BML SIMULATION REPORT

Yibing Luo
912491862

1. Approaches to simulate the BML Traffic Model

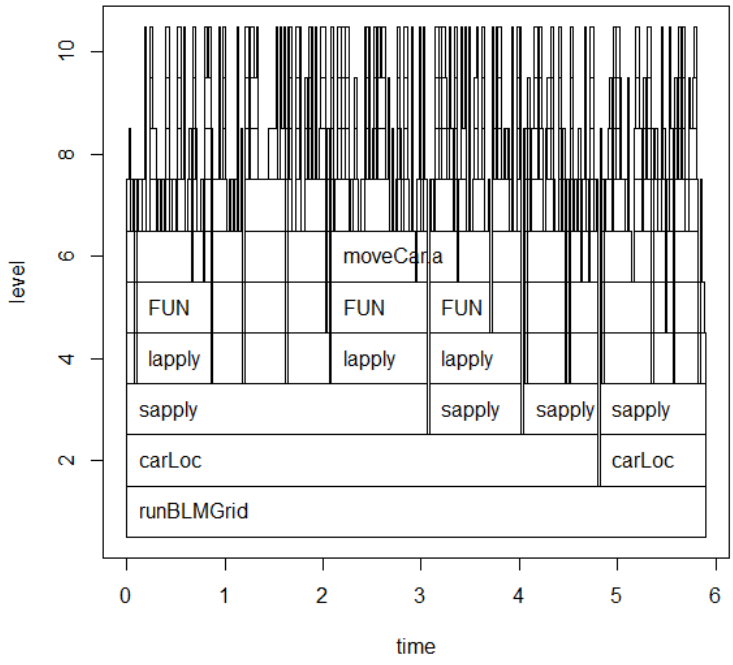
I totally applied three method to simulate the BML traffic model.

(1) The first approach

In this method, the grid is stored by a matrix. Then blue car and red car are denoted by 1 and 2 separately. This approach regards the matrix as an array and process just one element which has 1 or 2 value one time. The process is below:

First, find a position of one car. Then find this car's next position in the grid. According to the next position, we can decide whether this car can move. And we will move this car if it is able to. Finally, use sapply to apply above process to all cars in the grid.

Below is the profiling of this R code:
We create grid with 100 rows and 100 columns and density of cars is 0.5.
Then run the steps 100 times.



Even there are many functions which are called not shown in the plot, it's enough to see that too many functions are called repeatedly. Sapply/lapply was used whole time to loop over in order to find cars locations and move the cars. And in summaryRprof() we noticed that function 'dim' has 41.09 self.pct which implies the first approach spent too much time on dim().

As a result, I want to apply vectorization as an alternative in order to avoid using too much loop, and use another way to store or search locations in order to avoid using too much dim().

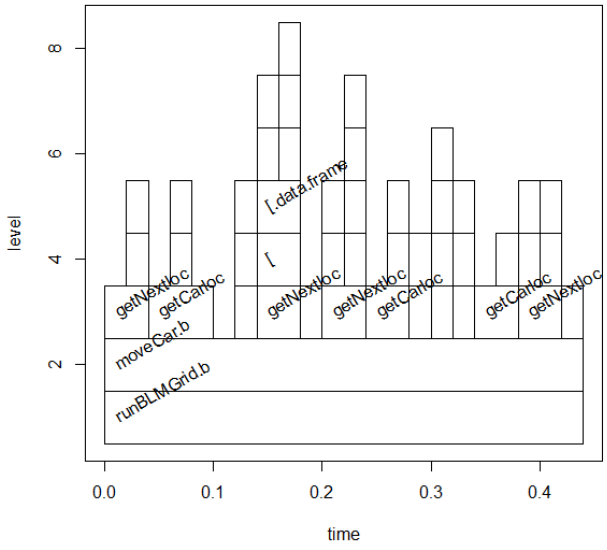
(2) The second approach

In this approach, the way to create a BML grid is the same as the first approach. However, instead of treating grid as an array, we treat it as a matrix. Therefore, we use form (x,y) to represent a car location. So in this approach, function getCarloc() returns a dataset which includes coordinate x, coordinate y, and type of car.

Then use function getNextloc() to get the next location of the cars. When we want to obtain blue cars' location, we can remain coordinate y same, and miner coordinate x by 1. If x-1 equal to 0, then this car gets to the edge of the grid, and wraps around. When we move red cars, we can remain coordinate x same, and plus coordinate y by 1. If y+1 larger than the number of columns, then this car gets to the edge of the grid, and wraps around. This function **manipulates vectors** without loop. So it will be faster than the one in the first approach.

Final, use moveCar.b() to move the car. This function doesn't use loop, too.

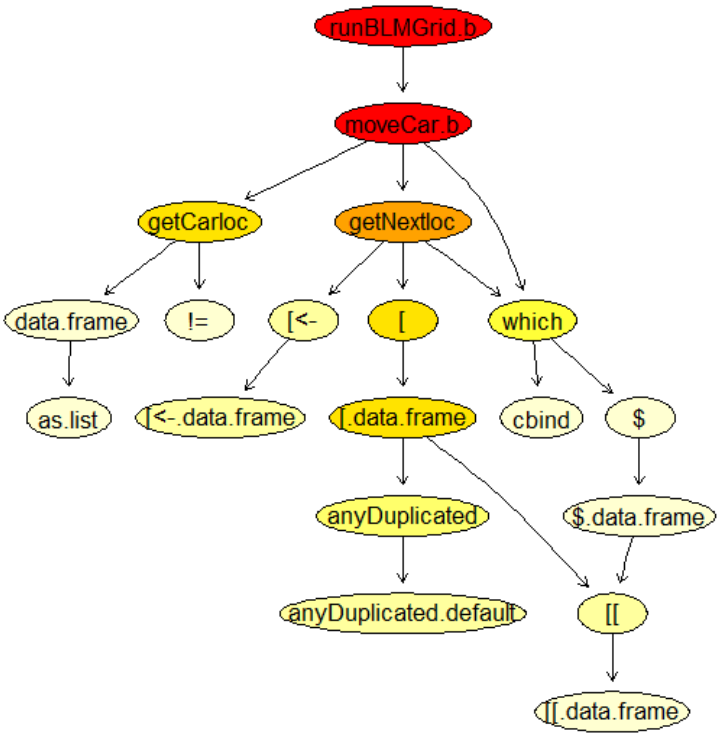
Below is the profiling of this R code:
We create grid with 100 rows and 100 columns and density of cars is 0.5.
Then run the steps 100 times.



Notice that there is no sapply/lappy or for loop using in the program, so it becomes much faster than the first approach. However, we can observe from above plot that functions getNextloc() and getCarloc() are called many times and from the figure below we can see those two functions called many other functions. From summaryRprof we

notice that function '[.data.frame' has self.pct at 17.65 which is pretty high.

As a result, I think these two functions should be more concise in order to increase the efficiency. And the way to record the locations of cars in 'dataframe' may be modified.



(3) The third approach

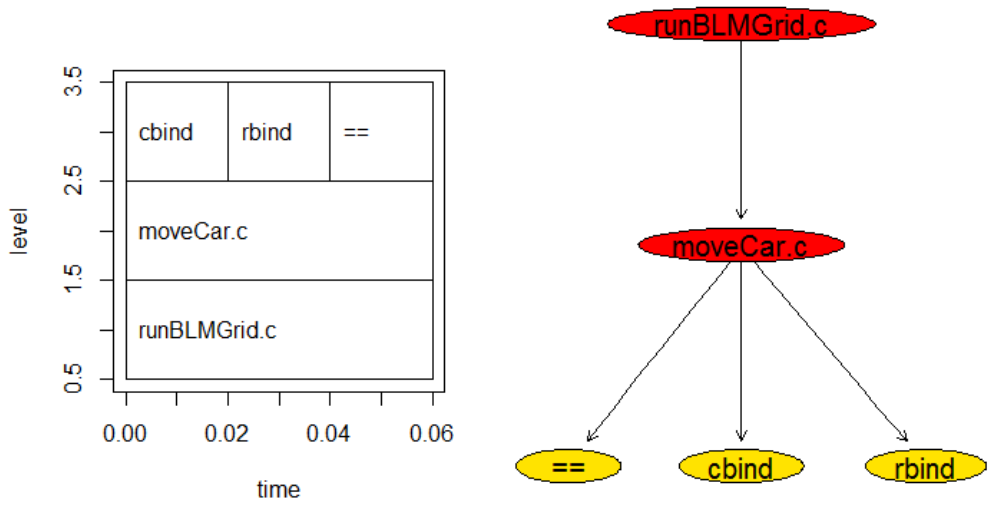
In the second approach, we first search all the cars' positions and their next move, then judge whether the cars are able to move. It's not really wise to do so because it will waste some time to pay attention to cars which cannot move. In the third approach, the cars which can move next step are decided first, then to find how they move.

In the third approach, the way to store the grid remains the same. But we use integer 3 to denote red cars and integer 1 to denote blue cars, and 0 for empty. These denotations are supposed to be useful when we find the cars which can move.

We define a function called moveCar.c(). In this function, we create a new grid (matrix) based on the original one. This new grid can be considered as all elements in the old grid move upwards (or rightwards). Thus, by finding the difference between two grids each corresponding element, we can easily find which cars can move. Such as when it's time to move the blue cars, we move downward the original grid (matrix) to get the new grid and calculate a difference matrix. The elements of that matrix which have value -1 are the locations we should move the cars. For red cars, we need to move the grid leftwards and find locations which have value -3 in difference matrix. And these are the locations we should move the cars.

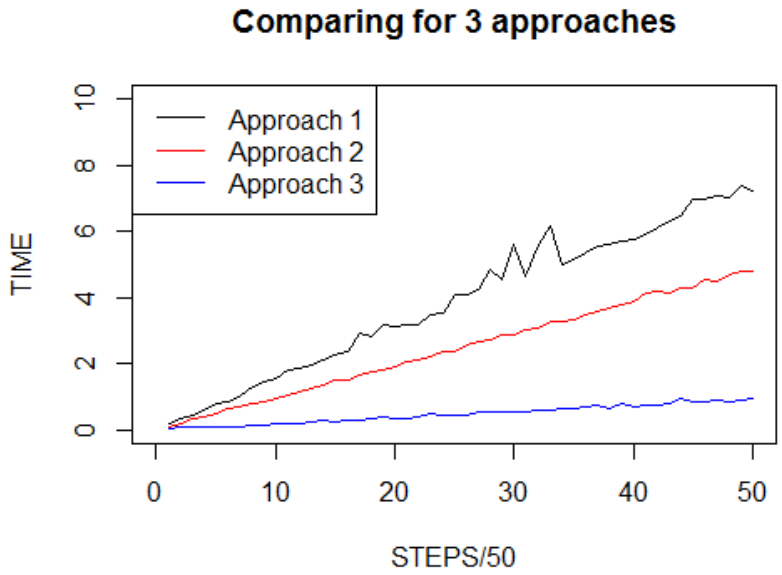
Below is the profiling of this R code:

We create grid with 100 rows and 100 columns and density of cars is 0.5.
Then run the steps 100 times.



The plot above shows a really simple organization and this approach can be really fast.

Blew is the plot draw a rough picture of three approaches' efficiency.



The three approaches are given the same original grid and run for 2500 steps. (Use identical() to compare the results. The return is TRUE which shows the approaches are correct.)

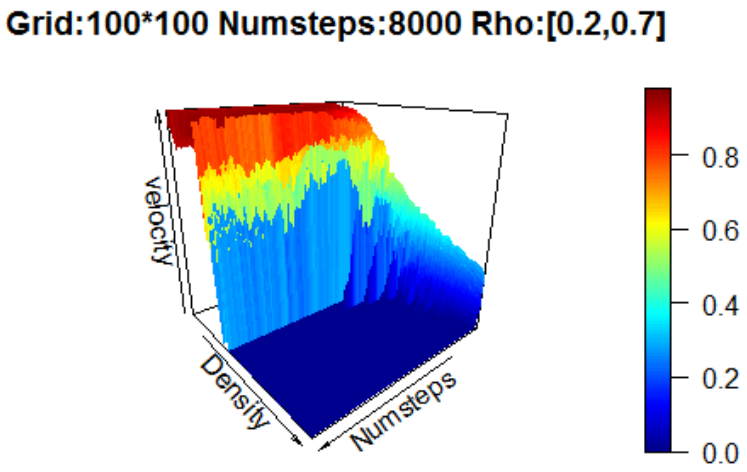
2. Explore the stochastic process

(1) Overview of the stochastic process

Some articles suggest that there should be two primary phases: a free-flowing phase and a jammed phase. In a free-flowing phase, cars with low density will produce a smooth traffic flowing and in that status cars have a relatively high velocity. In a

jammed phase, no car can move and velocity become to 0. Further, there is an intermediate phase, which combines both jammed phase and free- flowing phase.

As the 3D plot shown below, in a 100*100 grid, it shows clearly that the transitions among these three phases are existent. From this plot, we can see the phase changes along with the density of cars. And the number of steps is to make one specific phase stable, namely when the number of steps is large enough, the phase of traffic no longer change with a specific density.

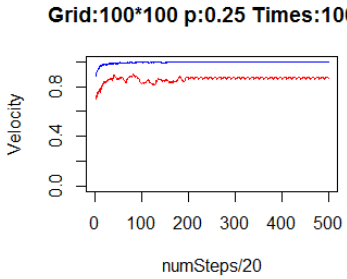


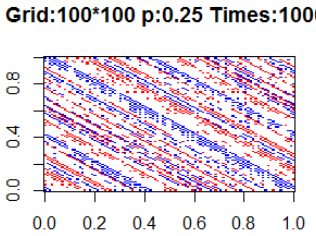
(2) Phase transition with density

The initial positions of cars and lattice dimensions is related closely to the phase and phase transitions. The phases and the density to change phases are quite different among grids which have different size. However, the general process is roughly similar, namely the phase transitions are along with density of the cars. And there is a small interval of density that phase transitions happen.

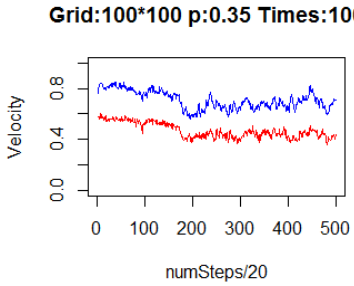
In this part, we will only look at 100*100 grid. We will look into several different density ρ, and plot each grid under the ρ. Also the change of velocity in each grid is drawn. It can somehow reflect how the 'traffic' in grid move into phase. How quick it will be under different ρ.

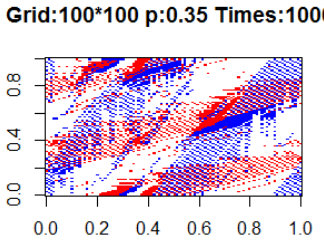
The first set of plots shows a free-flowing phase with density equal to 0.25. The velocity is close to 1 shows almost no jam exists.



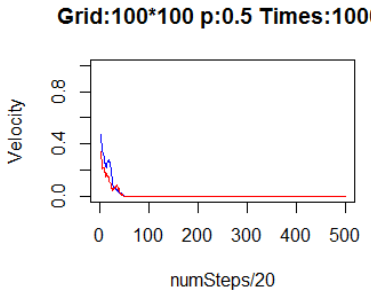


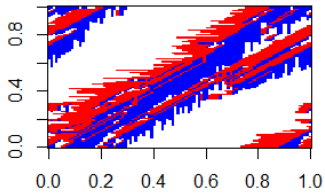
The second set of plots shows an intermediate phase. The velocity shows the cars can keep moving but some jams happened. And the BMLGrid class plot supports that.





The third set of plots shows a jammed phase.





So, when density less than 0.3, the ‘traffic’ will move to a free-flowing phase. When the density is around 0.35, the ‘traffic’ will go to an intermediate phase. And when density is larger than 0.4, the ‘traffic’ will tend to a jammed phase. Further, on condition density>0.4, the larger the density, the faster the ‘traffic’ moves into a jammed phase.

3. More findings…

The grid size will influence the time that ‘traffic’ move to one phase. And according to a website (https://www.jasondavies.com), we find by simulation that coprime dimensions will lead to a periodic manner with high probability and non-coprime dimensions will tend to disordered phase. But which phase the “traffic” will tend to depend more on density and initial positions of cars rather than the size of the grid.

However, I pay a little attention to grid with different length of row and column. And the number of density’s value I selected is a little bit small.