Analysing the performance of Monte Carlo cluster algorithms

R. Singh, rs2910

29th April 2024

Abstract

In statistical mechanics, the partition function is used to find average thermodynamic properties (e.g. average magnetisation), it involves summing over all possible states of the system, which is unfeasible for systems of practical interest due to the immense number of states, it's computationally expensive. The objective of Monte Carlo methods in statistical mechanics is to generate different equilibrium states in configuration space to estimate thermodynamic variables, rather than explicitly finding the partition function. Local Monte Carlo update algorithms have the disadvantage of being slow in reaching equilibrium states and long autocorrelation times. Cluster algorithms attempt to remediate this by changing a larger proportion of the system and don't suffer from critical slowing down. The performance of some cluster algorithms in statistical mechanics, especially the Ising model, is investigated in this report.

1 Introduction

The central object in statistical mechanics problems is the partition function (Z) of the system, which in the canonical ensemble is:

$$Z = \sum_{\substack{system \\ microstates}} \exp(-\beta E_{microstate}) \qquad \beta = \frac{1}{k_b T}$$

In general, it's very difficult to find a closed form expression for the partition function or to find it numerically as the phase space of microstates is very large.

Instead of finding the partition function, we devise an algorithm that cycles through the equilibrium states of the system, using these equilibrium states we can estimate thermodynamic variables.

Local Monte Carlo methods such as the Metropolis algorithm, accomplish the task of cycling through equilibrium states, but they take a lot of steps to reach an equilibrium state and show large "similarity" between successive states, as only local updates are carried out. This leads to very large autocorrelation times.

This problem of large autocorrelation times naturally leads to the concept of cluster Monte Carlo methods. Where instead of local updates, clusters of spins in the system are flipped instead.

The Ising Model will be the main model of discussion to demonstrate the general concept, which can often be applied to other problems.

Monte Carlo simulations in statistical mechanics must obey ergodicity; the system must be able to access all states in phase space with equal probability.

If the algorithm evolves equilibrium state i into j, it must equally be likely to evolve state j into i. This is ensured through the condition:

$$P(j) * T(j \to i) = P(i) * T(i \to j) \qquad (1)$$

Where P(i) is the probability of being in state i and $T(i \to j)$ is the probability of transitioning to to state j. Detailed balance ensures that each configuration is sampled in proportion to its equilibrium probability, thus providing an accurate representation of the system's behavior at equilibrium.

Detailed balance is the "trade off" of not having to compute the partition function but still getting equilibrium configurations representative of the phase space.

Without detailed balance, the simulation may spend disproportionate amounts of time in some regions of phase space, leading to inefficient sampling and inaccurate results. By obeying detailed balance, the simulation can efficiently explore configuration space and provide reliable estimates of equilibrium properties.

1.1 Autocorrelation time

The autocorrelation time of an algorithm quantifies how long it takes for the algorithm to reach a configuration that is sufficiently "different" to the initial configuration. If the autocorrelation time is short, fewer Monte Carlo steps are needed to obtain independent samples, which can significantly improve the efficiency of the simulation, as thermodynamic quantities can be estimated more accurately.

Define the autocorrelation time τ as the number of steps for $\langle m_i m_{i+\tau} \rangle$ to drop to $\langle m_i m_i \rangle / e$, where m_j is the magnetisation after j steps. The ith index doesn't need to be specified due to the time translation symmetry (i.e. You can take any equilibrium configuration as your initial configuration).

Autocorrelation times will be used as a measure of the efficiency of an algorithm in producing independent equilibrium configurations.

2 Local Monte Carlo methods

To motivate the need for cluster Monte Carlo algorithms, I will first investigate the problems with local Monte Carlo methods. The Metropolis algorithm will be used to illustrate this problem.

2.1 Metropolis Algorithm

The Metropolis algorithm (Metropolis, Rosenbluth, Rosenbluth, Teller & Teller 1953) is one of the first Monte Carlo algorithms for the Ising Model and it works by locally updating a spin on an $L \times L$[1] lattice.

The Ising Hamiltonian at zero magnetic field is:

$$H = -J \sum_{i,j} \sigma_i \sigma_j \tag{2}$$

Where the sum is over all the neighbouring pairs on the lattice, and σ_i are the Pauli spin matrices. Although the sum is only over nearest neighbours, the system has a large correlation length near the critical temperature.

The corresponding partition function in the canonical ensemble is:

$$Z = \sum_{\sigma_1=\pm1, \sigma_2=\pm1 \ldots} exp(-\beta H) \tag{3}$$

This system is of particular interest as it was analytically solved in two dimensions by Onsager (Onsager 1944).

The algorithm (Metropolis et al. 1953):

1. Randomly select a spin in the lattice
2. Calculate the energy change (ΔE) if the spin is flipped
3. If $\Delta E < 0$, flip the spin
4. If $\Delta E \geq 0$, flip the spin with probability $\exp(-\beta \Delta E)$

It can be shown that this satisfies the ergodic hypothesis:

Let the initial state(i) have energy E_i and the state(j) after one step of the Metropolis algorithm have energy E_j.

Case 1 ($E_i \neq E_j$):
Without loss of generality, let i be a state with lower energy and j be a state with greater energy. The transition probability for the Metropolis algorithm is $\min(1, \exp(-\beta E_i))$

$$P(j) \times T(j \to i) = (A \frac{\exp(-\beta E_j)}{Z}) \times (\exp(-\beta(E_i - E_j))) \tag{4}$$

$$= \left(A \frac{\exp(-\beta E_i)}{Z} \right) \times (1) \tag{5}$$

$$= P(i) \times T(i \to j) \tag{6}$$

[1]L will refer to the lattice side length throughout the report.

Case 2 ($E_i = E_j$):
The transition probability is $\min(1, \exp(-\beta * \Delta E)) = 1$ since $\Delta E = 0$. So the transition probability $i \to j$ is the same as $j \to i$.

2.2 Advantages of the Metropolis algorithm

The Metropolis algorithm's biggest advantage is the very low runtime of the algorithm, O(1) for each step (In practice we define a Monte Carlo step of the Metropolis algorithm to consist of L^2 attempted spin flips. So, it's computational complexity is O(L^2)). It is also much easier to implement than the more complex cluster algorithms.

2.3 Disadvantages of the Metropolis algorithm and the need for cluster algorithms

The Metropolis algorithm is slow to reach an equilibrium configuration. We expect at least $\sim L^2/2$ single moves for a lattice to reach equilibrium, which would require 1 Monte Carlo step. But in practice it takes significantly longer than that, as shown in *Figure* 1.

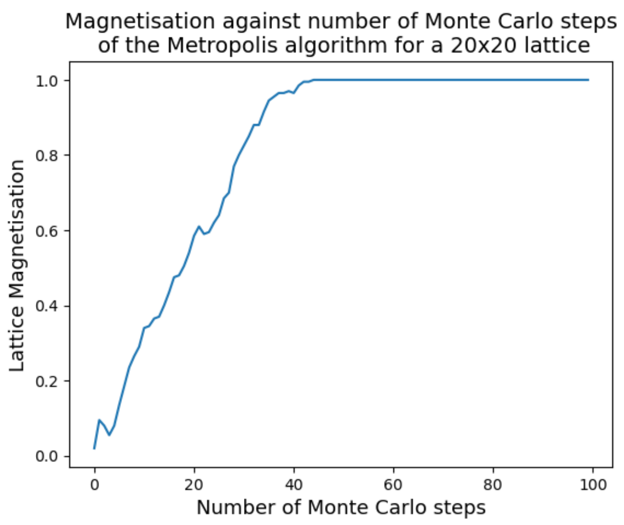


Figure 1: Magnetisation against number of Metropolis steps for a 20x20 lattice.

The local updates after each Metropolis move lead to "similar" configurations after each move so that the there is high correlation between Metropolis steps. Which leads to large autocorrelation times for the Metropolis algorithm, as seen in *Figure* 2 and *Figure* 3.

This means a large number of steps is required to be able to produce independent configurations.

This motivates the idea of flipping clusters of spins in the lattice rather than flipping individual spins, to decrease autocorrelation times.

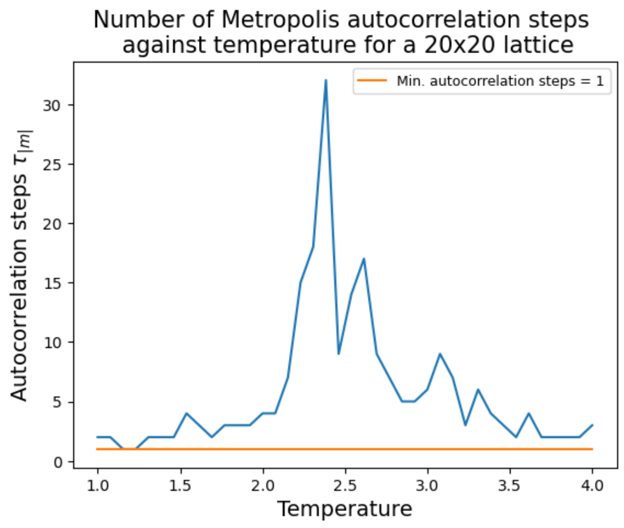


Figure 2: Number of autocorrelation steps for the Metropolis algorithm against temperature for a 20x20 lattice

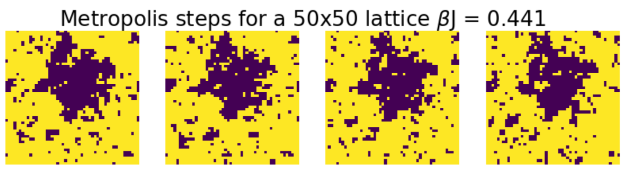


Figure 3: 0, 1, 2, 3 Monte Carlo steps of Metropolis algorithm at critical temperature on a 50×50 lattice

2.4 Critical slowing down

Critical slowing down is best illustrated by *Figure* 2 and the .mp4 file attached. Near the critical temperature T_c [2] the autocorrelation of the Metropolis algorithm diverges. This is due to large clusters forming near the critical temperature. The probability of flipping a spin in such a cluster is $\exp(-\beta_c(8J)) = (1+\sqrt{2})^{-4}$ which is around 3%.

The algorithm rejects most proposed spin flips, so it has effectively been "stalled" by the large clusters.

Most of the changes to the lattice happen when the Metropolis algorithm picks a spin at the boundary of a cluster, so that the energy change from flipping that spin isn't as large. This is seen in Figure 3.

3 Ising Model Cluster algorithms

3.1 Wolff algorithm

The Wolff Algorithm is a rejection free algorithm, that grows clusters in the lattice and flips them. The main issue is choosing the size of the cluster to be flipped, in such a way that the system obeys detailed balance.

The algorithm (Wolff 1989):

1. Randomly select a spin(i) in the lattice.
2. Form a cluster by adding the nearest neighbours(j) with probability $p = 1 - \exp(-2\beta J)$ if the spins i and j are parallel.
3. For the spins added to the cluster, recursively add the neighboring spins that are parallel to (i) with probability p, until no more spins can be added.
4. Flip all the spins in the cluster

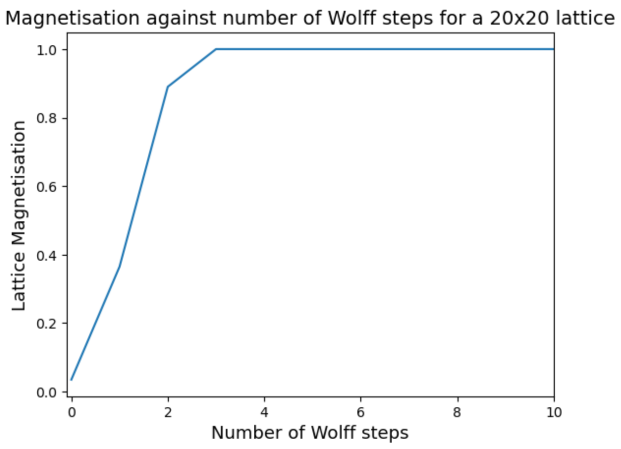


Figure 4: Magnetisation against number of Wolff steps for a 20x20 lattice.

This approach leads to much lower autocorrelation times than the Metropolis algorithm as shown in *Figure* 5[3]. (Wolff 1989). The autocorrelation times are a factor of ~ 2 smaller. Flipping large cluster leads to the algorithm being able to explore the phase space much faster, and being able to form very "different" lattice configurations.

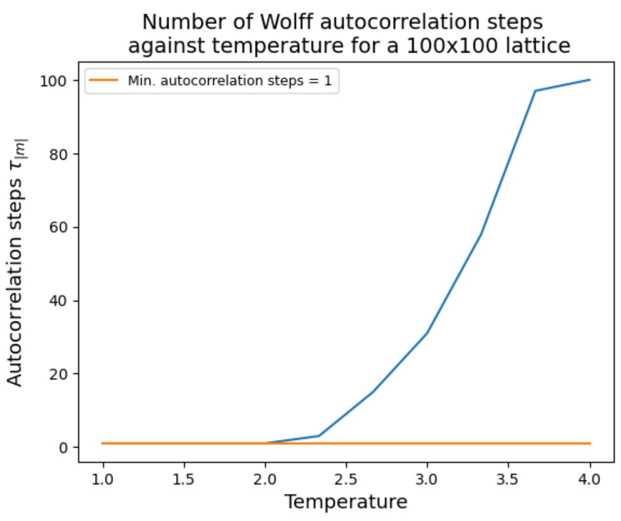


Figure 5: Number of autocorrelation steps for the Wolff algorithm against temperature for a 20x20 lattice

The Wolff algorithm is better than the Metropolis algorithm at producing independent configurations near

[2] $T_c = \frac{2J}{k_b ln(1+\sqrt{2})} \approx 2.269 \frac{J}{k_b}$ (Onsager 1944)

[3] Above T_c the system remains in the same state (all spin up/down) hence the higher autocorrelation times.

the phase transition temperature. Due to critical slowing down the Metropolis Algorithm doesn't change the lattice much and rejects most of the proposed spin flips. The Wolff algorithm doesn't suffer from this, since it's a rejection free algorithm.

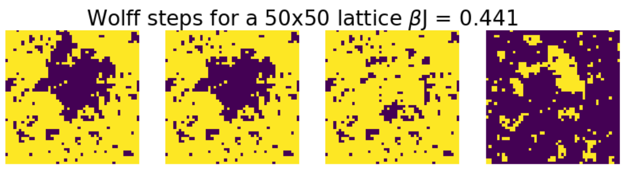


Figure 6: 0, 1, 2, 3 Monte Carlo steps of Wolff algorithm at critical temperature on a 50×50 lattice

3.2 Swendsen–Wang Algorithm (SW)

The Swendsen–Wang algorithm is significantly different from the Metropolis algorithm, as it can potentially flip all the spins in one iteration. The SW algorithm forms clusters in the lattice and flips each cluster with probability 1/2, rather than always flipping the clusters.

The algorithm (Swendsen & Wang 1987):

1. A bond between all neighbouring spins is made with probability
   p $= 1 - \exp(-2\beta J)$ if the spins are parallel.
2. All spins bonded together (directly or indirectly) form a cluster
3. Flip individual clusters with probability 1/2

The autocorrelation times can be seen in *Figure 7*, which are a factor of ~ 10 smaller than the Metropolis algorithm.

The reason for this is the same reason the Wolff algorithm had lower autocorrelation times.

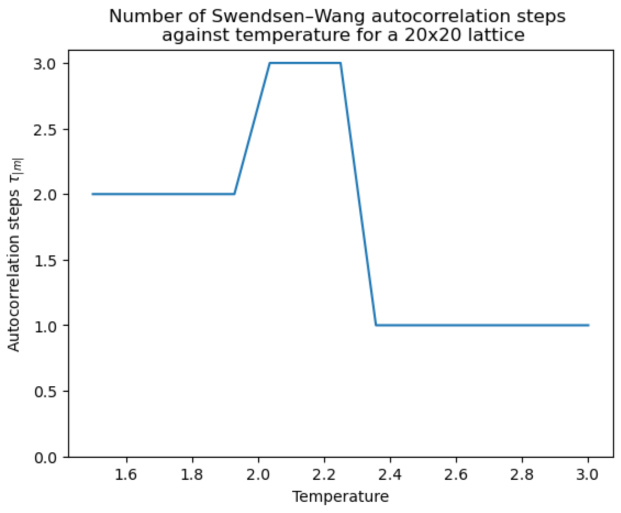


Figure 7: Number of autocorrelation steps for the Swendsen–Wang algorithm against temperature for a 20x20 lattice

The SW algorithm can flip many spins in the lattice, with its large updates, so it doesn't suffer from critical slowing down. However, because of such large changes in configurations, it is harder to ensure the lattice is still at an equilibrium configuration.

3.3 Performance Comparison between the Wolff and the SW algorithm

Although the autocorrelation times of the SW algorithm are lower than the Wolff algorithm's, it isn't enough to conclusively decide that the SW algorithm is "better" than the Wolff algorithm. There are some important differences:

1. Memory usage: The Wolff algorithm requires less memory than the SW algorithm since you don't need to store the labels of the clusters. The SW algorithm's memory requirements grows as $\mathrm{O}(L^2)$, whilst the Wolff algorithm's memory usage can grow between $\mathrm{O}(1)$ and $\mathrm{O}(L^2)$, depending on the size of the clusters.
2. Implementation: The SW algorithm is harder to implement due to the identification of clusters, whilst the Wolff algorithm "grows" its own cluster.
3. Runtime: The Wolff algorithm is much faster at large L. This is because the SW algorithm iterates over all lattice spins, so its computational complexity scales with $\mathrm{O}(L^2)$ whilst the Wolff algorithm's scales as $\mathrm{O}(L^{1.8})$.

4 Implementation and time complexity

4.1 Metropolis algorithm performance

The implementation of the Metropolis algorithm is very straightforward compared to the other algorithms. The complexity of Monte Carlo steps is $\mathrm{O}(L^2)$, since each step tries to flip L^2 spins.

4.2 Wolff algorithm performance

The implementation of the Wolff algorithm provided is similar to that given in the lecture notes (Budd 2022), but small modifications were made to lower the runtime and improve the readability. It is $\sim$ two times faster than the implementation outlined in the lecture notes (Budd 2022).

Rather than dealing with matrix indices the matrix is mapped onto a list with L^2 entries, the mapping is shown below:

$$\begin{bmatrix} 0 & 1 & 2 & \dots & L-1 \\ L & L+1 & L+2 & \dots & 2L-1 \\ \vdots & \vdots & \vdots & \vdots & \\ L^2-L & L^2-L+1 & L^2-L+2 & \dots & L^2-1 \end{bmatrix} \tag{7}$$

The (n, m) matrix entry is mapped onto the $(n \times L + m)$ entry in the list.

The main advantage from this mapping onto an array is seen for the SW algorithm. For the Wolff algorithm this provides a small improvement in runtime.

The time complexity of the implementation of the algorithm is $\sim O(L^{1.8})$, which is due to the growing of the cluster being dependent on the size of the lattice, near the critical temperature the clusters become very large, hence you get similar time complexity as the SW algorithm.[4]

4.3 Swendsen-Wang algorithm performance

The implementation of the SW algorithm attached in the .py file takes advantage of the mapping of the matrix onto a list shown in the matrix (7). This approach translates the issue of cluster identification into a problem of finding the connected components of a graph.

When a bond between neighbouring spins is made, their indices are stored in a tuple to represent the cluster formed by this bond, for example if the 1st list entry is bonded to the 2nd entry and the 4th entry, you get the cluster (1,2,4). Once all the bonds are made you obtain a list of clusters [(1, 2, 3), (3, 12), (16, 12, 10, 11)...]. The clusters formed are found by merging all the tuples that share at least one element.[5]

The Wolff algorithm doesn't need to do such a cluster identification so it's in general faster.

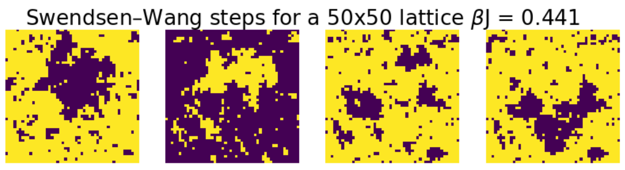


Figure 8: 0, 1, 2, 3 Monte Carlo steps of SW algorithm at critical temperature on a 50×50 lattice

5 Geometric cluster algorithms

The advantages brought by the cluster algorithms described in Sect. 3, in particular the suppression of critical slowing down, made it a widely pursued goal to generalize the SW and Wolff algorithms to fluid systems in which particles are not confined to lattice sites but can take arbitrary positions in a continuum.

The hard disks model is used to study the behaviour of systems composed of non interacting particles confined to move within a 2-dimensional region, such as ideal gases or colloidal suspensions.

[4]Notice the exponent is between 0 and 2, as expected for Wolff algorithm.

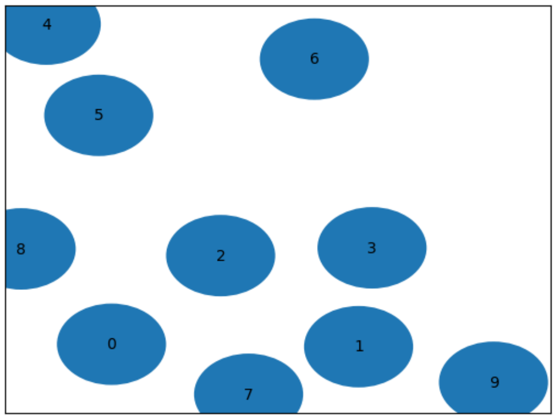
[5]Another approach is to use the Hoshen-Kopelman algorithm to identifty the clusters, but the implementation of this turned out to be slower.

Geometric Cluster Algorithms (e.g. (Luijten 2006)) are used to model more general inter-particle potentials, and also offer the advantage of lower autocorrelation times and don't suffer from critical slowing down.

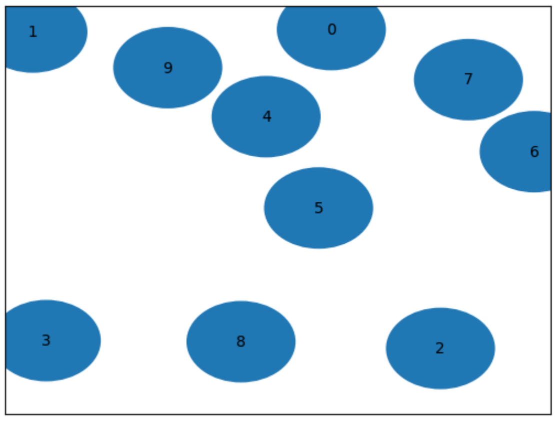
5.1 Non-Interacting Disk model

A simple rejection free algorithm for non-interacting disks consists of "taking advantage of the translational and point-reflection symmetry of the disk model on the torus" (Budd 2022). The challenge in designing such an algorithm is creating a new configuration that doesn't have any overlapping disks whilst still maintaining detailed balance.

The configuration of the system is described by a list of the positions of the disks as well as the radius of all the disks, an example of a possible configuration is given in *Figure* 9, where the disks have been given an arbitrary numbering.



(a)



(b)

Figure 9: a) Example of a possible configuration of the hard disk model.
b) Example of an iteration of the hard disks algorithm, the initial configuration is in figure 9a.

The algorithm works as follows (Luijten 2006):

1. Pick a random point in the disk configuration and let it be the "pivot".

2. Point reflect all the disks about the pivot.

3. By superimposing the original configuration and its point reflected version, group all neighbouring disks into clusters. I will refer to these disks as the interacting disks.

4. For each cluster, with probability 0.5, either retain the original configuration of the cluster or adopt the version of the cluster that is point reflected about the pivot.

An example of an iteration of the algorithm is given in *Figure* 9.

The code given in the .py file is a variation of the code given in the lecture notes at: (Budd 2022).

The algorithm is similar to the SW algorithm, after producing the clusters they are flipped with probabilty 0.5. It efficiently generates a new possible configuration, however since it doesn't handle more general potentials it's applicability is limited.

5.2 Interacting disk model

The non-interacting case can be modified to take into account more general potentials by adding a Metropolis-like criterion based on the energy change if the proposed change is accepted.

In practice, this is by done point reflecting the disks about the pivot probabilistically. If the energy change of the system is negative, the point reflection of the disk about the pivot is more likely.

It is important to stress that the radius of the disk in the interacting case is in general not the same as the radius of the particle being modelled, instead it represents the region in which the potential due to the disk is not negligible i.e. the interaction region. Importantly, defining such a radius is simple for short ranged potentials such as the Yukawa potential or the hard disk potential, but it is more complicated for "soft" potentials such as the Lennard-Jones potential, due to its long range ($\propto 1/r^6$).
For such potentials a lot of moves are rejected, since the disks are likely to overlap.

An alternative approach, motivated by the SW algorithm and the Wolff algorithm, is to first grow the particle cluster and then flip all the particles within the cluster. Notice the similarity of this to the Wolff algorithm. I'll refer to this as the GGCA (Generalised Geometric Cluster Algorithm).

The ergodicity and detailed balance of the algorithm are shown in (Liu & Luijten 2005).

The algorithm (Liu & Luijten 2005) works as follows:

1. Pick a random point in the disk configuration and let it be the "pivot".

2. Choose a random particle i, with position r_i and point reflect it about the pivot. Add i to the stack

3. Remove a particle (j) from the stack (FIFO[6] method), and all the particles interacting with j (As defined in the non-interacting disk model) are potentially added to the cluster

4. For each particle k interacting with j, at position r_k, point reflect it about the pivot with probability $\max(1,\, 1-\exp(-\beta\Delta E_{jk}))$.
Where $\Delta E_{jk} = V(|\underline{r}\prime_j - \underline{r}_k|) - V(|\underline{r}_j - \underline{r}_k|)$

5. Add each particle k that was point reflected, to the stack.

6. Repeat steps 3, 4 and 5 until the stack is empty.

NOTE: The primed version of each vector $\underline{r}\prime_j$denotes point reflecting $\underline{r}_j$ about the pivot.

As an example, consider a Coulomb potential[7] of the form $\beta V(r) = -1/r^2$. We expect to see the particles joining together to form clusters. The observed behaviour is shown in *Figure* 10. The algorithm successfully models the particle interactions and it forms particle clusters, as expected.

5.3 Performance of GGCA in binary mixtures

The GGCA is exceptionally good at handling systems with asymmetric particles compared to a conventional local update algorithm. In a binary system with particles of size σ_1 and σ_2, the GGCA's autocorrelation times exhibit very small dependence on the ratio σ_1/σ_2 (Luijten 2006), whilst a Metropolis-like algorithm's auto correlation times grow strongly with σ_1/σ_2. So, the GGCA gets increasingly advantageous with increasing size asymmetry and doesn't suffer from a large number of rejected moves near the critical temperature (Liu & Luijten 2005) (analogous to critical slowing down in the Ising Model).

5.4 Implementation challenges

The implementation of GCA involves some challenges:

1. Point reflection of the particles is complicated by the periodic boundary conditions and once one of the particles is point reflected it must not be point reflected again, even if it may interact with other particles.

2. Systems where interactions may change depending on the particles (size asymmetric) involved are more difficult to deal with as you have to keep track of the particle "type" you are dealing with.

3. The periodic boundary conditions make it so that you cannot just take the length of the difference of two position vectors, to find the distance between them.

[6]FIFO stands for First in First out. It is used here so that the first particle added to the stack is the first particle whose neighbours are considered for addition into the cluster.

[7]Such a potential is unphysical due to there being no short range repulsion.

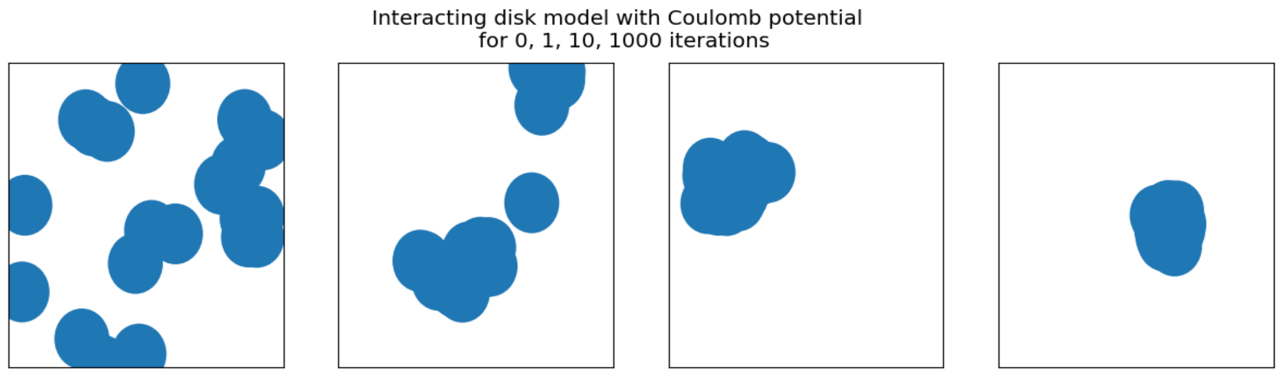


Figure 10: Evolution of identical particle configurations under Coulomb potential

6 Conclusions

Overall, Monte Carlo cluster algorithms in statistical mechanics allow for lower autocorrelation between successive system configurations. This allows for more accurate sampling of the phase space at equilibrium and hence better estimates of average thermodynamic variables; as well as faster convergence towards equilibrium.
The effects of this are more evident for large systems near critical temperature, where local Monte Carlo techniques are slow at producing independent configurations, where both the Metropolis algorithm and the Metropolis-like algorithm for the disk model showed they suffer from critical slowing down.

References

Budd, T. (2022), 'Criticality cluster algorithms'. [Online; accessed 20-04-2024].
URL: *https://hef.ru.nl/~tbudd/mct/lectures/cluster_algorithms.html*

Liu, J. & Luijten, E. (2005), 'Generalized geometric cluster algorithm for fluid simulation', *Phys. Rev. E* **71**, 066701.

Luijten, E. (2006), 'Introduction to cluster monte carlo algorithms', *Lect. Notes Phys.* **703**, pp. 13–38.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. & Teller, E. (1953), 'Equation of state calculations by fast computing machines', *J. Chem. Phys.* **21**, pp. 1087–1092.

Onsager, L. (1944), 'Crystal statistics. i. a two-dimensional model with an order-disorder transition', *Phys. Rev. Lett.* **65**(3-4), pp. 117–149.

Swendsen, R. H. & Wang, J. S. (1987), 'Nonuniversal critical dynamics in monte carlo simulations', *Phys. Rev. Lett.* **58**(2), pp. 86–88.

Wolff, U. (1989), 'Collective monte carlo updating for spin systems', *Phys. Rev. Lett.* **62**(4), pp. 361–364.

Appendix

The submitted ZIP file contains:

1. `project-rs2190.pdf` (Report)
2. `stat_mech_library` containing the modules `ising.py` and `disks.py`.

   `disks.py` contains the numerical algorithms used in the Disk Model.

   `ising.py` contains the numerical algorithms used in the Ising Model. As well as the code for a different implementation of the SW algorithm using the Hoshen-Kopelman cluster finding Algorithm.
3. `Cluster_Algorithms_rs2190.ipynb` (and the .py version, as instructed) containing all the plots and code for the plots.
4. `Ising_Model_crit_temp.mp4` animation of the Metropolis, SW, Wolff algorithms near critical temperature.

The libraries used are: numpy, collections, random, networkx (for cluster finding) and matplotlib.pyplot.