CS150 Spring 2018

Project : Traffic Flow Modeling - The Nagel-Schreckenberg Model
Due: May 18, 2018 at 4:00PM

1 Introduction

Traffic flow modeling is an example of broad topic of cellular automation. In cellular automation, discrete cells within the model domain have states that change from one time step to the next using clearly defined rules. The specific traffic flow model used in this project is the **Nagel-Schreckenberg** model. Cells within the model represent the length of a vehicle. A cell is either occupied by a vehicle at a given time or is "empty." The speed of the vehicle in an occupied cell is the other component of the cell's state.

The initial model domain consists of a single lane in which traffic flows from right to left. The lane is comprised of cells joined end to end. A 11-cell track is depicted in Figure 1. Each cell has a horizontal dimension equal to an automobile length. The example track has $N = 4$ vehicles distributed through the length of the track. (N cannot be greater than the number of cells.) The current vehicle locations are stored in a location list with length equal to the number of cells making up the track. The list elements are either 0 (unoccupied cell) or 1 (occupied).

A companion list gives the current speed of each vehicle. Vehicle speeds are given by integer values ranging from 0 (auto is not moving) to 5 (the maximum speed `v_max` that represents the speed limit). Additionally, the speed represents how may cells (call lengths) the vehicle will travel over the time interval (time step) between the current time and the next time for which the location and speed list are to be determined.

$loc\_c =$	0	1	1	0	0	1	0	0	1	0	0

$spd\_c =$	0	2	1	0	0	5	0	0	3	0	0

Figure 1: The location (loc_c) and speed (spd_c) lists for the current time step. The "first" cell on the left-hand-side has index value equal to zero.

2 The Nagel-Schreckenberg Model (NSM) Rules (single lane case)

The four following rules must be enacted in the order listed for all vehicles in a left-to-right direction sweep. The autos are traveling left to right, so that is the sweep direction.

1. **Acceleration:** If vehicle n ($1 \leq n \leq N$) has velocity v_n that is less than the maximum velocity v_{max}, the vehicle will increase its velocity by, at most, one unit. That is, `if v_n < v_max, then v_n = v_n + 1`

   Figure 1 Example: The four vehicles have locations $x_1 = 1$, $x_2 = 2$, $x_3 = 5$ and $x_4 = 8$ and speeds $v_1 = 2$, $v_2 = 1$, $v_3 = 5$, and $v_4 = 3$. After step 1, the new speeds will be $v_1 = v_1 + 1 = 3$, $v_2 = v_2 + 1 = 2$, $v_3 = v_3 = 5$ (at the v_{max} value) and $v_4 = v_4 + 1 = 4$

2. **Braking**: If a vehicle is at site i, and the next vehicle is at site $i + d$, and after step 1 its speed v_n is greater than d, then speed v_n is reduced to a value of $d - 1$. That is, `if v_n >= d, then v_n = d - 1`

   Figure 1 Example: The distances d for the vehicles in Figure 1 are $d_1 = x_2 - x_1 = 1$, $d_2 = x_3 - x_2 = 3$, $d_3 = x_4 - x_3 = 3$ and $d_4 = x_1^* - x_4 = 12 - 8 = 4$. Distance d_4 is determined by calculating the "wrap around" distance. The x_1^* location is determined by adding $x_1 + 11$. With the distances determined, the *adjusted* speeds, due to the need for braking, may be determined. For vehicle 1, v_1 and d_1 make $v_1 >= d_1$ true, so $v_1 = d_1 - 1 = 1 - 1 = 0$. For vehicle 2, v_2 and d_2 make $v_2 >= d_2$ false, so $v_2 = 2$. For vehicle 3, v_3 and d_3 make $v_3 >= d_3$ true, so $v_3 = d_3 - 1 = 2$. For vehicle 4, v_4 and d_4 make $v_4 >= d_4$ true, so $v_4 = d_4 - 1 = 3$.

3. **Randomization (reaction)**: For a given deceleration probability p, the speed v_n of vehicle n is reduced by one unit. That is, `if random.random() >= p, then v_n = v_n - 1`

   Figure 1 Example: Suppose for this step only vehicle 3 has its speed reduced, so that $v_3 = 1$, and all other speeds remain that same as determined in step 2.

4. **Driving**: After steps (a) - (c) have been completed for all vehicles, vehicle n at cell number x_n "drives" forward a distance (the number of cells) equal to v_n. That is `x_n = x_n + v_n`. If the distance places the vehicle beyond the last cell of the track, the vehicle "wraps around" the track and emerges on the left end.

   Figure 1 Example: Vehicle 1: $x_1 = x_1 + v_1 = 1 + 0 = 1$. Vehicle 2: $x_2 = x_2 + v_2 = 2 + 2 = 4$. Vehicle 3: $x_3 = x_3 + v_3 = 5 + 2 = 7$. Vehicle 4: $x_4 = x_4 + d_4 = (8 + 3)\%11 = 0$.

The result of applying steps 1 - 4 above is shown in Figure 2. After completion of the four steps outlined above, the new location and speed lists become the current lists, and the process of applying the four steps is repeated for a prescribed number of time steps.

$loc\_n =$	1	1	0	0	1	0	1	0	0	0	0
$spd\_n =$	3	0	0	0	2	0	1	0	0	0	0

Figure 2: The location `loc_n` and speed `spd_n` lists for the new time step.

3 Tasks

1. Write a Python program that includes the following functions.

   (a) `def init_positions(n_cells, n_veh)` - This function takes parameters `n_cells` (the number of cells for the track) and `n_veh` (the number of vehicles on the track) and returns a list of length `n_cells` of 1's and 0's. The locations of 1's denote the randomly selected initial locations of the `n_veh` vehicles.

   (b) `def init_speeds(loc_c)` - The initial (current) location list is input and a list of speeds is returned. Vehicle locations are denoted by '1' in the `loc_c` list, and randomly selected speeds from the interval [0, 5] are placed in the list. All other location have speed zero in the returned speed list.

   (c) `def calc_accel(loc_c, spd_c)` - The input parameters are the location list (`loc_c`) and the speed list (`spd_c`). They give the location and speed of the autos for the current (c for "current") time. The function uses the NSM rules to determine and return a new speed list `spd_n`.

   (d) `def calc_dist(loc_c)` - The location list, as an input parameter, is used to determine the distance between consecutive vehicles. The distance list is returned.

   (e) `def calc_braking(loc_c, spd_c, dis_c)` - Input parameters are the location, speed and distance lists. The function returns the speed list determining using the braking rules.

   (f) `def calc_reaction(loc_c, spd_c, p)` - Input parameters are the location and speed lists as well as the probability parameter `p`. The function returns the adjusted speed list.

   (g) `def calc_loc_spd(loc_c, spd_c)` - Input parameters are the location and speed lists. The function returns the updated (new) location and speed lists.

   (h) `def simulate_flow(rho, p, n_cells, t_steps)` - Input parameters are the highway density in vehicles per cell `rho` ($0 \leq$ rho ≤ 1), the probability of reaction braking `p` ($0 \leq$ p ≤ 1), the total number of highway cells `n_cells`, and the total number of time steps `t_steps` in the simulation. This function uses all of the functions listed above to create a flow simulation. It returns two objects, each is a list of lists (an array). One is the location array, and the other is a speed array. For example,

```
loc_a = [[0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0], [1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0], ... ]
spd_a = [[0, 2, 1, 0, 0, 5, 0, 0, 3, 0, 0], [3, 0, 0, 0, 2, 0, 1, 0, 0, 0, 0], ... ]
```

(i) `def plot_flow(rho, p, n_cells, t_steps)` - This function uses the four input parameters and the function `def simulate_flow(rho, p, n_cells, t_steps)` to generate location and speed arrays. These arrays are used with turtle graphics to plot the speed of the vehicle at its given location using the `dot()` turtle method and a color scheme that differentiates the speed of the vehicle. For example, an open cell would be represented by a white dot. Cells occupied by vehicles should have a dot color that corresponds to the integers speeds 0 - 5. The vertical axis of the graph represents the time step for the corresponding speed list. The first line (at the top of the graph) is the initial speed of the vehicles, so time increases in the downward direction. The horizontal axis corresponds to the cell location, increasing to the right. An example of such a plot is shown in Figure 1i. The darker the point, the slower the traffic is moving. The lightest portions of the graph correspond to either empty cells or traffic moving at the maximum speed.

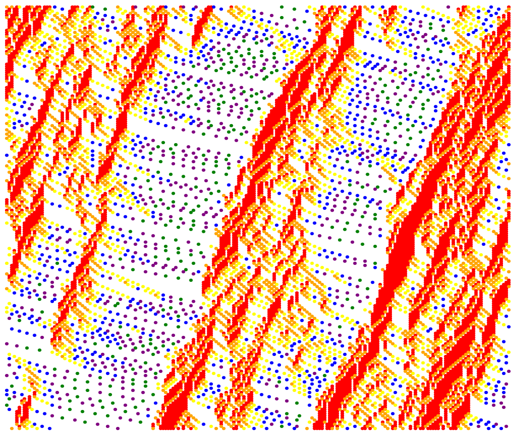

Figure 3: Simulated flow diagram.

(j) `def plot_ave_spd_v_rho(p_l, n_cells, t_steps, d_rho)` - For a given reaction braking probability `p`, this function plots the average speed derived from a flow simulation call versus the density `rho` of the flow for values of `p` provided by the parameter `p_l` (a list of `p` values). Parameter `d_rho` specifies the step size (spacing) between consecutive `rho` values used by the flow simulation function. The average speed is calculated by summing the speed of the vehicles at the conclusion of the simulation and dividing by the number of vehicles.

(k) `def plot_gflow_v_rho(p_l, n_cells, t_steps, d_rho)` - For a given reaction braking probability `p`, this function plots the global flow (`gflow`) derived from a flow simulation call versus the density `rho` of the flow for values of `p` provided by the parameter `p_l` (a list of `p` values). Parameter `d_rho` specifies the step size (spacing) between consecutive `rho` values used by the flow simulation function.The global flow value (`gflow`) is calculated by multiplying the average speed of the vehicles by the track density `rho`.

2. After successfully writing the required function above, generate a location versus time plot of the simulated flow (flow diagram as depicted in Figure 3) for various `rho` and `p` values including (`rho`, `p`) = (0.15, 0.00), (0.20, 0.00), (0.30, 0.30), and (0.40, 0.50).

3. Generate a plot of average velocity versus road density `rho` for `p_l = [0.0, 0.1, 0.3, 0.5, 0.7, 0.9]` using the `def plot_ave_spd_v_rho(p_l, n_cells, t_steps, d_rho)` function. From it, determine the density `rho` at which the average velocity drops to 4 for each value of `p`.

4. Generate a plot of global flow versus road density `rho` for `p_l = [0.0, 0.1, 0.3, 0.5, 0.7, 0.9]` using the `def plot_gflow_v_rho(p_l, n_cells, t_steps, d_rho)` function. From it, determine the road density `rho` at which the global flow is maximum for each value of `p`

4 Assessment

Requirement	Percentage
Completing Tasks 1(a) - 1(g)	60%
Completing Tasks 1(a) - 1(h)	70%
Completing Tasks 1(a) - 1(i)	80%
Completing Tasks 1(a) - 1(k)	90%
Completing Tasks 1 - 4	100%
Completing Tasks 1 - 4 by 05/07	110%