The Swendson-Wang algorithm

Submitted by: Yashasvee Goel and Anushka Menon

In [27]:

```
# Necessary modules

%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import numpy.random as rnd
#from random import choice
```

1. Swendson-Wang algorithm

In [28]:

```
def SW(self, grid_coordinates, spin_site_numbers, grid_spins):
    islands = []               #list of clusters which form islands of the same spin
    cluster_flips = []         #list which states if the cluster is flipped  .
    not_visited = np.ones((self.L, self.L), dtype= bool) #Boolean for spin site visited(=False) or not visited(=True)

    bonds = bond_eval(self, grid_spins)

    for i in spin_site_numbers:
        cluster = []
        flip_cluster = 2*rnd.randint(2) - 1
        spin_site_x = grid_coordinates[0][i]
        spin_site_y = grid_coordinates[1][i]
        cluster, grid_spins = back_track(self, spin_site_x, spin_site_y, bonds, not_visited, cluster, grid_spins, flip_cluster)

        if cluster != []:
            islands.append(cluster)
            cluster_flips.append(flip_cluster)

    return islands, grid_spins, cluster_flips
```

In [29]:

```
#Function to go over the spins and check the bonds
#If spins are opposite bonds set to zero
#Otherwise set to infinity with prob, p = 1- e^-J

def bond_eval(self, grid_spins):
    bonds = np.zeros((2, self.L, self.L,),dtype=float)
    chance_value = np.minimum(1, np.exp(-2*self.J))

    #For horizontal bonds
    delta_spin_hor = np.abs(grid_spins+np.roll(grid_spins,-1,axis=1))/2 # Divided by 2 to normalise
    nz_delta_spin_hor = np.asarray(np.nonzero(delta_spin_hor)) # Gives array with indices for non-zero elements

    for i in range(np.shape(nz_delta_spin_hor)[1]):
        if rnd.binomial(1, chance_value) == 1:
            bonds[0, nz_delta_spin_hor[0,i], nz_delta_spin_hor[1,i]] = 0
        else:
            bonds[0, nz_delta_spin_hor[0,i], nz_delta_spin_hor[1,i]] = np.inf


   #For vertical bonds
    delta_spin_ver = np.abs(grid_spins+np.roll(grid_spins,-1,axis=0))/2 # Divided by 2 to normalise
    nz_delta_spin_ver = np.asarray(np.nonzero(delta_spin_ver)) # Gives array with indices for non-zero elements


    for j in range(np.shape(nz_delta_spin_ver)[1]):
        if rnd.binomial(1, chance_value) == 1:
            bonds[1, nz_delta_spin_ver[0,j], nz_delta_spin_ver[1,j]] = 0
        else:
            bonds[1, nz_delta_spin_ver[0,j], nz_delta_spin_ver[1,j]] = np.inf

    return bonds
```

In [30]:

```
#Checks neighbour of the spins, if they are equal this functions jumps over to that spin and repeats itself

def back_track(self, x, y, bonds, not_visited, cluster, grid_spins, flip_cluster):
    if not_visited[x, y]:
        not_visited[x, y] = False  #Visited spin site
        cluster.append([x, y])
        grid_spins[x, y] = grid_spins[x, y] * flip_cluster

        if bonds[0][x][y] == np.inf:
            n_x = x
            n_y = (y + 1)%self.L
            cluster, grid_spins = back_track(self, n_x, n_y, bonds, not_visited, cluster, grid_spins, flip_cluster)

        if bonds[0][x][(y - 1)%self.L] == np.inf:
            n_x = x
            n_y = (y - 1)%self.L
            cluster, grid_spins = back_track(self, n_x, n_y, bonds, not_visited, cluster, grid_spins, flip_cluster)

        if bonds[1][x][y] == np.inf:
            n_x = (x + 1)%self.L
            n_y = y
            cluster, grid_spins = back_track(self, n_x, n_y, bonds, not_visited, cluster, grid_spins, flip_cluster)

        if bonds[1][(x - 1)%self.L][y] == np.inf:
            n_x = (x - 1)%self.L
            n_y = y
            cluster, grid_spins = back_track(self, n_x, n_y, bonds, not_visited, cluster, grid_spins, flip_cluster)

    return cluster, grid_spins
```

This doesn't account for if you have already visited the site what to do

you have to flip the entire cluster to the same no.

2. |m| and ϵ/J as a function of J

In [ ]:

```
#Incomplete

def m_absolute():
    m_absolute=np.append(m_absolute,np.absolute(np.mean(cluster)))
    return m_absolute

J_range = np.linspace(0.25,2,100)
N=20


#Plot for m_absolute vs. J
plt.figure(figsize=(12,10))
plt.title("|m| vs J ",fontsize=12)
plt.grid()
plt.xlabel("J")
plt.ylabel("|m|")
plt.plot(J_range, m_absolute)
plt.show()
```