BML Model with C Code

Zhewen Shi(912501820)

May 12, 2015

1. C routines design

Three C routines were designed, two different methods for moveCars(moving cars at one time period) and one for runBMLGrid(moving cars for several time periods).

1.1 Moving cars at one time period.

The R funciton 'moveCars' is writen by c to move cars at one time period. Using 'color' to move red cars or blue cars. Two different methods were designed to finish the same task. The functions are designed as follows.

void move(int *r, int *c, int *grid, int *color, double *velocity)

where *r is the number of rows of a grid, *c is the number of columns of a grid, grid is an array stores the content of grid matrix (stored by columns), *color means which kind of car need to move(1 means red and 2 means blue), *velocity stands for the velocity of cars at this time period.

The first method first records current position and next position of moveable cars and then moves them to the correct positions. The pseudo-code is as follows:

Algorithm 1 Function: void move(int *r, int *c, int *grid, int *color, double *velocity)

```
1: int numColor ← the number of cars in that color(1 for red and 2 for blue)
2: int * curPos ← allocate memory sizeof(int) * numColor to record current positions of moveable cars
3: int * nexPos ← allocate memory sizeof(int) * numColor to record next positions of moveable cars
4: int k ← 0 to record the number of moveable cars
5: if *color == 1 then
6:     for i in range 0:(r[0]*c[0] - 1) do
7:         if grid[i] == color[0] then
8:             nexPosI ← next position of i-th cars(red)
9:             if grid[nexPosI] == 0 then
10:                update curPos[k] = i; nexPos[k] = nexPosI; k = k + 1
11:            end if
12:        end if
13:    end for
14: else
15:    for i in range 0:(r[0]*c[0]) do
16:        if grid[i] == color[0] then
17:            nexPosI ← next position of i-th cars(blue)
18:            if grid[nexPosI] == 0 then
19:                update curPos[k] = i; nexPos[k] = nexPosI; k = k + 1
20:            end if
21:        end if
```

Algorithm 1 move function (continued)

```
    end for
end if
*velocity ← k/numColor
for i in range 0:(k-1) do
    grid[curPos[i]] ← 0
    grid[nexPos[i]] ← *color
end for
```

The second method copy BML grid first and use it as a reference to move cars.The pseudo-code is as follows:

Algorithm 2 Function: void move2(int *r, int *c, int *grid, int *color, double *velocity)

```
nexGrid ← the copy of grid
int numColor ← 0 (use this variable to compute the number of cars in that color)
int numMove ← 0 (use this variable to compute the number of moveable cars)
if *color == 1 then
    for i in range 0:(r[0]*c[0] - 1) do
        if grid[i] == color[0] then
            numColor getsnumColor + 1
            nexPosI ← next position of i-th cars(red)
            if nexGrid[nexPosI] == 0 then
                update grid[nexPosI] = color[0]; grid[i] = 0; numMove++
            end if
        end if
    end for
else
    for i in range 0:(r[0]*c[0] - 1) do
        if grid[i] == color[0] then
            numColor getsnumColor + 1
            nexPosI ← next position of i-th cars(blue)
            if nexGrid[nexPosI] == 0 then
                update grid[nexPosI] = color[0]; grid[i] = 0; numMove++
            end if
        end if
    end for
end if
*velocity ← numMove/numColor
```

1.2 Moving cars several times.

The function is designed as follows:

void runSteps(int *numSteps, int *r, int *c, int *grid, double *velocity)

where *numSteps is times to run move function, *r is the number of rows of a grid, *c is the number of columns of a grid, grid is an array stores the content of grid matrix (stored by columns), velocity is an array stores the velocity of cars at each time period. In this function, move function is called several times. The code is attached in appendix.

2. Code Test

2.1 Unit Test

I choose a package (RUnit) to do unit test and test 7 functions (createBMLGrid, moveCars, cmoveCars, cmoveCars2, runBMLGrid, crunBMLGrid, crunBMLGrid2).

When testing createBMLGrid function, I tested whether a BML Grid can be created correctly, whether the code can handle exceptions such as density <0, the number of rows/columns <0, the number of cars larger than number of cells in the grid, the number of one kind of car < 0.

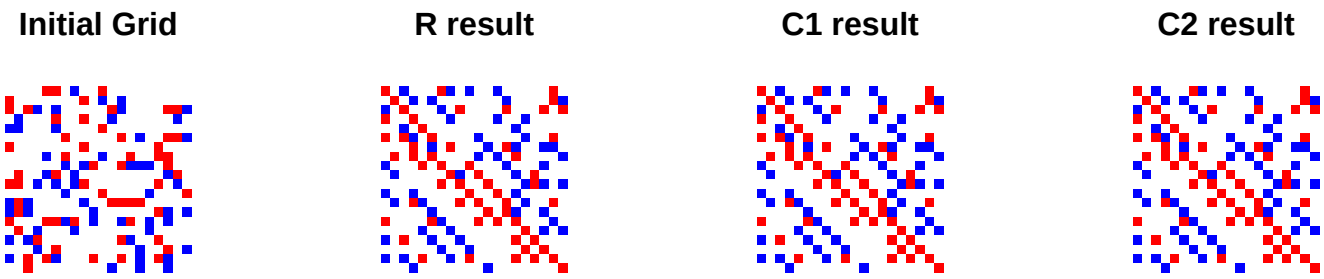
When testing moveCars, cmoveCars, cmoveCars2(moving cars at one time period according to color), I give test cases to check whether the algorithm can move correctly when there is only one kind of car, the number of rows/columns is 1 and dealing with normal cases.

When testing runBMLGrid, crunBMLGrid, crunBMLGrid2(moving cars at several time periods), I give test cases to check whether the algorithm can move correctly. I also test whether these three functions can get the same results in different cases.

The test code is in appendix.

2.2 Result Show

A 20*20 BML Grid with density 0.3 is created. Then let cars move in 1000 time periods. The results are as follows. We can see that all results are same.



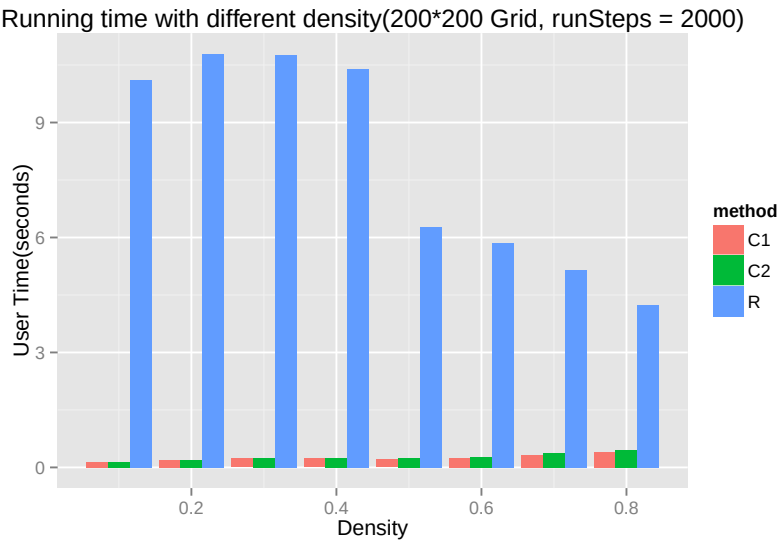
3. Performance Change

In this section, the three moving cars methods are compared(R method, C method 1, C method 2). We use R, C1, C2 to describe these three methods. All experiments are simulated three times and get the average value.

3.1 Is it worth implementing the moveCars function in C?

3.1 Running time with different density.

We use a 200*200 BML grid to test performance of each method. Cars were moved 2000 steps.



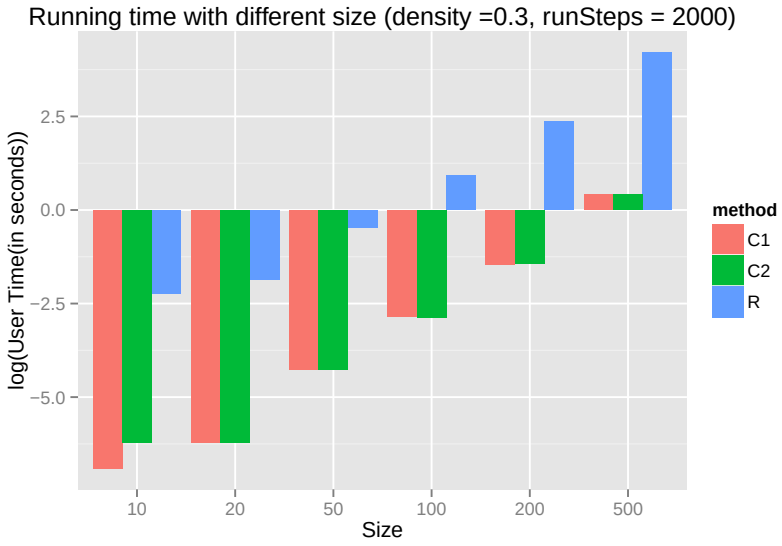
From the above figure, we can see that C code is much faster than R. Running time of R code reaches a peak around density 0.3. The reason is that there is an inflection density point between 0.3 and 0.4. If density is smaller than that point, all cars tend to move freely with the time increasing. If density is larger than that point, all cars will be blocked with the time increasing. If all cars are blocked, there is no need to move cars. If all cars can move freely, the time of changing grid will increase with the number of cars increase. So, with the density increasing, the running time first increases and then decreases.

We can also see that the running time of C code increases with the density increasing. The reason is that if the density increases, the number of cars in the grid will become larger. C code is not vectorized. So, we need more time to see which car can move to the new position.

The third point we can see from the figure is that between two C functions, method 1 is better than method 2. The reason is that if density is small, the number of cars is not large. The memory size we need to record current position and next position of moveable cars is smaller in Method 1(In method 2, we need to copy the whole grid).If density has a large value, all cars tends to be blocked with time increasing. So, we do not need to record positions of moveable cars(No car is moveable). In this case, method 1 is also faster than method 2.

3.2 Running time with different grid size.

Since the running time of R code is too large, we use log(running time) in the following figure.



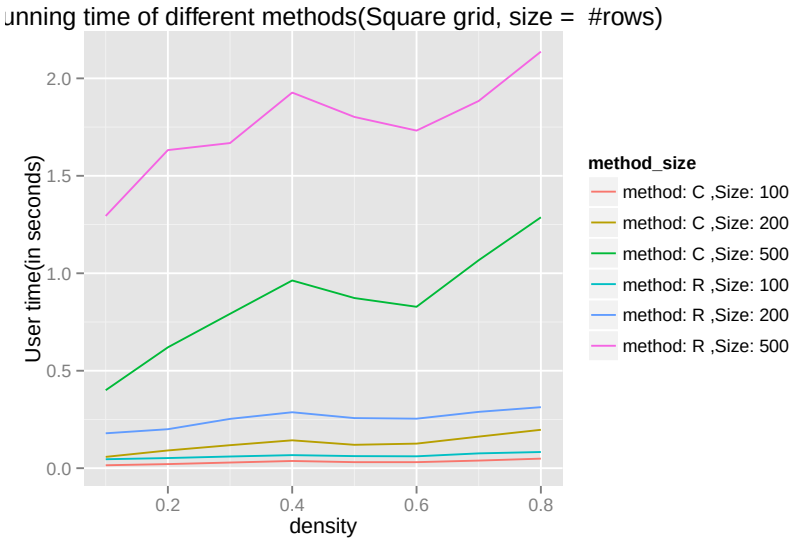
In the above figure, we use a square grid with density 0.3 and test the difference in running times with different number of rows/columns. We can see that running time of all methods increases with size increasing. We can also see that C1 is a little better than C2.

We can see that it is worth to implement moveCars function in C.

3.2 Is it worth implementing the runBMLGrid function in C?

In this section, we only test two methods. All of them use moveCars function in C. The difference is that the first method uses R to invoke cmoveCars(), the second methods uses C code to implement runBMLGrid function directly. We use R and C to distinguish them.

The performance is as follows(numSteps = 1000):



We can see that C is faster than R in all cases. So, it is also worth to implement the runBMLGrid in C.

4. Discussion: Is it worth implementing the grid creation function in C?

We can find the running time decreases much when implementing the moving function in C. So , it is worth to tranfer moving function to C code. When creating new grid, we do not need to implement in C. The reason is that we only need to create BML Grid once and then move cars for several times. Althogh C is faster than C, we can easily find sample functions in R. R is easier to use. Since we only create BML Grid once, I do not think we can save much time using C.