Biham-Middleton-Levine (BML) Traffic Model Simulation

Hong Fan 912524085

I. Simulation Process

In this simulation, traffic behavior is investigated by BML model. This model involves two types of cars: “red” and “blue”. Initially, the cars are placed in random on the r * c dimensional grid but without occupying the same cell. In order to simplify the process, we assume the number of red cars and blue cars are equal, but callers can adjust the quantities of each type of cars by themselves using CreateGrid function.

In this model, the blue cars move vertically upward at time periods t = 1, 3, 5, …, while red cars move horizontally rightwards at time periods t = 2, 4, 6, …. When a blue car gets to the top row, it will go to the bottom row of the same column when it moves next time. Similarly, when a red car gets to the right edge of the lattice will move to the first column of the lattice next time. But one cell on the grid cannot be occupied by two cars simultaneously. In this model, cars that advance are treated as having velocity v = 1, while cars are blocked when v = 0, indicating severe traffic congestions.

II. Findings from Simulations

Due to the predefined rules mentioned above, the only randomness is the model’s initial condition. However, there are some interesting patterns found in the simulation process. In this part, we simplify the process by setting the number of red and blue cars equal to each other and the grid to be a square: r = c.

To investigate the effects of density on traffic flow, we adjust density from 0.20 to 0.70 under the condition that grid size is fixed at 200*200 and the number of steps is fixed at 5000.

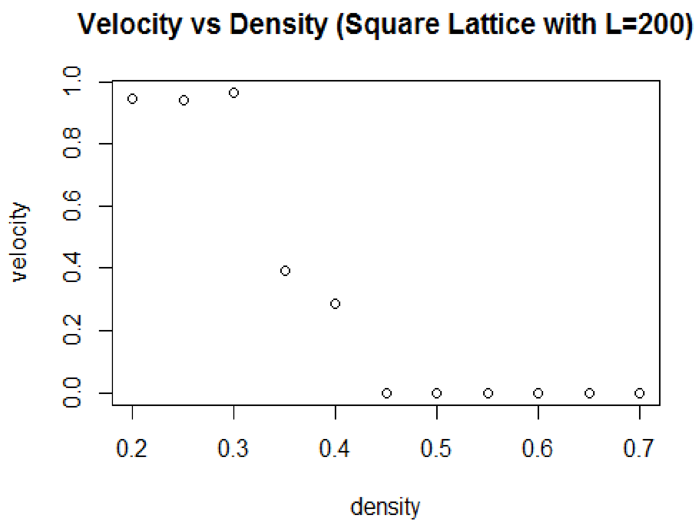

Figure 1 Density against Velocity with Lattice fixed at L = 200

Figure 1 above shows the relations between velocity and density (from 0.20 to 0.70) when grid size and steps are fixed. It is clear to see the average velocity reduces to 0.4 dramatically when density changes from 0.3 to 0.35 and eventually decreases to 0, indicating all the cars are blocked and cannot move.

Judging by the images in Figure 2, it seems that the behavior of the model strongly depends on the density of cars. For low densities (0.2, 0.25, 0.30), the average velocity is approximately equal to 1 (Figure 1) implying the traffic is completely free flowing and the top left graph in Figure 2 is consistent with this result. For densities 0.35 and 0.40, at which average velocities dramatically changes from 1 and decrease to 0 afterwards, top right and bottom left graphs in Figure 2 shows large-scale bands. It seems that cars have arranged themselves into several wide bands which avoid each other and cause traffic congestions. For high densities, the average velocity of cars is 0 which means severe traffic congestions and no car can advance.

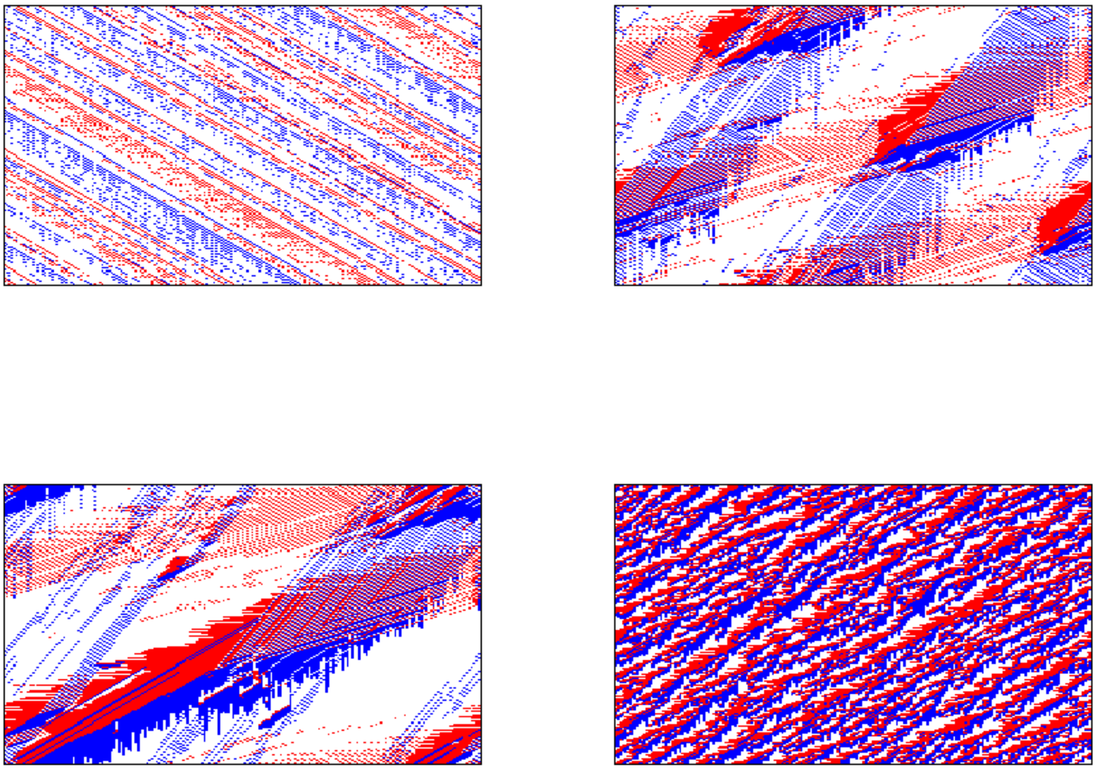

Figure 2 BML Model Simulation. Lattice: L*L = 200*200, Steps = 5000.
Top left: density p = 0.20 Top right: density p = 0.35
Bottom left: density p = 0.40 Bottom right: density p = 0.70

To investigate the effects of grid size on traffic flow, we adjust grid size from 50*50 to 300*300 and fix the density at p = 0.4, steps = 5000. In Figure 3 below, it is clear to see that cars' average velocity varies on different grid sizes and it is not monotonically changes as grid size increases. When the grid size is 200*200, average velocity of cars is approximately equal to 0.2530. When the grid size is 300*300, average velocity of cars is around 0.10. In the other 4 cases, velocities are 0 indicating all the cars are blocked. Interestingly, the 4 corresponding graphs in Figure 4 (grid sizes are 50*50, 100*100, 150*150, 250*250) have two very clear and wide diagonal bands that avoid each other. While it is clear to see there are "intermediate phases" at grid size = 20*20 and 300*300 in which some cars are still free to advance.

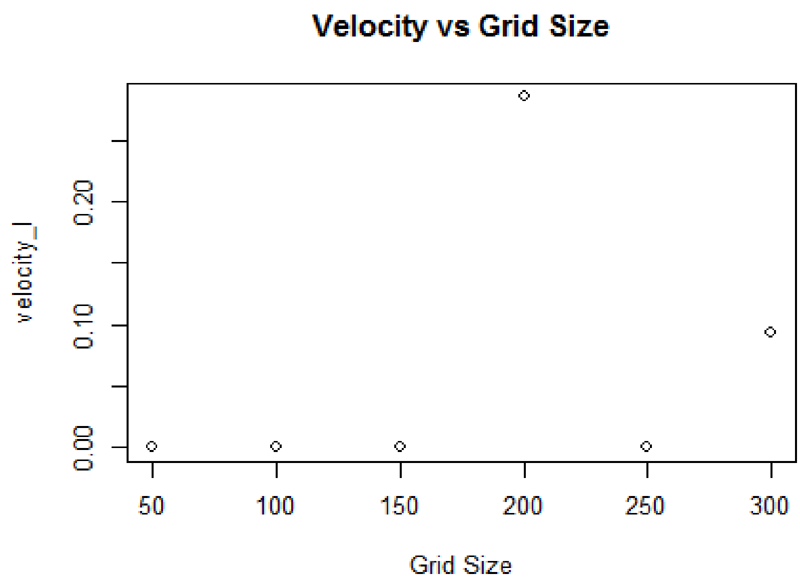

Figure 3 Grid Size against Velocity with Density fixed at p = 0.4

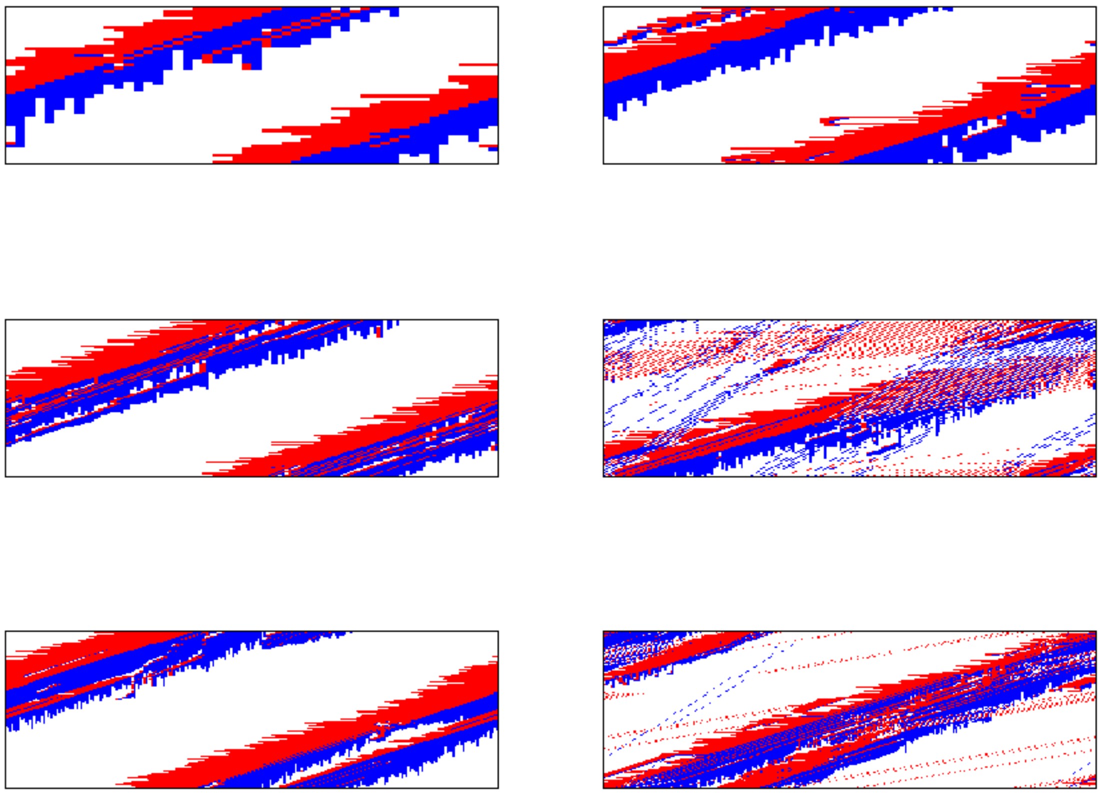

Figure 4 BML Model Simulation with Density p = 0.4 and Steps = 5000
Top left: Grid size = 50 **Top right: Grid size = 100**
Middle left: Grid size = 150 **Middle right: Grid size = 200**
Bottom left: Grid size = 250 **Bottom right: Grid size = 300**

There are three main functions for the simulation process:
CreateGrid
MoveCars
RunBMLgrid

After profiling each of them, some adjustments were made on the RunBMLgrid function and it is shown in the R code.

R code:

```
CreateGrid = function (r, c, ncars = c(red, blue)) #ncars is a vector allowing callers to specify the number of blue cars and red cars.
{
  if (sum(ncars) >= r*c) stop ("overload")

  grid = matrix("", nrow = r, ncol = c)
  set.seed(1)
  pos = sample(1:(r*c), sum(ncars))
  grid [pos] = rep(c("red", "blue"), c(ncars[1], ncars[2]))
  class(grid) = c("BMLgrid", class(grid))
  return(grid)
}


location_car = function (grid)
{
  i = row(grid)[grid != ""]
  j = col(grid)[grid != ""]
  pos = cbind(i,j)
  data.frame(i=i, j=j, colors = grid[pos])
}


##The first part of MoveCars function is quoted from professor Duncan Temple Lang's class notes in STA141.
MoveCars = function(grid, color="red") #Movecars returns a new grid and velocity
{
  cars = location_car(grid)
  index = which(cars$color == color)
  rows = cars [index,1]
  cols = cars [index,2]

  if(color == "red"){
    nextrows = rows
    nextcols = cols + 1L
    nextcols[nextcols > ncol(grid)] = 1L
  } else {
    nextrows = rows + 1L    #position in image() is different from matrix
    nextrows[nextrows > nrow(grid)] = 1L
    nextcols = cols
  }
```

```
  nextlocation = cbind (nextrows, nextcols) #n*2 matrix
  emptycheck = grid[nextlocation] == ""   ##check if next location is empty
  grid[nextlocation[emptycheck, , drop = FALSE]] = color
  grid[cbind(rows, cols)[emptycheck, , drop = FALSE]] = ""

  unblocked_car = sum(emptycheck)
  tot_car = nrow(nextlocation) #same color cars as unblocked
   v = unblocked_car/tot_car #velocity of cars at time t. velocity is defined as ratio of unblocked
cars and total number of cars at time t.

  returnlist = list(grid,v)
}


RunBMLgrid = function (r, c, ncars = c(red, blue), NumSteps)
{
  grid = CreateGrid(r, c, ncars)
   colors = rep(c("blue", "red"), ceiling(NumSteps/2)) #t = 1,3,5...blue cars run; t =2,4,6...red cars
run.
  locations = MoveCars (grid, color = colors[1])[[1]]
  velocity = MoveCars (grid, color = colors[1])[[2]]
  for (i in 2: NumSteps){
    locations = MoveCars (locations, color = colors[i])[[1]]
    velocity = MoveCars (locations, color = colors[i])[[2]]
  }
  list(locations, velocity, NumSteps)
}

###Slower version (old version) of RunBMLgrid (this is replaced by a faster function after
profiling)
RunBMLgrid = function (r, c, ncars = c(red, blue), NumSteps)
{
  grid = CreateGrid(r, c, ncars)
  colors = rep(c("blue", "red"), NumSteps)
  locations = list()
  velocity = list()
  locations[[1]] = MoveCars (grid, color = colors[1])[[1]]
  velocity[[1]] = MoveCars (grid, color = colors[1])[[2]]
  for (i in 2: NumSteps){
    locations[[i]] = MoveCars (locations[[i-1]], color = colors[i])[[1]]
    velocity[[i]] = MoveCars (locations[[i-1]], color = colors[i])[[2]]
  }
  list(locations, velocity)
}


BMLgridPlot = function(grid)
{
  z = matrix(match(grid, c("", "red", "blue")), nrow(loca), ncol(loca))
  image(t(z), col = c("white", "red", "blue"),
       axes = FALSE)
  box()
}
```

```
##summary of RunBMLgrid
summary = function(x) {
  dim_grid = dim(x[[1]])
  Numsteps = x[[3]]
  velocity = x[[2]]
  blue_cars = sum(x[[1]]=="blue")
  red_cars = sum(x[[1]]=="red")
  density = (blue_cars + red_cars)/prod(dim_grid)
  moving_car = ifelse(x[[3]]%% 2 == 0,"red", "blue")
   out = list(Grid_Dimension = dim_grid, Current_Steps = Numsteps, Current_Velocity = velocity,
Tot.Blue_Cars = blue_cars, Tot.Red_Cars = red_cars, Current.Moving_car = moving_car, Density
= density)
  return(out)
}


###Density changes from 0.2 to 0.7 (we force the number of red cars and number of blue cars to
be equal and fix grid to be 200*200)
p = seq(0.2, 0.7, by =0.05)
r = c =200

quant_cars = matrix(rep(p*r*c/2,2), nrow = 2, byrow = TRUE)

change.density = apply(quant_cars, 2, function(ncars) RunBMLgrid(200, 200, ncars, 5000))

rep.times = length(p)
velocity = rep(0, rep.times)
for (i in 1:rep.times) {
  velocity[i] = change.density[[i]][[2]]
}

plot1 = plot(p, velocity, main = "Velocity vs Density (Square Lattice with L=200)", xlab =
"density")


grid = list()  #Getting grids at different densities
for (i in 1:rep.times){
  grid[[i]] = change.density[[i]][[1]]
}

#plot grids at p = 0.35 and p = 0.4 and compare with grid plots at p = 0.2 and 0.7
par(mfrow = c(2,2))
image_p0.20 = BMLgridPlot(grid[[1]])
image_p0.35 = BMLgridPlot(grid[[4]])
image_p0.40 = BMLgridPlot(grid[[5]])
image_p0.70 = BMLgridPlot(grid[[11]])

###Grid size changes: L =50, 100, 150, 200, 250, 300 and fixed density p = 0.40 and steps = 5000
grid_l50 = RunBMLgrid(50, 50, ncars=c(500,500), 5000)
grid_l100 = RunBMLgrid(100, 100, ncars=c(2000,2000), 5000)
grid_l150 = RunBMLgrid(150, 150, ncars=c(4500, 4500), 5000)
grid_l250 = RunBMLgrid(250, 250, ncars=c(12500, 12500), 5000)
grid_l300 = RunBMLgrid(300, 300, ncars=c(18000,18000), 5000)
```

```
velocity_l = c(grid_l50[[2]], grid_l100[[2]], grid_l150[[2]], velocity[5], grid_l250[[2]], grid_l300[[2]])
L = seq(50, 300, by = 50)
plot(L, velocity_l, main = "Velocity vs Grid Size", xlab = "Grid Size")

par(mfrow = c(3,2))
image_l50 = BMLgridPlot(grid_l50[[1]])
image_l100 = BMLgridPlot(grid_l100[[1]])
image_l150 = BMLgridPlot(grid_l150[[1]])
image_l200 = BMLgridPlot(grid[[5]])
image_l250 = BMLgridPlot(grid_l250[[1]])
image_l300 = BMLgridPlot(grid_l300[[1]])


###Profiling (choose r = c = 1000, ncars = c(800, 800), NumSteps = 5000)
#CreateGrid function
Rprof("CreateGrid.out")
y = CreateGrid(1000, 1000, ncars = c(800,800)) # Call the function to be profiled
Rprof(NULL)
summaryRprof("CreateGrid.out")

#location_car function
Rprof("location_car.out")
y = location_car(CreateGrid(1000, 1000, ncars = c(800,800)))
Rprof(NULL)
summaryRprof("location_car.out")

#MoveCars function
Rprof("MoveCars.out")
y = MoveCars(CreateGrid(1000, 1000, ncars = c(800,800)), color = "red")
Rprof(NULL)
summaryRprof("MoveCars.out")

#RunBMLgrid function
Rprof("RunBMLgrid.out")
y = RunBMLgrid(1000, 1000, ncars = c(800,800), 5000)
Rprof(NULL)
summaryRprof("RunBMLgrid.out")
```