5CCP211C — Introduction to Numerical Modelling
Physics Department — King's College London

Kinetic Monte Carlo

Akshat Chandel; Taimur Ahmad Khan

November 18, 2018

Date performed .....November 13, 2018
Group 2
Pair .....Pair#

1 Background and Theory

Kinetic Monte Carlo Monte Carlo methods use random sampling in order to simulate time evolution of natural processes. Applications of Monte Carlo include finding areas through integration and the modelling of systems in statistical mechanics. Two of the most commonly used forms of the Monte Carlo method are Metropolis Monte Carlo and Kinetic Monte Carlo. Metropolis Monte Carlo is when random sampling is used through Markov Chains with reference to a detailed balance of a canonical distribution. For example in the case of the Maxwell-Boltzmann distribution and the Ising Model values of energy are used to determine changes in the system. Kinetic Monte Carlo is different however as it depends on transition rates or energy barriers between two different states to determine how a system evolves over time . In this report Kinetic Monte Carlo is used to model the DDA model to simulate epitaxial growth on a substrate, which is modelled as a 8 by 8 grid. Within this 8 by 8 grid occupied sites are given the number 1 and unoccupied sites are given the number 0. This is an infrequent process as there is no defined regularity with which evolution occurs.The substrate is illustrated by the images of the two lattices below, the first image showing an initial lattice and the second image showing a lattice after a process has occurred:-

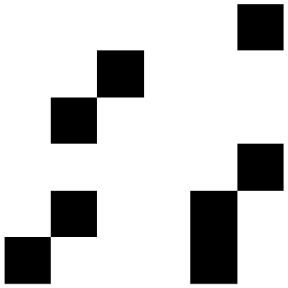

Figure 1: The initial lattice before any event has occurred

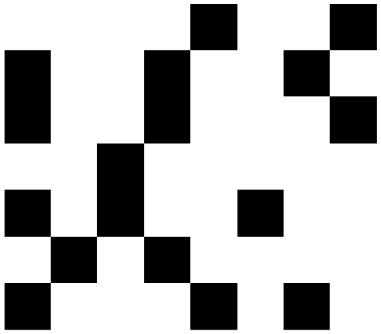

Figure 2: The lattice after an event has occurred

Overall, there are 3 processes which are modelled in this simulation:-

- Deposition(Prefactor=0.001f/s/Energy Barrier 0.01 eV: The placement of a new particle on a previously unoccupied place on the substrate. In terms of the grid a random site (i,j) is selected and if the site is unoccupied its value changes from a 0 to 1.
- Diffusion(Prefactor:0.1(f/s)/Energy Barrier=0.35eV) This is the movement of a molecule from an area of higher concentration to an area of

lower concentration. In terms of the lattice an occupied site (i,j) is selected. Then one of its unoccupied neighbours are randomly chosen and the molecule moves to that neighbour so the new site is any of (i+1,j), (i-1,j), (i,j+1) or (i,j-1). i.e none of the neighboring positions must be occupied.

- Agglomeration/Aggregation(Prefactor=0.1(f/s)/Energy Barrier 0.5eV): The movement of a particle similar to that of diffusion. The neighbouring positions to which it can move must also have unoccupied neighbours i.e neighbours and neighbours' neighbours must be unoccupied.In terms of the lattice an occupied site (i,j) is selected. Then one of its neighbours(i+1,j), (i-1,j), (i,j+1) or (i,j-1) is selected however the molecule only moves if all of the neighbour's neighbours are unoccupied. Note that agglomeration and aggregation are the same process but assigned different names due to there being two *barrier values* given.

The transition rates or 'barrier' which are used to determine the evolution of the system are given by the following equation;

$$r_i^{TST} = r_0 e \tag{1}$$

The rate given from the equation above,this rate is based on an idea known as transition state theory which assumes a quasi equilibirum between between the two states.The values of r_i for each evolution are summed and then multiplied by a random number between 0 and 1. The value of ur_i is then used to determine which event occurs. There is then a time increment, this is calculated by taking a natural logarithm of a new random number between 0 and 1,w, and dividing this by the cumulative probability. This time increment is then added to the time after each event occurs this is known as the Bortz-Kalos-Liebowitz algorithm, this is given by the equation below:-

$$t_{draw} = log(w)/k_{tot} \tag{2}$$

The time is initialized as being 0. The substrate is illustrated by the images of the two lattices below, the first image showing an initial lattice and the second image showing a lattice after a process has occurred.

The Kinetic Monte Carlo Algorithm is summarized as follows:-

- 1.Set time, t=0.
- 2.Form a list of all the transitions r_i π_i
- 3.Calculate the cumulative rate of k.
- 4.Get a uniform random number between 0 and 1.
- 5.Find the event to carry out, i, by multiplying u by k_{tot} and
- the i for which $k_{j-1} < uk_{tot} < k_j$

- 6. Carry out event i.
- 7.Find all k_j and recalculate the total rate k_tot
- 8. Get a new random number w between 0 and 1 and apply the Bortz-Kalos-Liebowitz algorithm to get the time increment and add this on to the overall time.

2 Overview of Algorithms

2.1 Plan and description

KMC Model A class for the Kinetic Monte Carlo is defined. The evolution method is initialised. It requires two methods; *evolve* which evolves the model system according to given specified events and *get rate* which calculates the probability of the said event taking place. This rate is calculated using equation (1) which can be written in code as shown;

```
    self.rate=self.prefactors*exp(-self.barriers*self.beta)
```

The following line of code shows how the events are chosen according to rate;

```
    u = uniform(0,1)
    events_index = sum(probabilities < u)
```

Where probabilities is calculated from the cumulative sum of the predefined rates given for each event. Overall for the system, a time increment *dt* is calculated for each event taking place. It is calculated using Voter's algorithm and acts as an internal system clock for this evolution. By adding a small line of code, one can calculate the total time passed after a random event takes place;

```
    self.time = self.time+dt
```

Object Lattice A separate class is defined for the gas lattice on which this process is simulated. The lattice is initialised and set according to the initial prefactors with a given density.

```
    2*int(ceil(sqrt(n_sites)/2))
    self.state=
choice([0,1],size=(self.size,self.size),p=[1-density,density])
```

This square lattice acts as a matrix with dual values of; 1,0. 1 acts as an occupied site within the lattice while 0 is empty of unoccupied. The most important part of the algorithm/code comes in the evolve function, which defines the events, the prerequisites for the events to transpire and classifies how the evolution takes place for the system. with four initial prefactors, four events are set which are deposition or adsorption, diffusion, agglomeration and aggregation. each event is assigned its own *events index*. Randomly produced, if the index

for a specific event comes up, the evolve function will continue that event as part of evolution of the system.

2.2 Evolution events

Adsorption/Deposition The first event for index=0 involves a particle dropped onto the sub-lattice into an empty site. Computationally this is shown to select a random empty site with i and j coordinates. If the coordinate site value is 0, it is turned into a 1 to show its evolution from unoccupied to occupied. Using a counter, the number of depositions are counted for each iteration.

Diffusion The second event for index=1, involves selecting a random occupied site or a singular particle which is unbound. The direct neighbours i.e the Northern, Eastern, Western and Southern neighbours are checked to see if they are unoccupied. If this *if/else* condition is satisfied, a random number generator is used to choose which movement the particle will perform. In terms of code, movement is shown by the following example;

```
self.state[i,((j+1)%self.size)]=1
self.state[i,j] = 0
print 'diffused north'
```

The above code shows a particle diffusing from site of coordinates i,j to site with coordinates i,j+1 or its northern coordinate.

Aggregation/Agglomeration Similar to diffusion, this event has the same directions of possible movements. If the index is equal to 2 or 3, again a random occupied site is chosen from the square lattice. The direct neighbours are checked if they are empty and if satisfied, the code moves forward to a secondary nested condition regarding the positions and occupational statuses of the neighbours' neighbours. The following example is of the secondary condition for a particle moving to a northern neighbour;

```
if (self.state[i-1,((j+1)%self.size)]==0
and self.state[((i+1)%self.size),((j+1)%self.size)]==0
and self.state[i,((j+2)%self.size)]==0):
self.state[i,((j+1)%self.size)] = 1
self.state[i,j] = 0
print 'aggregated north'
```

Random number generator is used to choose which direction, the particle will aggregate or agglomerate to. To explain simply, here a particle in a site of coordinates i,j can only move to neighbour i.e site 2 e.g i+1,j if the neighbours of site 2, i.e i+1,j+1 and i+2,j and i+1,j-1 are all unoccupied. This can be understood clearly from Figure 3;

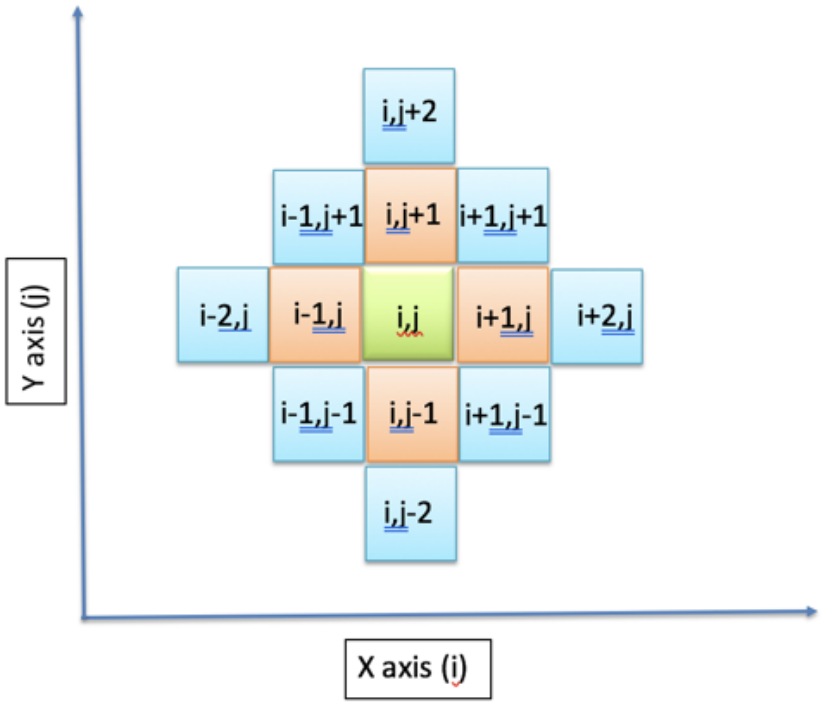

Figure 3: Lattice as i,j coordinate configuration.

2.3 Periodic Boundary

While manipulating the two dimensional array or the lattice, a periodic boundary problem comes up. If the randomly selected coordinate or its neighbour is the maximum site on the lattice, further calculations are deemed impossible and the code breaks down. The simplest solution is to *wrap* the coordinates around from the zeroth or minimum coordinates on the lattice. For example, if the randomly generated coordinate values include imax or jmax; at the end points of the square lattice then imax+1 or jmax+1 coordinates do not exist. Using modular arithmetic, the imax+1 or jmax+1 coordinate values are warpped around to the other side of the square lattice i.e imin and jmin values for the coordinate. The following code is an example of the computational approach to this theory;

```
    self.state[i,((j+1)%self.size)] = 1
```

The percentage symbol is used to wrap the value around. Here self.size is the magnitude of the length of this square lattice. Note that this periodic boundary condition is only required for i+n and j+n values and not for negative

n values. These conditions allow for the code to not break down at extreme valued positions in the lattice and the iterations continue for the designated amount of loops.

3 Results and Discussion of Data

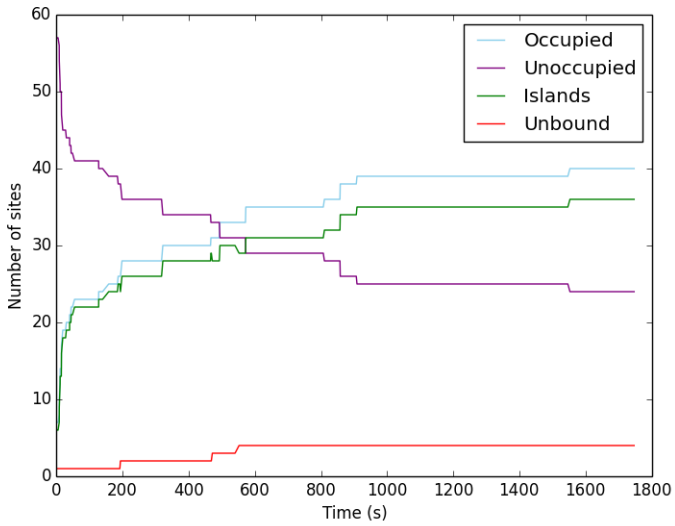

Figure 4: A graph illustrating how the system changes with time over 500 iterations.

The above graph illustrate how the state of the sites on the substrate changes over time. The number of unoccupied sites rapidly decreases until around 1000 seconds when it stagnates, this rapid increase indicates deposition occurring. This trend is, of course, mirrored by the trend for occupied sites. The number of unbound sites increases though at a very slow rate, indicating that that particles are diffusing and agglomerating in order to leave other particles free. Finally the number of islands increases over time at a moderate rate before finally stagnating, another feature is that the line for islands closely correlates with that for occupied sites implying that deposition is the main cause for islands. Overall the lattice gets more and more congested with particles eventually becoming more and more bound.