STA 242 Assignment 4

BML Model with C
Xueting Shao
Bitbucket : git@bitbucket.org:sxshao/sta242.git

1. Introduction

1.1. Biham-Middleton-Levine Traffic Model

BML Model is a simulation of cars moving on a grid. There are two types of cars: "red" and "blue", which are moving on a $r \times c$ two-dimensional grid.

In the grid, $\rho \times r \times c$ cars are randomly placed at difference cells. And the type of each car is randomly selected from "red" and "blue" with equal probability.

Blue cars move at time periods t = 1,3,5… and red cars move at time periods t = 2,4,6… Blue cars move vertically upward and red cars move horizontally rightwards. When a blue/ red car gets to the edge of the grid: the top row/ the last column, it moves to the bottom row of the same column/ first column of the same row. Meanwhile, a car cannot move to another cell if that cell is already occupied by another car (no matter what color it is).

1.2. Model Behavior

As time period t increases, BML model process goes in to a certain moving pattern. It seems cars form a collective group and build a pattern of self-organization motion.

As density ρ changes from 0.2 to 0.7, the moving pattern exhibits a phase transition. Two different moving patterns appear in low density and high density, as shown in Figure 1.

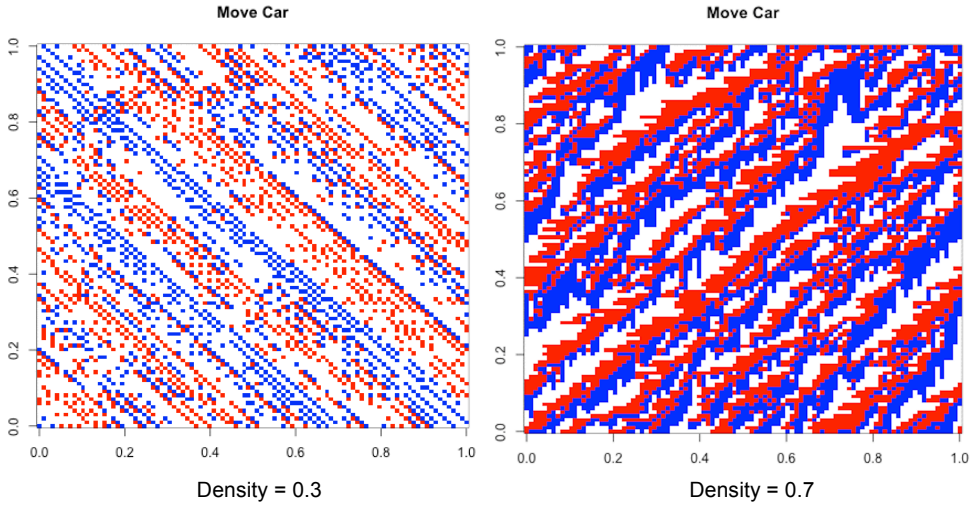


Figure 1 Two Phases

In low-density car moving process, cars automatically form a line-flowing phase. Cars can move smoothly without traffic jam. In high-density car moving process, cars will jam and cannot move at all at later time periods.

2. Code Performance

To simulate BML, R functions: runBMLGrid(), crunBMLGrid() and crun2BMLGrid() are constructed. All three functions have input parameters g and s, which call functions to moving grid g in s steps. Functions runBMLGrid() and crunBMLGrid() will return a list containing all the intermediate grids while crun2BMLGrid will only return the final grid.

The intuitive ideas of three functions are the same. There are two processes in three functions. First one is a process moving a car in one step; the second one is loop the first one over steps.

But those functions have three approaches to complete the two processes. (1) Both two processes in runBMLGrid() are written in R and R compute those via its own way in C. (2) The first process of crunBMLGrid() is written in C, pass the result back to R and R completes the second process. crunBMLGrid() needs $7\times r\times c$ loops in C to complete the first process once and s loops in R to complete the second process. (3) For crun2BMLGrid(), both two processes are written in C and R only takes the final grid and converts it into R object. There are $7\times r\times c\times s$ loops in total and all the loops are done in C.

Therefore, it is not surprising to see those three functions have various performances.

In Figure 2, it shows running times of three functions moving cars for 1000 steps in grids of different dimensions and car densities. It is obvious that runBMLGrid() takes much more time than crunBMLGrid() and crun2BMLGrid(). Besides, running times increase as grids become bigger.

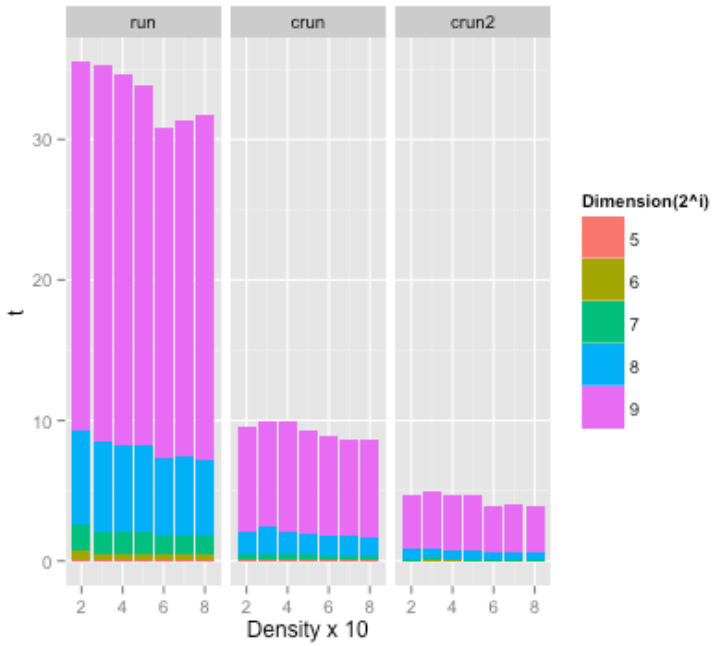


Figure 2 Running Times of Three Functions

However, the increasing pattern varies in function. runBMLGrid() increases most drastically and crun2BMLGrid() increases least drastically, which is not beyond expectation. Since C is lower level language, loop in C is super fast. Additionally, most loops are needed in the first process and improvement of first process will have the biggest effect in total running time. This is exactly what crunBMLGrid() achieved, and this results the bigger gap between runBMLGrid() and crunBMLGrid() than the gap between crunBMLGrid() and crun2BMLGrid().

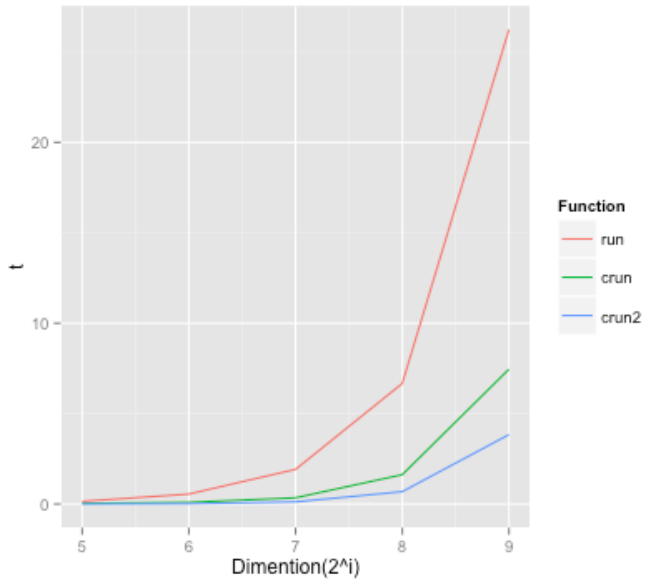
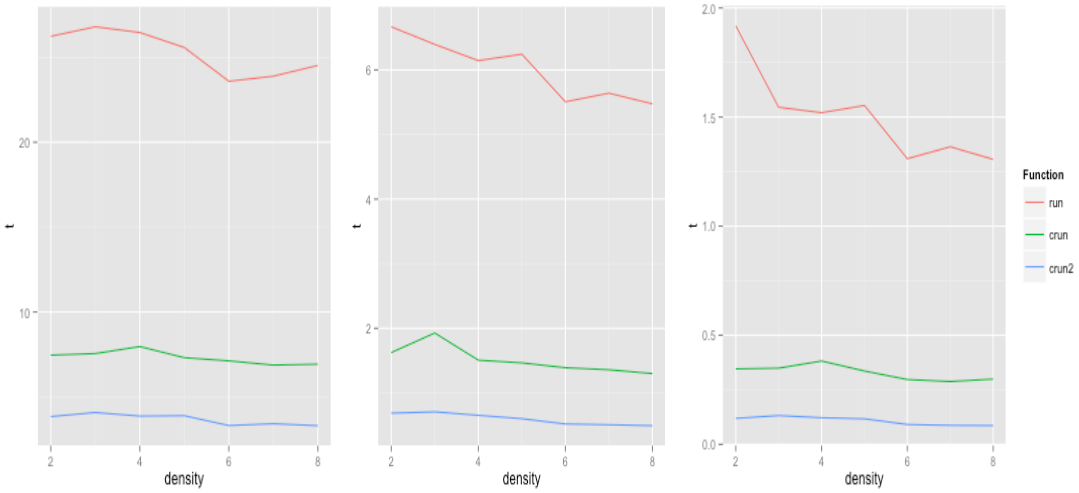


Figure 3 Running Time over Different Dimensions (Density = 0.2)

Meanwhile, changing patterns over densities are not the same. runBMLGrid() decreases dramatically over density while running times for the other two appear to be more consistent. It may suggest R has an advantage on increasing car density. One guess is that the c process will still loop for the same times even if cars are jammed while R will "be lazy" and do "inner C loop" for less time.



Dimension = 2^9 x 2^9 Dimension = 2^8 x 2^8 Dimension = 2^7 x 2^7

Figure 4 Running Time over Different Car Densities

3. Conclusion

In general, BML Model functions written in C are faster than runBMLGrid(), which is written in R.

Since C is the basic and lower level language, it is foreseen C will do a better job in loops than R. Besides it does not take much time to pass results and convert them into R objects, although the matrix() function in R counts for the most time in the process.

As grid dimension increasing, running times of three functions increase but with different speed. Function runBMLGrid() has the highest acceleration and function crun2BMLGrid() has the lowest. Time of function runBMLGrid() also change dramatically when car density changes. Overall, runBMLGrid() is the most susceptible to grid variation. The most interesting fact is the running time of runBMLGrid() decreases drastically when density increase. This may be related to the R's advantage over vector/matrix/data frame.

Appendix1: crunBMLGrid()

1. R Code

```
movecarc = function(m, t, no, ro){
 result =  .C("movcar",as.integer(m),
           as.integer(t) , as.integer(no), as.integer(ro))[[1]]
 result = matrix(result, ro, no)
 class(result) = c("BMLGrid", "matrix")
 return(result)
}
crunBMLGrid = 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]] = movecarc(map[[i]], col[i], n, c)
 }
 #map is a list containing all the matrix in moving process
 class(map) = c("BMLGrid", "List")
 return(map)
}
```

2. C Code

```
#include <R.h>

void logim(int result[],int m1[], int m2[],int n);
void equalv( int result[], const int m1[], const int val, const int n);
void matrixassign(int tmp[],int m[], const int logi[], const int n, int val );
void tranblue(int tmp[], int m[], int no, int ro);
void tranblue1(int tmp[], int m[], int no, int ro);
void tranred(int tmp[], int m[], int no, int ro);
void tranred1(int tmp[], int m[], int no, int ro);

void movcar(int *m, int *t, int *no, int *ro)
{
 // m is ro*no long vector//
 int row = ro[0], col = no[0];
 int n = row*col;
 int car[n], blank[n], blankt[n], moveable[n];
 int newm[n], newm1[n],moveablet[n];
 if (*t == 1){
  //move blue cars//
```

```
 equalv(car, m, 2, n );
 equalv(blank, m, 0, n);

 tranblue(blankt, blank, col, row);

 logim( moveable, blankt, car, n);


 matrixassign(m, m, moveable, n, 0);
 tranblue1(moveablet, moveable, col, row);
 matrixassign(m, m, moveablet, n, 2);
}
else {
// move red cars
 equalv(car, m, 1, n );
 equalv(blank, m, 0, n);

 tranred(blankt, blank, col, row);

 logim(moveable, blankt, car, n);


 matrixassign( m,m, moveable, n, 0);
 tranred1( moveablet, moveable, col, row);
 matrixassign( m,m, moveablet, n, 1);
};

}




void logim(int result[],int m1[], int m2[],int n){
//m1, m2 are both arrays with length n//

 int i;

 for(i = 0; i<n; i++){
 if ( (m1[i]+m2[i]) == 2 )
   result[i] = 1;
 else
   result[i] = 0;
 }
}

void equalv( int result[], const int m1[], const int val, const int n){
```

```
// val is a single value//

 int i;

 for(i = 0; i<n; i++){
 if ( m1[i] == val) result[i] = 1;
 else result[i] = 0;
 }
}

void matrixassign(int tmp[],int m[], const int logi[], const int n, int val ){
 int i;
 for (i = 0; i<n; i++){
 if (logi[i] == 1) tmp[i] = val;
 else tmp[i] = m[i];
 }
}

void tranblue(int tmp[], int m[], int no, int ro){

 int i, j, n = ro*no;

 for(i = 0; i< no; i++) {
   tmp[i*ro] = m[(i+1)*ro -1];
   for(j = 1; j< ro; j++) tmp[ ro*i + j] = m[ ro*i + j -1];
 }
}

void tranblue1(int tmp[], int m[], int no, int ro){
 int i, j, n = ro*no;

 for(i = 0; i< no; i++) {
   tmp[(i+1)*ro -1] = m[i*ro];
   for(j = 1; j< ro; j++) tmp[ ro*i + j-1] = m[ ro*i + j];
 }
}

void tranred(int tmp[], int m[], int no, int ro){
 int i, j, n = ro*no;

 for(i = 0; i< ro; i++) {
  for(j = 1; j< no; j++) tmp[ (j-1)*ro + i] = m[ ro*j + i ];
 }
 for(i=0; i< ro; i++) tmp[ (no-1)*ro + i] = m[i];
}
```

```
void tranred1(int tmp[], int m[], int no, int ro){
 int i, j, n = ro*no;

 for(i = 0; i< ro; i++) {

   for(j = 1; j< no; j++) tmp[ ro*j + i ] = m[ (j-1)*ro + i];
 }
 for(i =0; i< ro; i++) tmp[i] = m[(no-1)*ro+i];

}
```

Appendix2: crun2BMLGrid()

1. R Code

```
crunBMLGrid = function(grid, step){
  n = ncol(grid)
  c = nrow(grid)
  result = .C("crun", as.integer(grid), as.integer(step), as.integer(n),
as.integer(c))[[1]]
  result = matrix(result, c , n)
  class(result) = class(grid)
  return(result)
}
```

2. C Code

```
#include <R.h>

void logim(int result[],int m1[], int m2[],int n);
void equalv( int result[], const int m1[], const int val, const int n);
void matrixassign(int tmp[],int m[], const int logi[], const int n, int val );
void tranblue(int tmp[], int m[], int no, int ro);
void tranblue1(int tmp[], int m[], int no, int ro);
void tranred(int tmp[], int m[], int no, int ro);
void tranred1(int tmp[], int m[], int no, int ro);
void movcar(int *m, int t, int *no, int *ro);

void crun(int *m, int *step, int *no, int *ro){
 int i;
 int t=1;
 for(i = 0; i< *step; i++){
 movcar(m, t, no, ro);
 if(t == 1) {t=0;}
 else {t = 1;}
```

```
 }
}

void movcar(int *m, int t, int *no, int *ro)
{
 // m is ro*no long vector//
 int row = ro[0], col = no[0];
 int n = row*col;
 int car[n], blank[n], blankt[n], moveable[n];
 int newm[n], newm1[n],moveablet[n];
 if (t == 1){
  //move blue cars//
  equalv(car, m, 2, n );
  equalv(blank, m, 0, n);

  tranblue(blankt, blank, col, row);

  logim( moveable, blankt, car, n);


  matrixassign(m, m, moveable, n, 0);
  tranblue1(moveablet, moveable, col, row);
  matrixassign(m, m, moveablet, n, 2);
 }
 else {
 // move red cars
  equalv(car, m, 1, n );
  equalv(blank, m, 0, n);

  tranred(blankt, blank, col, row);

  logim(moveable, blankt, car, n);


  matrixassign( m,m, moveable, n, 0);
  tranred1( moveablet, moveable, col, row);
  matrixassign( m,m, moveablet, n, 1);
 };

}




void logim(int result[],int m1[], int m2[],int n){
//m1, m2 are both arrays with length n//
```

```
 int i;

 for(i = 0; i<n; i++){
 if ( (m1[i]+m2[i]) == 2 )
   result[i] = 1;
 else
   result[i] = 0;
 }
}

void equalv( int result[], const int m1[], const int val, const int n){
// val is a single value//

 int i;

 for(i = 0; i<n; i++){
 if ( m1[i] == val) result[i] = 1;
 else result[i] = 0;
 }
}

void matrixassign(int tmp[],int m[], const int logi[], const int n, int val ){
 int i;
 for (i = 0; i<n; i++){
 if (logi[i] == 1) tmp[i] = val;
 else tmp[i] = m[i];
 }
}

void tranblue(int tmp[], int m[], int no, int ro){

 int i, j, n = ro*no;

 for(i = 0; i< no; i++) {
   tmp[i*ro] = m[(i+1)*ro -1];
   for(j = 1; j< ro; j++) tmp[ ro*i + j] = m[ ro*i + j -1];
 }
}

void tranblue1(int tmp[], int m[], int no, int ro){
 int i, j, n = ro*no;

 for(i = 0; i< no; i++) {
   tmp[(i+1)*ro -1] = m[i*ro];
   for(j = 1; j< ro; j++) tmp[ ro*i + j-1] = m[ ro*i + j];
```

```
 }
}

void tranred(int tmp[], int m[], int no, int ro){
 int i, j, n = ro*no;

 for(i = 0; i< ro; i++) {
   for(j = 1; j< no; j++) tmp[ (j-1)*ro + i] = m[ ro*j + i ];
 }
 for(i=0; i< ro; i++) tmp[ (no-1)*ro + i] = m[i];
}

void tranred1(int tmp[], int m[], int no, int ro){
 int i, j, n = ro*no;

 for(i = 0; i< ro; i++) {

    for(j = 1; j< no; j++) tmp[ ro*j + i ] = m[ (j-1)*ro + i];
 }
 for(i =0; i< ro; i++) tmp[i] = m[(no-1)*ro+i];

}
```