappings that use the partition defined in Section 11.1.1 cannot converge when it is impossible to fully enumerate all local modes, and the overabundance of local minima and barrier information can make the DG visualization too complex to be a useful summary of the landscape.

A highly non-convex landscape can still have a simple and recognizable *global* structure. A well-known example is the "funnel" structure of potential energy surfaces associated with protein folding. A funnel shape is well-suited for guiding an unfolded or partially-folded protein to its native state. Weakly stable intermediate states might occur along the folding path, but random perturbations from the environment are enough to upset these shallow minima and allow the folding process to continue. Once the protein has reached its native state, its configuration is stable and resistant to small perturbations. The macroscopic landscape has a single global basin, despite the astronomical number of weakly stable intermediate states along the "sides" of the funnel.

Motivated by this observation, one can define a new framework for mapping that seeks to identify *macroscopic* structure in non-convex landscapes while ignoring the noisy local structure. Intuitively, one can imagine the high-dimensional state space Ω as a vast and mostly empty universe, the non-convex energy U as gravitational potential energy, and the local minima of U as dense stars. Groups of related local minima separated by low energy barriers (states with similar properties in the system) form connected low-energy regions, which are "galaxies" in the state space "universe" (see Figure 11.25). There are usually only a handful of minima "galaxies" in an energy landscape, each of which contains a virtually infinite number of local minima with nearly identical system properties. Defining basins as minima "galaxies" rather than defining a separate basin for each local minima can vastly improve mapping efficiency and interpretability.

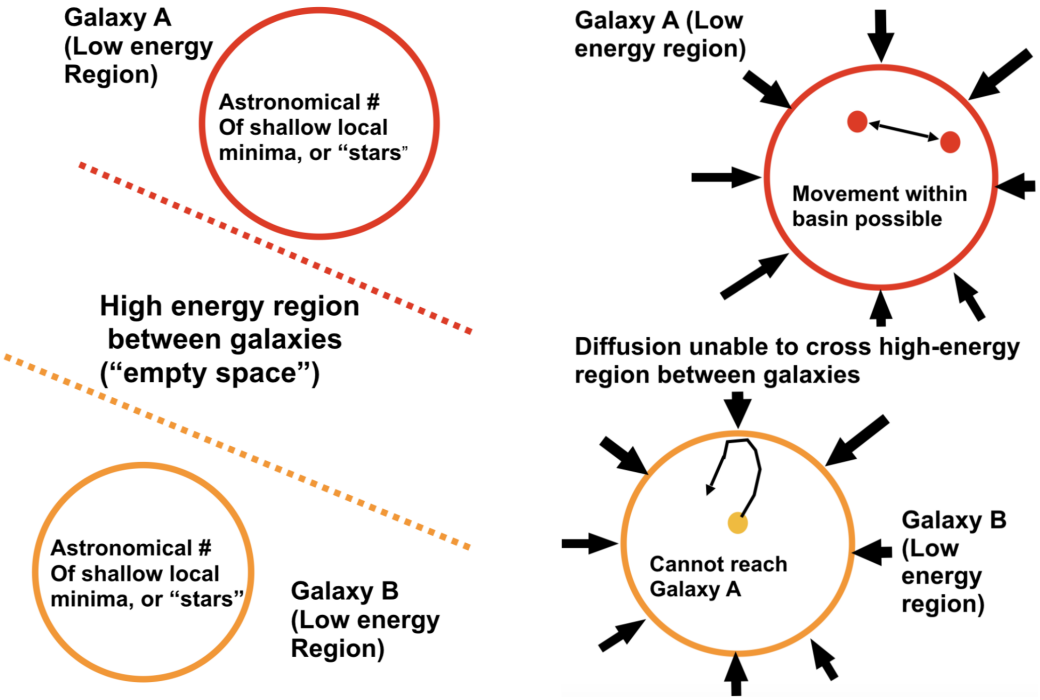


Fig. 11.25: *Left:* Simplified diagram of local minima "galaxies". The circles are low-dimensional manifolds with high density. Between the galaxies are high-energy, "empty" regions (in practice, empty space is vast compared to galaxy size). *Right:* Diagram of metastable behavior within image galaxies. The Attraction-Diffusion algorithm is used to detect such behavior.

11.4.1 Metastability and a macroscopic partition

"Galaxies" of local minima in non-convex landscapes can be formalized using the concept of *metastability*. Instead of dividing the state space into basins of attraction for each local minimum, one can divide the image space into metastable regions such that (1) a diffusion process on U mixes over short time-scales within a region, and (2) a diffusion process on U mixes over long time-scales between regions. In other words, a local MCMC sample of p initiated from a minima galaxy will travel in the same galaxy for a very long time, because random fluctuations are enough to overcome small energy barriers within the galaxy, while much larger energy barriers restrict movement between galaxies.

Following Bovier [5], one can formally define the "galaxies" as disjoint sets $\{D_j\}_{j=1}^J$ such that $\cup_{j=1}^J D_j \subset \Omega$ and

$$\frac{\sup_{x \notin \cup_j D_j} \mathrm{E}[\tau(x, \cup_j D_j)]}{\inf_{x \in \cup_j D_j} \mathrm{E}[\tau(x, \cup_{k \neq \varphi(x)} D_k)]} < \varepsilon \tag{11.13}$$

where $\tau(x,S)$ is the hitting time for a set S starting from a point x, $\varphi(x)$ gives the index j for which $x \in D_j$, and $\varepsilon > 0$ is a small parameter that characterizes the degree of metastability. The numerator term quantifies how strongly the galaxies attract states in $\Omega \setminus (\cup_{j=1}^J D_j)$. An MCMC sample initialized outside of the galaxies should visit one of the galaxies in a short time period. The denominator quantifies the separation between galaxies in terms of expected mixing time between galaxies. An MCMC sample initiated from any D_j should visit another D_k only after a long period of diffusion. Metastability is satisfied when the mixing time between galaxies is slow relative to the attractor time from outside of the galaxies. Crucially, the definition of hitting time $\tau(x,S)$, and by extension the definition of the partition $\{D_j\}_{j=1}^J$, implicitly depends on the MCMC sampler that is used. When detecting metastability, it is actually desirable to use a sampler with *poor* mixing rather than good mixing, because the definition of metastability presented here actually relies on the high auto-correlation exhibited by MCMC samples when trapped in strong modes.

11.4.2 Introduction to attraction-diffusion

Attraction-Diffusion (AD) [16] is a method for characterizing the relative stability of local minima in highly non-convex landscapes. Given an energy function U and two local minima, one minimum is designated as the starting location X_0 and the other as the target location X^*. An MCMC sample is initiated from X_0 using an altered density

$$p_{T,\alpha,X^*}(X) = \frac{1}{Z_{T,\alpha,X^*}} \exp\{-(U(X)/T + \alpha||X - X^*||_2)\} \tag{11.14}$$

whose energy function is the sum of the original energy U and a "magnetization" term penalizing the distance between the current state and the target location X^*. T gives the temperature of the system and α is the strength of the "magnetic field" penalizing distance from the target minimum. The roles of starting and target location are arbitrary and diffusion in both directions is possible. The idea is illustrated in Figure 11.26. The state space Ω can be continuous or discrete.

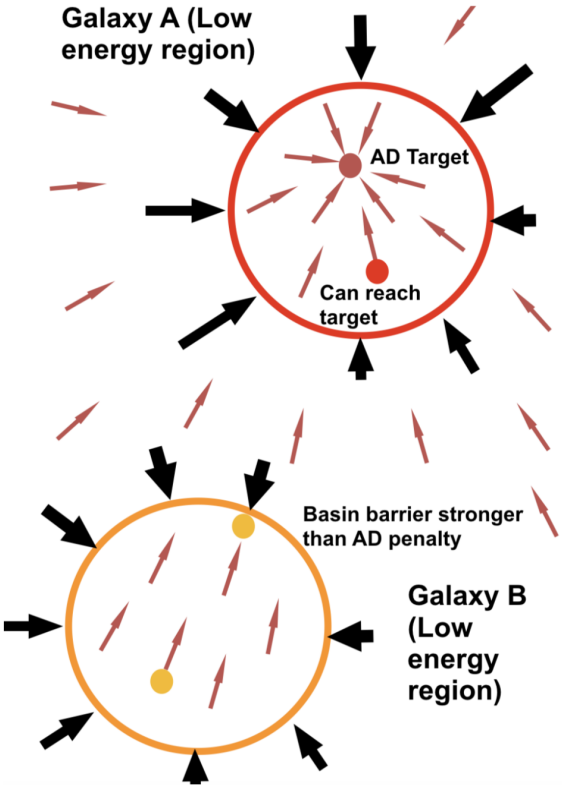


Fig. 11.26: Visualization of detecting metastable behavior with AD. The AD penalty points towards the target with constant strength α throughout the landscape. An MCMC sample initialized from the same image galaxy as the target will travel quickly to the target. An MCMC sample initiated from a different galaxy can approach the target for a short time, but will eventually be trapped by the strong obstacles that separate galaxies.

By adjusting the value of α and T, the altered landscape can be tuned so that a diffusion path can overcome small obstacles in the original landscape while remaining trapped in strong basins. If the Markov chain comes within a close distance of the target state, then the starting state belongs to the same energy basin as the target state at an energy resolution implicitly defined by the strength of magnetization. If the chain cannot improve on the minimum distance between the previous states of the chain and the target state for M consecutive iterations, then there must be an energy barrier between the starting and target location that is stronger than the force of the magnetization. Figure 11.27 demonstrates the basic principles of AD in a simple 1D landscape with two global basins.


Mitchell K. Hill

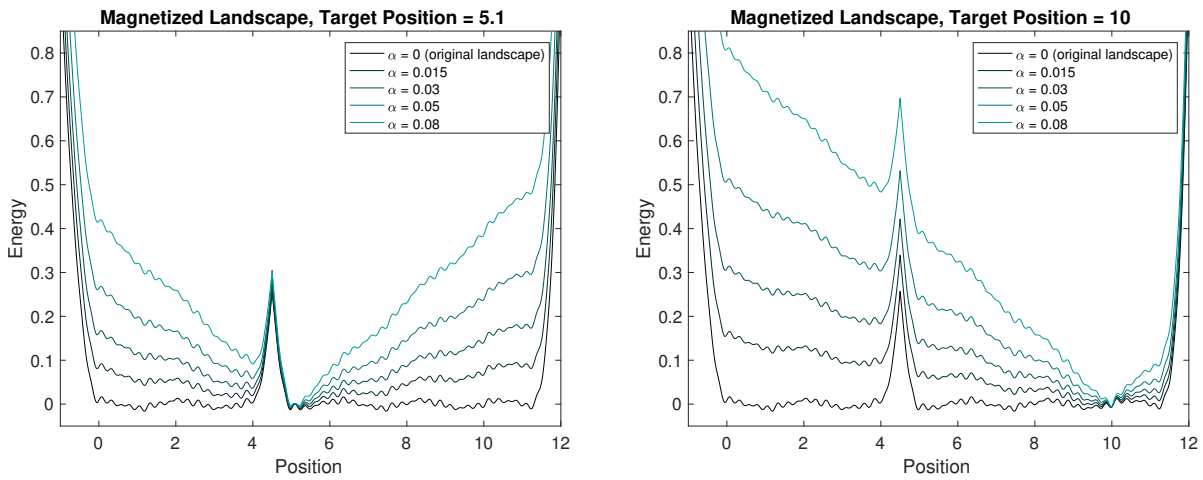


Fig. 11.27: Magnetization of a toy 1D landscape with target positions $X = 5.1$ (left) and $X = 10$ (right). The original landscape has two flat and noisy basins. Both target positions belong to the same basin, even though they are distant in Euclidean space. The magnetized landscapes have easily identifiable minima, and preserve the large barrier separating the two basins. Since diffusion in the left-hand landscape from initiated from $X = 10$ will reach $X = 5.1$, and vice-versa in the right-hand landscape, these points belong to the same basin. Low-temperature diffusion initiated from the left of the barrier will be unable to reach the target position in either landscape. © [2019] Brown University. Reprinted, with permission, from ref. [16].

AD is meant to accelerate mixing within metastable galaxies while preserving the long time-scales that separate galaxies. Metastable behavior is determined by whether states mix on a "slow" or "fast" time-scale, but the so-called "fast" time-scales of mixing within galaxies are still too long for efficient simulation. It is highly unlikely that two Markov chains will meet in a high-dimensional state space in a computationally feasible amount of time, even if they belong to the same metastable galaxy. The attraction term in AD alters the dynamics of the original system to decrease hitting time to the magnetization target. With correct tuning, this accelerated convergence should only occur for minima within the same galaxy and the long time-scales of mixing will remain even after magnetization for minima from different galaxies. As long as α is sufficiently small, the metastable partition $\{\tilde{D}_j\}_{j=1}^{\tilde{J}}$ of the altered density p_{T,α,X^*} should have approximately the same structure as the metastable partition $\{D_j\}_{j=1}^{J}$ of the original density p and metastable behavior on p_{T,α,X^*} should approximate metastable behavior on the target p.

AD can also be used to estimate the energy barrier between minima, since the maximum energy along a successful diffusion path is an upper bound for the minimum barrier height. This estimate can be refined by setting α just above the threshold where the diffusion path fails to reach the target. By using a *local* MCMC method such as Random-Walk Metropolis-Hastings, Component-Wise Metropolis Hastings, Gibbs sampling, or Hamiltonian Monte Carlo, one can limit the maximum Euclidean distance between points in the diffusion path and ensure that the step size is small enough so that the 1D landscape between successive images is well-behaved. An AD chain moves according to the geodesic distance in the magnetized landscape, which should be similar to the geodesic distance in the raw landscape as long as the strength of magnetization is not too strong.

The choice of the L_2-norm as the magnetization penalty is motivated by the observation that $\frac{d}{dX}||X||_2 = X/||X||_2$, which means that the AD magnetization force points towards the target minimum with uniform strength α throughout the energy landscape. This can be seen in the Langevin Equation

$$dX(t) = -\left(\nabla U(X(t))/T + \alpha \frac{X(t) - X^*}{||X(t) - X^*||_2}\right) dt + \sqrt{2}\, dW(t) \tag{11.15}$$

associated with the magnetized dynamics. An L_1 penalty would probably give similar results. The penalty $\alpha||X - X^*||_2^2$ would *not* have desirable properties because the strength of magnetization would depend on the distance between the points, and the magnitude of alteration would vary throughout the landscape.

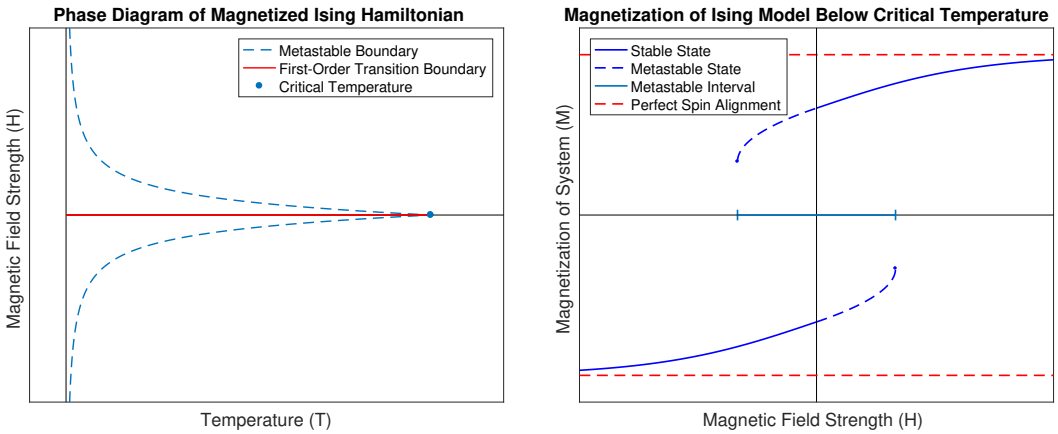


Fig. 11.28: *Left:* Phase diagram of the magnetized Ising model. Below the critical temperature, sweeping the magnetic field H from positive to negative (or vice-versa) results in a jump between the basins of σ^+ and σ^-. However, if the magnetization force is weak, states in the opposite basin can remain stable for long time periods. *Right:* Magnetization $M = \sum_i \sigma_i$ as a function of H for a fixed $T^* < T_c$. The metastable interval is the region between the dashed lines along the vertical line $T = T^*$ in the left figure. © [2019] Brown University. Reprinted, with permission, from ref. [16].

11.4.3 Attraction-diffusion and the Ising model

The AD penalty term is closely related to the magnetization term found in energy functions from statistical physics. Consider the N-state magnetized Ising energy function

$$U_{T,H}(\sigma) = -\frac{1}{T}\sum_{(i,j)\in\mathcal{N}} \sigma_i\sigma_j - H\sum_{i=1}^{N}\sigma_i \tag{11.16}$$

where $\sigma_i = \pm 1$, $\mathcal{N}$ is the set of neighboring nodes, $T > 0$ gives the temperature, and H gives the strength of an external magnetic field. This energy function is sometimes parameterized by the slightly different form $U_{T,H}(\sigma) = \frac{1}{T}\left(-\sum\sigma_i\sigma_j - H\sum\sigma_i\right)$, but the same properties and diagrams hold either way. The first term $-\frac{1}{T}\sum\sigma_i\sigma_j$ is the energy function of the standard Ising model, and $-H\sum\sigma_i$ represents a uniform magnetic field with strength H acting on each node. When $H > 0$, the field has a positive magnetization, encouraging every node to be in state +1. In this case, $U_{T,H}$ can be rewritten as

$$\begin{aligned} U^*_{T,H}(\sigma) &= U_{T,H}(\sigma) + NH \\ &= -\frac{1}{T}\sum_{(i,j)\in\mathcal{N}} \sigma_i\sigma_j + H\sum_{i=1}^{N}(1-\sigma_i) \\ &= -\frac{1}{T}\sum_{(i,j)\in\mathcal{N}} \sigma_i\sigma_j + H||\sigma - \sigma^+||_1 \end{aligned}$$

where σ^+ is the state with $\sigma_i^+ = 1$ for all nodes. The probability distribution defined by $U^*_{T,H}$ is the same as the distribution defined by $U_{T,H}$ because they differ only by a constant. Similarly, when $H < 0$ and the magnetic field is negative, the energy function can be rewritten as

$$U^*_{T,H}(\sigma) = -\frac{1}{T}\sum_{(i,j)\in\mathcal{N}} \sigma_i\sigma_j + |H|\,||\sigma - \sigma^-||_1$$

where σ^- is the state with all $\sigma_i^- = -1$. This shows that the role of H in the magnetized Ising model is the same as the role of α in (11.14), because $U^*_{T,H}$ is the sum of the unmagnetized Ising energy and a term that penalizes distance to either σ^+ or σ^-, the mirror global minima. Introducing the magnetization term upsets the symmetry of the standard Ising energy function and causes either σ^+ or σ^- to become the sole global minimum, depending on the sign of H.

The behavior of the system with respect to the parameters (T,H) can be represented by the simple phase diagram in Figure 11.28. The dot is the critical temperature of the system, and the solid line is a first-order phase transition boundary. When the parameters of the system are swept across the first-order transition boundary, a discontinuous change in the state space occurs as the system flips from a predominantly positive state to a predominantly negative state, or vice-versa. On the other hand, sweeping the magnetic field H across 0 above the critical temperature results in a smooth transition where positive and negative nodes coexist.

Let $H > 0$ be a weak magnetic field, and suppose the temperature T is below the critical temperature T_c. In this situation, the magnetized Ising model exhibits strong metastable behavior. If the system is initialized from a random configuration (each node $+1$ or -1 with probability 1/2), the influence of the magnetic field will cause the system to collapse to σ^+, or a nearby predominantly positive region of the state space, with high probability. However, if the system is initialized from σ^-, and if H is sufficiently small, the system will exhibit metastability, because magnetic force H will be unable to overcome the strength of the bonds in σ^-, which are very strong below the critical temperature. The system will stay in a stable, predominantly negative state for a long period of time, even though the global minimum of the energy landscape is σ^+, because the magnetic field force cannot overcome the barriers between σ^+ and σ^- in the raw Ising energy landscape.

11.4.4 Attraction-diffusion ELM algorithm

ELM methods have three basic exploration steps:

1. Get a state X as the starting point for a minima search.
2. Find a local minimum Y starting from X.
3. Determine if Y is grouped with a previously found minima basin or if Y starts a new minima basin.

These steps are repeated until no new local minima are found for a certain number of iterations and barrier estimates between basins have converged.

Step 2 can be accomplished with standard gradient descent methods, and the GWL Algorithm provides a principled way to propose X in Step 1. Previous ELM methods lack a reliable way to tackle Step 3. Traditionally, ELM studies have attempted to enumerate *all* basins of attraction of the energy landscape (or the N lowest-energy minima), no matter how shallow. Minima are only grouped together if they are identical in discrete spaces, or if they are extremely close in continuous spaces. This approach is doomed to failure in all but the simplest cases, because the number of distinct local minima grows exponentially with landscape complexity and/or dimension. On the other hand, for some families of energy functions, the *macroscopic* structure might remain unchanged as landscape complexity/dimension increases. For example, the Ising energy landscape will always have two global basins, regardless of neighborhood structure or number of nodes.

The ADELM Algorithm below illustrates the use of AD to group newly found minima with known basins in Step 3 of the ELM procedure. Grouping occurs when successful travel is possible between a new minimum and a representative minimum at the fixed magnetization α used throughout mapping. If a new minimum cannot successfully diffuse to any of the previously identified minima, then the new minimum defines a new metastable basin. Steps 1 and 2 of the ADELM algorithm are left open-ended. In particular, GWL and ADELM can be seamlessly integrated by using GWL to perform the proposals of new initial states for gradient descent in Step 1.

The MCMC sampler S used for mapping should be local in the sense that displacement after a single step is small relative to landscape features with high probability. MCMC methods with step size parameter ε such as Metropolis-Hastings with a Gaussian proposal or HMC/Langevin Dynamics are local samplers, since ε can be tuned to limit displacement. Gibbs sampling is also local, because only a single dimension is changed in each update. The requirement that S be local is needed to ensure that a Markov chain updated using S cannot escape from local modes at low temperatures. For example, one should use Gibbs sampling rather than Swendsen-Wang Cuts when using ADELM to map the local minima of the Ising model because the SWC method can easily escape from local modes, which undermines the entire AD procedure. Usually, high auto-correlation is considered an undesirable feature of MCMC methods, but in AD it is essential that the Markov samples remain trapped in the absence of magnetization. Upsetting this baseline behavior by introducing a magnetic field enables the discovery of landscape features.

Algorithm 18 Attraction-Diffusion ELM (ADELM)

Input: Target energy U, local MCMC sampler S, temperature $T > 0$, magnetization force $\alpha > 0$, distance resolution $\delta > 0$, improvement limit M, number of iterations N

Output States $\{X_1, \ldots, X_N\}$ with local minima $\{Y_1, \ldots, Y_N\}$, minima group labels $\{l_1, \ldots, l_N\}$, and group global minima $\{Z_1, \ldots, Z_L\}$, where $L = \max\{l_n\}$

for $n = 1 : N$ **do**

1. Get proposal state X_n for minima search. (Random initialization or a GWL MCMC proposal)
2. Start a local minimum search from X_n and find a local minimum Y_n.

if $n = 1$ **then**

Set $Z_1 = Y_1$ and $l_1 = 1$.

else

Determine if Y_n can be grouped with a known group using AD. Let $L_n = \max\{l_1, \ldots, l_{n-1}\}$, and let minimum group membership set $G_n = \emptyset$.

for $j = 1 : L_n$ **do**

a) Set $C = Y_n$, $X^* = Z_j$, $d_1 = ||C - X^*||_2$, $d^* = d_1$, and $m = 0$.

while $(d_1 > \delta)$ & $(m < M)$ **do**

Update C with a single step of sampler S using the density

$$P(X) = \frac{1}{Z} \exp\{-(U(X)/T + \alpha||X - X^*||_2)\}$$

and find the new distance to the target minimum: $d_1 \leftarrow ||C - X^*||_2$.

If $d_1 \geq d^*$ **then** $m \leftarrow m + 1$, **else** $m \leftarrow 0$ and $d^* \leftarrow d_1$.

end while

b) Set $C = Z_j$, $X^* = Y_n$, $d_2 = ||C - X^*||_2$, $d^* = d_1$, and $m = 0$, and repeat the loop in Step a).

c) If $d_1 \leq \delta$ or $d_2 \leq \delta$, then add j to the set G_n, and let B_j be the barrier along the successful path. If both paths are successful, let B_j be the smaller of the two barriers.

end for

if G_n is empty **then**

Y_n starts a new minima group. Set $l_n = \max\{l_1, \ldots, l_{n-1}\} + 1$, and $Z_{l_n} = Y_n$.

else

Y_n belongs to a previous minima group. Set $l_n = \text{argmin}_j B_j$.

if $U(Y_n) < U(Z_{l_n})$ **then**

Update the group global minimum: $Z_{l_n} \leftarrow Y_n$.

end if

end if

end if

end for

In the ADELM algorithm, the global minima Z_j of each basin are used as the targets for AD trials. One reason for this choice is the intuition that, for the same strength α, an AD chain should be more likely to successfully travel from a higher-energy minimum to a lower-energy minimum than vice-versa. While not true in general, in practice the intuition holds in most cases, especially for very deep minima. A more nuanced implementation could consider multiple candidates from the same basin as targets for diffusion instead of just the global basin minimum.

Ideally, in each step of the ADELM Algorithm, diffusion to only one basin representative Z_j should be successful. Successful diffusion to a large number of previously found basins is a sign of poor tuning — in particular, either the value of T or α (or both) is too high, causing leakage between basins. On the other hand, some leakage between minima is usually inevitable, because there are often plateau regions that sit between stronger global basins. This is not too much of a problem as long as the basin representatives remain separated. The global basin representatives $\{Z_j\}$ should be checked periodically to make sure they remain well-separated at the current parameter setting. If an AD chain successfully travels between two of the $\{Z_j\}$, these minima should be consolidated into a single group. This is especially important in the early stages of mapping, when good basin representatives have not yet been found. A

single basin can split into multiple groups if the early representatives are not effective attractor states for the entire basin. When consolidating minima, the lower-energy minimum is kept as the group representative.

The ADELM algorithm has two computational bottlenecks: the local minima search in Step 2, and the AD grouping in Step 3. The computational cost of Step 2 is unavoidable for any ELM method, and the MCMC sampling in Step 3 is not unreasonable as long as it has a comparable running time. In our experiments, we find that the running time for local minimum search and a single AD trial are about the same. Step 3 of the ADELM algorithm involves AD trials between a new minimum and several known candidates, and the efficiency of ADELM can be greatly increased by running the AD trials in parallel.

11.4.5 Tuning ADELM

Correct tuning of T and α is essential for good results. The temperature T must be set low enough so that movement is restricted to the current mode, but not so low that the chain becomes totally frozen. In our experiments, we first tune the temperature independently of α by initializing unmagnetized chains from a local minimum and observing the change in energy that occurs over a long trajectory. The change in energy should be small relative to the barriers that exist in the landscape. If the temperature is too high, MCMC samples can easily cross between metastable regions even without magnetization and the mapping fails to recover meaningful structure. The ideal temperature for AD trials appears to be approximately an order of magnitude smaller than the system critical temperature T_c. See Figure 11.29 for an example of tuning AD temperature.

The magnetization strength α must be strong enough to overcome the noisy shallow barriers in the landscape while respecting the large-scale barriers. Once the temperature T has been tuned and fixed so that chains can diffuse in a limited metastable region, one can run trial mappings across the spectrum of α to locate the critical range where α yields meaningful mapping results. In the limiting case $\alpha \to 0$, each distinct minimum defines its own metastable region, while in the limiting case $\alpha \to \infty$, all minima merge in a single basin. By plotting the number of minima that are discovered in a small number of trial steps as a function of α, it is possible to quickly identify the critical range where magnetization and energy features compete on approximately equal footing. See Figure 11.29 for an example of tuning AD magnetization. In our experiments we find that the behavior of AD is quite consistent across a range of T below the critical temperature. Choosing α seems to be the most important tuning decision.

11.4.6 Barrier estimation with AD

AD can be used to estimate the energy barriers between the representatives of each basin to construct a DG after exploration. This is done by setting the temperature T below critical temperature (see Section 11.4.5) and tuning α to find a threshold where successful travel between minima is just barely possible. The AD barrier estimates are lowest when α is just strong enough to overcome barriers in the landscape, and the estimates will increase as α increases. In the limit $\alpha \to \infty$, the AD barriers are identical to the 1D linear barriers, because the MCMC samples will simply move in a straight line towards the target. Estimated barrier height appears consistent for a range of T below critical temperature T_c.

In our mappings, we are primarily interested in the energy barriers between the global basin representatives, which are the most significant features of the macroscopic landscape. The global minima within each basin and the barriers between the global minima are used to construct a DG from the results of an ADELM implementation using the same method described in Section 11.1.2. The basins D_j from ADELM contain multiple local minima, in contrast to the basins D_j defined in Section 11.1.1 which contain only a single local minimum. In the DG visualization of ADELM results, we can include visualizations of local minima from each basin under the terminal node for each basin, sorted from top to bottom in order of descending energy. Circles can also be added to an ADELM DG to indicate the proportion of minima from the mapping that are contained within each basin. See Figures 11.36 and 11.38.

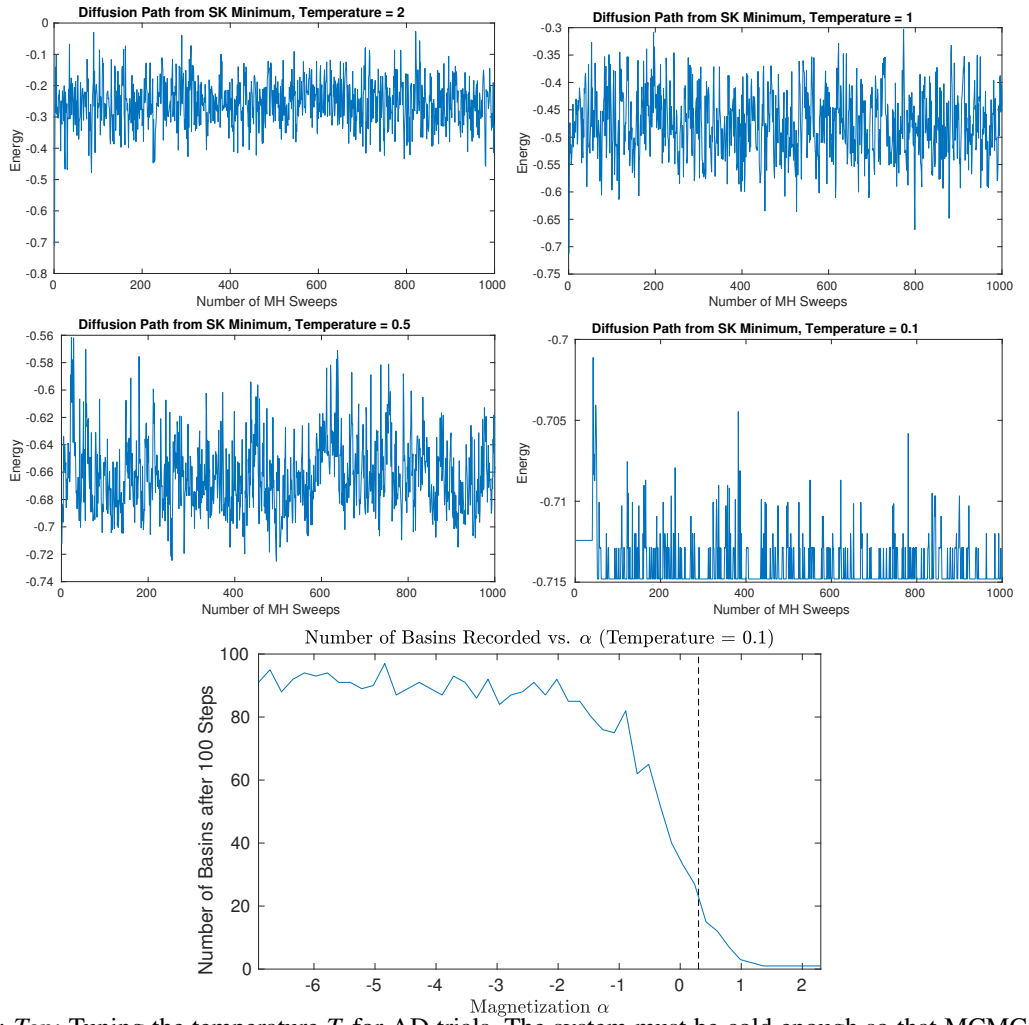


Fig. 11.29: *Top:* Tuning the temperature T for AD trials. The system must be cold enough so that MCMC chains do not travel in an energy spectrum above the minimum energy barriers. The critical temperature $T = 1$ is too warm, and we use $T = 0.1$ instead. *Bottom:* Tuning the magnetization α for AD trials. We run 100 mapping iterations and record the number of distinct basins encountered. As $\alpha \to 0$, we find a new minima for nearly every iteration. As $\alpha \to \infty$, all minima merge into a single basin. In a critical range between the limiting cases, macroscopic behavior can be detected. We use $\alpha = 1.35$, which is shown by the vertical dotted line. © [2019] Brown University. Reprinted, with permission, from ref. [16].

11.5 Mapping the SK Spin Glass Model with GWL and ADELM

In our first ADELM experiment, we map the structure of a sample from the 100-state SK spin glass model. The N-state SK spin glass is a generalization of the standard N-state Ising model where the coefficients for couplings are unspecified. The energy function for the N-state SK spin glass is

$$U(\sigma) = -\frac{1}{TN} \sum_{1 \le i < k \le N} J_{ik}\, \sigma_i \sigma_k \tag{11.17}$$

where $\sigma_i = \pm 1$, $T > 0$ is the temperature, and J_{ik} are couplings. In standard Ising model, the coupling coefficients are either 1 (i.e. the nodes are adjacent) or 0 (i.e. the nodes are not adjacent). The energy landscape of an SK spin glass contains multiple well-separated global basins that have noisy local structure. Like the Ising model, the landscape is exactly symmetric, since $U(\sigma) = U(-\sigma)$. Zhou [40] has shown that the GWL algorithm can accurately identify the

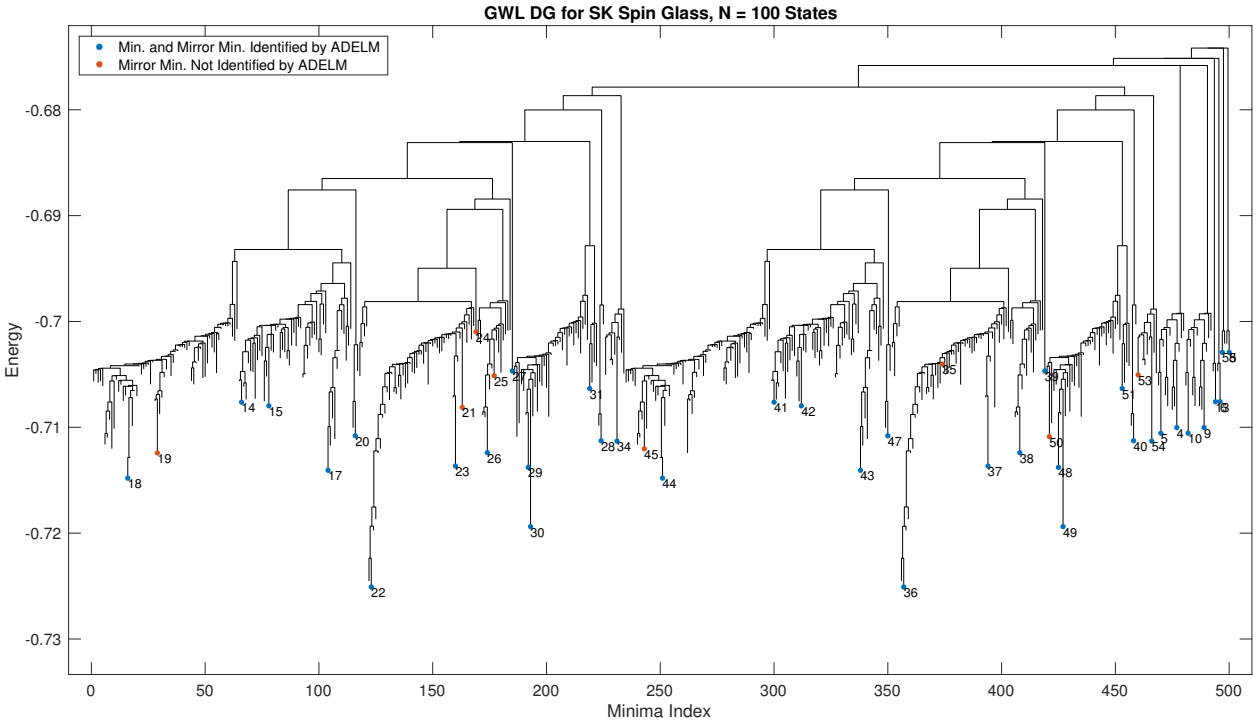


Fig. 11.30: GWL DG for SK Spin Glass. The map records the 500 lowest energy minima in the landscape and the tree is nearly symmetric. The blue and orange dots indicate basins that were identified by our ADELM mapping. The blue dots show minima whose mirror state was also identified during the ADELM run, while the orange dots show minima whose mirror state was not identified. The blue dots cover the most important features of the landscape, which are very stable for the AD parameters $(T = 0.1, \alpha = 1.35)$, while the orange dots record substructures within stronger basins that are close to the metastable border. © [2019] Brown University. Reprinted, with permission, from ref. [16].

lowest-energy minima and barriers for as many as $N = 100$ states. Mapping a 100-dimensional SK spin glass with both ADELM and GWL reveals how ADELM efficiently captures the same coarse-scale features that appear in the fine-scale GWL mapping.

We replicated the GWL mappings in [40], and the result is shown in Figure 11.30. The couplings J_{ik} are independent Gaussians with mean 0 and variance $1/N$, as in the original experiment. We ran our mapping for 5×10^8 iterations using the same GWL parameters described in the original paper, and searched for the 500 lowest minima in the landscape. After running the GWL mapping, the 500 lowest minima identified were exactly symmetric, meaning that for each minima discovered we also identified its mirror state as a minima in the mapping. The number of local minima in an SK spin glass is far more than 500 even with only $N = 100$ states, but previous mappings show that the 500 lowest-energy local minima capture the main landscape features. In more complex landscapes or larger spin glasses, even the lowest-energy regions can contain an astronomical number of local minima, making the GWL approach problematic.

We mapped the same energy landscape using ADELM to compare results and to see if ADELM can reliably identify the most important features of the landscape. We used the temperature $T = 0.1$, which is well below the critical temperature $T_c = 1$, and magnetization strength $\alpha = 1.35$ as the AD parameters. Setting T exactly at the critical temperature yielded poor results, because the energy fluctuation of the chains in the absence of magnetization was greater than the depth of the landscape features, and a colder system is needed to restrict diffusion to the lowest energy levels. After tuning and fixing T, we tuned α by running 100 mapping iterations for different α spaced evenly on a log scale and recording the number of minima identified. See Figure 11.29 for plots showing tuning results. We use the same approach to tune T and α in each of the experiments.

We ran our algorithm for 5,000 iterations, set the AD improvement limit to $M = 100$ Gibbs sweeps of all states, and set our distance resolution $\delta = 0$, which requires that AD chains travel exactly to their target for a successful trial. Our ADELM result is shown in Figure 11.31, and a side-by-side comparison of the ADELM and GWL mappings is shown in Figure 11.32. The ADELM mapping identifies the lowest energy minima for all of the major basins of the landscape,

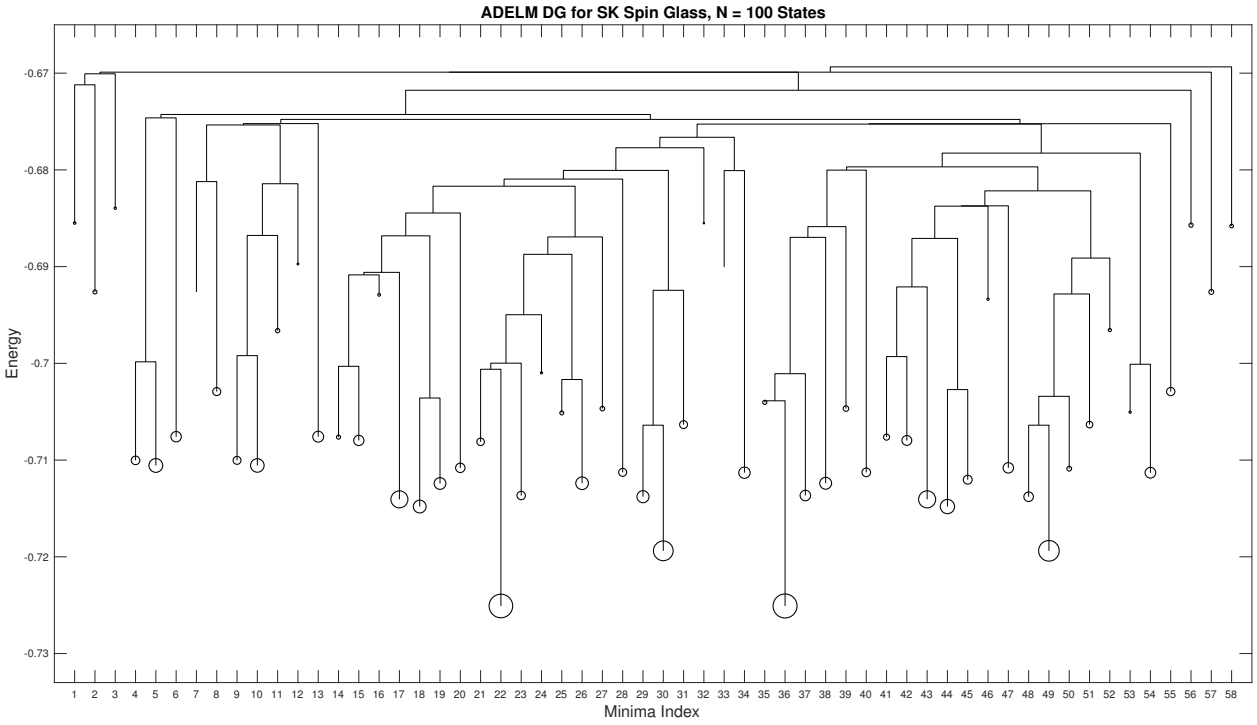


Fig. 11.31: AD DG for SK Spin Glass. The AD diagram is quite symmetric (see Figure 11.30) and the structure of the DG is very consistent with the DG created from the GWL mapping (see Figure 11.32). 44 of the AD minima are also located by GWL, while 14 of the ADELM minima are not among the 500 lowest energy minima. The GWL mapping, which records only lowest-energy minima, misses significant stable features in higher-energy regions. The size of circles around minima nodes is proportional to the number of minima sorted to each basin, as described in Section 11.4.6. © [2019] Brown University. Reprinted, with permission, from ref. [16].

as well as substructures within the basins. ADELM is also able to identify a number of basins which are stable but not recorded by the GWL mapping, since these local minima are not among the 500 lowest-energy minima in the landscape. Overall, 44 of the AD basins were also included in the GWL mapping, while 14 stable basins identified by AD were beyond the energy threshold of inclusion in the GWL mapping.

The barriers estimated by the GWL mapping and the ADELM mappings are very similar, although in most cases the GWL barriers are slightly lower than the barriers estimated by AD. This shows that using a large number of minima during barrier estimation can be helpful, because shallow minima can help bridge the gap between stronger basins of attraction. Even though nearly all of the individual barriers identified by GWL are higher than the barriers identified by AD (see Figure 11.33), the total information of barrier estimates between 500 minima can lead to overall barriers that are lower than the estimates obtained using only 58 minima. On the other hand, it might not be possible to exhaustively identify all of the relevant lowest-energy minima in other landscapes, and it is important to be able to accurately estimate barriers between distant minima without many shallow intermediate minima to connect the basins. Figure 11.33 shows an AD path between the two global minima of the SK spin-glass. The maximum energy along the path is only slightly above the barrier identified in GWL and ADELM DG's. This is evidence that AD can provide reliable interpolations between distant locations.

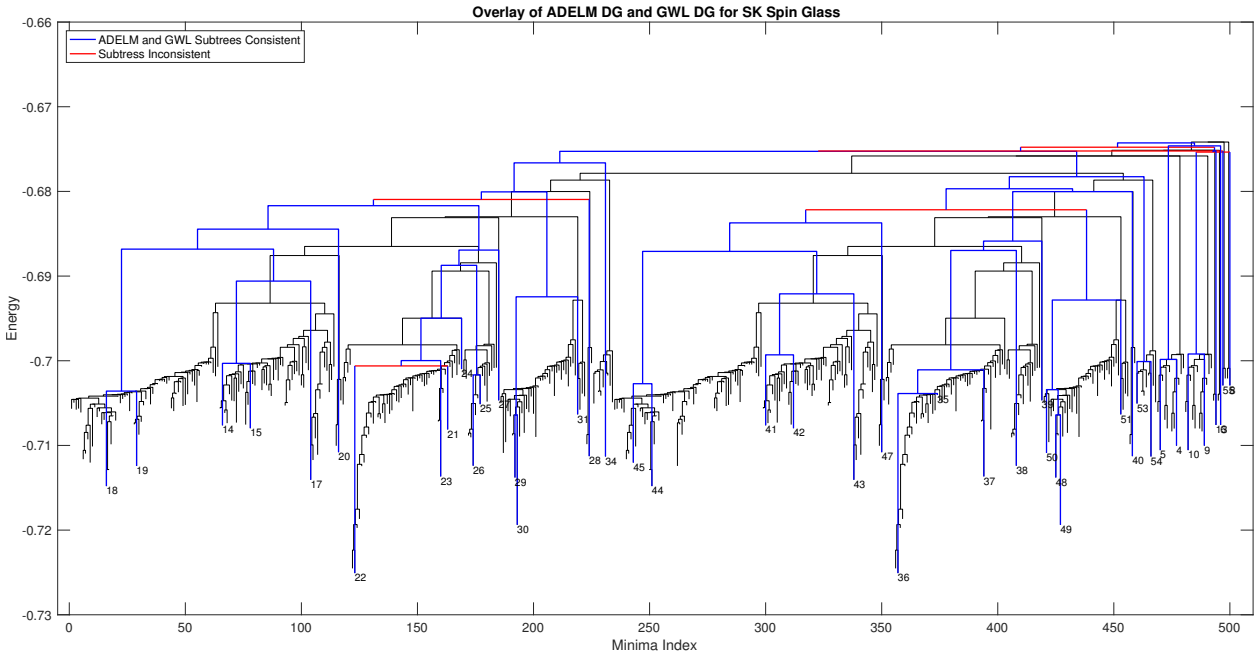


Fig. 11.32: Overlay of Ising AD and GWL mapping. Blue horizontal lines indicate nodes of the ADELM DG where branch merges are consistent with the GWL DG. Red horizontal lines indicate nodes where the ADELM DG and GWL DG merge branches in a different order. The inconsistencies are minor and mostly occur in higher energy regions. Most inconsistencies only occur for a single merging, and are corrected by the next merge. The ADELM mapping effectively captures the macroscopic features of the GWL mapping. © [2019] Brown University. Reprinted, with permission, from ref. [16].

11.6 Mapping Image Spaces with Attraction-Diffusion

11.6.1 Structure of image galaxies

This section investigates the structure of a Gibbs density p (or equivalently, energy $U = -\log p$) that is trained to model an unknown image density f. During training, the density learns to form modes around the samples of f, and local minima of U can be interpreted as "memories" of the training data. The metastable galaxies of image densities represent different concepts in the training images, and by finding the galaxies of an image density we can reduce the vast high-dimensional image space to a few groups that summarize the typical image appearances. We are also interested in measuring the energy barriers between galaxies, because they encode similarity between images or concepts. The structure of a learned image density encodes memories of the pattern manifold, but this information is hidden in p and must be recovered through mapping.

Landscape structure varies according to the image pattern being modeled (see Figure 11.35). In particular, image scale should have a strong influence on the structure of image memories. In one of the central paradigms of pattern representation, Julesz identifies two major regimes of image scale: texture and texton. *Textures* are high-entropy patterns defined as groups of images sharing the same statistics among nearby pixels [17]. *Textons*, on the other hand, are low-entropy patterns, and can be understood as the atomic building elements or local, conspicuous features such as bars, blobs or corners [18].

As illustrated in Figure 11.34, texton-scale images have explicit structure that is easily recognizable, and this structure allows humans to reliably sort texton images into coherent groups. Texture-scale images have implicit structure, and it is usually difficult or impossible to find groups among images of the same texture, because no distinguishing features can be identified within a texture ensemble. As image scale increases, the number of recognizable image groups tends to increase until one reaches the threshold of perceptibility, where texton-scale images transition into

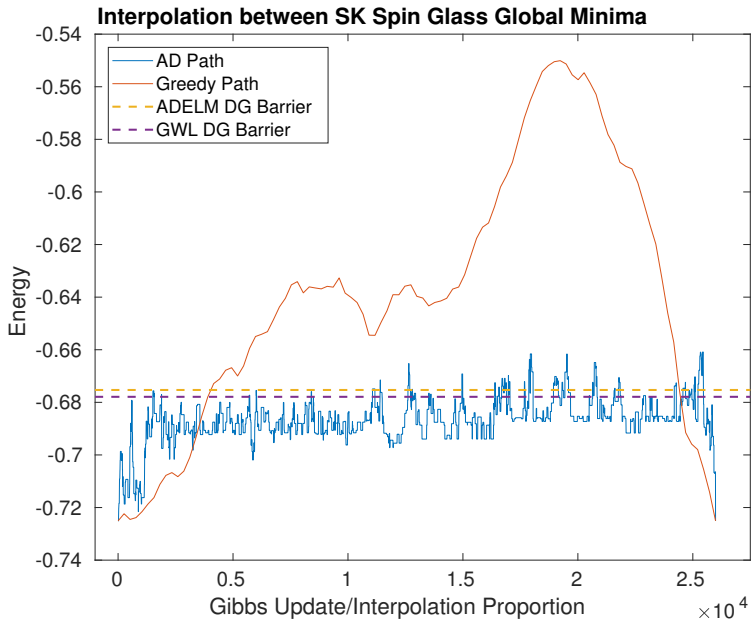


Fig. 11.33: Interpolation between SK spin-glass global minima. The AD path travels in an energy spectrum very close to the barriers in the GWL and ADELM DG's. © [2019] Brown University. Reprinted, with permission, from ref. [16].

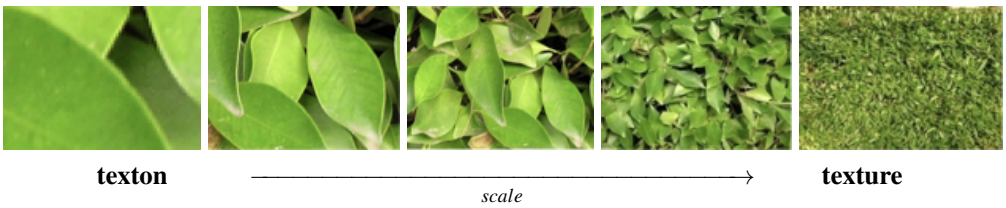


Fig. 11.34: Ivy leaves at different scales. As image scale increases from left to right, an increasing variety of image groups can be identified, until one reaches the threshold of perceptibility, after which it becomes difficult to distinguish between images. The fourth scale is close to the perceptibility threshold of humans, while the fifth scale is beyond human perceptibility. A regime transition from explicit, sparse structure to implicit, dense structure occurs as the threshold is crossed. A similar transition occurs in the energy landscape (see Figure 11.39). © [2019] Brown University. Reprinted, with permission, from ref. [16].

texture-scale images and humans begin to lose the ability to identify distinguishing features. Beyond the threshold of perceptibility, texture images cannot be told apart or reliably sorted into groups. Change of image scale causes a change in the statistical properties of an image, and we call this phenomenon *Information Scaling* [37].

Information Scaling is reflected in the structure of the image landscape, and there is a connection between the perceptibility of differences between pattern images and the stability/depth of local minima images. When the landscape models texton-scale images, where groups among the images can easily be distinguished, one expects to find many separate, stable basins in the landscape encoding the separate appearances of the groups. Landscapes that model texture-scale images, on the other hand, should exhibit behavior similar to human perception and form a single macroscopic basin of attraction with many shallow local minima to encode the texture. By mapping images from the same pattern at multiple scales, one can show that the transition in perceptibility that occurs between scales results in a transition in the landscape structure of image memories (see Figure 11.35 and Figure 11.39).

11.6.2 Experiments

The target energy function throughout this section is

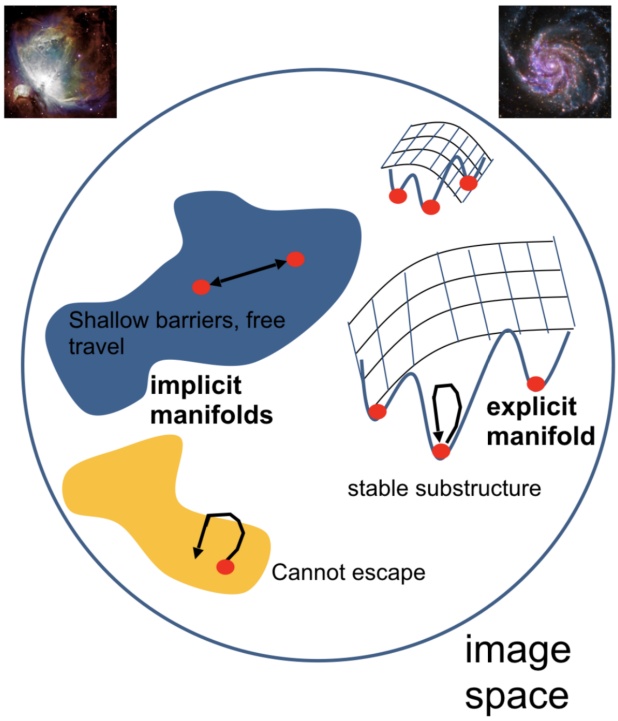


Fig. 11.35: Diagram of image galaxies for textures (implicit manifolds) and textons (explicit manifolds). Textures form wide and cloudy regions with no internal structure, while textons form galaxies with stable substructure to encode the different typical appearances of textons.

$$U_{w_1,w_2}(Z) = U(g(Z|w_2)|w_1) \tag{11.18}$$

where $U(\cdot|w_1)$ and $g(\cdot|w_2)$ are learned according to the Co-Op Net Algorithm [38]. The proposal in Step 1 of ADELM can be obtained by sampling from the latent distribution of the generator network. The formulation in (11.18) provides a way to efficiently map DeepFRAME functions defined over images of realistic size using ADELM.

11.6.2.1 Digits 0-9 ELM in Latent Space

We apply ADELM to map the energy (11.18) of Co-Op Networks modeling all of the digits of MNIST. We used the first half of the MNIST testing set as our training data (about 500 examples of each digit). This time, we increase image size to 64×64 pixels. Since we will only sample in the *latent* space, which has low dimension, we can use realistically-sized images during training.

The descriptor network had three layers: two convolutional layers sized 5×5 of 200 filters and 100 filters, and a fully-connected layer of 10 filters. Each layer is followed by a ReLU activation function. The latent generator distribution was the 8-dimensional normal $\mathrm{N}(0, I_8)$. The generator network had three layers of size 4×4, 7×7, and 7×7 with 200, 100, and 10 filters respectively. ReLU was used as the activation for the first two layers, and tanh was used as the activation of the last layer. An upsampling factor of 4 was used after each generator layer. The AD parameters were $T = 1200$ and $\alpha = 230$. The other ADELM parameters used were the same as in the Spots/Stripes Latent Space ELM. For mapping, 500 burn-in iterations and 5000 testing iterations were used, and the results are shown in Figure 11.36.

The ELM in Figure 11.36 has many strong, well-separated energy basins. A close look at the DG shows that all 10 digits are represented by at least a single strong minima basin. The basin members and the global structure of the DG both match closely with human visual intuition.

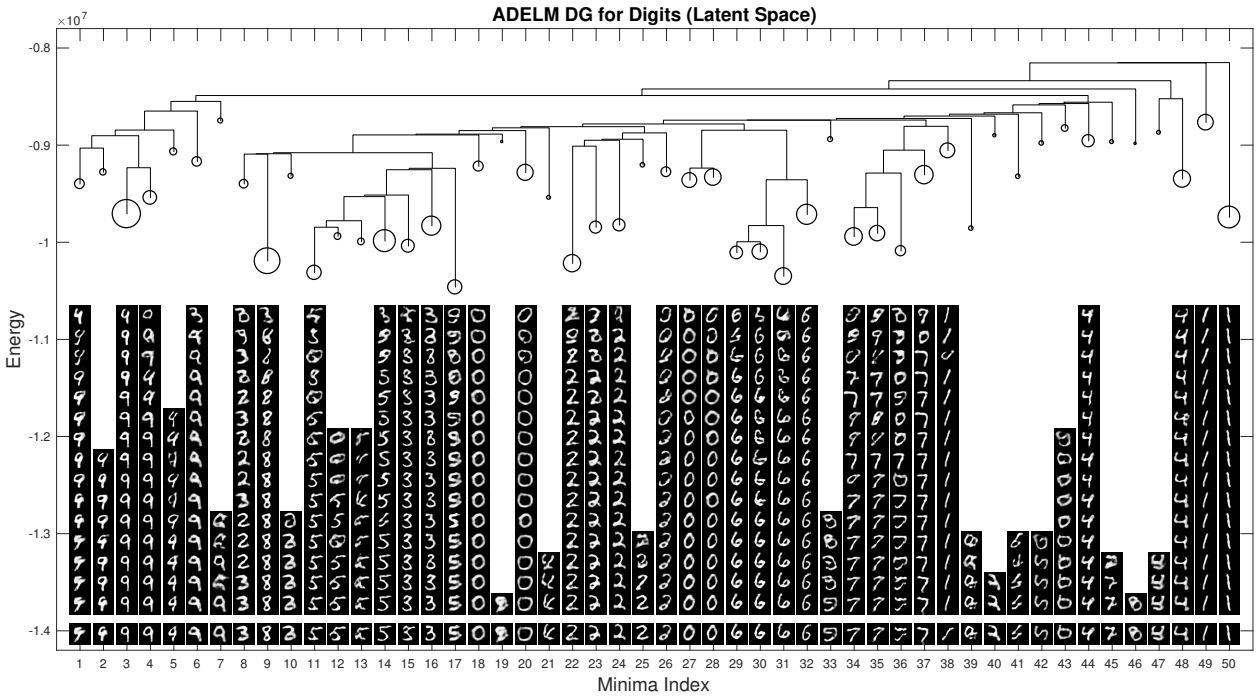


Fig. 11.36: DG of Digits 0-9 ELM in latent space. The descriptor network is over 64×64 images, but the generator latent space has only 8 dimensions, allowing for efficient mapping. Remarkably, all 10 digits have at least one well-separated branch in the DG. Minima representing the same digit generally merged at low energy levels. © [2019] Brown University. Reprinted, with permission, from ref. [16].

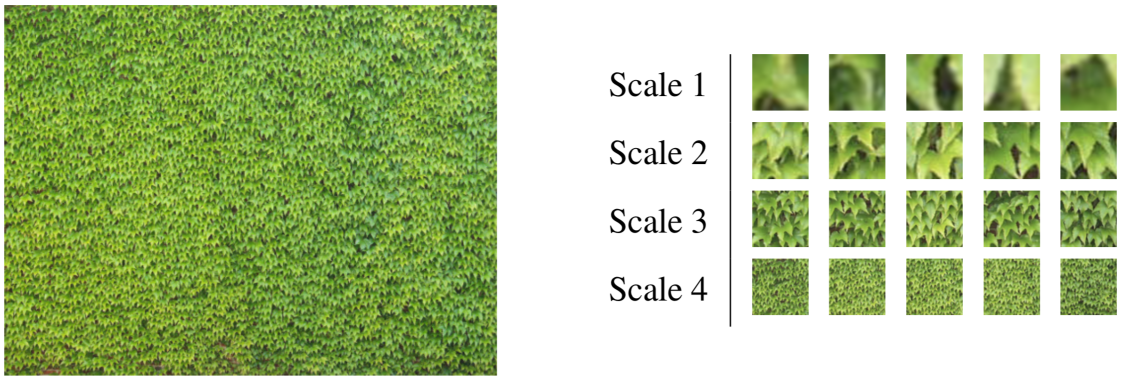


Fig. 11.37: Ivy texture image and image patches from four scales. © [2019] Brown University. Reprinted, with permission, from ref. [16].

11.6.2.2 Ivy Texton ELM in Latent Space

We now map a Co-Op Network trained on image patches from an ivy texture. At close range, ivy patches have distinct and recognizable structure, and the goal of the mapping is to identify the main patterns that recur in the ivy textons. Figure 11.37 shows the entire ivy texture image along with image patches from the texture taken at four different scales. The networks in this experiment are trained to model 1000 image patches from Scale 2.

The DG's for the ivy texton mapping in Figure 11.38 show that the landscape is dominated by 3 or 4 global basins. The images within basins are very consistent, and the barriers between the basins are representative of visual similarity between the minima images. Unlike the digits mapping, there is no ground-truth for the minima groupings, so it is useful to explore the landscape at different energy resolutions to identify image groupings at different degrees of visual similarity. One major advantage of ADELM is the ability to perform mappings at different energy resolutions

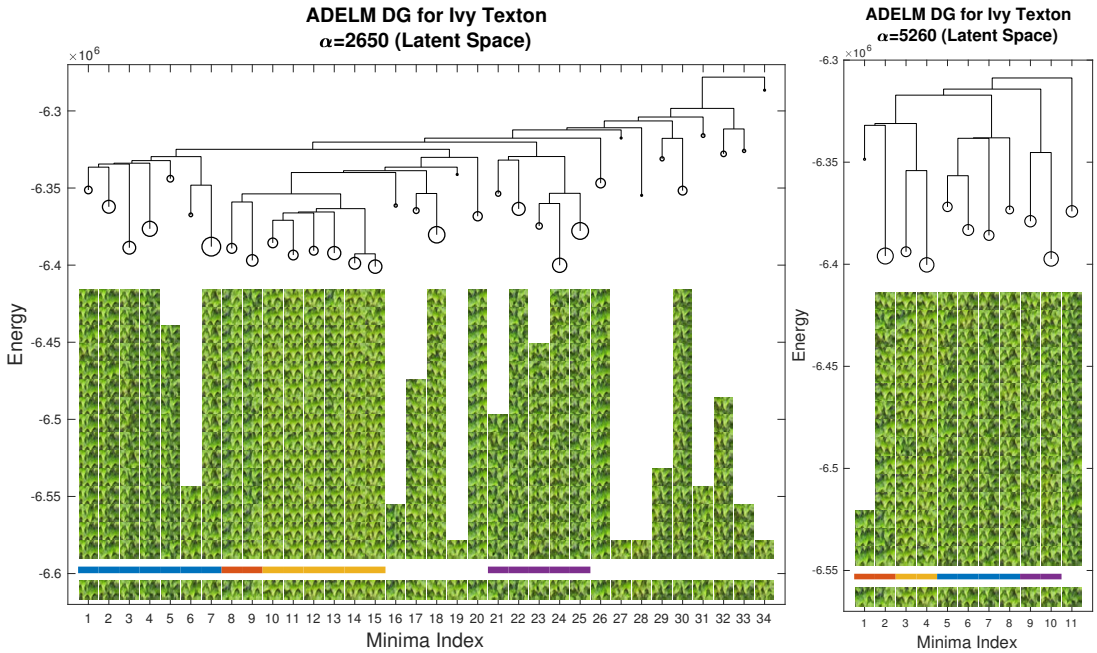


Fig. 11.38: DG's of Ivy Textons for two different values of magnetization α. Both mappings show 3 strong global basins and substructures within these basins that are stable at different magnetizations. There is no ground-truth grouping for texton image patches, so it is useful to map image structures at multiple resolutions to identify "concepts" at different degrees of visual similarity. The colors below the basin representatives indicate regions that appear in both mappings. © [2019] Brown University. Reprinted, with permission, from ref. [16].

simply by changing the magnetization strength α used during the AD trials. Figure 11.38 presents two mappings of the same landscape at different energy resolutions. The same landscape features appear in both mappings with more or less substructure depending on the magnetization strength.

11.6.2.3 Multiscale Ivy ELM in Latent Space

We continue our investigation of the ivy texture image from the previous section by mapping a Co-Op Network trained on 1000 image patches from each of the four scales shown in Figure 11.37. In this experiment, we want to investigate the differences in memory formation between the different scales. In particular, we are interested in identifying a relation between the *metastability* of local minima in the landscape and the *perceptibility* of visual difference among the minima. We expect to find fewer structures at the extreme scales. Image patches from Scale 1 are mostly solid-color images with little variety, which should form a few strong basins in the landscape. Image patches from Scale 4 have no distinct features and cannot be told apart by humans, so we expect these images will form a wide basin without much substructure. For the intermediate scales, we expect to find a richer assortment of stable local minima, because the intermediate scales contain more variation than Scale 1, but the variation still can be distinguished visually, in contrast to the Scale 4 images.

The structure of the landscape does indeed differ between the image scales. As expected, the memories from Scale 1 form a few strong and large basins. Scale 2 accounts for the majority of the basins in the landscape, since this scale contains the most variety of perceptible image appearances. The Scale 2 basins merge with the Scale 1 basins in the DG visualization, indicating that there are accessible low-energy connections between these regions of the landscape. The images from Scale 3 and Scale 4 each form a separate region of the energy landscape with little substructure. The mapping shows that the perceptibility threshold for ivy texture images (at least in terms of memories learned by the Co-Op Network) lies somewhere between Scale 2 and Scale 3. Above the perceptibility threshold, the network cannot reliably distinguish variation between images, and the landscape forms a single region with no significant substructure. It is difficult for a human to distinguish groups among images from Scale 3, so the perceptibility threshold for the

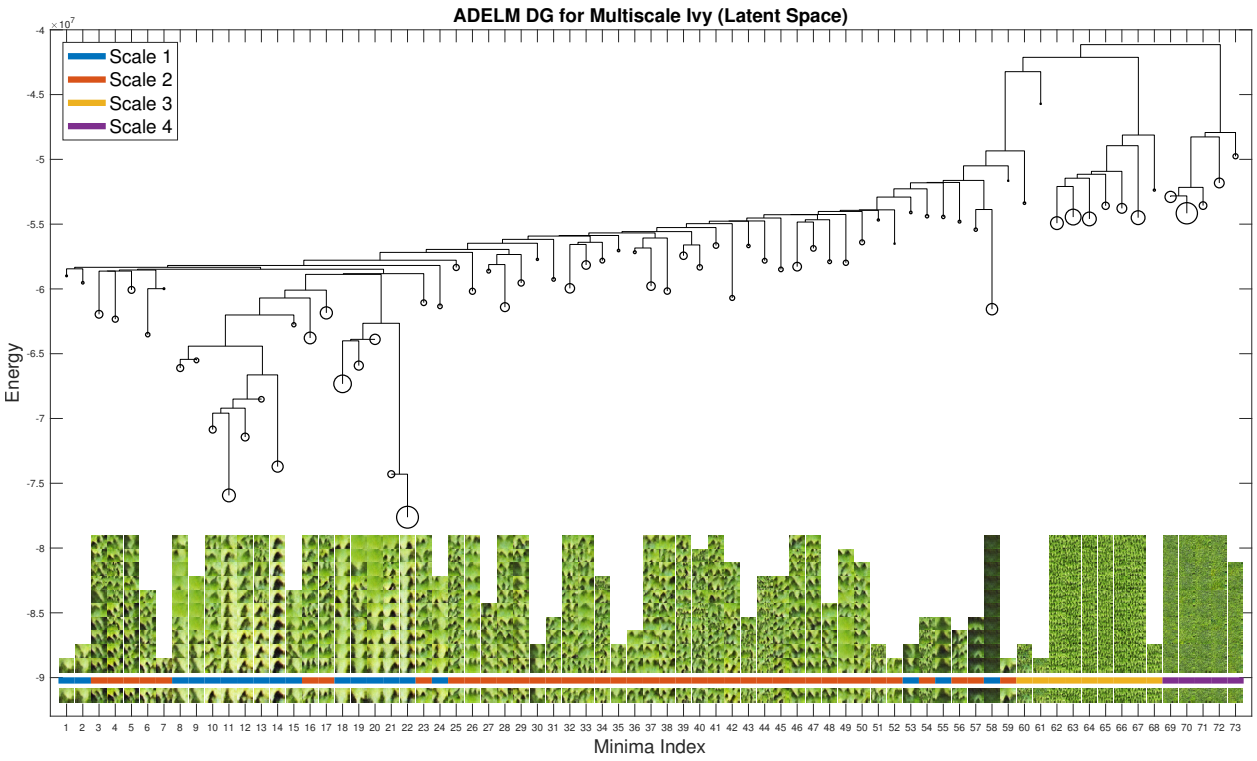


Fig. 11.39: Landscape of ivy image patches at four different scales. Images from Scale 1 and Scale 2 are textons, while images from Scale 3 and Scale 4 are textures. The texton-scale images account for the majority of the basins in the landscape. More basins are identified for Scale 2 than Scale 1 because Scale 2 has a richer variety of distinct appearances, while the Scale 1 minima have lower energy, since appearances from this scale are more reliable. The texture-scale images form separate basins with little substructure. See Section 11.6.1 for details. © [2019] Brown University. Reprinted, with permission, from ref. [16].

network seems similar to that of humans.

11.6.2.4 Cat Faces ELM in Latent Space

For our final experiment, we map a Co-Op Network trained on aligned cat face images gathered from the internet. The results of our mapping is shown in Figure 11.40. The DG has a single branch and the energy barriers are quite shallow. The main features of the local minima are the geometry and color of the cat faces, but these can be smoothly deformed during interpolation without encountering improbable images, in contrast to images such as digits, which must enter an improbable geometric configuration along an interpolation path. For this reason, the energy barriers throughout the cat landscape are very low. Nonetheless, the global basins found by ADELM coherently identify major groups of cat faces. AD can effectively identify landscape structure even when the majority of basin members have energy that is higher than the barrier at which the basin merges.

11.7 Exercises

Problem 1. Find the energy landscape of a one hidden layer convolutional neural network (CNN) with 16 filters of size 5×5 followed by ReLU and cross-entropy loss on two of the MNIST digits (digits 5 and 9). Compare with the energy landscape of a two hidden layer CNN with 8 filters of size 5×5 in each layer, followed by ReLU.

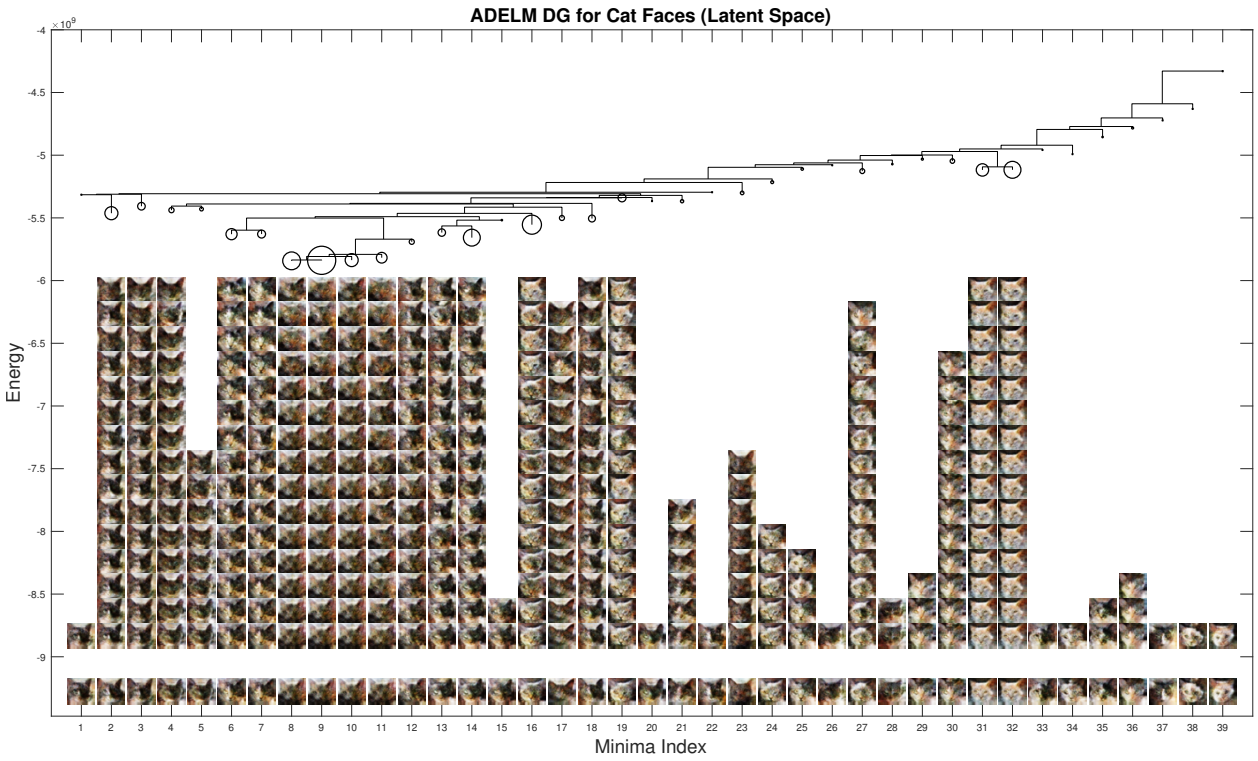


Fig. 11.40: DG of Cat Faces in latent space. The landscape has a single global basin, likely because interpolations between cat faces that respect geometry and color constraints are easily found, unlike interpolations between digits, which must pass through a high-energy geometric configuration along the path. Despite the lack of overall landscape structure, AD is able to find meaningful image basins that show a variety of cat faces. © [2019] Brown University. Reprinted, with permission, from ref. [16].

Problem 2. Consider the k-D XOR classification problem $y(\mathbf{x}) = I(\prod_{i=1}^{k} x_i > 0)$ defined for p-dimensional observations $\mathbf{x} = (x_1, ..., x_p) \in \mathbb{R}^p$, with $\mathbf{x} \sim \mathcal{N}(0, I_p)$. Thus the problem has $p - k$ irrelevant variables $x_{k+1}, ..., x_p$.

1. Find the energy landscape of a one hidden layer neural network with 10 hidden nodes followed by ReLU and cross-entropy loss on the 3D XOR problem for $p = 3$ and $n = 1000$ observations. Compare the energy landscape with the landscape of a network with 500 nodes.
2. Repeat point a) for $p = 10$ and $p = 100$, keeping $k = 3$ and $n = 1000$. Observe that the energy landscape becomes more and more complex as the number of irrelevant variables increases.

References

1. Adrian Barbu and Song-Chun Zhu. Generalizing swendsen-wang to sampling arbitrary posterior probabilities. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(8):1239–1253, 2005.
2. Oren M Becker and Martin Karplus. The topology of multidimensional potential energy surfaces: Theory and application to peptide structure and kinetics. *The Journal of chemical physics*, 106(4):1495–1517, 1997.
3. Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ICML*, pages 41–48, 2009.
4. C L Blake and C J Merz. Uci repository of machine learning databases, 1998. Robustness of maximum boxes.
5. Anton Bovier and Frank den Hollander. Metastability: A potential theoretic approach. *International Congress of Mathematicians*, 3:499–518, 2006.
6. Stephen P. Brooks and Andrew Gelman. General Methods for Monitoring Convergence of Iterative Simulations. *Journal of Computational and Graphical Statistics*, 7(4):434–455, December 1998.
7. Eugene Charniak. Immediate-head parsing for language models. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 124–131, 2001.
8. Michael Collins. *Head-driven statistical models for natural language parsing*. PhD thesis, University of Pennsylvania, 1999.
9. Sanjoy Dasgupta and Leonard J. Schulman. A two-round variant of em for gaussian mixtures. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, UAI'00, pages 152–159, 2000.

10. Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.
11. Jeffrey L Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99, 1993.
12. Kuzman Ganchev, João Graça, Jennifer Gillenwater, and Ben Taskar. Posterior regularization for structured latent variable models. *Journal of Machine Learning Research*, 11:2001–2049, 2010.
13. Andrew Gelman and Donald B Rubin. Inference from iterative simulation using multiple sequences. *Statistical science*, pages 457–472, 1992.
14. Charles J Geyer and Elizabeth A Thompson. Annealing markov chain monte carlo with applications to ancestral inference. *Journal of the American Statistical Association*, 90(431):909–920, 1995.
15. William P Headden III, Mark Johnson, and David McClosky. Improving unsupervised dependency parsing with richer contexts and smoothing. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 101–109, 2009.
16. Mitch Hill, Erik Nijkamp, and Song-Chun Zhu. Building a telescope to look into high-dimensional image spaces. *Quarterly of Applied Mathematics*, 77(2):269–321, 2019.
17. Béla Julesz. Visual pattern discrimination. *IRE Trans. Information Theory*, 8(2):84–92, 1962.
18. Béla Julesz. Textons, the elements of texture perception, and their interactions. *Nature*, 290:91, Mar 1981.
19. Dan Klein and Christopher D Manning. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 478, 2004.
20. Sandra Kübler, Ryan McDonald, and Joakim Nivre. Dependency parsing. *Synthesis Lectures on Human Language Technologies*, 1(1):1–127, 2009.
21. Faming Liang. A generalized wang-landau algorithm for monte carlo computation. *Journal of the American Statistical Association*, 100(472):1311–1327, 2005.
22. Faming Liang, Chuanhai Liu, and Raymond J Carroll. Stochastic approximation in monte carlo computation. *Journal of the American Statistical Association*, 102(477):305–320, 2007.
23. Enzo Marinari and Giorgio Parisi. Simulated tempering: a new monte carlo scheme. *EPL (Europhysics Letters)*, 19(6):451, 1992.
24. Igor' Aleksandrovič Mel'čuk. *Dependency syntax: theory and practice*. SUNY Press, 1988.
25. José Nelson Onuchic, Zaida Luthey-Schulten, and Peter G Wolynes. Theory of protein folding: the energy landscape perspective. *Annual review of physical chemistry*, 48(1):545–600, 1997.
26. Maria Pavlovskaia. *Mapping Highly Nonconvex Energy Landscapes in Clustering, Grammatical and Curriculum Learning*. PhD thesis, Doctoral Dissertation, UCLA, 2014.
27. Maria Pavlovskaia, Kewei Tu, and Song-Chun Zhu. Mapping the energy landscape of non-convex optimization problems. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 421–435. Springer, 2015.
28. Douglas LT Rohde and David C Plaut. Language acquisition in the absence of explicit negative evidence: How important is starting small? *Cognition*, 72(1):67–109, 1999.
29. Rajhans Samdani, Ming-Wei Chang, and Dan Roth. Unified expectation maximization. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 688–698. Association for Computational Linguistics, 2012.
30. Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. From baby steps to leapfrog: How "less is more" in unsupervised dependency parsing. In *NAACL*, 2010.
31. Robert H Swendsen and Jian-Sheng Wang. Nonuniversal critical dynamics in monte carlo simulations. *Physical Review Letters*, 58(2):86–88, 1987.
32. Kewei Tu and Vasant Honavar. On the utility of curricula in unsupervised learning of probabilistic grammars. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1523, 2011.
33. Kewei Tu and Vasant Honavar. Unambiguity regularization for unsupervised learning of probabilistic grammars. In *Proceedings of the 2012 Conference on Empirical Methods in Natural Language Processing and Natural Language Learning (EMNLP-CoNLL 2012)*, 2012.
34. David J Wales and Jonathan P. K. Doye. Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms. *J. Phys. Chem.*, 101(28):5111–5116, 1997.
35. David J Wales and Semen A Trygubenko. A doubly nudged elastic band method for finding transition states. *Journal of Chemical Physics*, 120:2082–2094, 2004.
36. Fugao Wang and David P Landau. Efficient, multiple-range random walk algorithm to calculate the density of states. *Physical review letters*, 86(10):2050, 2001.
37. Ying Nian Wu, Cheng-En Guo, and Song-Chun Zhu. From information scaling of natural images to regimes of statistical models. *Quarterly of Applied Mathematics*, 66(1):81–122, 2007.
38. Jianwen Xie, Yang Lu, and Ying Nian Wu. Cooperative learning of energy-based model and latent variable model via mcmc teaching. *AAAI*, 2018.
39. Qing Zhou. Multi-domain sampling with applications to structural inference of bayesian networks. *Journal of the American Statistical Association*, 106(496):1317–1330, 2011.
40. Qing Zhou. Random walk over basins of attraction to construct ising energy landscapes. *Physical review letters*, 106(18):180602, 2011.
41. Qing Zhou and Wing Hung Wong. Reconstructing the energy landscape of a distribution from monte carlo samples. *The Annals of Applied Statistics*, 2:1307–1331, 2008.

Index