

UNIVERSITY OF CALIFORNIA, DAVIS
STA242
SPRING 2015

Project 2

Junxiao Bu
999452701

April 30, 2015

Purpose and Organization

In this project, The first part is about the characteristics of the BML model. In the second part, I will present some of the code refinements to speed up my code including vectorization and parallel computing.

Part I: Algorithm and Package Introduction

The package to simulate the BML traffic model is called **BMLsimulation**. In this simulation process, the blue cars will move upwards first. Then the red cars will move to the right. If cars move to the borders of the grid, red cars will jump back to the left border of the same row. Blue cars will jump back to the bottom border of the same column. The process can simulated in the next plots. Here, the time step I defined in the package is a little bit different. In my definition, each time step means blue cars and red cars move once in their own orders.

If the next position of one color's car is occupied, then this particular car cannot move. The red car cannot move if its right cell has a car. The blue car cannot move if its upper cell has a car.

In order to verify if the simulation process is correct using **BMLsimulation** package. A 4*4 grid with 4 red cars and 4 blue cars with 4 steps of movements for each color is simulated. The initial grid is shown below.

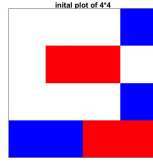


Figure 1: Initial plot of 4*4 grid with 4 red cars and 4 blue cars

All the plots that show the moving steps are shown below.

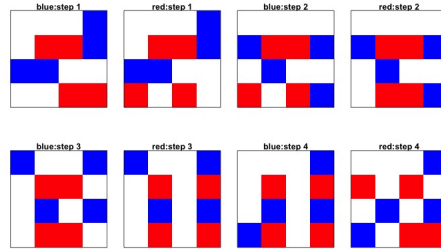


Figure 2: Moving process for a 4*4 grid with 4 steps

From the plots, one can observe the moving process by eyes and verify the result step by step. In this $4*4$ grids, the moving result is correct.

Part II: Behavior of the BML model

In this BML traffic model process, different mode of car moving may lead to different results. In my definition, I choose to let cars move simultaneously rather than sequentially. The behaviour of the model depends on the exact dimensions of the grid as well as the initial density of the cars. I simulated the process of $128 * 128$, $256 * 256$, $512 * 512$ and $1028 * 1028$ grids. In order to find the transition points. I use the density from 0.2-0.7. (I suppose that the numbers of red cars are equal to the number of blue cars.) Here I only present the result from $256 * 256$ grid. To balance the efficiency and accuracy, the grid is run for 5000 times (red cars: 5000 times; blue cars: 5000 times). All the grids are represented below.

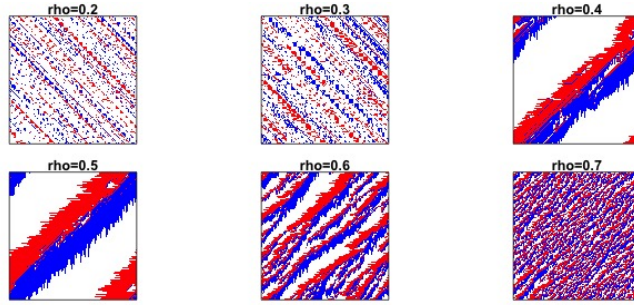


Figure 3: BML grids with different density

If initialized with a low enough density of cars, the system eventually self-organizes into a configuration where all cars can move at each time step (each car has asymptotic velocity equal to unity). From figure 3, in the top-left, we have a free-flowing grid with a density of 0.2 in which the cars arrange themselves into diagonal lines.

If initialized at slightly higher density, the cars are blocked by other cars, until eventually all cars end up participating in one large global jam, where no car can move (asymptotic velocity equal to zero). At a higher density of 0.4, I see a structure with some deadlocked cars in two parallel groups running diagonally. We also see bands of red and blue cars moving. These are intermediate phases that combine jammed and free-flowing phases, in either a periodic or disordered manner. Some cars will move freely, while in the 0.5 picture the cars have arranged themselves into several diagonal bands which avoid each other. At a higher density (larger than 0.4), I see additional groups but the same pattern of moving cars. For higher densities, I see more parallel groups but the same deadlock.

Different Grid Size

Next, the impacts of different grid size on the behavior of the model are going to be explored. From previous plots, the intermediate phases seem to be different. For grids with coprime dimensions, the intermediate states are self-organized bands of jams and free-flow with detailed geometric structure, that repeat periodically in time. In non-coprime rectangles, the intermediate states are typically disordered rather than periodic.[1]

Choosing the grid size to be 144×289 , the plots of 5500, 6000, 6500, 7000, 7500 and 8000 iteration times are shown below. The density is 0.38.

From the plots in Figure 4, the coprime grid size tends to produce periodic intermediate phases near the transition density. The periodic images are quite clear. The red cars and blue cars gather into two groups. The red group moves from topleft to bottomright. The blue group moves from left to right. The states are self-organized bands of jams and free-flow with detailed geometric structure, that repeat periodically in time.

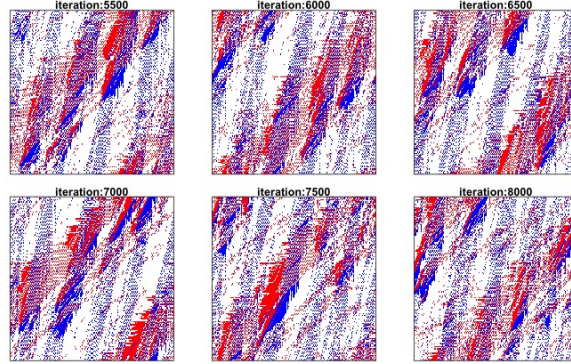


Figure 4: rectangle grid

Different Density

Next, some explorations about different car numbers are presented. In figure 5, the first situation is numbers of red cars are more than the numbers of blue cars. The grid is still 256×256 . I suppose the ratio of red cars vs. blue cars is firstly 7:3 and then 3:7. The overall density is still in the range of 0.3-0.5.

In the density = 0.5 picture there is a single jam spanning the entire grid, while in the density = 0.3 picture the cars have arranged themselves into wide diagonal bands which avoid each other. In the density = 0.4 picture (red: blue=7:3), all cars move some of the time and wait some of the time, and this is achieved by semi-regular geometric patterns of jams feeding into each other. This is the intermediate phase.

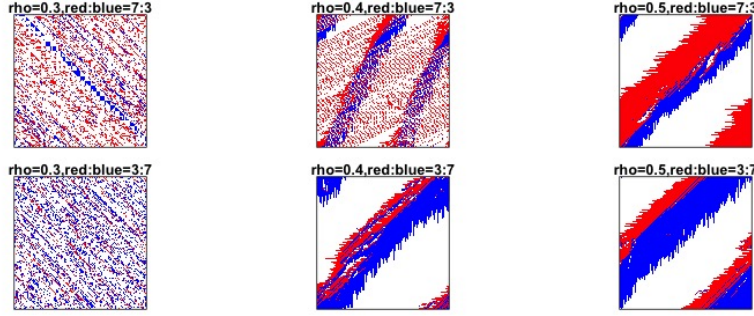


Figure 5: BML grids with different numbers of cars in each color

Comparing with the previous cases, the parallel group patterns are still present in these two cases. Obviously, most of the deadlock cars are still in two parallel groups running diagonally. But the widths of the color bands are different now. The majority colors bands are wider. If the density of the grid increases, there would be more parallel groups as shown in the previous cases (red cars = blue cars) but with different widths.

Next, the average velocities of cars are present when giving different grid size and density range. Here the average velocity is defined as the ratio of number of cars to the total number of cars.

Notice that the numbers of cars that can move were computed when all the cars have moved for given time steps. The density was evenly split into 50 blocks in the range of 0.2 to 0.7. There would be two sets of plots with different iteration times. One is 500 times, another one is 1000 times. The grid sizes are 64×64 , 128×128 , 256×256 . The different grids' average velocities are drawn on one plot.

For small size systems like 64×64 I observe the predicted behavior of a relatively sharp transition from freely flowing to dead-lock phase. when we implement systems with larger than 64×64 , we observe a bifurcation where the traffic free-flow phase and dead-lock phase start to coexist, as we go from low to intermediate values of density. The plots show that there are different points of transition phase, not simply free-flowing or deadlocked.

For different iteration times, there are no big differences between two sets of plots for different grid sizes. The only difference is that there are less points in the intermediate phase when iteration times increase. This is reasonable because cars are more easily to be blocked when their move times increase. If the iteration time is larger enough, i.e. 100000 times. There will be fewer points in the intermediate phase. Most of the points (different densities) will have value of average velocity 0 or 1.

Besides, if drawing three grids' average velocities on one plot, there exists a trend the density at which average velocity decreasing occurs decreases as the grid size increases.

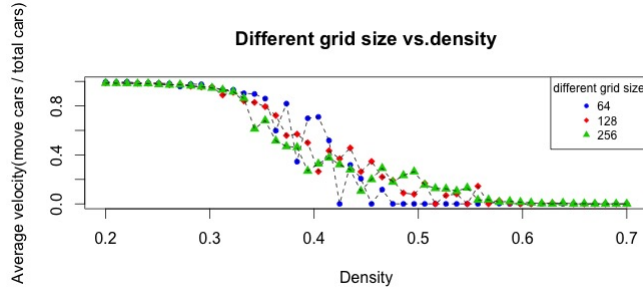


Figure 6: Average velocity vs. density (500 times)

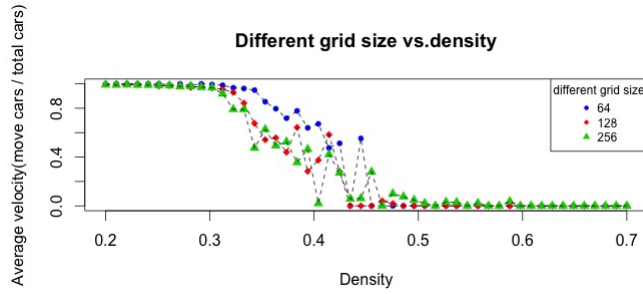


Figure 7: Average velocity vs. density (1000 times)

Part III: Evaluating the performance of the code

In order to speed up my code, my refinements include two parts. The first part is to use vectorization method and matrix subset instead of for loop. The second part is parallel computing for iterating over time steps. As the benchmark, I use a 150*150 grid with 3000 number cars for each color. The iteration will be 100 times.

For the first part, in my function to move the cars, I use **for** loop and **if else** simulate the process. I use profiling to find out where the computations spend most of their time. The user time is roughly 70 seconds. I use profiling to find out where the computations spend most of their time. The result is in the following tables. As I expect, my function movecar1 is taking a large proportion of the overall time, since I haven't vectorized my functions. More specifically, I checked the profiling for movecar1 function.

After some refinement including vectorization and matrix subset in the function, the system time is significantly lower than before. The new time is in the following table. As the table shows, the old user time is over 200 times of new user time, which means my code is over 200 times faster than before.

Table 1: Profiling to show each part's time

| Time Type(seconds) | self.time | self.pct | total.time | total.pct |
|--------------------|-----------|----------|------------|-----------|
| "movecars1" | 63.76 | 85.45 | 74.58 | 99.95 |
| "[[.data.frame" | 2.08 | 2.79 | 6.44 | 8.63 |
| "match" | 1.38 | 1.85 | 2.08 | 2.79 |
| "Anonymous" | 1.14 | 1.53 | 1.36 | 1.82 |
| "\$" | 0.88 | 1.18 | 8.76 | 11.74 |

Table 2: Time for running different code

| Time Type(seconds) | User Time | System Time | Elapsed Time |
|----------------------|-----------|-------------|--------------|
| before vectorization | 69.117 | 12.023 | 81.312 |
| after vectorization | 0.319 | 0.082 | 0.401 |

At last, the faster code's performance will be discussed. First, the performance of the code over different grid sizes will be discussed. Here I use the grid sizes of 8*8, 16*16, 32*32, 64*64, 128*128, 256*256, 512*512 and 1024*1024. And I choose three densities to check the difference. The first density is 0.2(traffic flow). The second density is 0.35(transition phase). The third density is 0.7(dead-locked phase). The iteration will be 100 times. The plot is shown below.

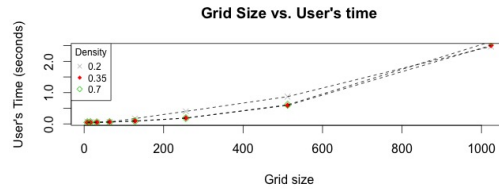


Figure 8: Grid size vs. user time

This shows that as the number of grid size, and the total number of cars, doubles, the time taken approximately doubles.

Second, the performance of the code over different densities will be discussed.

Here I use the grid size 128×128 . The density is in the range of 0.2 to 0.7.(equally split in 10 blocks). The iteration will also be 500 times. The plot is shown below.

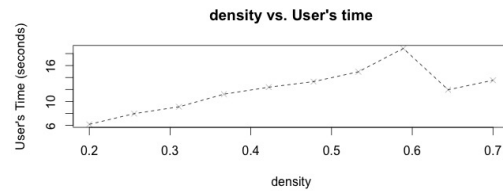


Figure 9: density vs. user time

Generally ,the system time is increasing along with the density increases.

Conclusion

This report explores the BML traffic model. I am able to simulate some of the characteristics of this model. For example: the periodically intermediate phase.

Reference

1. <http://en.wikipedia.org/wiki/Biham>
2. classnotes
3. Piazza: 168,181,183.