STA 242 Assignment 2

Biham-Middleton-Levine Traffic Model

Xueting Shao

SID:912457262

1. Behavior of the BML model

1.1. Phase Transition

Cars in BML model are moved for 1000 steps with car density 0.3 and 0.7. There appears two different phases.

In the case with car density 0.3, the model seems to have a smooth flow of traffic. The case with 0.7 density seems to jam and little cars can move.

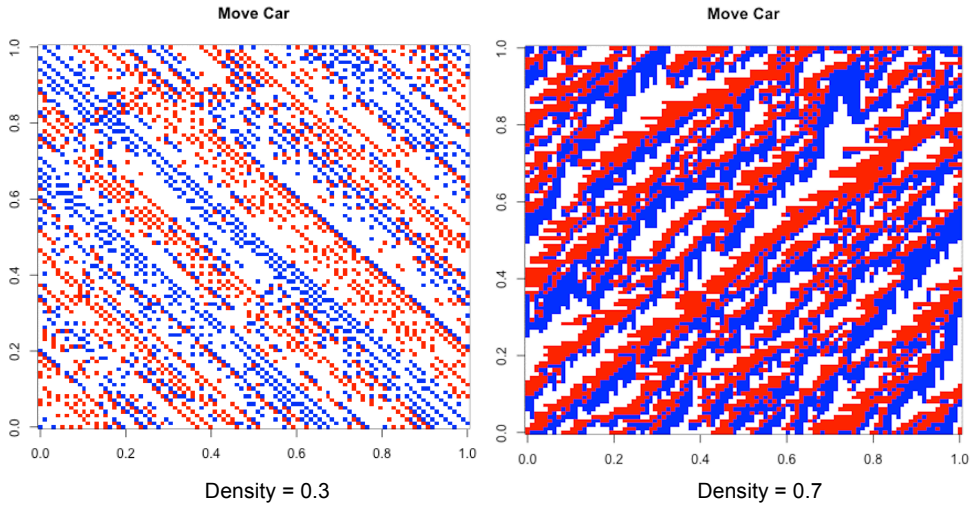

Figure 1 Two Phases

1.2. Effect of Car Density

It is clear that the model will achieve to a certain phase. Questions come: Is there only two phases or more than two? And how is it decided? Car density is one potential reason. So, simulations of BML model of car density from 0.2 to 0.7 are finished to check the overall velocity of cars.

In Figure 2, it appears to be two different phases.

(1) In the low car density plots (0.2, and 0.3), velocity increases dramatically at first. After moving certain steps, velocity fluctuates around a non-zero value.

(2) In the higher car density plots (0.4, 0.5, 0.6, and 0.7), the velocity decrease sharply and keeps zero after moving some steps.

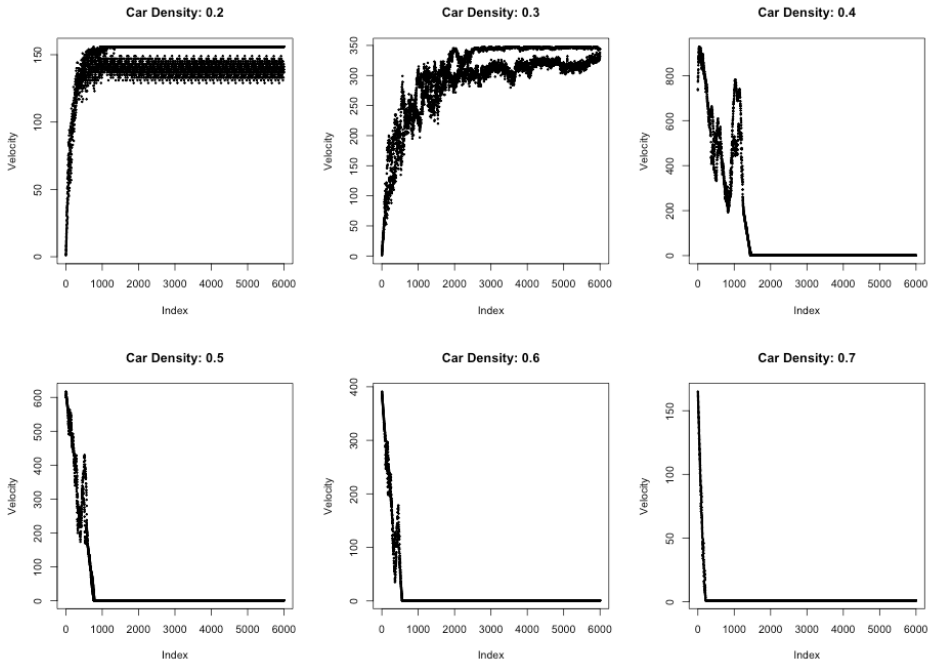

Figure 2 Velocities in Different Car Density

1.3. Effect of Grid Shape

Since red cars always move rightward and blue cars always move forward, the shape of grid may also influence the velocity of red cars and blue cars.

1.3.1. Square Shape

Each red car is moving in one row and each blue car is moving in one column. In one row, if there are more cells no matter whether they are empty, red cars have more possible cells to move in. It is normal to make a prediction that red cars velocity will increase when the number of columns increases. So, in square shape, the velocity for both red and blue cars should be the same.

To eliminate the effect of car density, simulation is done with low car density (0.3) and high car density (0.7).

According to Figure 3 and Figure 4, both red cars and blue cars tend to have similar velocity in both low car density and high car density.

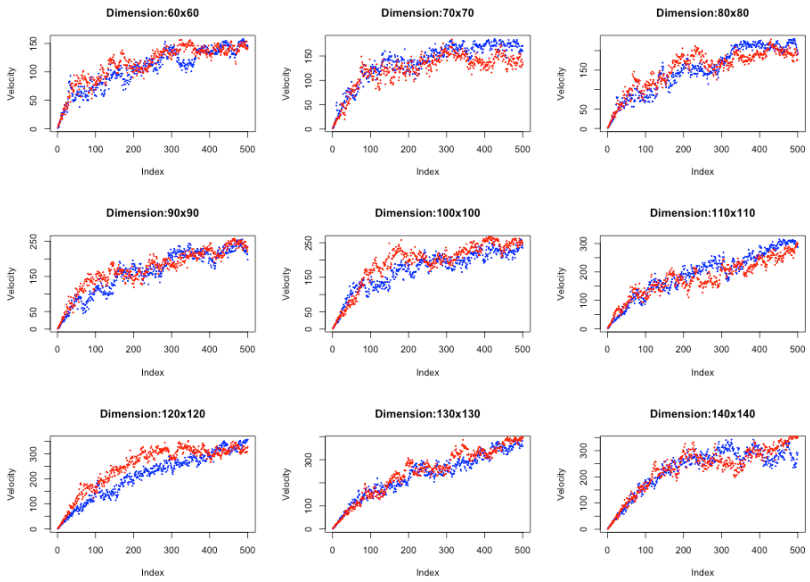

Figure 3 Velocities of Red Cars and Blue Cars with Low Density

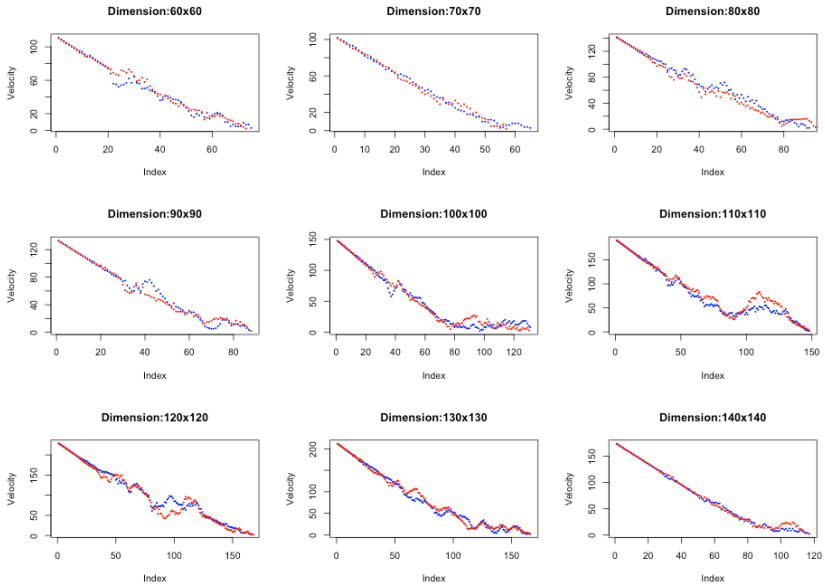

Figure 4 Velocities of Red Cars and Blue Cars with High Density

1.3.2. Rectangle

Only the case with larger number of columns is discussed here. The case with smaller number of columns will have similar conclusion.

For low car density case, there is no significant evidence that red cars have higher velocity than blue cars or in inverse. It is surprisingly different with former assumption. But compare to square case, the difference between red cars and blue cars seem to be larger.

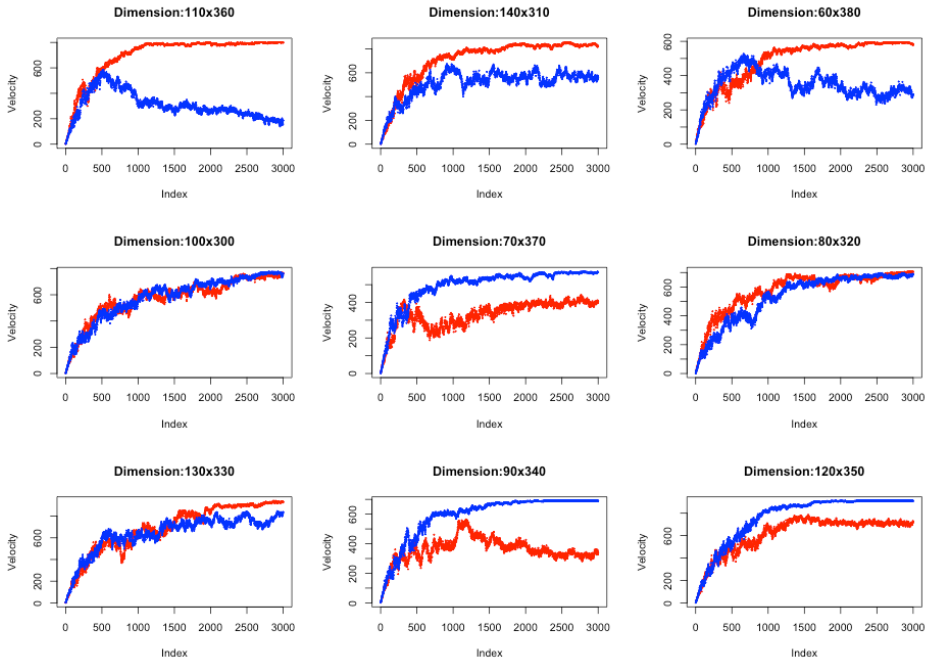

Figure 5 Velocities of Red Cars and Blue Cars with Low Density

For high car density case, there is either no significant evidence that red cars have higher velocity than blue cars or in inverse. And the difference between them is small.

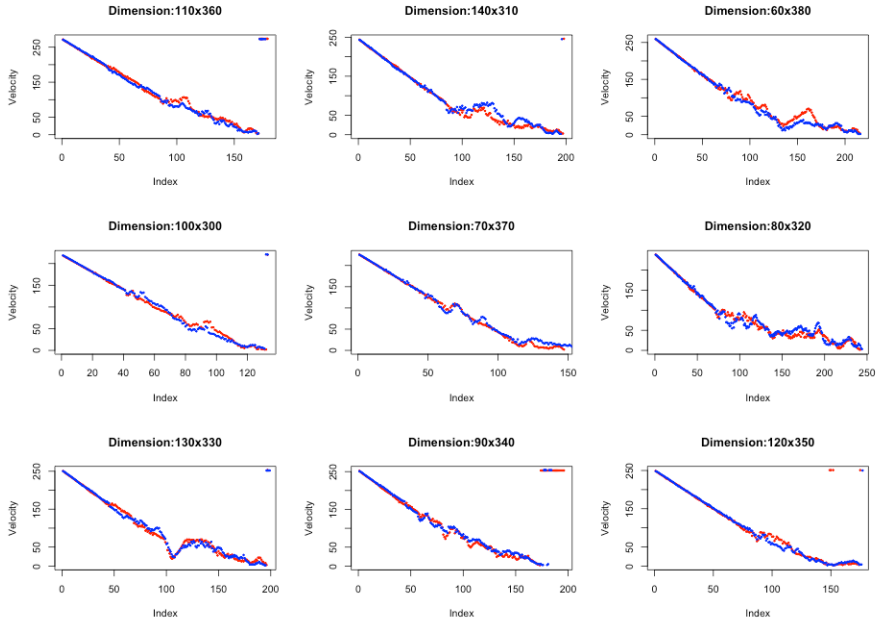

Figure 6 Velocities of Red Cars and Blue Cars with High Density

In general, larger column numbers does not suggest any advantage of moving red cars. But it is different with the square case. The lower

density case shows bigger gap between velocity of red and blue cars. It is reasonable to conclude that different relationship between row number and column number does affect the difference between velocity of red and blue cars.

2. Performance of Code

2.1. Grid Size

To test the performance of code, run different size grids (square) in 1000 steps. The running time increase dramatically as the dimension increase. This should be resulted by the logical matrix computation in the code. As the dimension goes up, the logical matrix needs more time to logical computation since all the logical matrixes have the same dimension as grid.

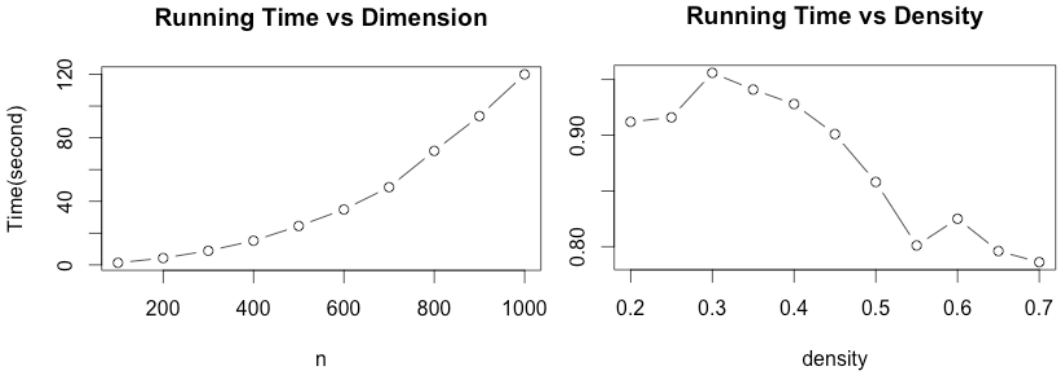

Figure 7 Performances in Different Dimension and Density

2.2. Car Density

Times for running 100 x 99 grids with different car density in 1000 steps are recorded and shown in Figure 7. As density increase, the running time decreases since only a few cars can be moved.

Appendix R Code

- **Code in BML Package**

```
createBMLGrid = function(r, c, ncars){
 #create a matrix, blank = 0, red = 1, blue = 2
 if( r<= 0 || c <=0 || ncars <=0){
  stop(" All the arguement must be positive!")
 }
 else{
  if( sum(ncars) >= r*c){
   stop(" Number of cars must be smaller than the number of grid cells!")
  }
  else{
   map = matrix(rep(0, r*c), r , c)
   if(length(ncars) == 1){
    #ncars is the percent of cars
    if(ncars > 1){
     stop(" The percent of cars must be smaller than 1!")
    }
    else{
     #tmp contains which grid number has car
     n = as.integer(ncars*r*c)
     tmp = sample( seq(1, r*c, 1), n, replace = FALSE)
     #randomly allocate the color
     map[tmp] = sample( rep(c(1,2), n ) , n)
    }
   }
   else{
    #ncars = c( red car number, blue car number)
    tmp = sample( seq(1,c*r,1) , sum(ncars), replace = FALSE )
    map[tmp] = 2
    map[sample(tmp, ncars[1], replace = FALSE)] = 1
   }
   class(map) = c("BMLGrid", "matrix")
   return(map)
    }
 }
}

plot.BMLGrid = function(grid){
 image(t(grid)[,nrow(grid):1 ], col = c('white', 'red', 'blue'), main = 'Move
Car')
}

compare = function(grid1, grid2){
 n = ncol(grid1)
```

```
  c = nrow(grid2)
  com = grid1 != grid2
  if( sum(com) ){
    #moven is the number of cars moving from grid1 to grid2
    if(any(grid1[com] == 2 )) {
      #blue cars are moved
      if( identical( movecar(grid1, TRUE,n,c), grid2 ) ){
        moven = sum(com)/2
        col = 'blue'
        n = sum(grid1 == 2)
        blockn = n - moven
      }
      else stop("grid2 is not moved from grid1")

    }
    else{
      #red cars are moved
      if( identical(movecar(grid1, FALSE, n, c) ,grid2) ){
        moven = sum(com)/2
        n = sum(grid1 == 1)
        blockn = n - moven
        col = 'red'
      }
      else stop("grid2 is not moved from grid1")
    }
  }
  else{
    col = "None"
    moven = 0
    blockn = 0
    v = 0
  }

  result = c( color = col, move = moven, block = blockn, v = moven/n)

  return( result )
}

summary.BMLGrid = function(grid){
  if(is.list(grid)){
    #grid is a list contains all the BMLgrids
    tmp = sapply(2:length(grid), function(i)
      compare( grid[[i-1]], grid[[i]] )  )
    tmp = data.frame(t(tmp))
  }
  else{
```

```
  redn = sum( grid == 1)
  bluen = sum( grid == 2)
  blank = prod(dim(grid)) - redn - bluen
  tmp = data.frame( red = redn, blue = bluen, blank = blank)
 }
 return(tmp)
}

movecar= function(m,t, no, ro){
 #if t = TRUE then move blue cars
 #if t = FALSE then move red cars
 if(t){
  #move blue cars
  #blue is a logical matrix who has the same dimension with m
  #blue indicates which one is blue cars in the grid m
  blue = m == 2
  #blank is a logical matrix who has the same dimension with m
  #blank indicates which one is empty in the grid m
  blank = m == 0
  #move blank down and add the last column ->blankt
  blankt = blank[ c(ro, 1:(ro-1) ), ]
  #if a TRUE cell in blue has the same location with that in blankt
  #then this TRUE cell (blue car) is moveable
  moveable = blankt & blue
  #then moveable cells will be empty (0)
  tmp = m
  tmp[moveable] = 0
  #then new location should be filled with 2
  tmp[ moveable[c( 2: ro, 1 ),] ] = 2
 }
 else{
  #move red
  #the logical is similar with moving blue cars
  red = m == 1
  blank = m == 0
  blankt = blank[,c(2:no, 1)]
  moveable = blankt & red
  tmp = m
  tmp[moveable] = 0
  tmp[ moveable[, c(no, 1:(no-1) )] ] = 1
  #class(tmp) = c("BML grid","matrix", paste0('move red = ', sum( moveable)
) )
 }
 return(tmp)
}
```

```
runBMLGrid = function(grid, step){
 map = list(start = grid)
 n = ncol(grid)
 c = nrow(grid)
 #col is a length(step) logical vector
 #each element in col indicates whether to move blue car(TRUE) or red
car(FALSE)
 col = rep(c( TRUE, FALSE), step/2 + 1)[1:step]
 for ( i in 1:step){
  map[[i+1]] = movecar(map[[i]], col[i], n, c)
 }
 #map is a list containing all the matrix in moving process
 class(map) = c("BMLGrid", "List")
 return(map)
}

playBMLGrid = function(grid, name = 'moving.gif', step){
 #output a gif to show the moving process
 ani.options(interval = 0.7, loop = TRUE )
 if( is.list(grid) ){
  saveGIF({
   lapply(grid, function(i)
    plot.BMLGrid(i) )
   ani.pause()
  }, movie.name = name)
 }
 else{
  map = grid
  n = ncol(grid)
  c = nrow(grid)
  col = rep(c( TRUE, FALSE), step/2 + 1)[1:step]
  saveGIF({
   plot.BMLGrid(map)
   for ( i in col){
    map = movecar(map,i, n,c)
    plot.BMLGrid(map)
   }
   ani.pause()
  }, movie.name = name )
 }
}
```

- **Analysis**

```
g = createBMLGrid(100, 99, 0.3)
playBMLGrid(g,'0.3density.gif', 1000)
g = createBMLGrid(100, 99, 0.7)
```

```
playBMLGrid(g,'0.7density.gif', 1000)

#1for a 100x99 dimenson, density from 0.2 to 0.7
g = list()
g = sapply( 2:7, function(i) createBMLGrid(100, 99, i/10))
su = lapply(g, function(i){
  summary(runBMLGrid(i, 6000))
})
density = c(2:7)/10
par( mfrow= c(2,3))
lapply(g, plot)

sapply(1:length(su), function(i)
  plot( as.numeric(su[[i]]$v), ylab = 'Velocity', main = paste0("Car Density: ",
density[i]), cex = 0.3))
par( mfrow= c(1,1))
plot(as.numeric(su[[7]]$v), main = density[7], cex = 0.3)

#2effect of grid shape
#grid shape might influcen the v of certain color
#2.1square dimension
#low car density
n = (6:14)*10
g = list()
g = sapply( n, function(i) createBMLGrid( i, i, 0.3))
su = lapply(g, function(i){
  summary(runBMLGrid(i, 1000))
})
#su = lapply(su, function(i) as.numeric(i$v))
blue = lapply(su, function(i) i[i$color == 'blue',])
red = lapply(su, function(i) i[i$color == 'red',])
par( mfrow= c(3,3))
sapply(1:length(blue), function(i){
  plot( as.numeric(blue[[i]]$v), col = 'blue', cex = 0.2,
        main = paste0("Dimension:", n[i], "x", n[i]), ylab = "Velocity")
  points( as.numeric(red[[i]]$v), col = 'red', cex = 0.2)
})

#high car density
n = (6:14)*10
g = list()
g = sapply( n, function(i) createBMLGrid( i, i, 0.7))
su = lapply(g, function(i){
  summary(runBMLGrid(i, 1000))
})
```

```
blue = lapply(su, function(i) i[i$color == 'blue',])
red = lapply(su, function(i) i[i$color == 'red',])
par( mfrow= c(3,3))
sapply(1:length(blue), function(i){
  plot( as.numeric(blue[[i]]$v), col = 'blue', cex = 0.2,
        main = paste0("Dimension:", n[i], "x", n[i]), ylab = "Velocity")
  points( as.numeric(red[[i]]$v), col = 'red', cex = 0.2)
})



#2.2c(columns) is bigger
#low car density
r = sample(6:14)*10
c = sample(30:38)*10

g = list()
g = sapply( 1:length(r), function(i) createBMLGrid( r[i], c[i], 0.3) )
su = lapply(g, function(i){
  summary(runBMLGrid(i, 6000))
})

blue = lapply(su, function(i) i[i$color == 'blue',])
red = lapply(su, function(i) i[i$color == 'red',])
par( mfrow= c(3,3))
sapply(1:length(blue), function(i){
  plot( as.numeric(red[[i]]$v), col = 'red', cex = 0.3, ylim = c(0,
max(as.numeric(su[[i]]$v)) ),
        main = paste0("Dimension:", r[i], "x", c[i]), ylab = "Velocity")
  points( as.numeric(blue[[i]]$v), col = 'blue', cex = 0.3)
})

sapply( 1:length(blue), function(i)
 c( nrow(blue[[i]]), nrow(red[[i]]) )
  )

sapply(1:length(blue), function(i){
  plot( as.numeric(red[[i]]$move), col = 'red', cex = 0.2, ylim = c(0,
max(as.numeric(su[[i]]$move)) ),
        main = paste0("Dimension:", r[i], "x", c[i]), ylab = "Move")
  points( as.numeric(blue[[i]]$move), col = 'blue', cex = 0.2)
})


#high density
g = list()
```

```
g = sapply( 1:length(r), function(i) createBMLGrid( r[i], c[i], 0.7) )
su = lapply(g, function(i){
  summary(runBMLGrid(i, 6000))
})

blue = lapply(su, function(i) i[i$color == 'blue',])
red = lapply(su, function(i) i[i$color == 'red',])
par( mfrow= c(3,3))
sapply(1:length(blue), function(i){
  plot( as.numeric(red[[i]]$v), col = 'red', cex = 0.2, ylim = c(0,
max(as.numeric(su[[i]]$v)) ),
       main = paste0("Dimension:", r[i], "x", c[i]), ylab = "Velocity")
  points( as.numeric(blue[[i]]$v), col = 'blue', cex = 0.2)
})

sapply( 1:length(blue), function(i)
  c( nrow(blue[[i]]), nrow(red[[i]]) )
)

sapply(1:length(blue), function(i){
  plot( as.numeric(red[[i]]$move), col = 'red', cex = 0.2, ylim = c(0,
max(as.numeric(su[[i]]$move)) ),
       main = paste0("Dimension:", r[i], "x", c[i]), ylab = "Move")
  points( as.numeric(blue[[i]]$move), col = 'blue', cex = 0.2)
})


#3.code performance
#size
n = (1:10)*100
g = list()
g = sapply( n, function(i) createBMLGrid( i, i, 0.3))
su = lapply(g, function(i){
  system.time(runBMLGrid(i, 1000))
})

time = unlist(lapply(su, function(i) i[1]))
time = data.frame(n,time)

plot(time, type = 'b', ylab = 'Time(second)', main = 'Running Time vs
Dimension')

#density
g = list()
density = c(4:14)/20
g = lapply( density, function(i) createBMLGrid(100, 99, i))
```

```
su = lapply(g, function(i){
  system.time(runBMLGrid(i, 1000))[1]
})
time = unlist(su)
time = data.frame(density, time)
plot(time, type = 'b', ylab = 'Time(second)', main = 'Running Time vs
Density')
```

- **Code Refine**

```
#Code Refine
#Original one
createBMLGrid = function(r, c, ncars){
  #create a matrix, blank = 0, red = 1, blue = 2
  map = matrix(rep(0, r*c), r , c)
  if(length(ncars) == 1){
    #ncars is the total number of cars
    tmp = sample( seq(1, r*c, 1), ncars, replace = FALSE)
    map[tmp] = sample( rep(c(1,2), ncars))
  }
  else{
    #ncars = c( red = ??, blue = ??)
    tmp = sample( seq(1,c*r,1) , sum(ncars), replace = FALSE )
    map[tmp] = 2
    map[sample(tmp, ncars[1], replace = FALSE)] = 1
  }
  class(map) = c("BML grid", "matrix")
  return(map)
}

mover = function(m){
  tp = m
  no = ncol(m)
  tp[, 2:no] = m[, 1:no-1]
  tp[, 1] = m[, no]
  return(tp)
}
movel = function(m){
  no = ncol(m)
  tp = m
  tp[, 1:no-1] = m[, 2:no]
  tp[, no] = m[, 1]
  return(tp)
}
moveu = function(m){
  ro = nrow(m)
  tp = m
```

```
  tp[1:ro-1, ] = m[2:ro, ]
  tp[ro, ] = m[1, ]
  return(tp)
}
moved = function(m){
  ro = nrow(m)
  tp = m
  tp[2:ro, ] = m[1:ro-1, ]
  tp[1, ] = m[ro, ]
  return(tp)
}
movelist = list(right = mover, left = movel , up = moveu , down = moved)

movecar= function(m,t){
  if(t%%2 ==1){
    #move blue cars
    blue = m == 2
    blank = m == 0
    blankt = moved(blank)
    tmp = m
    moveable = blankt & blue
    tmp[moveable] = 0
    tmp[ moveu(moveable) ] = 2
    # class(tmp) = c("BML grid","matrix", paste0('move blue = ', sum(
moveable) ) )
  }
  else{
    #move red
    red = m == 1
    blank = m == 0
    blankt = movel(blank)
    tmp = m
    moveable = blankt & red
    tmp[moveable] = 0
    tmp[ mover(moveable) ] = 1
    #class(tmp) = c("BML grid","matrix", paste0('move red = ', sum( moveable)
) )
  }
  return(tmp)
}
runBMLGrid = function(grid, step){
  map = list(start = grid)
  for ( i in seq(1, step, 1)){
    map[[i+1]] = movecar(map[[i]],i)
  }
  return(map)
```

```
}
playBMLGrid = function(grid, step){
 library(animation)
 ani.options(interval = 0.7, loop = TRUE )
 if( is.list(grid) ){
  saveGIF({
   lapply(grid, function(i)
    image(t(i)[,nrow(i):1 ], col = c('white', 'red', 'blue'), main = 'Move Car') )
   ani.pause()
  })
 }
 else{
  map = grid
  saveGIF({
   for ( i in seq(1, step, 1)){
    map = movecar(map,i, ncol(map), nrow(map))
    image(t(map)[,nrow(map):1 ], col = c('white', 'red', 'blue'), main = 'Move
Car')
   }
   ani.pause()
  }, outdir = "/Users/Shawn/Dropbox/242")
 }
}

Rprof("/Users/Shawn/Dropbox/242/WorkStudio/Assignment2/move.prof"
)
g = createBMLGrid(100, 99 ,c(100, 100))
glist = runBMLGrid(g, 20)
playBMLGrid(glist)
b = system.time( runBMLGrid(g,10000) )
Rprof(NULL)
summaryRprof("/Users/Shawn/Dropbox/242/WorkStudio/Assignment2/m
ove.prof")$by.self
self.time self.pct total.time total.pct
"<Anonymous>"     4.30    32.33      4.84     36.39
"movecar"         4.24    31.88     11.58     87.07
"&"               1.64    12.33      1.64     12.33
"runBMLGrid"      1.48    11.13     13.06     98.20
"=="              0.76     5.71      0.76      5.71
"-"               0.30     2.26      0.30      2.26
"gc"              0.24     1.80      0.24      1.80
":"               0.20     1.50      0.20      1.50
"[<-"             0.10     0.75      0.10      0.75
"ncol"            0.04     0.30      0.04      0.30
user  system elapsed
12.834   0.890  14.094
```

```
#Refine 1
mover = function(m){
  no = ncol(m)
  tp = m[, c(no, 1:(no-1) )]
  return(tp)
}
movel = function(m){
  no = ncol(m)
  tp = m[,c(2:no, 1)]
  return(tp)
}
moveu = function(m){
  ro = nrow(m)
  tp = m[c( 2: ro, 1 ),]
  return(tp)
}
moved = function(m){
  ro = nrow(m)
  tp = m[ c(ro, 1:(ro-1) ), ]
  return(tp)
}

Rprof("/Users/Shawn/Dropbox/242/WorkStudio/Assignment2/move1.prof
")
b1 = system.time( runBMLGrid(g,10000) )
Rprof(NULL)
summaryRprof("/Users/Shawn/Dropbox/242/WorkStudio/Assignment2/m
ove1.prof")$by.self
self.time self.pct total.time total.pct
"movecar"        3.46    39.59      7.52     86.04
"&"              2.18    24.94      2.18     24.94
"runBMLGrid"     1.00    11.44      8.52     97.48
"=="             0.88    10.07      0.88     10.07
"mover"          0.26     2.97      0.26      2.97
"movel"          0.22     2.52      0.28      3.20
"gc"             0.22     2.52      0.22      2.52
"moveu"          0.22     2.52      0.22      2.52
"moved"          0.18     2.06      0.18      2.06
"%%"             0.04     0.46      0.04      0.46
"c"              0.04     0.46      0.04      0.46
":"              0.02     0.23      0.02      0.23
"[<-"            0.02     0.23      0.02      0.23
user  system elapsed
8.777   0.597   9.420
```

```
#2
movecar = function(m,t){
 if(t%%2 ==1){
   #move blue cars
   tmp = m
   moveable = ( moveu(m) == 2 ) & (m == 0)
   tmp[moveable] = 2
   tmp[ moved(moveable) ] = 0
   # class(tmp) = c("BML grid","matrix", paste0('move blue = ', sum(
moveable) ) )
 }
 else{
   #move red
   tmp = m
   moveable = ( mover(m) == 1 ) & (m == 0)
   tmp[moveable] = 1
   tmp[ movel(moveable) ] = 0
   #class(tmp) = c("BML grid","matrix", paste0('move red = ', sum( moveable)
) )
 }
 return(tmp)
}

Rprof("/Users/Shawn/Dropbox/242/WorkStudio/Assignment2/move2.prof
")
b2 = system.time( runBMLGrid(g,10000) )
Rprof(NULL)
summaryRprof("/Users/Shawn/Dropbox/242/WorkStudio/Assignment2/m
ove2.prof")$by.self
self.time self.pct total.time total.pct
"movecar"        3.80    42.89       7.60     85.78
"&"              1.76    19.86       1.76     19.86
"runBMLGrid"     1.16    13.09       8.76     98.87
"=="             0.70     7.90       0.70      7.90
"mover"          0.54     6.09       0.54      6.09
"moveu"          0.34     3.84       0.36      4.06
"moved"          0.20     2.26       0.20      2.26
"movel"          0.16     1.81       0.20      2.26
"gc"             0.10     1.13       0.10      1.13
"[<-"            0.04     0.45       0.04      0.45
":"              0.02     0.23       0.02      0.23
"["              0.02     0.23       0.02      0.23
"c"              0.02     0.23       0.02      0.23
user  system elapsed
9.146   0.506   9.668
```

```
#3
movecar= function(m, t , no, ro){
 if( t%% 2 == 1) {
  blue = which( m == 2)
  blank = which( m == 0 )
  boundlog = blue%%ro == 1
  future = blue
  future[ boundlog] = blue[ boundlog] + ro
  future = future -1
  movei = future %in% blank
  tmp = m
  tmp[ blue[ movei ] ] = 0
  tmp[ future[movei] ] = 2
 }
 else {
  m = t(m)[no:1,]
  blue = which( m == 1)
  blank = which( m == 0 )
  boundlog = blue%%no == 1
  future = blue
  future[ boundlog] = blue[ boundlog] + no
  future = future -1
  movei = future %in% blank
  tmp = m
  tmp[ blue[ movei ] ] = 0
  tmp[ future[movei] ] = 1
  tmp = t(tmp[no:1,])
 }
 return(tmp)
}

runBMLGrid = function(grid, step){
 map = list(start = grid)
 n = ncol(grid)
 c = nrow(grid)
 for ( i in seq(1, step, 1)){
  map[[i+1]] = movecar(map[[i]], i, n, c)
 }
 return(map)
}

playBMLGrid = function(grid, step){
 library(animation)
 ani.options(interval = 0.7, loop = TRUE )
 if( is.list(grid) ){
```

```
   saveGIF({
    lapply(grid, function(i)
     image(t(i)[,nrow(i):1 ], col = c('white', 'red', 'blue'), main = 'Move Car') )
    ani.pause()
   })
  }
  else{
   map = grid
   saveGIF({
    for ( i in seq(1, step, 1)){
     map = movecar(map,i, ncol(map), nrow(map))
     image(t(map)[,nrow(map):1 ], col = c('white', 'red', 'blue'), main = 'Move
Car')
    }
    ani.pause()
   }, outdir = "/Users/Shawn/Dropbox/242")
  }
}

Rprof("/Users/Shawn/Dropbox/242/WorkStudio/Assignment2/move3.prof
")
b3 = system.time( runBMLGrid(g,10000) )
Rprof(NULL)
summaryRprof("/Users/Shawn/Dropbox/242/WorkStudio/Assignment2/m
ove3.prof")$by.self
b3
self.time self.pct total.time total.pct
"match"              4.76    44.49      4.76     44.49
"movecar"            2.24    20.93     10.00     93.46
"which"              1.04     9.72      1.84     17.20
"=="                 0.78     7.29      0.78      7.29
"t.default"          0.66     6.17      0.66      6.17
"runBMLGrid"         0.62     5.79     10.62     99.25
"standardGeneric"    0.42     3.93      2.88     26.92
"gc"                 0.08     0.75      0.08      0.75
"%%"                 0.04     0.37      0.04      0.37
"%in%"               0.02     0.19      4.78     44.67
"t"                  0.02     0.19      1.06      9.91
"loadMethod"         0.02     0.19      0.04      0.37
> b3
user  system elapsed
10.698   0.821  11.619

#4
movecar= function(m,t, no, ro){
 if(t%%2 ==1){
```

```
   #move blue cars
   blue = m == 2
   blank = m == 0
   blankt = blank[ c(ro, 1:(ro-1) ), ]
   tmp = m
   moveable = blankt & blue
   tmp[moveable] = 0
   tmp[ moveable[c( 2: ro, 1 ),] ] = 2
   # class(tmp) = c("BML grid","matrix", paste0('move blue = ', sum(
moveable) ) )
  }
  else{
   #move red
   red = m == 1
   blank = m == 0
   blankt = blank[,c(2:no, 1)]
   tmp = m
   moveable = blankt & red
   tmp[moveable] = 0
   tmp[ moveable[, c(no, 1:(no-1) )] ] = 1
   #class(tmp) = c("BML grid","matrix", paste0('move red = ', sum( moveable)
) )
  }
  return(tmp)
}
runBMLGrid = function(grid, step){
  map = list(start = grid)
  n = ncol(grid)
  c = nrow(grid)
  for ( i in seq(1, step, 1)){
   map[[i+1]] = movecar(map[[i]], i, n, c)
  }
  return(map)
}

Rprof("/Users/Shawn/Dropbox/242/WorkStudio/Assignment2/move4.prof
")
b4 = system.time( runBMLGrid(g,10000) )
Rprof(NULL)
summaryRprof("/Users/Shawn/Dropbox/242/WorkStudio/Assignment2/m
ove4.prof")$by.self
b4
self.time self.pct total.time total.pct
"movecar"        4.54    51.47       6.94     78.68
"runBMLGrid"     1.70    19.27       8.64     97.96
"&"              1.32    14.97       1.32     14.97
```

```
"=="             0.82    9.30      0.82     9.30
"gc"             0.18    2.04      0.18     2.04
":"              0.12    1.36      0.12     1.36
"[<-"            0.10    1.13      0.10     1.13
"c"              0.04    0.45      0.04     0.45
> b4
user  system elapsed
8.649   0.655   9.366
#5

#6
movecar= function(m,t, no, ro){
 if(t){
   #move blue cars
   blue = m == 2
   blank = m == 0
   blankt = blank[ c(ro, 1:(ro-1) ), ]
   tmp = m
   moveable = blankt & blue
   tmp[moveable] = 0
   tmp[ moveable[c( 2: ro, 1 ),] ] = 2
   # class(tmp) = c("BML grid","matrix", paste0('move blue = ', sum(
moveable) ) )
 }
 else{
   #move red
   red = m == 1
   blank = m == 0
   blankt = blank[,c(2:no, 1)]
   tmp = m
   moveable = blankt & red
   tmp[moveable] = 0
   tmp[ moveable[, c(no, 1:(no-1) )] ] = 1
   #class(tmp) = c("BML grid","matrix", paste0('move red = ', sum( moveable)
) )
 }
 return(tmp)
}
runBMLGrid = function(grid, step){
 map = grid
 n = ncol(grid)
 c = nrow(grid)
 col = rep(c( TRUE, FALSE), step/2 + 1)[1:step]
 for ( i in col){
   map = movecar(map, i, n, c)
 }
```

```
  return(map)
}
#little better

#7
movecar= function(blue,red, blank, color , no, ro){
  if(color ){
    #move blue cars
    blankt = blank[ c(ro, 1:(ro-1) ), ]
    #tmp = m
    moveable = blankt & blue
    newfuture = moveable[c( 2: ro, 1 ), ]
    blue[ moveable] = FALSE
    blue[ newfuture] = TRUE
    blank[ newfuture ] = FALSE
    blank[ moveable] = TRUE
    # class(tmp) = c("BML grid","matrix", paste0('move blue = ', sum(
moveable) ) )
  }
  else{
    #move red
    blankt = blank[,c(2:no, 1)]
    moveable = blankt & red
    newfuture = moveable[, c( no, 1 : (no-1) )]
    red[ moveable] = FALSE
    red[ newfuture] = TRUE
    blank[ newfuture ] = FALSE
    blank[ moveable] = TRUE
    #class(tmp) = c("BML grid","matrix", paste0('move red = ', sum( moveable)
) )
  }
  return(list( blue,red,blank))
}
runBMLGrid = function(grid, step){
  blue = grid == 2
  blank = grid == 0
  red = grid == 1
  map = list(blue, red, blank)
  n = ncol(grid)
  c = nrow(grid)
  col = rep(c( TRUE, FALSE), step/2 + 1)[1:step]
  for ( i in col){
    map = movecar( map[[1]], map[[2]], map[[3]], i, n, c)
  }
  grid[map[[1]]] = 2
  grid[map[[2]]] = 1
```

```
  grid[map[[3]]] = 0
  return(grid)
}
#much slower
```