4

Monte Carlo for classical many-body problems

Our discussion of Markov chains, with the exception of mentioning the Metropolis and heat-bath algorithms, has so far been very general with little contact with issues and opportunities related to specific applications. In this chapter, we recall that our target is many-body problems defined on a lattice and introduce several frameworks exploiting what is special about Markov processes for these types of problems. We consider here classical many-body problems, using the Ising model as the representative. Our discussion will be extended to various quantum many-body problems and algorithms in subsequent chapters.

4.1 Many-body phase space

The numerical difficulty of studying many-body problems on a lattice arises from the fact that their phase space Ω is a direct product of many phase spaces of local degrees of freedom. Generally, a local phase space is associated with each lattice site. If n is the size of this phase space and N is the number of lattice sites, then the number of states $|\Omega|$ available to the whole system is n^N. In other words, the number of states in the phase space grows exponentially fast with the physical size of the system. For example, in the Ising model, the Ising spin s_i on each site can take one of the two values ± 1, and hence the number of states in the total phase space is 2^N.

In Chapter 1, we noted that this exponential scaling thwarts deterministic solutions and is a reason why we use the Monte Carlo method. The exponentially large number of states implies that the enumeration of all states, which requires a computational effort proportional to the size of the phase space, is not an option for a problem with a large number of sites. As discussed in Section 2.7, the Monte Carlo method generally samples from a compressed phase space, avoiding the need to solve the entire problem. However, even with this advantage, can we reduce the computational effort to a manageable level?

An obvious requirement is that we can equilibrate the simulation in a reasonable amount of computer time. We know that the Markov process converges to a stationary state sampling some distribution (Section 2.4), and for detailed balance algorithms, Rosenbluth's theorem (Section 2.6) guarantees monotonic convergence. However, as we noted before, these theorems do not tell us how rapid the convergence is. While we know that the convergence rate is controlled by the value of the second largest eigenvalue λ_2 of the transition probability matrix, in general, we do not know this eigenvalue.

We do know from the properties of stochastic matrices that $|\lambda_2| < 1$. The difference between it and unity is a finite margin of the order $\mathcal{O}(|\Omega|^{-1})$ or larger (Section 2.4.2). This margin characterizes the slowest relaxation mode and sets an upper bound to the computational auto-correlation time discussed in Sections 3.3 and 3.4,

$$\tau_{\text{comp}} = \frac{-1}{\log|\lambda_2|} \leq \mathcal{O}(|\Omega|),$$

which implies that importance sampling, as embodied, for example, in the Metropolis and heat-bath algorithms, is at least as good as random sampling. However, if the upper bound is fulfilled, the simple enumeration of all states is just as fast, and it has the advantage of being exact.

Fortunately, in many important many-body simulations, τ_{comp} is of the order of unity, independent of the size of the system. Even in the vicinity of a critical point, where Monte Carlo simulations usually have longer relaxation times than in noncritical cases, τ_{comp} depends on the system size polynomially rather than exponentially. Hence, Monte Carlo simulation has become a standard technique for solving many-body problems.

But why can τ_{comp} be independent of the system size? While a mathematically rigorous answer is beyond the scope of this book, we can obtain a rough idea why by reminding ourselves that the transition probability matrix of a many-body problem is not an arbitrary Markov matrix. To see what we mean by this last statement, let us divide the physical system into many identical subsystems, the size of each being independent of the size of the whole system, and also assume that they are large enough that their properties are approximately independent of each other. Then, the distribution function of the whole system is the product of the distribution functions of the subsystems, and the relaxation rate of the whole system is therefore controlled by the relaxation rates of the subsystems. Since by assumption all subsystems are identical, the Markov chain auto-correlation time of the whole system is that of any one of the subsystems with the latter being independent of system size. This situation applies to most Monte Carlo simulations of many-body problems. There usually exists a characteristic length scale beyond which the spatial correlations do not extend. If we are simulating continuous phase transitions, the divergence of the correlation length as we approach a critical point is the reason why the auto-correlation time becomes unusually large in many Monte Carlo simulations. The cause is the physics of the problem and not per se the physical size of the many-body phase space.

4.2 Local updates

We adopt the Ising model as a working example and explore three different Monte Carlo algorithms for simulating its properties. As we remarked in Chapter 1, we can use Monte Carlo simulations to compute both ground state and finite-temperature properties. As we also remarked, for the Ising model the ground state properties are not as interesting as the finite-temperature ones. We defer the discussion of ground state Monte Carlo methods to other chapters (Chapters 9, 10, and 11). To simulate finite-temperature properties we have to sample from the Boltzmann distribution, which means it is sufficient that the algorithms satisfy the detailed balance condition. The Metropolis algorithm sets the standard here. We start by revisiting this method and discuss what is general about it as opposed to dwelling on its specific use for the Ising model.

Our goal is sampling from a distribution whose partition function[1]

$$Z = \sum_{C \in \Omega} W(C) \tag{4.1}$$

is defined by the non negative weight

$$W(C) = \prod_{\langle ij \rangle} w(s_i, s_j), \qquad w(s_i, s_j) = e^{K s_i s_j}, \tag{4.2}$$

where the product is over all pairs of neighboring sites and $K \equiv J/kT$. We seek a procedure for updating a configuration C to another configuration C', where $C \equiv (s_1, s_2, \ldots, s_N)$ denotes a configuration of Ising spin variables for a lattice of N sites.

A simple way of accomplishing this sampling is to use the Metropolis algorithm with single spin-flip updates. It proposes and then accepts or rejects the flipping of the spin on a given site, one site at a time. Its Markov chain transition matrix $P(C'|C)$, which satisfies the detailed balance condition, is a product of a proposal and an acceptance probability. To be specific, we pick a random site with probability $1/N$ and propose to flip the spin on that site. If ΔE is the change in total energy associated with this spin flip, then the spin flip is accepted with a probability $\min(1, \exp(-\Delta E/kT))$. Otherwise we keep (and measure) the old configuration. This simple algorithm is sketched in Algorithm 6 and illustrated in Fig. 4.1.

[1] The goal is computing the derivatives of $\log Z$ with respect to various control parameters, not the calculation of Z per se.

Algorithm 6 Single spin-flip Metropolis algorithm for the Ising model.

```
Input: A configuration of $N$ spins.
  Randomly select one of the $N$ spins ;
  Compute the energy change $\Delta E$ associated with the spin flip ;
  Flip the spin with probability $\min(1, \exp(-\Delta E/kT))$ ;
  return the updated configuration.
```

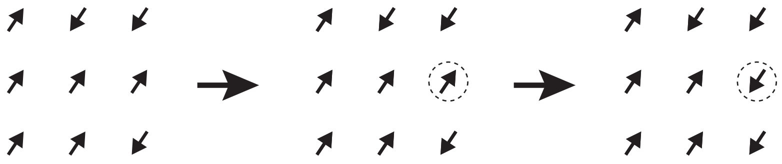

Figure 4.1 A cycle in the single spin-flip update of the Ising model.

4.3 Two-step selection

The single spin-flip Metropolis algorithm can be viewed as a two-step sampling process with the following steps:

1. Select a given spin with some probability that is independent of the current configuration,
2. Select the new state of this spin with the appropriate probability.

The transition matrix for this procedure has the form

$$P(C'|C) = \sum_{i=1}^{N} p_i q_i(C'|C), \tag{4.3}$$

where p_i is the probability that the spin s_i is chosen in the first step, and q_i is the probability that the configuration C is changed to C' in the second step. We can show (Exercise 4.1) that if $q_i(C'|C)$ satisfies the detailed balance condition, then so does $P(C'|C)$.[2] Because of this property, we can select the spins deterministically; that is, we can implement a sweep over the lattice sites j, and for the individual updates use $p_i = \delta_{i,j}$, instead of selecting the spins at random ($p_i = 1/N$).

In the two-step procedure, instead of choosing the next state directly from the vast space of all states, we first restrict the set of states from which we select the

[2] It is rather obvious that we can generalize the single-spin algorithm to a *single unit algorithm* where a unit u consisting of more than one spin is flipped. More specifically, if we express all degrees of freedom as a sum of units, then in (4.3) we can replace i by u and the summation over i by the summation over all units. Now $q_u(C'|C) = 0$ if C' differs from C on some i outside of u. Although it involves flipping several spins at once, the procedure still has two steps.

final state. More specifically, if we are currently in the state $C = (1, 1, 1, 1, \dots)$ and select the second spin in this state as the candidate for flipping, then we must choose the next state from a set of two states: C itself and $C' = (1, -1, 1, 1, \dots)$. Thus, in the selection of the second spin, we are actually selecting from a two-state subset of the phase space. We can define this type of two-state subspace for all initial and final spin configurations. In the example of the single-spin update of the Ising model, every single state is covered by N elements of the family of such subspaces since there are N choices for the spin to be updated.

In an abstract language, for a family $\mathcal{F}$ of subspaces G covering Ω, a single-spin update Markov process consists of the following steps:

1. Select a subspace $G \in \mathcal{F}$ with the probability $P(G|C)$,
2. Select the new state C' out of G with probability $P(C'|C, G)$,

where $P(G|C) = 0$ if C is not in G. While these statements are a rather formal way of viewing the process, they in fact provide a simplifying point of view: Instead of selecting a state out of the whole phase space, we first restrict the subspace and then select a state out of this restricted space. As we will see, we can classify several important algorithms discussed in later chapters as such "two-step selection" algorithms.

We can regard the overall transition matrix of the two steps as the sum of all possible paths between the two states:

$$P(C'|C) = \sum_G P(C'|C, G)P(G|C). \tag{4.4}$$

Therefore, the detailed balance condition holds if it does so for each term in the sum; that is, it holds if

$$P(C'|C, G)P(G|C)W(C) = P(C|C', G)P(G|C')W(C'), \tag{4.5}$$

where $W(C)$ is the Boltzmann weight (equilibrium distribution).

4.4 Cluster updates

We now explore a second example of a two-step selection algorithm, so-called cluster algorithms. When first proposed, they were a marked departure from the Metropolis or heat-bath type algorithms, because they involve the flipping of clusters of spins without rejection. For many years, algorithms that did more than a single-spin update were sought to mitigate the expense often encountered in equilibrating simulations, particularly near a continuous phase transition, but simply defining a unit of local spins generally had minor, if any, benefits. In many applications, cluster algorithms (Swendsen and Wang, 1987) enjoy a much shorter computational auto-correlation time than single-spin algorithms. Sometimes they

Algorithm 7 Swendsen-Wang cluster algorithm for the ferromagnetic Ising model.

```
Input: Current Ising configuration, $K \equiv J/kT$.
  for every pair of interacting spins $s_i$ and $s_j$ do
    Place an edge connecting them with probability $\delta_{s_i,s_j}(1 - e^{-2K})$ ;
  end for
  for every cluster of connected spins in the graph do
    With probability $\frac{1}{2}$, flip all spins in the cluster ;
  end for
  return the updated configuration.
```

reduce the sign problem (Chandrasekharan et al., 1999, 2003). Another important feature is that they can be parallelized (Chapter 13), while other methods that reduce critical slowing down often do not yet share this property. While still a two-step algorithm, they gain their advantage by changing the configuration on length scales that are relevant to the physics of the problem, as opposed to updating the degrees of freedom that define the microscopic model.

4.4.1 Swendsen-Wang algorithm

The first cluster algorithm was proposed by Swendsen and Wang (1987) for the Ising model (Algorithm 7). In their algorithm, a graph whose edges connect two parallel spins characterizes the restricted phase space G.[3] A *graph G* is defined by a non null set of *vertices* and a possibly null collection of *edges*. The edges specify undirected relationships between pairs of vertices. The Ising model, for example, is defined on a graph. The vertices are the lattice sites and the edges are the bonds specifying the pairs of sites whose spins interact.

The first step of the Swendsen-Wang algorithm is the construction of a graph. Here, the vertices of the graph are the lattice sites. The edges are a stochastically constructed subset of the bonds. The second step samples a configuration C from the ones that satisfy the restriction imposed by the graph. We do this sampling by flipping "clusters," where a cluster is a collection of connected spins in the graph.

With the addition of graph degrees of freedom, our transition probability matrix is now of the form (4.4), thereby transferring our task of proving the detailed balance condition for the whole phase space to the task of showing the condition in the restricted phase space (4.5). To this end, we introduce a new set of variables $\{g_{ij}\}$ representing connection ($g_{ij} = 1$) and disconnection ($g_{ij} = 0$) of lattice sites (presence or absence of edges), and we let G represent the set of all g_{ij}. We relate

[3] In what follows, because a graph imposes a restriction on the phase space and therefore defines a subset of phase space, we use the same symbol G for subsets of phase space and for graphs defined on a lattice.

these variables to subsets of the local phase space as follows: $g_{ij} = 1$ is compatible with $(s_i, s_j) = (-1, -1)$ or $(1, 1)$, while $g_{ij} = 0$ is compatible with $(s_i, s_j) = (-1, -1)$, $(-1, 1)$, $(1, -1)$, or $(1, 1)$. To be more specific, we write the probability $P(G|C)$ in (4.5) as

$$P(G|C) = \prod_{\langle ij \rangle} P(g_{ij}|s_i, s_j), \tag{4.6}$$

$$P(1|s_i, s_j) = 1 - P(0|s_i, s_j) = \delta_{s_i,s_j}(1 - e^{-2K}). \tag{4.7}$$

We now justify these choices for the conditional probabilities.

We can view the lattice itself as a graph with an edge between each pair of neighboring lattice sites. At each site (vertex) is an Ising spin variable. Let us overlay on this graph another one that places an edge between neighboring spins only if they are both up or both down. The overlaid graph creates disconnected clusters of aligned spins, and we obtain a new spin configuration by flipping each cluster with probability $\frac{1}{2}$. In the rest of this chapter, we call this overlaid graph simply a graph, and represent it by the symbol G.

If either C or C' is incompatible with G, (4.5) is trivially satisfied since both sides of the equation vanish. If, on the other hand, both states are in G, because of the random coin-flipping nature of the second step of the algorithm, the probability of selecting the final state is always $1/2^{N_{\text{cluster}}}$, independent of the initial or final state. Therefore, in (4.5), $P(C'|C, G) = P(C|C', G)$. Thus, the condition that remains to be verified is

$$C, C' \in G \;\Rightarrow\; P(G|C)W(C) = P(G|C')W(C').$$

To show this, we use (4.6) and factorize both sides of this equation to obtain

$$\prod_{\langle ij \rangle} P(g_{ij}|s_i, s_j)w(s_i, s_j) = \prod_{\langle ij \rangle} P(g_{ij}|s'_i, s'_j)w(s'_i, s'_j),$$

where $w(s_i, s_j) = e^{Ks_is_j}$ (see (4.2)). With (4.7), we find by simple inspection that corresponding factors on both sides of the equation are equal, if the pairs of spins are compatible with g_{ij} (constraint imposed by G):

$$(s_i, s_j), (s'_i, s'_j) \in g_{ij} \Rightarrow P(g_{ij}|s_i, s_j)w(s_i, s_j) = P(g_{ij}|s'_i, s'_j)w(s'_i, s'_j). \tag{4.8}$$

Explicitly, if $g_{ij} = 1$, then $s_i = s_j$ and $s'_i = s'_j$, so that (4.8) is satisfied. If $g_{ij} = 0$, all spin configurations are compatible with the graph and $P(0|s_i, s_j)w(s_i, s_j) = (1 - \delta_{s_i,s_j}(1 - e^{-2K}))e^{Ks_is_j} = e^{Ks_is_j} - \delta_{s_i,s_j}(e^K - e^{-K}) = e^{-K}$, and (4.8) is again satisfied. Thus, we have verified that the transition probability (4.6) for the Swendsen-Wang algorithm satisfies (4.5), and therefore proved that the Swendsen-Wang algorithm satisfies detailed balance. We summarize the Swendsen-Wang algorithm in Algorithm 7, and illustrate a cluster updating cycle in Fig. 4.2.

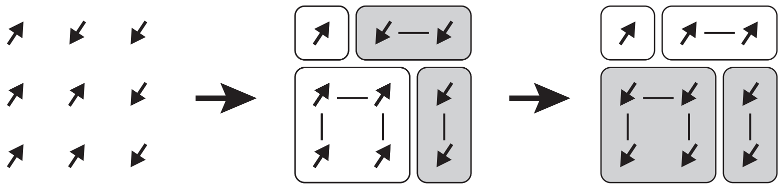

Figure 4.2 A cycle in the Swendsen-Wang cluster update of the Ising model.

To identify the clusters of connected spins, one can use for example the Hoshen-Kopelman clustering algorithm (Hoshen and Kopelman, 1976). It works as follows: For each site, one stores a cluster index and a root index. The cluster index is initially set equal to the site index. The *root index* of a site is determined iteratively by moving to the site stored in the cluster index until the cluster index becomes identical to the site number (which then defines the root index). If two sites are connected by a new bond, their root indices are computed and the cluster indices of the sites, as well as the cluster indices of the root sites, are set to the smaller root index. Once all the bonds have been inserted, the cluster index of each site is set to its root index. The sites with identical cluster indices belong to the same cluster. The algorithm is discussed in more detail in Section 13.4.

There is an even simpler and more efficient version of the Monte Carlo cluster algorithm, called Wolff algorithm (Wolff, 1989). This algorithm builds only a single cluster, starting from a randomly chosen site, and always flips it. A pseudo-code is provided in Algorithm 8. We leave it as an exercise to prove that the Wolff algorithm satisfies detailed balance.

4.4.2 Graphical representation

It is often useful to view the Swendsen-Wang cluster algorithm as a Markov process in an extended phase space where each state is expressed as $\tilde{C} = (C, G)$. We realize this extension by using a graphical decomposition of the Ising model weight originally proposed by Kasteleyn and Fortuin (1969) and Fortuin and Kasteleyn (1972),

$$e^{Ks_i s_j} = e^{-K} + \delta_{s_i,s_j}(e^K - e^{-K}). \tag{4.9}$$

The utility of this very simple formula becomes more apparent when we write the weight (4.2) as

$$w(s_i, s_j) = e^{Ks_i s_j} = e^{-K} + \delta_{s_i,s_j}(e^K - e^{-K}) = \sum_{g_{ij}=0,1} \Delta(s_i, s_j | g_{ij}) v(g_{ij})$$

with

$$v(0) = e^{-K}, \quad v(1) = e^K - e^{-K}, \quad \Delta(s_i, s_j|0) = 1, \quad \Delta(s_i, s_j|1) = \delta_{s_i,s_j}.$$

Algorithm 8 Wolff Algorithm for the ferromagnetic Ising model.

```
Input: Current Ising spin configuration, $K \equiv J/kT$.
  Randomly choose a lattice site $i$ (root of the cluster) ;
  for each added site $i$ do
    for each interacting site $j$ not yet on the cluster do
      if spins at $i$ and $j$ are parallel then
        add $j$ to the cluster with probability $1 - e^{-2K}$ ;
      end if
    end for
  end for
  Flip all spins in the cluster ;
  return the updated configuration.
```

Plugging this relation into (4.1) and using (4.2) yields

$$Z = \sum_C \sum_G \Delta(C,G)V(G), \tag{4.10}$$

with $V(G) \equiv \prod_{\langle ij\rangle} v(g_{ij})$ and $\Delta(C,G) \equiv \prod_{\langle ij\rangle} \Delta(s_i, s_j|g_{ij})$ being the generalized Kronecker delta expression that is one if and only if all spins connected by the graph are parallel to each other and zero otherwise. Hence, we have expressed the partition function as $Z = \sum_{C,G} W(C,G)$, with $W(C,G) = \Delta(C,G)V(G)$.

With these definitions, we now state the algorithm as a two-step process characterized by the transition probabilities

$$P_{\text{graph}}(C',G'|C,G) = P_{\text{graph}}(G'|C) = \frac{W(C,G)}{W(C)}$$

for the graph assignment step and by

$$P_{\text{flip}}(C',G'|C,G) = P_{\text{flip}}(C'|G) = \frac{W(C',G)}{W(G)}$$

for the cluster flipping step, with $W(C) = \sum_G W(C,G')$ and $W(G) = \sum_C W(C,G)$.

We also see that taking the partial summation over spin variables in (4.10) is easy, since the factor $\Delta(C,G)$ restricts the states to those generated by random cluster flips. If there are $N_c(G)$ connected clusters in G, then there are only $2^{N_c(G)}$ such states. All these states have the same weight. Therefore, taking the partial trace of spin variables simply produces $2^{N_c(G)}$, and we obtain

$$Z = \sum_G V(G) q^{N_c(G)}, \tag{4.11}$$

with $q = 2$ for the Ising model. Taking $q = 3, 4, \ldots$ we get the partition function of the q-state Potts model. This formula can also be used to study the q-state Potts model with non integer q (Blöte and Nightingale, 1982). Similar techniques also exist for the SU(N) quantum Heisenberg model (Beach et al., 2009).

4.4.3 Correlation functions and cluster size

The reason why the cluster algorithm can have a shorter auto-correlation time than a single spin-flip algorithm is that the cluster algorithm updates the state of the system by units whose physical size is comparable to the system's intrinsic spatial correlation length. For example, in the Swendsen-Wang algorithm, the typical range of the spatial correlations is related to the average size of the clusters by the formula

$$\langle s_i s_j \rangle = \text{Prob}(\text{“}i \text{ and } j \text{ are in the same cluster”}). \tag{4.12}$$

To prove this, we express the left-hand side as

$$\begin{aligned}\langle s_i s_j \rangle &= \sum_{C,G} W(C,G) s_i s_j = \sum_G W(G) \sum_C P(C|G) s_i s_j \\ &= \sum_G W(G) \chi(\text{“}i \text{ and } j \text{ are in the same cluster”}) \\ &= \langle \chi(\text{“}i \text{ and } j \text{ are in the same cluster”}) \rangle_{\text{MC}}\end{aligned}$$

where $\chi(\text{“statement”}) = 1$ if the statement is true, and $\chi(\text{“statement”}) = 0$ otherwise. From the last equation (4.12) directly follows.

Equation (4.12) says that the linear size of a typical cluster is roughly proportional to the spatial correlation length. During a Monte Carlo simulation, various spin configurations appear. If the system is near or already in equilibrium, then there are many clusters of up-spins and down-spins, and in each cluster the spins are aligned. The typical size of such clusters is comparable to the spatial correlation length. When we use a single-spin update algorithm, the process leading to the appearance or disappearance of a cluster takes a long time because it usually involves a boundary propagation.

Figure 4.3 illustrates this situation for the one-dimensional Ising model in which the system relaxes through the diffusion of the kinks. This way of propagating is why the relaxation time depends on the size of the clusters and therefore on the correlation length. The autocorrelation time in fact becomes proportional to the correlation length raised to some power (usually close to 2, reflecting the diffusive nature of the domain-wall motion). In a cluster algorithm, we modify spatial structures of the size of the correlation length in just one Monte Carlo step.

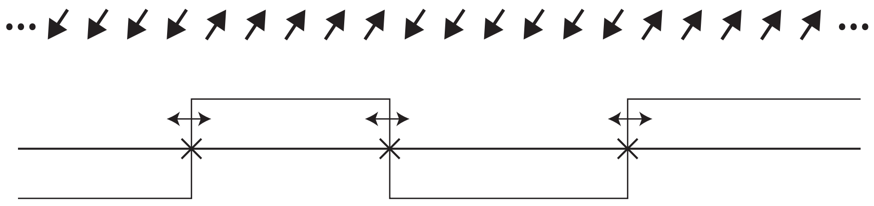
Figure 4.3 Relaxation through kink diffusion in the one-dimensional Ising model with single-spin update.

We note that the partition functions in (4.1) and (4.11) have the same value but express the Boltzmann distribution for the model as summations over two different sets of degrees of freedom. Whereas the spin configurations in (4.1) are the most obvious degrees of freedom in terms of the definition of the model, the graphs used in (4.11) are the most natural ones in terms of the physics of the model near its critical point. If we simulate away from the critical point at high temperatures, the Swendsen-Wang algorithm loses its efficiency advantage because the physics of the model changes. There, the computational cost of the cluster construction becomes a disadvantage.

4.5 Worm updates

So far we have seen two different examples of two-step Monte Carlo algorithms for many-body problems defined on lattices: the local update and the cluster update. In the case of the Ising model, because it is defined in terms of spin variables, it is natural to consider a Markov process defined in the space of spin configurations $\Omega \equiv \{C\} = \{(s_1, s_2, \dots, s_N)\}$. However, it is in general possible, and often quite advantageous, to reformulate the original problem as one defined in terms of a new set of degrees of freedom. We now discuss such a reformulation of the problem (Prokof'ev and Svistunov, 2001).

We start from the high-temperature series expansion of the Ising model, which can be obtained by decomposing the local Boltzmann factor and using the simple identity $e^{\pm x} = (\cosh x)(1 \pm \tanh x)$,

$$Z = \sum_C \prod_{\langle ij \rangle} e^{K s_i s_j} \propto \sum_C \prod_{\langle ij \rangle} (1 + t s_i s_j),$$

where $t \equiv \tanh K$. In the last expression, we neglected the constant factor $(\cosh K)^{N_{\text{bonds}}}$, with N_{bonds} being the number of interacting spin pairs in the lattice. By expanding the parentheses, we obtain many terms. Each term corresponds to a graph G that consists of the lattice sites as vertices and bonds for the edges associated with ts_is_j. (The edges for which 1 is chosen instead of ts_is_j are not included in G.) The resulting formula for the partition function is

$$Z = \sum_C \sum_G t^{|G|} \prod_i s_i^{n_i(G)}. \tag{4.13}$$

Here, the symbol G represents a graph defined on the lattice, and $|G|$ stands for the number of edges in G, while $n_i(G)$ is the number of edges in G that share the vertex i.

Let us take the summation over C before the summation over G. If $n_i(G)$ is an odd number for any given i, the summation over C kills the term because $\sum_{s_i=\pm 1} s_i^n = 0$ if n is an odd number. Therefore, we can neglect all graphs with "odd" vertices. We call G a *closed graph* if it does not have any odd vertices. With all this in place, we write the partition function, apart from an overall numerical constant, more simply as

$$Z = \sum_{G \in \Omega_0} t^{|G|}. \tag{4.14}$$

Now, the phase space Ω_0 is the set of all *closed* graphs defined on the lattice, and our task is to construct a Markov process for generating closed graphs with a frequency proportional to $t^{|G|}$. Note that both (4.11) and (4.14) express the partition function as a sum over graphs, but the graphs are quite different. Additionally, unlike the cases studied so far, the new phase space Ω_0 has a complicated structure and cannot be expressed as a simple product of many local phase spaces because of the "no-odd-vertex" constraint: We cannot propose a new state, or a closed graph, simply by choosing a pair of nearest neighbor sites at random and then removing or adding an edge connecting them, because doing so results in a state with two odd vertices and such a state does not contribute to the partition function.

What we can do is to allow in our phase space "artificial" states that do not contribute to the partition function. In the sampling, we generate states stochastically (some are closed graphs and others are artificial ones), but for the measurements, we discard all artificial states, and count only the "real" states. If we keep at most two odd vertices in the artificial states, we have a good chance of observing closed graphs with sufficient frequency. To keep this condition, when we add a new edge we make one of its ends cover an odd vertex, and when we remove an edge we remove one that has an odd end. As a result, we shift the location of the odd vertex by one lattice spacing. Algorithm 9 describes the procedure. In this algorithm, we introduce two odd vertices, which we call *head* and *tail*. We add or remove an edge by moving the head from the current site to one of its nearest neighbors.

Allowing up to two odd vertices means that we have extended the phase space from the zero odd-vertex space Ω_0 to $\Omega_0 \cup \Omega_2$, where Ω_2 is the space of graphs with

Algorithm 9 Worm algorithm for the Ising model.

```
Input: A high-temperature series configuration with or without worms.
  if there is no head and tail then
    Choose a site uniformly and randomly and place head and tail on it ;
  else if the head is on the same position as the tail then
    Remove head and tail ;
  end if
  if there are head and tail then
    Choose the direction of the head's motion randomly ;
    With probability $p$ move the head in this direction and change the edge state
    ($p = 1$ if the edge exists, otherwise $p = t$) ;
  end if
  return the updated configuration.
```

two odd vertices. Since the weight of a state in Ω_2 is undefined a priori, it becomes a parameter we can choose for our convenience. We choose a weight that not only covers Ω_2 but also reproduces the previous definition of the weight for Ω_0:

$$W(G) = a^{\nu(G)} t^{|G|}.$$

Here a is a constant we fix later and $\nu(G)$ is the number of odd vertices in G. Now our new task is to generate graphs G in $\tilde{\Omega} = \Omega_0 \cup \Omega_2$ with the weight $W(G)$. For a uniform lattice, we can do this with Algorithm 9 (Prokof'ev and Svistunov, 2001).

We note that the artificial states in Ω_2 include the case where two odd vertices happen to be on the same lattice site. Such a state is the same as a state in Ω_0 except that the position of the imaginary pair of odd vertices is defined and contributes a weight a^2 relative to the corresponding state in Ω_0. The first two cases in the conditional in Algorithm 9 represent transitions between these two states, say, $G \in \Omega_0$ and $G' \in \Omega_2$. The detailed balance condition between them is

$$\frac{1}{N} \times W(G) = 1 \times W(G'), \tag{4.15}$$

from which it follows that a^2 should be $1/N$.

The description of this head creation/annihilation procedure is rather formal and is mainly a reminder of the phase space structure we are using (in particular, that Ω_0 is still present). In practice, we skip going back and forth between Ω_0 and Ω_2 and simplify the procedure as described in Algorithm 10.

Now we consider the third case in the conditional in Algorithm 9, where the worm exists and the head is separated from the tail. In this case, we stochastically move the head. We first select the trial direction at random, and then move the head

Algorithm 10 Worm algorithm for the Ising model on a uniform lattice (simplified).

```
Input: A worm configuration with head and tail.
  if the head is on the same position as the tail then
    Randomly select a site and move the head and tail there ;
  end if
  Choose the direction of the head's motion randomly ;
  With probability $p$ move the head in this direction, and change the edge state
  ($p = 1$ if the edge exists, otherwise $p = t$) ;
  return the updated configuration.
```

in this direction with probability $p = 1$ ($p = t$) if an edge is present (no edge is present). Afterward, we change the edge state. Suppose we have no edge in the trial direction. Then, the final state of the motion, say, G', has one edge more than G, and because each edge carries the weight t, we have $W(G') = tW(G)$. The probability flow from the initial state G to the final state G' is the product of three probabilities: the probability $1/z_i$ for choosing the direction, where z_i is the coordination number of the site i at which the head is currently located, the probability t for accepting that motion, and the target probability of the state ($\propto W(G)$). We express the probability flow in the opposite direction in a similar fashion and then write the detailed balance condition as

$$\frac{1}{z_i} \times t \times W(G) = \frac{1}{z_j} \times 1 \times W(G').$$

This equality holds since $z_i = z_j$ is constant on a uniform lattice and $W(G') = tW(G)$ as we noted above. Thus, we confirmed detailed balance. In the same way, we can confirm detailed balance when an edge is in the direction of the head motion. We illustrate a cycle in the worm algorithm in Fig. 4.4.

In the above derivation, the introduction of Ω_2 might merely look like a trick for making the whole procedure work. However, it plays another important and useful role in the measurement of two-point correlation functions. As in the cluster update algorithm, the way the two-point correlation function is calculated within the present framework is the key to understanding the essential property of the worm update algorithm. In the high-temperature series expansion, the two-point correlation function is

$$\langle s_i s_j \rangle = \frac{Z_{ij}}{Z}, \quad Z_{ij} \equiv N \sum_{G \in \Omega_{ij}} W(G),$$

where Ω_{ij} is the set of graphs that has odd vertices on i and j and thus is a subset of Ω_2. Note that we have the factor N in the above expression because we have defined

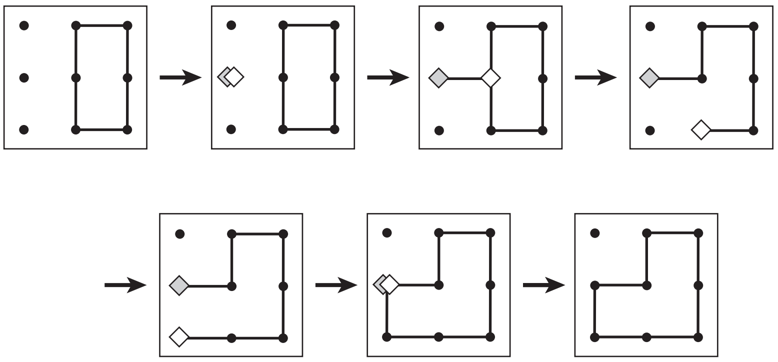

Figure 4.4 A worm update cycle for the Ising model, based on the high-temperature series representation. The head is depicted by the empty diamond and the tail by the shaded diamond.

the weight of the wormful states with this factor in (4.15). In the Markov process previously defined, the probability that the head and the tail are on i and j or vice versa occurs with a frequency proportional to $\sum_{G\in\Omega_{ij}} W(G)$, whereas the frequency of having a closed graph is proportional to $\sum_{G\in\Omega_0} W(G) = Z$. Therefore, the two-point correlation function equals the total number of the "i, j" events divided by the number of no odd-vertex events, multiplied by N. The factor N again comes from the definition of the weights of the wormful states relative to the wormless states. In a uniform system, we can further simplify the procedure: Because of translational invariance, instead of multiplying by N, we count all events where the location of the head relative to that of the tail is $r_{ij} \equiv r_i - r_j$ or is $-r_{ij}$ where r_i and r_j are the position vectors of sites i and j. Hence,

$$\langle s_i s_j \rangle = \langle \mathrm{Count}(r_{\mathrm{head}} - r_{\mathrm{tail}} = \pm r_{ij}) \rangle_{\mathrm{MC}},$$

where Count("event") is the number of times that "event" occurs during a head-creation/annihilation cycle.[4]

4.6 Closing remarks

We started with the Metropolis algorithm and focused on what is at its core as opposed to simply explaining its familiar application to the Ising model. The insight

[4] By now, it should be evident why the procedure is called *worm* update. Nonetheless, since it is only the current positions of the head and the tail that matter for the stochastic process that follows, and since the trajectory of the head has no significant meaning that makes it necessary to record it, the word "worm" misleads one to think about the wiggly "body" of a worm and may not be particularly appropriate. "Pac-Man" update, for example, would reflect the nature of the algorithm better. However, since the name "worm algorithm" has become standard among the people working in the field, we stick to this accepted name throughout this book.

of Metropolis et al. was the detailed balance condition. Their craftiness produced a general two-step algorithm for sampling in accord with this condition.

When we look at the "forest" and not the "trees," the cluster algorithm is a Metropolis-like two-step algorithm for a new set of degrees of freedom. These degrees of freedom are the edges of a graph and are not part of the Hamiltonian's dynamical variables. These edges link like spins on neighboring sites, thereby forming clusters of aligned spins reflecting those found in a physical system near a critical point. A nonlocal updating of spin configurations becomes possible and potentially more efficient. The graphs are introduced in such a way that they provide a path between an initial and final configuration of an overall transition probability matrix. We sample this path, that is, the graphs, in a way that satisfies detailed balance.

Finally, we discussed the worm algorithm, which is strictly speaking not a two-step Metropolis algorithm. However, it is in the same spirit as the cluster algorithm in that it also enlarges the configuration space of the model and imposes restrictions on it, by allowing loops with open ends and restricting the number of them to at most two.

The three algorithms illustrate the flexibility of the Monte Carlo approach as a tool for simulating many-body problems. Because the approach was applied to a finite-temperature problem, the algorithms needed to sample from an explicit distribution. In the construction of the algorithms, the detailed balance condition was as much an aid as it was a constraint. Although satisfying the detailed balance condition, the cluster and worm algorithms a priori are not guaranteed to be more efficient than the standard Metropolis algorithm with single-spin flips. They are in fact more efficient for only some applications, such as simulations near a second-order phase transition, for which they were designed.

In the next chapters, we move to quantum Monte Carlo algorithms. As we commented in Chapter 1, a quantum Monte Carlo algorithm is a classical Monte Carlo algorithm applied to a quantum problem recast into classical degrees of freedom. When we first learned quantum mechanics, we began by quantizing classical problems. In recasting a quantum problem into a form suitable for Monte Carlo sampling, we do not undo this step but rather "classicize" the problem. Once this is done, the algorithms discussed in this chapter reapply, or at least the basic concepts carry over to the simulation of quantum problems.

Suggested reading

J. S. Wang and R. H. Swendsen, "Cluster Monte Carlo algorithm," *Physica A* **167**, 565 (1990).

J. M. Yeomans, *Phase Transitions* (Oxford University Press, Oxford, 1992), chapters 6 and 7.

N. Prokof'ev and B. Svistunov, "Worm algorithm for problems in quantum and classical systems," in *Understanding Phase Transitions*, ed. L. D. Carr (Boca Raton, FL: Taylor and Francis, 2010).

Exercises

4.1 Single-Spin Update Algorithm. Prove that as long as $q_i(S'|S)$ satisfies the detailed balance condition itself, that is,

$$q_i(S'|S)\bar{P}(S) = q_i(S|S')\bar{P}(S'),$$

for all i with $\bar{P}(S)$ being the target distribution, the Markov matrix (4.3) satisfies the detailed balance condition regardless of the choice of p_i.

4.2 Two-Dimensional Random Walk. In a two-dimensional random walk on a square, the walker steps north, south, east, and west with equal probability. No "diagonal" moves are permitted.

1. Show that the probability of the walker returning to the starting point after $2n$ steps is

$$p_{2n} = \frac{(2n)!}{4^{2n}} \sum_{m=0}^{n} [m!\,(n-m)!]^{-2}.$$

2. Using the series expansion $(1+x)^{2n}$, establish the relations

$$\frac{(2n)!}{[m!\,(n-m)!]^2} = \begin{pmatrix} 2n \\ n \end{pmatrix}^2 \sum_{m=0}^{n} \begin{pmatrix} n \\ m \end{pmatrix}^2$$

and hence that

$$p_{2n} = \frac{1}{4^{2n}} \begin{pmatrix} 2n \\ n \end{pmatrix}^2.$$

3. Using Sterling's large n approximation, $n! \approx \sqrt{2\pi}\; n^{n+\frac{1}{2}} e^{-n}$, establish that

$$p_{2n} \approx \frac{1}{\pi n}.$$

4.3 Show that the Wolff algorithm (Algorithm 8) satisfies detailed balance.

4.4 Worm Update for the q-State Potts Model. Generalize the worm update for the Ising model discussed in Section 4.5 to the q-state ferromagnetic Potts model,

$$H = -J \sum_{\langle ij \rangle} \delta_{\langle s_i s_j \rangle} \quad (J > 0,\ s_i = 1, 2, \ldots, q).$$

4.5 Worm Update of a Nonuniform System. In Fig. 4.4, in the direction of the head's first move there are three options, whereas in the second move there are four. Therefore, the probability flow from the first state to the second is $(1/3) \times t \times w_1$, whereas the opposite is $(1/4) \times 1 \times w_2$, with $w_1 = t^6$ and $w_2 = t^7$ being the weight of the first and the second states, respectively. How can we modify the algorithm to make the algorithm work for nonuniform cases?