**Traffic Flow: Peachtree City Simulation (Cellular Automata)**

Dawar Ahmed, Robert Hill Pendley

Code: <https://github.gatech.edu/dahmed7/CX4230-Project-2>

**Problem Statement:**

We are going to be investigating the average travel time for vehicles to traverse a portion of Peachtree Street, the corridor from 10th street to 14th street. We will focus on differing various conditions to find the distribution of travel times through the corridor. We will implement random lanes closures (due to car’s breaking down with random probability p) and investigating how this will affect the travel time. We also be adding random probability that the driver is a rash driver which is affect their probability to change lanes, thus creating ripples behind the car and he moves in and out of lanes, affecting travel times.

**Conceptual Model:**

This part of the project will be using cellular automata to model and simulate the project. This means our world consists of a world of open cells in a grid which can either be occupied (meaning a car is present) or unoccupied (meaning a car is not present). Initially the grid is filled with N% of cars where N is the starting density. This density is specified by the specific simulation and can and will be changed throughout our project to see how this will affect how travel time. Our model is based in a simulated world which consists of a 2 x 135 Numpy array which will use as our backing for our cellular grid. The 2 represents the number of lanes on Peachtree Street and 135 represents the physical distance from 10th Street and Peachtree up to 14th. This number is the actual physical distance divided by the average length of car, meaning there are 135 car sized cells from 10th street to 14th street. We have also placed two traffic lights in the model, one 40 cells up and the other 80 cells up. These are just approximately placed as they seen on Peachtree Street. Once we start the simulation, either two or one cars will be generated per unit time based on the specific simulation and the traffic density in that specific simulation. Once the simulation begins the cars propagate forward with a given set of rules which we will discuss later. The cars also have the ability to change lanes and there is a random probability that a car breaks down which has initially been defined as 1/500. Cars may only the max velocity which we have defined as 3. Once we can run the simulation for a period of time we are able to determine data on how long each car stays in the system. In the Final Project we will be using this data to validate and verify our results.

Our model simplifies the changing lanes by only having the car move completely sideways as opposed to the more diagonal motion seen in real life. Also, our model assumes the area outside the visible world is empty. As in beyond the grid there are no cars and no speed restrictions which ultimately affects the simulation especially as you approach the end of the grid since the car may speed up even in high density traffic situation. We have also, for the time being, grossly over simplified the two traffic lights. Currently the traffic lights only exist as either on or off (green or red) and each state is for 10 seconds. So, the simulation using a timer, switches between the green or red every ten seconds. In the real world you would also have amber in the traffic light and there is a set of completely different car actions which happen in that situation. Also, our traffic lights are arbitrarily offset by three seconds just to make sure both weren’t stopping and starting at the same time.

Our model assumes that each car is travelling straight and is only travelling straight. So, each car should have no intention of switching lanes (in order to turn) since its sole object is to move forward as quickly as possible. We also assume that there will be no car accidents, which is certainly not true on real life roads. Once a car breaks down, it doesn’t move off the road or onto the side, it is assumed that the car will remain on the road and be fixed in a matter of time (5 units to be precise).

The major limitation we have faced with our model so far is solving the issue of making the cars move in a natural way. Because of the world being limited to just what can be seen, there is a sense of the cars always following a very predictable behavior. The cars will start slow, increase their speed and eventually start moving at max velocity as they approach the end and rarely change lanes. I believe this is because of the free world at the end. So, cars have the ability to increase their velocity at the end of the road, meaning cars before can also increase velocity and thus there is no need for a car to ever change lanes. Another issue is the appearance of a convoy. Cars will at times move in tight packs with almost no space in between, which I don’t believe is reflective of a real road. Cars in the real life would slowly spread out over time.

**Specific Code Implementations:**

1. Generating Cars. Here the method generates new cars, where *new\_cars* is the number of cars to add each iteration
2. Propagate Forward. Here is the method where the cars move forward, and other elements are updated in each unit time. The code for this method can be found in the propogateForward method in world.py.
   1. First Traffic Light times are updated
   2. The times of blocked cars are updated and are unblocked is the time has elapsed
   3. Next cars may break time (become blocked cars)
   4. Then the cars follow the propagation rules which are as following:
      1. If car velocity is below max and gap until next car allows speeding up, increase speed by 1
      2. If car velocity cannot be maintained because of the gap until next car is too small, look to change lanes
      3. If gap until next car is too small and you cannot change lanes, reduce velocity to match the gap
   5. Remove any cars which have moved off the world
   6. Time is incremented for the data plotting purposes
3. Changing Lanes. Before a car can change lanes, it must follow a specific set of checks. When a car wants to change lanes, it investigates all possibly lane changes (left and right). In our case it would only ever check left when right and right when left, but this method can be further applied to a large road with 3+ lanes. These are rules if checks for:
   1. Check if a car exists in the potential new lane. If so, it cannot change
   2. Find the distance to previous car in the new lane and the next car in the new lane
   3. If there is no car behind and the current velocity can be maintained in the new lane based on the distance to the next car, then swap lanes
   4. If the previous car can maintain its velocity and the car can maintain its velocity in the new lanes, then swap lanes

This method is in the changeLanes function in world.py

All other methods aren’t really part of the model but can be viewed on the GitHub repository.