STA 242

Assignment 2

Yin Zhang

912485250

1 Develop the speed of the code:

The first version I build is: first find the position of cars, then use for loop for every position to find the color of the car and have the corresponding behavior.

The result shows below:

```
$by.self
                      self.time self.pct total.time total.pct
"move"                    80.26    62.43     128.12     99.66
"dim"                     17.92    13.94      17.98     13.99
"ncol"                     3.64     2.83      21.16     16.46
"!="                       2.92     2.27       2.92      2.27
"[.data.frame"             2.90     2.26       4.12      3.20
"as.character"             2.64     2.05       2.64      2.05
"Ops.factor"               1.72     1.34       2.70      2.10
"as.vector"                1.14     0.89       1.14      0.89
```

The last version of the function:

```
$by.self
          self.time self.pct total.time total.pct
"ifelse"       4.24    36.81       4.32     37.50
"=="           2.68    23.26       2.68     23.26
"move"         2.52    21.88      11.52    100.00
"matrix"       0.40     3.47       0.40      3.47
"which"        0.32     2.78       3.66     31.77
```

1 From the results, “move” function costs lots of time because there are for loop and several “if” sentences in it that cost lots of times. Moreover, the “dim” also costs lots of time which means that I use lots of “which” function to find the position and the dimension of the grid. So I reorganization of code and use vectorization to developed the speed by avoiding the “for” loop. I use a matrix (number of cars * 3) to describe the statue of grid. The first column is the row of the cars and the second column is the column of the cars in the grid. The third column is the corresponding car color. The results of the last version shows that the move function has less run time.

2 The first version, the “move” function is big, so I can divided it into several small functions, which will develop the speed.

3 Because there are two times of if condition: one is to find the color of the car in

the “move” function, the other is to “find the time to move which car” in the runBMLGrid function. In order to reduce the “if” condition, I give a “time” parameter in the move function, in that way, if the time is odd, the blue car will move, else the red car will move. Though it does not reduce many time, it makes the function brief.

4 I found that the quicker the run speed (elapse time), more limitation the function have. So I should find the tradeoff of the two terms. In one of version, by combining the blue and red car to move together to be “movecombine” function, I reduced the half of times of for loop. In that way, it can develop the speed. But it can only run even number of times.

I repeat 5 times in order for avoid the initial position randomness influence. The user, system and elapse time for the first version is : 140.671, 5.571, 152.120 for the first version and 13.015 , 0.528, 13.641 for the last version. The output of summaryRprof of second version also shows that because there are all basic functions that cost most of time and the percentage for each running time are similar. I also change the == and if else, but there is no determinate time development but it makes the code not brief. So I think second version is good enough.

2 Analysis of the BML system

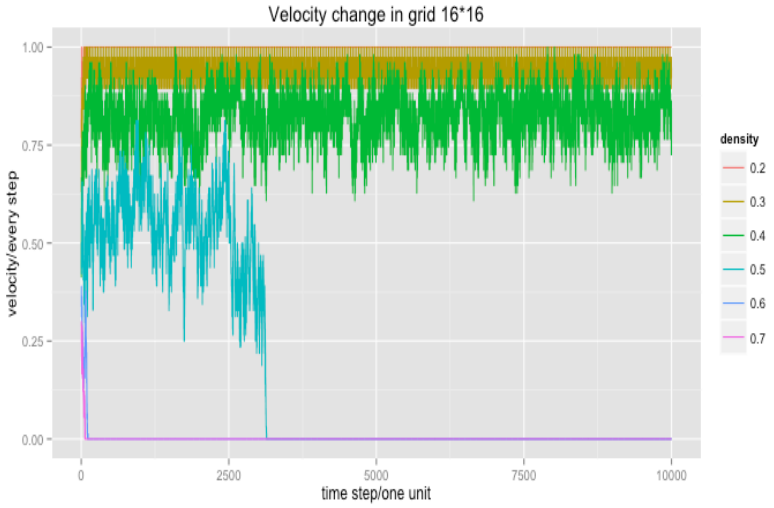
I analysis some variables which may have influence on the phase transition.

2.1 The grid size and density of the cars

Assumption: the grid is n*n, the number of red cars is equal to the number of blue cars.

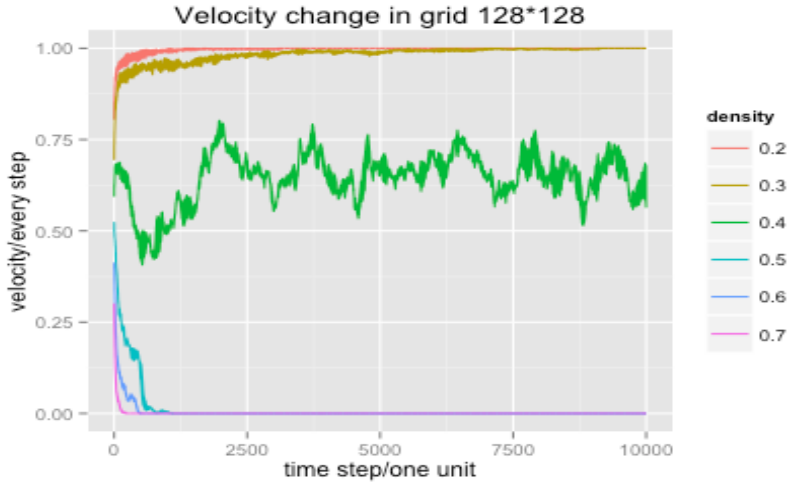
I did not repeat the function several times because I assume that the initial status of the grid does not influence the last status of the BML system (I will analysis it in the 2.2 part). I calculated the grid is: 16*16, 32*32, 64*64, 128*128, 256*256, 512*512 and chose the most representative figure shows below:

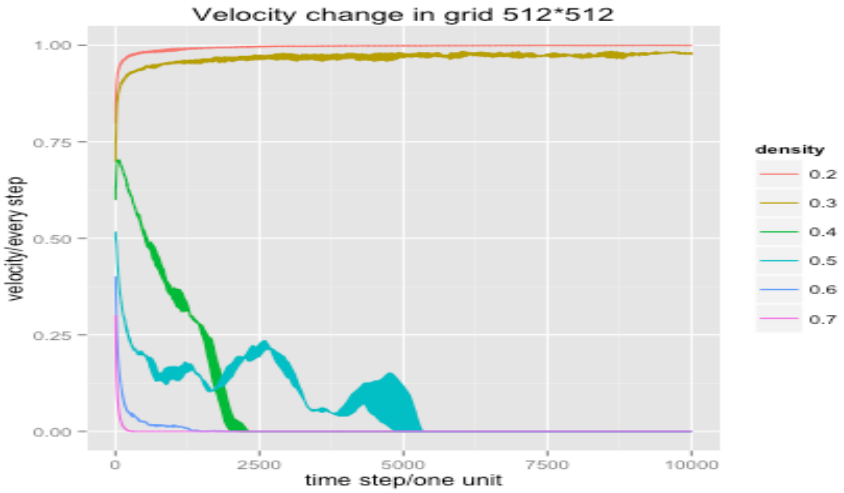
For grid 16*16



For the grid 16*16, from the plot, we can easily see there is a periodic when density is small. For density = 0.2, the velocity 0.96, 0.92, 1, 1 recur, which means it is stable at this status. For density = 0.4, the plot also shows a periodic function. For the different density, because there is a periodic but not a stable phase (hard to see a certain pattern), I did not contain them in the report.

For grid 128*128 and 512*512

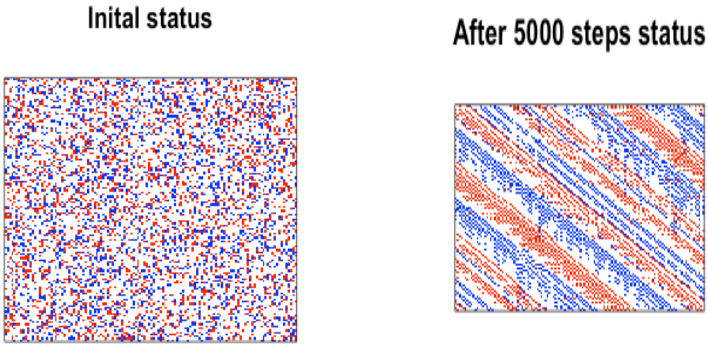




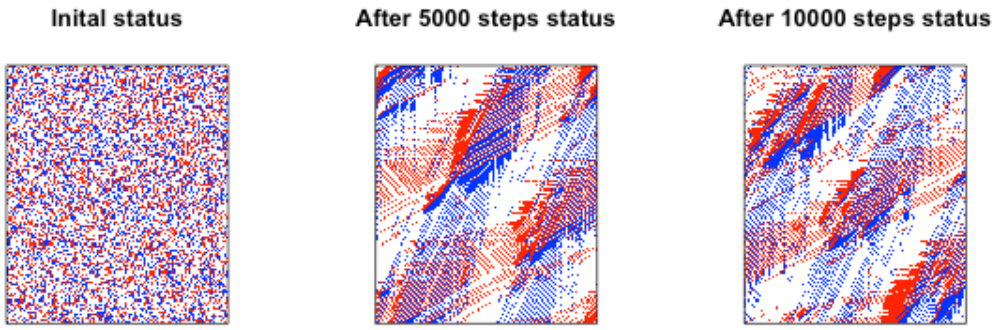
For the grid 128*128, for density = 0.4, there is also a periodic function. But comparing to the 16*16, the periodic is larger than 16*16. For the grid whose density lager than 0.4, the velocity did not have determined change; For the grid whose density smaller than 0.4, we also can see that there is no longer a periodic for the density 0.2 and 0.3, the velocity both converge to 1 which means they have a free-flowing phase. For the grid 512*512 on the bottom, the density=0.4, the velocity converge to 0 faster than 128*128. However, we can see a strange things, which is when density= 0.3, the velocity is not converge to 0 but fluctuate around 0.95 which means that it not necessary converge to velocity = 1 but just have a stable periodic. However, on the other hand, it is doubt that whether the timestep=10000 is long enough to see the converge status.

Based on the analysis above, because the size of grid not a sensitive parameter, I choose 128*128 grid (which is representative according to the velocity plot) as an example to see the exact phase transition for the BML grid:

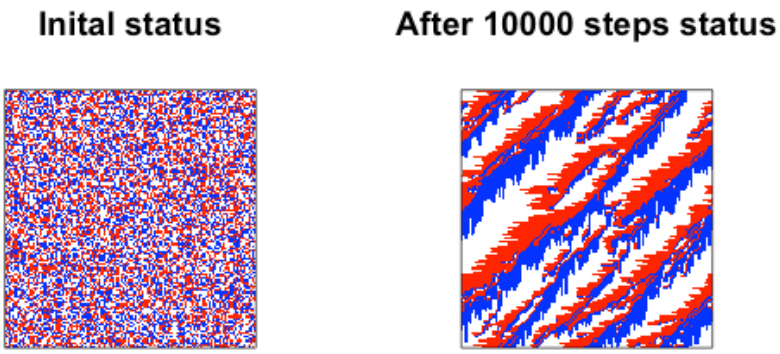
For density=0.3



For the density = 0.4,



For density = 0.6

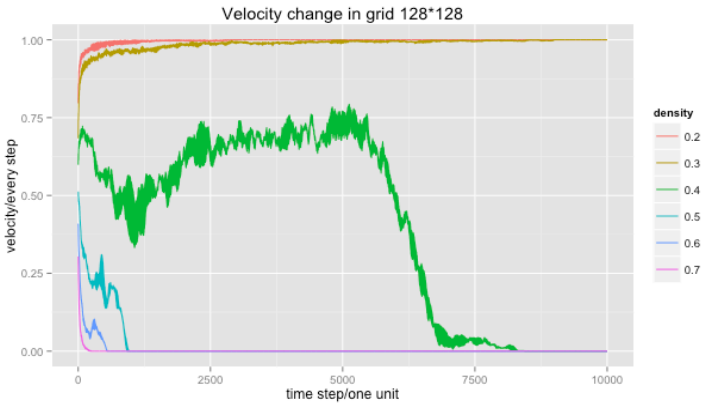


The three group of plot shows the tree phase transition status, which is all the car are move, only part of cars move and no car move. Additionally, The

second group of plot, the second and the third one have the same pattern, which means even the velocity is different, if they are stabled in a the period function, they will have the same pattern.

2.2 Initial position for

I also plot another grid which is 128*128



For the grid 128*128 above, we can see that at the density=0.4, after a long time, it converges to 0 which means there is a jammed phase. The other density, they show the similar pattern with the first 128*128 grid. We can see phase transition from the plot above, they show the initial status have an influence on the cut off point density, but no big influence for other density which is far away from the cut off point density. So we should pay attention to the initial status and calculate the average of repetition of the functions for the cut off point. But on the other hand, we should figure it out that the first situation of 128*128 when the density=0.4, whether the periodic time is stable or after a long time it will also converge to a jammed phase. For the other density will all converge to certain pattern, so there is no need to repeat the code in order to avoid the random effect of initial status for them. But when the grid gets bigger and bigger, it will take a lot of times to run the code. So I just run ones to explain.

2.3 Other:

For the different row and column of grid and the different proportion of red and blue cars. Those variables also will influence the behavior of the BML system. But they are not independent, so only analysis one of them is enough. The other variable will have the similar analysis. Also, red car and blue car are a whole

thing for the BML system, it is not important to analysis the red and blue car separately.

2.3 Conclusion:

1 The dimension of the grids and the density of cars will influence the status and phase transition. But apparently, the dimension of the grid is not as sensitive as the density for the phase transition.

2 There is a cut off point for the density, which means when the density is below 0.4, the phase is converge to be free-flowing phase; when the density is bigger than the cut off point, the phase converge to jammed phase. Some point around 0.4 is usually the cut off point. When the grid dimension is big, the cut off point could be smaller. The smaller the grid is, the harder it to be jammed. And when the density nearer the cut off point, it is harder to converge to a stable phase.

3 With the dimension of the grid become bigger and bigger, the periodic become longer and longer, and at the same time, the periodic can converge to infinite.

R code:

```
#The last version of functions:
createBMLGrid = function (r, c , ncars, prop = 0.5 ) # dim the grid dimention and carprop is the car proportion , the prop is the red:blue, if it is c(a,b), a is number of red car and b is num of blue car
{
  if ( length(ncars) == 1 && ncars < 1 ) #if input is proportion of the cars
    ncars = round(rep(r*c*ncars*prop, 2)) # one for the blue cars number and the other for the red car number
  grid = matrix("", r, c)
  pos = sample(1:(r*c),sum(ncars))
  grid[pos[1:ncars[1]]]="red"
  grid[pos[-(1:ncars[1])]]="blue"    #becuase the position is random, so the red and blue are random and have same prob
```

```
    class(grid) = c("BML",class(grid))
    grid
}

move = function (grid, time) #v=velocity
{
    #move cars
    if(time%%2) #in order to decrease the determine times, put the if out of for loop
    { colormove = "blue"
        curposition=which(grid == "blue", arr.ind = TRUE)
        nexposition=cbind(r = ifelse (curposition[,"row"] == 1L, nrow(grid), curposition[,"row"] - 1L), c = curposition[,"col"]) # blue car move upwards
    }else
    { colormove = "red"
        curposition=which(grid == "red", arr.ind = TRUE)
        nexposition=cbind(r = curposition[,"row"], c = ifelse (curposition[,"col"] == ncol(grid), 1L, curposition[,"col"]+ 1L))
        } # red car move rightwards

    #check the position
    find=(grid[nexposition] == "")
    movenum=sum(find) # the number of cars moves
    grid[nexposition[find, , drop = FALSE]] = colormove
    grid[curposition[find, , drop = FALSE]] = ""

    list(grid=grid, v=movenum/nrow(curposition), carmove=movenum)
}

plot.BML = function (grid, main="BML plot", ...)
{
    g=t(grid)[,nrow(grid):1] #because of the image funtion, in order to make the matric and the image same
```

```
    if(length(which(grid==""))==0)
    {
       matchcol = matrix(match(g, c("blue", "red")), nrow(g))
       image(matchcol, col = c("blue", "red"),axes = FALSE, main = main, xlab = "",
ylab = "" , ... )
    }else
    {matchcol = matrix(match(g, c("", "blue", "red")), nrow(g))
     image(matchcol, col = c("white", "blue", "red"),axes = FALSE, main = main,
xlab = "", ylab = "" , ... )}
    box()
}

summary.BML = function (grid, numSteps, ...)
{
    initial=grid
    num_red=sum(grid == "red")
    num_blue=sum(grid == "blue")
    prop=(num_red + num_blue) / length(grid)
    result=runBMLGrid(grid, numSteps)
    final=result$grid
    ave_velocity=mean(result$vbystep)
    car_move=mean(result$movestep)
    if (numSteps%%2)
    {block=num_blue-car_move
    }else
    {block=num_red-car_move}

F_result=list(initial=initial,final=final,prop=prop,num_red=num_red,num_blue=n
um_blue,ave_velocity=ave_velocity,car_move=car_move,block=block)
    class(F_result) = "summary.BML"
    F_result
}
```

```
print.summary.BML = function (x, ...)
{
   cat("The initial and final status see the plot:\n")
   par(mfrow=c(1,2))
   plot(x$initial, main = "Initial Status")
   plot(x$final, main = "Final Status")
   cat("\nThe number of red and blue cars:\n ")
   print(c(numred=x$num_red, numblue=x$num_blue))
   cat("\nThe density of cars:\n ")
   print(x$prop)
   cat("\nThe final status:\n")
   print(x$final)
   cat("\nThe average parameter:\n")
   print(c(ave_velocity=x$ave_velocity, car_move=x$car_move, block=x$block))
}
#The summary conclude two part : the initial status and final status. In the initial status, it conclude the
#gird, the number of red cars and blue cars, the density of car. In the final status, it conclude the final grid,
#the average velocity and the average number of moves cars and average number of block cars.



runBMLGrid = function ( g, numSteps)
{ vstep=numeric(numSteps) #velocity in every step
   movestep=integer(numSteps) #number of cars move
   for (i in 1:numSteps)
   {result=move(g,i)
    g=result$grid
#    vstep[i]=result$v #the blue car velocity
#     movestep[i]=result$carmove
```

```
#    plot(g)
   }
     g
# list(grid=g,vbystep=vstep, movestep=movestep)
}

#5
g=createBMLGrid(2,4,c(1,2))
par(mfrow=c(2,2))
plot(g, main='Initial grid')
final=runBMLGrid(g,4)
plot(final$grid, main='Fourth step')
summary(g,5000)

#6
library(ggplot2)
library(reshape)

gi=lapply(seq(0.2,0.7,0.1), function(x) {set.seed(10)
                                          createBMLGrid(256,256,x)})
g256=data.frame(1:10000,sapply(gi,function(x)
runBMLGrid(x,10000)$vbystep))
colnames(g256) <- c("timestep",seq(0.2, 0.7, 0.1))
g256= melt(g256, id ='timestep', variable_name="density")
ggplot(g256, aes(x=timestep, value, colour=density)) + geom_line()+xlab("time step/one unit")+ylab("velocity/every step")+ggtitle("Velocity change in grid 128*128")

###############
set.seed(10)
g=createBMLGrid(128,128,0.6)
plot(g,main="Inital status")
g2=runBMLGrid(g,10000)
```

```
plot(g2,main="After 10000 steps status")



#7
#should run several times
g=createBMLGrid(100,100,0.3)
Rprof("runBMLGrid.out")
runBMLGrid(g, 10000) #what should I put in this
Rprof(NULL) #turn it off
summaryRprof("runBMLGrid.out")
system.time(runBMLGrid(g, 10000))

plot(grid)

#9
############test
library(BMLGrid)

addition_test = function ()
{
   cat("Running additional test...\n")
   set.seed(2)
   g=createBMLGrid(2,4,c(1,1))
   resultb=runBMLGrid(g,1)
   nextp=which(resultb$grid == "blue", arr.ind = TRUE)
   if (all(nextp!=c(2,3)))
      stop ("Error in addition!")
   set.seed(2)
   resultr=runBMLGrid(g,2)
   nextp2=which(resultr$grid == "red", arr.ind = TRUE)
   if (all(nextp2!=c(2,2)))
      stop ("Error in addition!")
}
```

```
##############
#The first version of functions:
createBMLGrid = function (r, c , ncars, prop = 0.5 ) # dim the grid dimention and carprop is the car proportion , the prop is the red:blue
{
   if ( length(ncars) == 1 && ncars < 1 ) #if input is proportion of the cars
      ncars = round(rep(r*c*ncars*prop, 2)) # one for the blue cars number and the other for the red car number
   grid = matrix("", r, c)
   pos = sample(1:(r*c),sum(ncars))
   grid[pos[1:ncars[1]]]="red"
   grid[pos[-(1:ncars[1])]]="blue"    #becuase the position is random, so the red and blue are random and have same prob
   class(grid) = c("BML",class(grid))
   grid
}



move = function (grid, colormove = "blue")
{
   # find the position of cars
   position=which(grid!="", arr.ind = TRUE)
   car=data.frame(position,color=grid[position])

   curposition=as.matrix(subset(car[,-3], car$color== colormove ))
   if (colormove=="blue")
   { nextrow=c()
      for (i in 1 : nrow(curposition))
      {nextrow[i]=curposition[i,1]-1L
        if (nextrow[i] < 1)
            nextrow[i]=nrow(grid)
      }
```

```
    nextposition=cbind(nextrow,curposition[,2])
  }

  if (colormove=="red")
    { nextcol=c()
    for (i in 1 : nrow(curposition))
      {nextcol[i]=curposition[i,2]+1L
       if (nextcol[i] > ncol(grid))
          nextcol[i]=1L
    }
      nextposition=cbind(r=curposition[,1],c=nextcol)
  }

  #see whether can move--whether the next is empty
  find=(grid[nextposition] == "")
  movenum=sum(find)
  grid[nextposition[find, , drop = FALSE]] = colormove
  grid[curposition[find, , drop = FALSE]] = ""

  list(grid=grid,v=movenum/nrow(curposition))
}

plot.BML = function (grid,...)
{
  g=t(grid)[,nrow(grid):1] #because of the image funtion, in order to make the
matric and the image same
  matchcol = matrix(match(g, c("", "blue", "red")), nrow(g))
  image(matchcol, col = c("white", "blue", "red"),axes = FALSE, xlab="", ylab="",
main=""...    )
  box()
}

movecombine = function ( g )
```

```
{
   first=move(g,'blue')
   g=first[[1]]
   list(move(g,'red'),first[[2]]) #give the grid that we need
   #because the output g and input g should be the same, so I can not omit this
}

runBMLGrid = function ( grid, numSteps)
{ vstep=c()
   for (i in 1:(numSteps/2))
   {result=movecombine(g)
     g=result[[1]][[1]]
     vstep[2*i-1]=result[[2]] #the blue car velocity
     vstep[2*i]=result[[1]][[2]] #the red car velocity
   }
   list(grid=g,vbystep=vstep)
}

g=createBMLGrid(100,100,0.3)
Rprof("runBMLGrid.out")
runBMLGrid(g, 10000)
Rprof(NULL) #turn it off
summaryRprof("runBMLGrid.out")
system.time(runBMLGrid(g, 10000))
```