Homework 2 Report

Stephanie Chan

February 4, 2013

Contents

1 Create a package for BML grid.

The BMLgrid package is in schanBMLgrid. More detail can be found in the package manual.

1.1 Class:

The BMLgrid class has slots for dimension (the dimension of the grid), redcars (the location of the red cars), and bluecars (the location of the blue cars). The BMLgrid is implemented using an S4 class. The dimension is a size 2 integer array for the dimension of the grid. The red cars and blue cars is a vector of integers determining the location. The location is given by the single integer value of the location of the grid as a matrix. ie. A location at column x and row y would be given by $(x-1)$*numrows + y.

1.2 Functions:

1.2.1 generateBMLgrid

This function generates a new BMLgrid with the specified dimension and specified number of red cars and blue cars each

1.2.2 plot

This function is from the general method plot and it creates an image plot showing all the red cars and blue cars.

1.2.3 checkIfCarStuck

This is a helper function used by updateRed and updateBlue to check if there is a car in the way before a filled spot

1.2.4 getdim/redcars/bluecars

These functions return the slots in the BMLgrid class

1.2.5 updateRed/updateBlue

These functions move the red cars or the blue cars. In order to make these functions faster the numbers are updated by adding or subtracting the number of spaces to move right by one column (adding the row number) or up by one row (subtracting one). The positions that are at the edges, are taken the modular of in order to cycle to the other side of the grid.

1.2.6 summary

Returns the dimension of the grid and the number of red and blue cars on the grid.

1.2.7 as.data.frame

Returns the cars in the grid object as a data frame with each row representing one car and columns of color, row, column of current location.

1.2.8 checkMoves

Helper function used to check if the cars were moved.

1.2.9 getVelocity

Function to compute average velocity of all cars by number of cars moved divided by the total number of cars.

1.2.10 updateManySteps

This function will move the car multiple steps. There is a video flag which will display a video of the cars moving.

2 R profile and timing

The time was saved in car movement by moving each of the cars by representing each car by a single number to represent the location in the matrix and using vectorized numeric operations such as adding to move the car. The %in% function was used to check if the car was allowed to move.

The R profile results come from a 100x100 grid with 200 red cars and 200 blue cars. These cars were moved through 100 time steps.

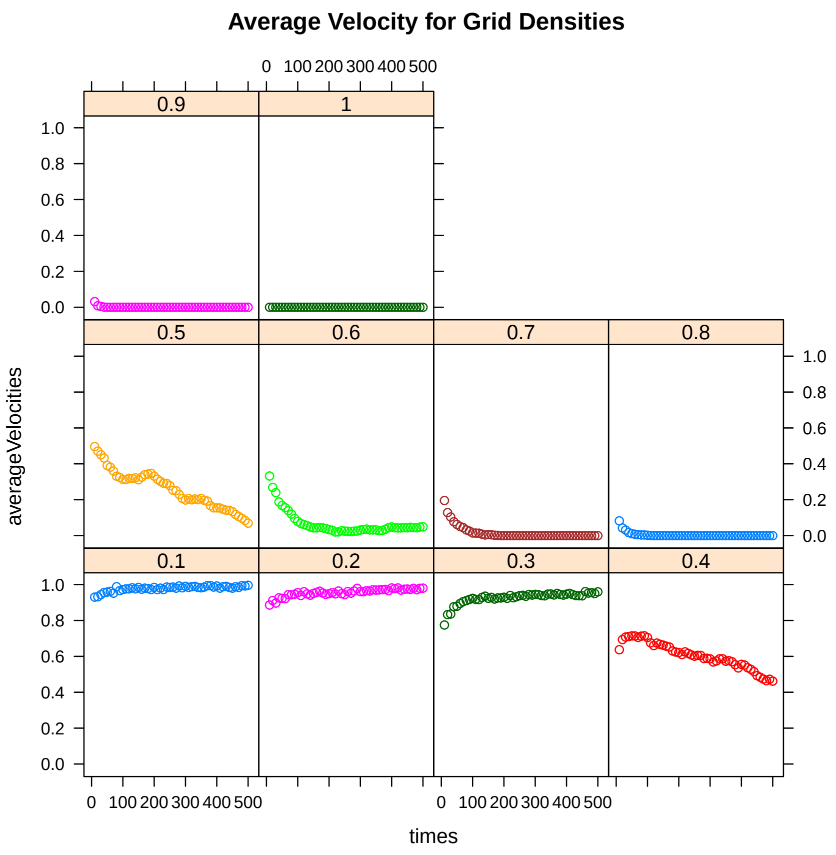
2.1 R profile results

```
> summaryRprof("Rprof.out")
$by.self
                   self.time self.pct total.time total.pct
"loadMethod"            0.08    57.14       0.10     71.43
"checkIfCarStuck"       0.02    14.29       0.04     28.57
```

```
"assign"               0.02    14.29       0.02     14.29
"match"                0.02    14.29       0.02     14.29

$by.total
                  total.time total.pct self.time self.pct
"standardGeneric"       0.14    100.00      0.00     0.00
"updateManySteps"       0.14    100.00      0.00     0.00
"loadMethod"            0.10     71.43      0.08    57.14
"updateBlue"            0.08     57.14      0.00     0.00
"updateRed"             0.06     42.86      0.00     0.00
"checkIfCarStuck"       0.04     28.57      0.02    14.29
"assign"                0.02     14.29      0.02    14.29
"match"                 0.02     14.29      0.02    14.29
"bluecars"              0.02     14.29      0.00     0.00
"%in%"                  0.02     14.29      0.00     0.00

$sample.interval
[1] 0.02

$sampling.time
[1] 0.14
```

3 Checking Velocity as ρ changes

We vary the density of cars in the grid and the velocities will change with it. These tests were done on a 100 by 100 grid with car densities of $\rho = 0.1, 0.2, 0.3, \ldots, 0.9, 1.0$, and equal red and blue cars. For $\rho \leq 0.3$, the average velocity of the cars tend to one, and the traffic jams occur in the beginning. For $\rho > 0.5$, the cars are quickly congested and there is little velocity and freedom to move. At $\rho = 0.4$ and $\rho = 0.5$, the trend is also toward getting stuck after enough time passes.


Average Velocity for Grid Densities
0.9
1
0.5
0.6
0.7
0.8
0.1
0.2
0.3
0.4
averageVelocities
times
0
100
200
300
400
500
0.0
0.2
0.4
0.6
0.8
1.0

4 Appendix

4.1 gridS4.R

```
## This is a BMLgrid object
setClass("BMLgrid",representation(dim="integer",
                                  redcars="integer",bluecars="integer"),
         prototype(dim=c(0L,0L),redcars=NULL,bluecars=NULL))


generateBMLgrid = function(nrow,ncol,nred,nblue) {
    ## newBMLgrid: function to create a BMLgrid with dimensions nrow x ncol
    ## and randomly selected red and blue cars
    ## nred is the number of red cars, and nblue is the number of blue cars

    ## randomly sample from nrow*ncol to get the correct number of red and blue cars
    n = nrow*ncol
    ncars = nred+nblue
    ## 1. randomly get ncars
    ## 2. of ncars, select the red ones
    ## 3. set the rest to be blue
    cars = sample(1:n,ncars)
    redcars = sample(cars,nred)
    bluecars = cars[!(cars %in% redcars)]

    obj = new("BMLgrid",dim=as.integer(c(nrow,ncol)),redcars=redcars,bluecars=bluecars
    return(obj)
}


setMethod("plot","BMLgrid",
          # This function plots the BMLgrid object with red and blue
          # cars in an image plot
          function(gridobj,x,y,...) {

              # make the BMLgrid into a matrix with -1 for red cars and 1 for blue car
              dim = gridobj@dim
              mat = matrix(0,dim[1],dim[2])

              mat[gridobj@redcars]=-1
              mat[gridobj@bluecars]=1

              # shift the matrix so it shows up right side up.
              shiftedmat = t(apply(mat,2,rev))

              red = "#FF0000FF"
              white = "#FFFFFFFF"
              blue = "#0000FFFF"

              image(shiftedmat,col=c(red,white,blue),xaxt='n',yaxt='n',...)
          })

checkIfCarStuck = function(toMove,filledSpots) {
    ## This function checks if the toMove cars are stuck behind an
    ## inPlace car.
    return(toMove %in% filledSpots)
}

setGeneric("getdim",function(obj) standardGeneric("getdim"))

setMethod("getdim","BMLgrid",
          function(obj) obj@dim )

setGeneric("redcars",function(obj) standardGeneric("redcars"))

setMethod("redcars", "BMLgrid",
          function(obj) obj@redcars )

setGeneric("bluecars",function(obj,...) standardGeneric("bluecars"))

setMethod("bluecars", "BMLgrid",
          function(obj,...) obj@bluecars )

setGeneric("updateBlue", function(obj,...) standardGeneric("updateBlue"))

setMethod("updateBlue","BMLgrid",
          ## Blue cars move one space up
          function(obj,...) {

              nrow = dim(obj)[1]
              blueOld = bluecars(obj)


              ## Move each car up by 1 spot
              ## If car is at top, then add nrow, move to end of line
              blueMove = blueOld-1 + as.integer(blueOld%%nrow==1)*nrow

              ## For debugging purposes
              ## print("blueOld")
              ## print(blueOld)
              ## print("blueMove")
              ## print(blueMove)

              ## print(blueOld-1)
              ## print(as.integer(blueOld%%nrow==1)*nrow)

              ## stuck cars do not move, because there is already some
              ## blue or red cars in their place
              stuckCars = checkIfCarStuck(blueMove,c(obj@bluecars,obj@redcars))
              blueMove[stuckCars] = blueOld[stuckCars]

              ## update the blue cars
              obj@bluecars = as.integer(blueMove)

              return(obj)
          })

setGeneric("updateRed", function(obj,...) standardGeneric("updateRed"))

setMethod("updateRed","BMLgrid",
          ## red cars move one space to the right
          function(obj,...) {

              d = dim(obj)
              nrow = d[1]
              n = d[1]*d[2]
              redOld = redcars(obj)

              ## Move each car right 1 spot, so nrow spaces
              ## If car is at the end, move to first line
```

```
                redMove = (redOld+nrow)
                redMove[redMove>n] = redMove[redMove>n]-n

                ## For debugging purposes
                ## print("redOld")
                ## print(redOld)
                ## print("redMove")
                ## print(redMove)
                ## print(nrow)
                ## print(n)

                ## stuck cars do not move
                stuckCars = checkIfCarStuck(redMove, c(obj@redcars,obj@bluecars))
                redMove[stuckCars] = redOld[stuckCars]

                ## update the red cars
                obj@redcars = as.integer(redMove)

                return(obj)
            })


setMethod("summary","BMLgrid",
            function(object,...) {
                cat("BMLgrid:\n")
                cat(paste("dim:",object@dim[1],"x",object@dim[2],"\n"))
                cat(paste(length(object@bluecars),"blue cars\n"))
                cat(paste(length(object@redcars),"red cars\n"))
            })


setMethod("as.data.frame","BMLgrid",
            function(x,...) {

                brow = x@bluecars %% x@dim[1]
                bcol = floor(x@bluecars / x@dim[2]) + 1
                bn = length(x@bluecars)

                rrow = x@redcars %% x@dim[1]
                rcol = floor(x@redcars / x@dim[2]) + 1
```

```
              rn = length(x@redcars)

              df = data.frame(color=c(rep("blue",bn),rep("red",rn)),
                row=c(brow,rrow),
                col=c(bcol,rcol))
          })


checkMoves = function(oldPlace,newPlace) {
    newPlace-oldPlace
}

getVelocity = function(oldPlace,newPlace) {
    if(length(oldPlace)!=length(newPlace))
      stop("oldPlace and newPlace different lengths")
    return(sum(oldPlace!=newPlace)/length(oldPlace))
}


## updateManySteps = function(obj,numsteps) {
##     ## This function updates the BMLgrid many steps

##     ## for every two steps, update blue, then update red
##     for(i in 1:floor(numsteps/2)) {
##         obj = updateBlue(obj)
##         obj = updateRed(obj)
##     }

##     ## if numsteps is odd, then update blue again, else, finish
##     if(!numsteps%%2)
##         obj = updateBlue(obj)

##     return(obj)
## }


updateManySteps = function(obj,numsteps,video=FALSE) {
    ## This function updates the BMLgrid many steps
    ## Blue moves first
    if(class(obj)!="BMLgrid")
      stop('not BMLgrid')

    if(video)
      plot(obj)

    toMove = "blue"

    for(i in 1:numsteps) {

        if(video) {
            dev.hold()
            Sys.sleep(1)
        }

        if(toMove=="blue") {
          obj=updateBlue(obj)
          toMove="red"
        } else if(toMove=="red") {
          obj = updateRed(obj)
          toMove="blue"
        }

        if(video) {
            plot(obj)
            dev.flush()
        }
    }

    return(obj)

}
```

4.2 testgrid.R

```
#source("gridS4.R")
library(schanBMLgrid)

## Initial Test
par(mfrow=c(3,3))
```

```
set.seed(100)
x = generateBMLgrid(20,20,10,10)
plot(x)
print(bluecars(x))
print(redcars(x))

# Test if blue car moves
for(i in 1:4) {
    x=updateBlue(x)
    plot(x)
    print(bluecars(x))
    x=updateRed(x)
    plot(x)
    print(redcars(x))
}



## Test with different parameters
par(mfrow=c(1,1))

set.seed(100)
x = generateBMLgrid(20,20,10,10)
plot(x)
print(bluecars(x))
print(redcars(x))

# Test if blue car moves
for(i in 1:5) {
    x=updateBlue(x)
    dev.hold()
    Sys.sleep(1)
    frame()
    plot(x,main=paste("step",i))
#    plot(x)
    dev.flush()
    print(bluecars(x))
    x=updateRed(x)
    dev.hold()
    Sys.sleep(1)
    frame()
    plot(x,main=paste("step i"))
#    plot(x)
    dev.flush()

    print(redcars(x))
}



set.seed(100)
y = generateBMLgrid(20,20,10,10)
print(redcars(y))
print(bluecars(y))

y = updateManySteps(y,10)
print(redcars(y))
print(bluecars(y))

y = updateManySteps(y,20,video=TRUE)

## do timing and profiling

Rprof("Rprof.out")
set.seed(100)
y = generateBMLgrid(100,100,200,200)
y = updateManySteps(y,100)
Rprof(NULL)
summaryRprof("Rprof.out")

## check to get the velocity of increasing the cars
getAverageFinalVelocity = function(BMLobj) {
    obj1 = updateBlue(BMLobj)
    blueVel = getVelocity(bluecars(BMLobj),bluecars(obj1))
    obj2 = updateRed(BMLobj)
    redVel = getVelocity(redcars(BMLobj),redcars(BMLobj))
    mean(blueVel,redVel)
```

```
}

## see what happens to the velocity as the traffic grid gets fuller
set.seed(100)
## create a set of grids of varying fullness
rho = 1:10*0.1
gridsize = 100
diffrhogrids = lapply(rho, function(x) generateBMLgrid(gridsize,gridsize,x*gridsize^2/

## create a function to record the velocity as time changes
velocityOverTime = function(obj,deltatimes) {

    output = numeric(length(deltatimes))
    for (i in 1:length(deltatimes)) {
        obj = updateManySteps(obj,deltatimes[i])
        output[i] = getAverageFinalVelocity(obj)
    }

    output
}

allVelocities = sapply(diffrhogrids,velocityOverTime,deltatimes = rep(10,50))

## plot the changed velocities
library(lattice)
times = rep(1:50*10,10)
rho = rep(1:10*0.1,each=50)
xyplot(allVelocities~times|as.factor(rho),groups=rho,main="Average Velocity for Grid d
```