

Topic 3: Monte Carlo Simulation of Magnetic Materials (continued)

Monte Carlo simulations close to a phase transition are affected by *critical slowing down*. In the 2-D Ising system, the correlation length ξ becomes very large, and the *correlation time* τ , which measures the number of steps between independent Monte Carlo configurations behaves like

$$\tau \sim \xi^z ,$$

where the *dynamic critical exponent* $z \simeq 2.1$ for the Metropolis algorithm.

The maximum possible value for ξ in a finite system of $N = L \times L$ spins is $\xi \sim L$, because ξ cannot be larger than the lattice size! This implies that $\tau \sim L^{2.1} \simeq N$. This makes simulations difficult because the Metropolis algorithm time scales like N , so the time to generate independent Metropolis configurations scales like $N\tau \sim N^2 = L^4$. If the lattice size $L \rightarrow \sqrt{10}L \simeq 3.2L$, the simulation time increases by a factor of 100.

There is a simple physical argument which helps understand why $z = 2$: The Metropolis algorithm is a *local algorithm*, i.e., one spin is tested and flipped at a time. Near T_c the system develops large domains of correlated spins which are difficult to break up. So the most likely change in configuration is the movement of a whole domain of spins. But one Metropolis sweep of the lattice can move a domain at most by approximately one lattice spacing in each time step. This motion is stochastic, i.e., like a random walk. The distance traveled in a random walk scales like $\sqrt{\text{time}}$, so to move a domain a distance of order ξ takes $\tau \sim \xi^2$ Monte Carlo steps.

This argument suggests that the way to speed up a Monte Carlo simulation near T_c is to use a *non-local algorithm*.

Swendsen-Wang Cluster Algorithm

The essential idea of this algorithm suggested by R.H. Swendsen and J.-S. Wang, *Phys. Rev. Lett.* **58**, 86 (1987), is to identify clusters of like spins and treat each cluster as a giant spin to be flipped according to a random criterion. It is necessary that the algorithm obey the detailed balance condition. Swendsen and Wang found the following algorithm based on ideas from *percolation theory*:

Freeze/delete bonds: The 2-D square lattice, periodic boundary conditions, has $N = L \times L$ spins and $2N$ bonds between spins. Construct a *bond lattice* as follows:

- If the bond connects opposite spins, then delete it, i.e., temporarily uncouple the two spins. Note that opposite spins have a higher bond energy $+J$ if $J > 0$ and thus a higher effective temperature. So if J is large we are effectively “melting” the bond.
- If the bond connects like spins (both up or both down), then delete the bond with probability $e^{-2J/(k_B T)}$, i.e., generate a random deviate r and delete the bond if $r < e^{-2J/(k_B T)}$. Note that a like-spin pair has bond energy $-J$: so the change in energy in flipping one spin of the pair, i.e., in going from like to unlike spins is $\Delta E = 2J$. Bonds which survive this test are “frozen”. The probability of this happening is $1 - e^{-2J/(k_B T)}$. If $T = 0$ all like-spin bonds get frozen, while at $T = \infty$ the freezing probability is zero and all the bonds melt.

Note that constructing the bond lattice takes time of $\mathcal{O}(N)$ because there are $2N$ bonds.

Cluster Decomposition: After the bond lattice has been set up, the spins are decomposed into clusters. A cluster is simply a domain of spins connected to one another by frozen bonds. The lattice obviously decomposes into clusters in a unique way, and the decomposition is a deterministic problem.

Cluster decomposition is potentially time consuming. A naive algorithm can take time of $\mathcal{O}(N^2)$, so it is essential to use a decomposition algorithm that scales linearly with lattice size like Metropolis!

Spin Update: So far, constructing the bond lattice and identifying clusters has not changed any of the spins. The spins in each cluster are now “frozen” and the bonds between different clusters have been deleted. Each cluster is now updated by assigning a random new value ± 1 to all of the spins simultaneously, i.e., generate a random deviate r and flip all spins in that cluster if $r < 0.5$. Note that T does not play a role in this flipping decision.

The spin update step scales like the number of clusters which is $< N$.

Swendsen and Wang showed that $z \approx 0.35$ for this algorithm in the 2-D Ising model. Assuming that each Swendsen-Wang step scales like N , the running time for the simulation scales like

$$N\tau \sim N\xi^{0.35} \sim NL^{0.35} \sim N^{1.175},$$

which is *much* better than $\mathcal{O}(N^2)$ with Metropolis.

Efficient Cluster Decomposition Algorithms

A lattice with sites and bonds can be viewed as a *graph*, and the problem of finding all clusters is the problem of identifying *connected components* in graph theory.

It is essential to find a cluster labeling algorithm that scales linearly or almost linearly with lattice size N . There are two popular algorithms which have this property:

Backtracking algorithm: This is straightforward to program using a recursive function $f(i)$ where the argument i labels a lattice spin. An array of size N is set up which keeps track of whether a spin s_i has been visited by the backtracking algorithm or not. The algorithm works as follows:

1. Mark all spins as not yet visited.
2. Call $f(i)$ on each spin in the lattice. The function does the following:
 - If s_i has been visited, then return. No further testing needed.
 - Otherwise mark s_i as visited.
 - Repeat for each of the 4 nearest neighbors s_j of s_i :
 - If the bond $i-j$ is frozen, then call $f(j)$. This is the recursive step!

A little thought shows that the recursive calls will systematically visit every spin in a cluster connected by frozen bonds. As the lattice is swept, all distinct clusters will be identified.

The recursive function can be given a second argument which can be

- an unique cluster number which can be used to mark all spins in the recursive tree, or
- the second argument can be a flag to update the cluster spin simultaneously with marking: recall that all spins in a cluster are later updated based on the random test $r < 0.5$.

This backtracking algorithm is easy to understand and program, but is not the most efficient available because many sites will be tested and found to have been visited before.

Hoshen-Koppelman Algorithm: A more efficient algorithm was found by J. Hoshen and R. Kopelman, *Phys. Rev. B*, 3428 (1976). It is somewhat difficult to understand and program. The essential ideas underlying this algorithm are as follows:

- The spins (sites) of the lattice are visited in regular order. Thus moving along columns left to right and rows top to bottom, the left and top neighbors of a spin will have been visited, and the right and bottom neighbors will not.
- Each spin is assigned a cluster label. If the spin is connected to its left and/or top neighbor by a frozen bond, then it is given the smaller of the two labels. Otherwise, it is given a new unique label larger than all labels assigned so far.
- The major problem with this procedure is that clusters will in general be assigned more than one label! To fix this problem, the labels in a cluster are classified as proper or improper (good or bad). A cluster has only one

proper label and the others are improper. The algorithm maintains an array A containing “labels of labels” which classify labels into these two categories. This array is indexed by the value of the label. The proper label of a cluster is the minimum value of all of the labels assigned to its spins. When a new label ℓ is created, it is marked as proper by setting $A[\ell] = \ell$. A labeling conflict can arise in scanning the lattice when a spin is connected to *both* its left and top neighbors, and these two neighbors have different labels. When this happens the array value of the larger label is set to the smaller label $A[\ell_{\max}] = \ell_{\min}$. Any label for which $A[\ell] \neq \ell$ is improper. Given an improper label, its proper value can be found using the iteration

- While $A[\ell] \neq \ell$ set $\ell = A[\ell]$.

Wolff Single Cluster Algorithm

Two years after Swendsen and Wang published their algorithm, U. Wolff, *Phys. Rev. Lett.* **69**, 361 (1989) published an even more efficient algorithm based on constructing and flipping one single cluster at a time. The algorithm works as follows:

- Choose a spin s_i at random in the lattice flip it and mark it as a cluster spin.
- Grow a cluster with this spin as “seed” by checking each of its 4 neighbor spins:
 - If the neighbor is marked as a cluster spin, continue with the next neighbor, or quit if 4 neighbors are done. Otherwise:

- If this neighbor is *opposite* to the seed spin, then add it to the cluster with probability $1 - e^{-2J/(k_B T)}$, i.e., generate a uniform deviate r and if $r < 1 - e^{-2J/(k_B T)}$ flip the spin, mark it as belonging to the cluster, and recursively check each of *its* 4 neighbor spins.

Note that the decision to add a spin to the cluster follows the same probability criterion $r < 1 - e^{-2J/(k_B T)}$ to freeze a bond in the Swendsen-Wang algorithm. This implies that Wolff clusters have the same statistical properties as Swendsen-Wang clusters.

A simple argument shows that the Wolff algorithm is potentially more efficient than the Swendsen-Wang algorithm. Imagine the lattice partitioned into Swendsen-Wang clusters. The Wolff algorithm flips a single cluster got by choosing the seed site at random. This random choice obviously favors larger clusters. Flipping larger clusters is more likely to result in uncorrelated configurations!