PHYC90010 - Statistical Physics Assignment 1

Mitchell de Zylva - 756539

April 5, 2019

Contents

1 Question 1 **2**
1.1 Question 1 2

2 Question 2 **2**
2.1 Question 2 2

3 Question 3 **2**
3.1 Question 3 2

4 Question 4 **2**

5 Question 5 **5**
5.1 Question 5 5
5.1.1 Case 1: 5
5.1.2 Case 2: 6
5.1.3 Case 3: 6

6 Question 6 **6**
6.1 Question 6 6
6.1.1 Case 1: 6
6.1.2 Case 2: 7
6.1.3 Case 3: 7

7 Question 7 **7**
7.1 Question 7 7
7.1.1 Case 1: 7
7.1.2 Case 2: 8
7.1.3 Case 3: 8

8 Question 8 **9**
8.1 Question 8 9

9 Question 9 **9**
9.1 Question 9 9

10 Question 10 **10**
10.1 Question 19 10

1 Question 1

1.1 Question 1

The first rule essentially means that since a site that is on fire, will, after some appropriate time step, have consumed all the fuel in the region, thereby moving from red to black.

The second rule tells us that sites with trees will catch fire if they are nearby another site that is also on fire, but it will only catch fire in isolation based on some probability

The third rule states that the sites which are empty will eventually regenerate

2 Question 2

2.1 Question 2

The regrowth acts as a driving factor for the system, because it is what provides the ability for the system to effectively 'avalanche' and for the fire to spread. I originally supposed that the fires acted as a driving force for the system but the fire is what effectively avalanches the system, and so without the trees to burn through, the system would be unable to avalanche.

3 Question 3

3.1 Question 3

Included in 4 is the implemented code, which assigns black cells a value of 0, green cells a value of 1, and burning cells a value of 100. This allows us to take advantage of a 2D convolve method to calculate if any nearby cells are on fire.

It then iterates over the array, and checks to see if the data contained in a given cell is red, green, or black.

1. If it is red, it converts it to black.
2. If is it green, it checks to see if there is a neighbor that was on fire in the previous step of the simulation. If there was, it converts the green cell to red. If there is no neighbour on fire, it calculates the probability of fire spontaneously occuring
3. If the site is black, it calculates the probability of it randomly regenerating

These can be converted to some pseudocode, which looks like

```
if self.data[i][j] == red:
  self.data[i][j] = black

if self.data[i][j] == green:
  if any neighbor == red:
    self.data[i][j] = red
  if random.probability < self.f:
    self.data[i][j] = red

if self.data[i][j] == black:
  if random.probability < self.p:
    self.data[i][j] = green
```

4 Question 4

```python
import numpy as np
import scipy as sp
import pandas as pd
import random as rand
import matplotlib.pyplot as plt
import matplotlib.colors as colors
import matplotlib.animation as animation
import scipy.signal

```

```python

class fire_sim_world:
    def __init__(self, dim, f, p):
        self.data = np.random.randint(0, 2, dim)
        self.p = p
        self.f = f
        if self.f == 0:
            ind_1 = np.random.randint(0,len(self.data))
            ind_2 = np.random.randint(0,len(self.data[ind_1]))
            self.data[ind_1][ind_2] = 100

    def __repr__(self):
        return f'World({self.data!r})\n with probabilities ({self.p!r},{self.f!r})'

    def calc_s(self, runtime):
        '''
        Simulates behaviour for a given runtime, and gives the S-value (total number of pixels
 on fire) pdf
        '''
        assert runtime > 1000 , "Runtime needs to be greater than 1000"
        s_values = np.ndarray([runtime-1000])
        for i in range(runtime):
            number_on_fire = 0
            self.simple_update()
            if i > 1000:
                for j in range(len(self.data)):
                    for k in range(len(self.data[j])):
                        if self.data[j][k] > 99:
                            number_on_fire += 1
                s_values[i-1000] = number_on_fire
        return(s_values)

    def get_wait_times(self,runtime):
        '''
        Calculates the Wait Time (amount of time a pixel must wait before being reignited) Pdf
        '''
        wait_times = np.zeros(np.shape(self.data))
        for i in range(runtime):
            self.simple_update()
            for j in range(len(self.data)):
                for k in range(len(self.data[j])):
                    if self.data[j][k] > 99:
                        wait_times[j][k] = 0
                    else:
                        wait_times[j][k] += 1
        wait_times = np.ravel(wait_times)
        return(wait_times)



    def simple_update(self):
        '''
        Utilises a simple method for updating the world based on conditions specified in
question
        '''

        'Convolution creates a check matrix, with Periodic Boundary Conditions which we can
check for behaviour '
        check = sp.signal.convolve2d(self.data, np.array(
            [[0., 1., 0.], [1., 0., 1.], [0., 1., 0.]]), mode='same', boundary='wrap')

        for i in range(len(self.data)):
            for j in range(len(self.data[i])):

                'Behaviour for Red Cells'
                if self.data[i][j] > 99:
                    self.data[i][j] = 0

                'Behaviour for Green Cells'
                if self.data[i][j] == 1:
                    if check[i][j] > 5:
                        self.data[i][j] = 100
                    if rand.random() < self.f:
                        self.data[i][j] = 100

                'Behaviour for Black Cells'
                if self.data[i][j] == 0:
                    if rand.random() < self.p:
                        self.data[i][j] = 1


    def complex_update(self):
        '''
        Utilises a more complex method for updating the world based on conditions specified in question, and realistic updates to the system

        Allows for jumping multiple cells.
        '''

        'Convolution creates a check matrix, with Periodic Boundary Conditions which we can check for behaviour '
        check = sp.signal.convolve2d(self.data, np.array(
            [[0., 0., 1., 0., 0.], [0., 0., 1., 0., 0.], [0., 1., 0., 1.,0],[0., 0., 1., 0.,0],[0., 0., 1., 0.,0]]), mode='same', boundary='wrap')

        for i in range(len(self.data)):
            for j in range(len(self.data[i])):

                'Behaviour for Red Cells'
                if self.data[i][j] > 99:
                    self.data[i][j] = 0

                'Behaviour for Green Cells'
                if self.data[i][j] == 1:
                    if check[i][j] > 5:
                        self.data[i][j] = 100
                    if rand.random() < self.f:
                        self.data[i][j] = 100

                'Behaviour for Black Cells'
                if self.data[i][j] == 0:
                    if check[i][j] < 5 and check[i][j] > 0:
                        if rand.random() < (self.p - 0.2):
                            self.data[i][j] = 1
                    if rand.random() < self.p:
                        self.data[i][j] = 1


def animate(i):
    sim.simple_update()
    ax.imshow(sim.data, interpolation='nearest', cmap=cmap, norm=norm)


def generate_pdf(sim, sim_runs):
    assert sim_runs>1000, "Sim run must exceed 1000 time-steps"
    s_values = np.ndarray(sim_runs)
    for i in range(sim_runs):
        s_values[i]= sim.generate_pdf(sim_runs)
    return(s_values)

def calc_wait_time(sim):
    index_1 = np.random.randint(0,len(sim.data))
    index_2 = np.random.randint(0,len(sim.data[0]))
    count = 0
    while sim.data[index_1][index_2] < 99:
        count += 1
        sim.simple_update()
    return(count)

if __name__ == '__main__':
    f_prob = float(
        input("Enter probability of a Green Site catching fire (f) : "))
    p_prob = float(
        input("Enter probability of a black site regenerating (p) : "))
    size = int(input("Enter dimension (NxN) of simulation: "))

    sim = fire_sim_world([size, size], f_prob, p_prob)

    cmap = colors.ListedColormap(['black', 'green', 'red'])
    bounds = [0, 0.8, 5, 105]
    norm = colors.BoundaryNorm(bounds, cmap.N)
    speed = 100

    fig, ax = plt.subplots(figsize=(15, 15))
    ani = animation.FuncAnimation(fig, animate,
                                  frames=100, interval=speed,
                                  save_count=50)
    plt.draw()
    plt.show()
```

Listing 1: Python Code for Automaton

5 Question 5

5.1 Question 5

In order to obtain a Probabilty distribution function of the number of cells on fire, I updated the automaton until it acheived some measure of equilibrium (in this case, for at least 1000 steps). Then for the remaining steps in the simulation runtime, you count the number of cells that are on fire, and return the array of these values. In all cases, I made an effort to normalise the vertical axis, but for some reason, some of my graphs didn't normalise correctly. In this case, they still illustrate a relative probability, but isn't normalised correctly.

5.1.1 Case 1:

For the case where $f = 0$, we have to initialise a fire cell in the original matrix, since it will not ever randomly generate a fire.

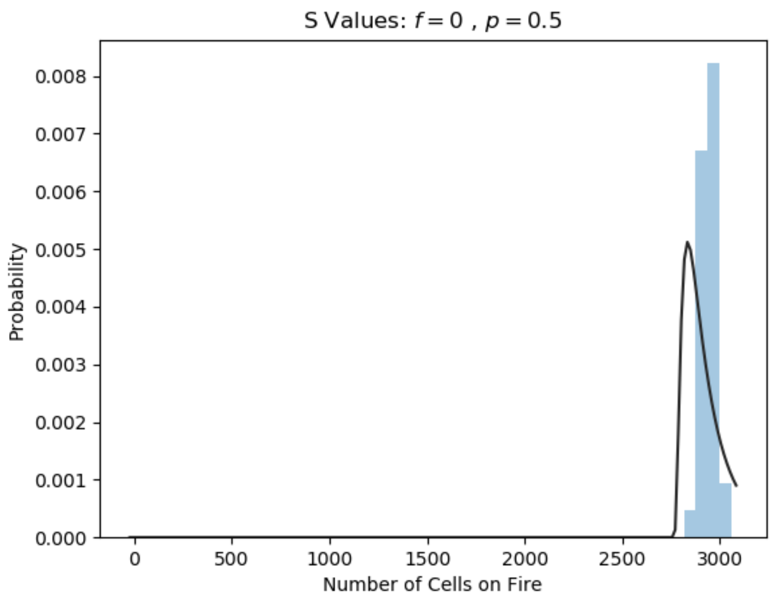

Figure 1: PDF of 100×100 simulation

5.1.2 Case 2:

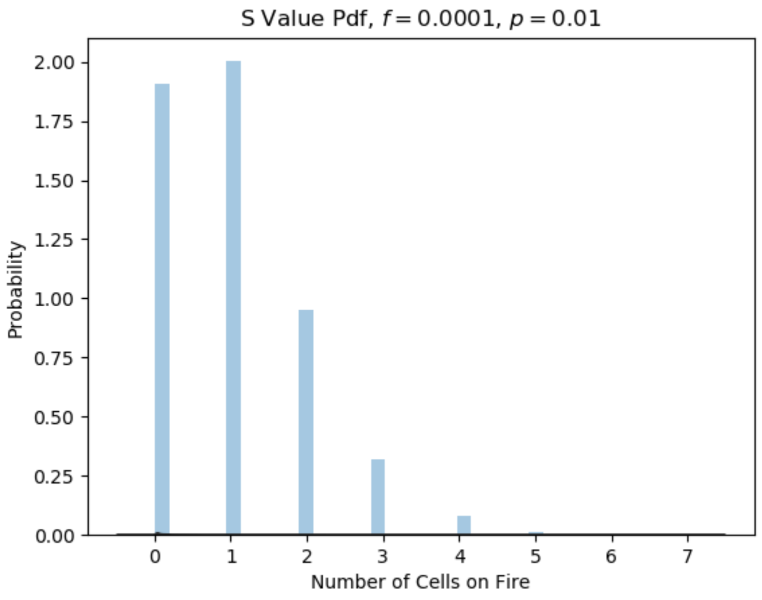

Figure 2: PDF of 100 × 100 simulation

5.1.3 Case 3:

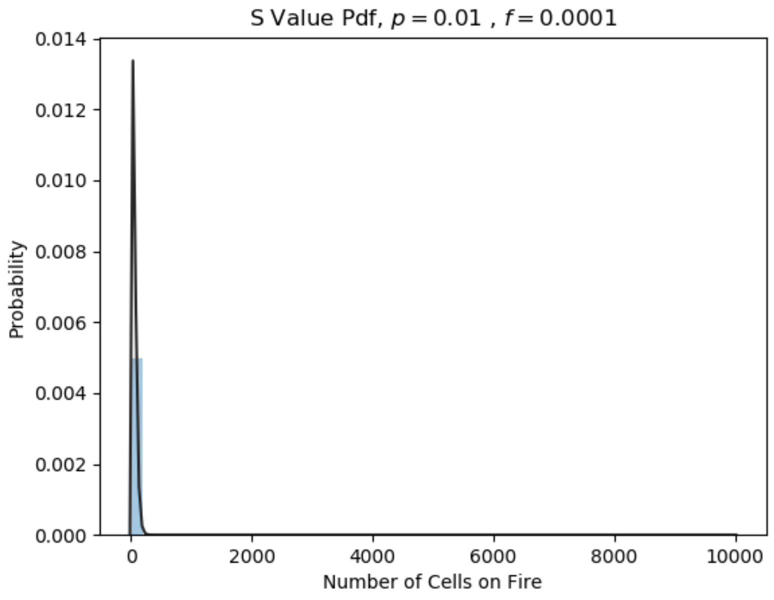

Figure 3: PDF of 100 × 100 simulation

6 Question 6

6.1 Question 6

6.1.1 Case 1:

For case 1, the inital fire spreads slower than the growth of the forest, and so the initial fire will never die out. When you watch the fire simulation, it appears that you have a continuous fire across the whole sim. This makes sense, since the forest is regenerating at a very rapid rate, and so the fire won't ever exhaust its supply. Its equilibrium state consists of being perpetually on fire, and based on the number of cells on fire, it appears that approximately 1/3 of the simulation is burning once this equilibrium state is reached.

6.1.2 Case 2:

For case 2, the probabilities used were taken so that the ratio of p/f was at least 100. This ensured they were sufficiently large enough and satistfied the condition $0 << p << f << 1$. Qualitatively, the fire initially burns through the forest, before settling into an equilibrium. Because the growth is so slow, it is unlikely that the forest will grow with clumps, and so when a fire starts, it will likely ignite only a single cell before burning itself out.

6.1.3 Case 3:

For case 3, the probabilites were reverse from the previous case. Qualitatively, this meant that the forest was growing faster than the fires might start. Once the initial forest burnt away, it regrows faster than in case 2. However, it still takes a while to get to a critical point where it cascades readily, and when it does, virtually the entire forest burns away again. This means that for the large majority of cases, there will be no fire.

7 Question 7

7.1 Question 7

The Waiting Times can be visualised by treating the system ergodically, and measuring the waiting times for each cell of a given system after it has run for a sufficiently long time and therefore reached equilibrium. Again I made efforts to normalise the graph, but it wasn't successful in one case.

7.1.1 Case 1:

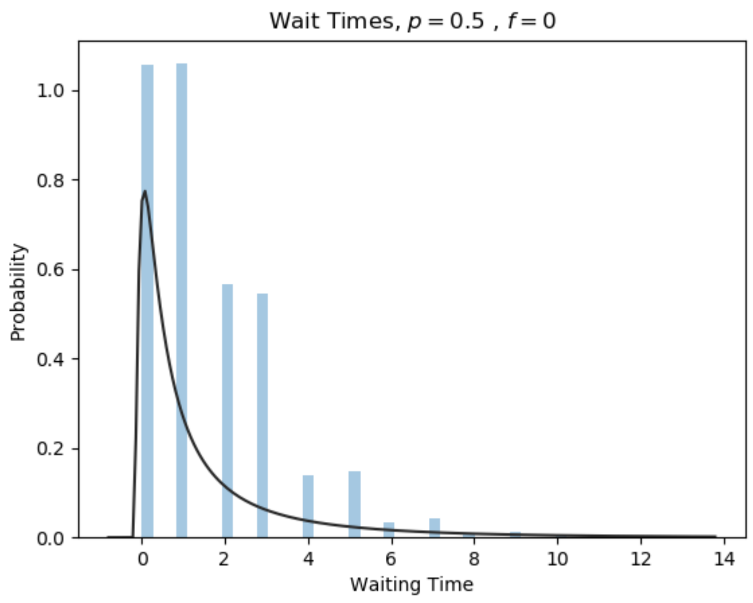

Figure 4: PDF of 100×100 simulation wait times

7.1.2 Case 2:

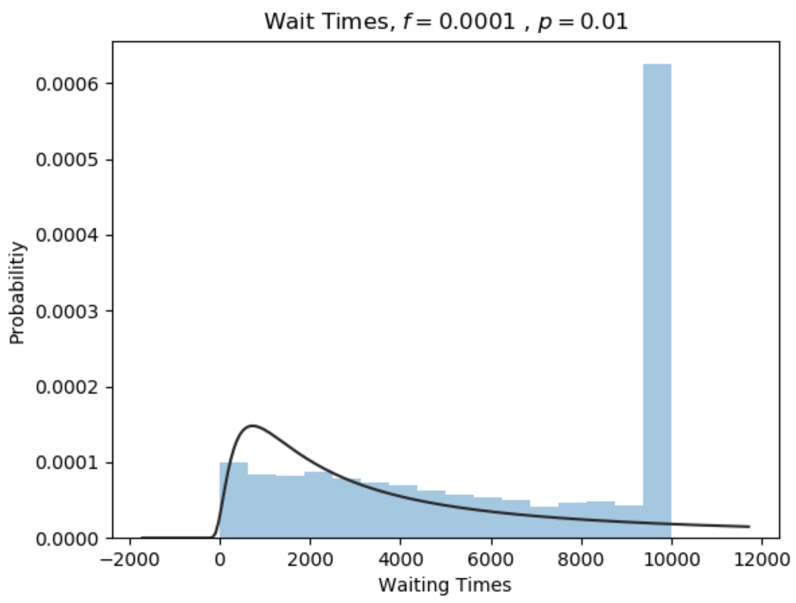

Figure 5: PDF of 100 $\times$ 100 simulation wait times

7.1.3 Case 3:

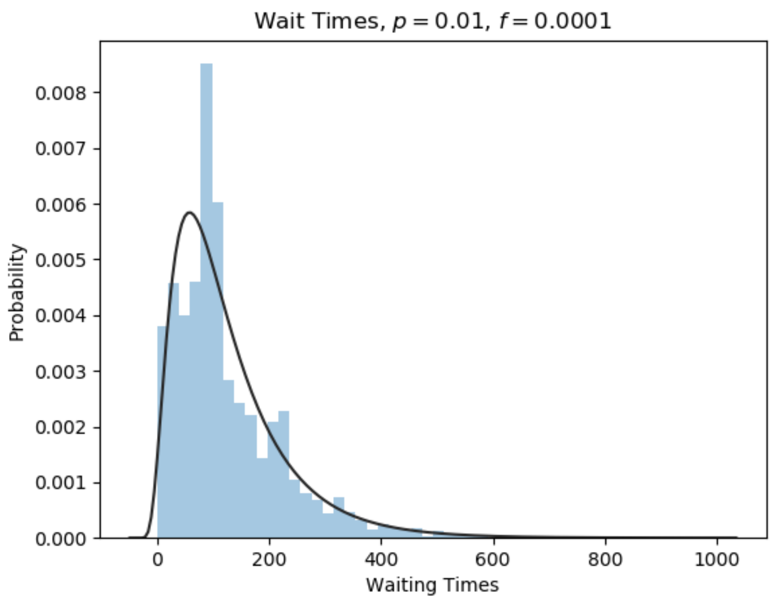

Figure 6: PDF of 100 $\times$ 100 simulation wait times

In all three cases, the simulation was run for 10000 before it was measured. What we see is that for the case where it is very likely that a cell will be on fire, the wait times are very low, with a peak at 0. For a system which regenerates slowly, this will be spread out much more, indeed, some cells might never catch fire again, which is what we see in the PDF of the wait times for Case 2. For Case 3, the cascades occur much more frequently, and so the waiting times are much less spread towards the higher times.

8 Question 8

8.1 Question 8

On average, the behaviour of the green and red sites will be governed by differential equations which broadly speaking follow an *inflow* - *outflow* model.

For Green Sites, the inflow is proportional to $1 - x_g - x_r$, since this number is representative of the number of cells that are black, related by the probability of them being converted from black to green p. Therefore the inflow is $p(1 - x_g - x_r)$.

Outflow is given by the sum of the probability of green cells turning red fx_g and the probability of finding a green and red site next to each other $4x_g x_r$. This therefore gives the differential equation governing the green sites as

$$\frac{dx_g}{dt} = p(1 - x_g - x_r) - fx_g - 4x_g x_r \tag{1}$$

For Red Sites, the inflow is the same as the outflow from green, in this case $fx_g + 4x_g x_r$ and the outflow is given by x_r since all red sites convert to black immediately.

This therefore makes the differential equation

$$\frac{dx_r}{dt} = fx_g + 4x_r x_g - x_r \tag{2}$$

Which are both exactly what we are expecting

9 Question 9

9.1 Question 9

The steady state solutions can be found by setting the left hand sides of (1) and (2) to zero, and solving the resulting system of equations.

$$0 = -fx_g - 4x_g x_r + p(1 - x_g - x_r) \tag{3}$$
$$0 = -x_r + fx_g + 4x_g x_r \tag{4}$$
$$\tag{5}$$

Taking (4) we see

$$4x_g x_r = x_r - fx_g$$

which we can substitute into (3), which yields

$$\begin{aligned} x_r(1 + p) &= p - px_g \\ x_r &= \frac{p(1 - x_g)}{p + 1} \end{aligned}$$

Substituting this now into (4) and expanding gives us a quadratic for x_g as follows

$$\begin{aligned} 0 &= -\frac{p(1 - x_g)}{p + 1} + fx_g + 4x_g \frac{p(1 - x_g)}{p + 1} \\ &= -p + px_g + pfx_g + fx_g + 4px_g - 4px_g^2 \\ \Rightarrow x_g &= \frac{-(4p + p + f + fp) \pm \sqrt{(4p + p + f + fp)^2 - 4(-4p)p}}{-8p} \\ \Rightarrow x_r &= \frac{p}{p + 1}\left(1 + \frac{-(4p + p + f + fp) \pm \sqrt{(4p + p + f + fp)^2 - 4(-4p)p}}{-8p}\right) \end{aligned}$$

For the case where $f = 0$ and $p = 0.5$, we obtain two steady state solutions

$$x_g = \frac{1}{4} \qquad x_r = \frac{1}{4} \tag{6}$$
$$x_g = 1 \qquad x_r = 0 \tag{7}$$

This makes some sense, since if we have a case where we don't initialise a cell on fire in the system, it will just grow until the entire automaton is green cells, with no fire cells. However, if you initialise the automaton with a single cell on fire, it will exist in roughly a state where one quater of the cells are on fire and one quater of the cells are green.

This contrasts slightly with the numbers we observe, where the number of cells on fire are approximately one third.

For the case where $f = 0.01$ and $p = 0.0001$, we again obtain two steady state solutions

$$x_g \approx 0.0095 \qquad x_r \approx 0.000099 \tag{8}$$
$$x_g \approx 26.243 \qquad x_r \approx -0.0025 \tag{9}$$

We can immediately discount the last solution, since you cannot have a fraction of cells over 100% or under 0. If we consider what this looks like with respect to our PDF, this means that about 0.01% of the cells will be red.

When we compare this to our PDF directly, this is close, but not exactly correct. The PDF has approximately 0.01% of cells (by counting directly, this is of the order 1-10), so there are actually slightly more.

For the case where $f = 0.0001$ and $p = 0.01$, we again obtain two steady state solutions

$$x_g \approx 0.249 \qquad x_r \approx 0.000743 \tag{10}$$
$$x_g \approx 1.003 \qquad x_r \approx -0.000036 \tag{11}$$

Again, we immediately discard the second solution. In this case, we expect approximately 0.07% of the system to be on fire in the long run.

Comparing this to our PDF, it appears that there is a bin which sits at roughly this number which is the most populous. Given the scale, it is somewhat difficult to see, but we expect that 0.07% of the simulation to correspond to roughly of the order 10 red cells, which is seems like it does.

For all three cases, the actual number of red cells is slightly higher than the predicted value, likely due to the inherent instability of the system. But to first order, the approximations made are sufficient to qualitatively describe the system.

10 Question 10

10.1 Question 19

In order to make the system more realistic, I would introduce a couple of changes to the simulation:

1. Allow for fire to jump, if for example, it was drive by wind, or some other factor. This would account for the fact that there are additional driving factors in the system other than just the probability of random fire generation.

2. Allow for increased clumping in regrowth, by esssentially preferencing the probability of trees growing next to an existing cell with trees. This would probably be incorporated by lowering the regeneration probability for a cell next to a green cell. Using the convolution method would actually be better for this, since it would allow for varying probability depending the number of neighboring green cells.

3. Make the tree growth time and the fire extinguishing time different orders of magnitude, since a tree cannot grow in the time it takes for a fire to burn through it. This correction perhaps needs to account for the number of neighboring fires, or neighboring forested cells. This fine tuning would require a bit more research into the precise conditions that might exist in the real world, to get a sense of the fire ignition probability and forest regeneration probability.