Code performance

Tianxia Zhou

2016-10-08

This is mean to assess the performance of the two different implementation of the main code in my package.

crunBMLGrid() calls C routines and it called in method such as **plotBMLPhases()**.

runBMLGrid() is implemented using only R and thus slower. It is the initial implementation. And it is not included in the name space and thus hidden from user.

Key findings

- The change of the size and density of the grid affect the run time (linearly).
- The use of C code improves the performance by a factor of about 5 (depending on the computing environment).

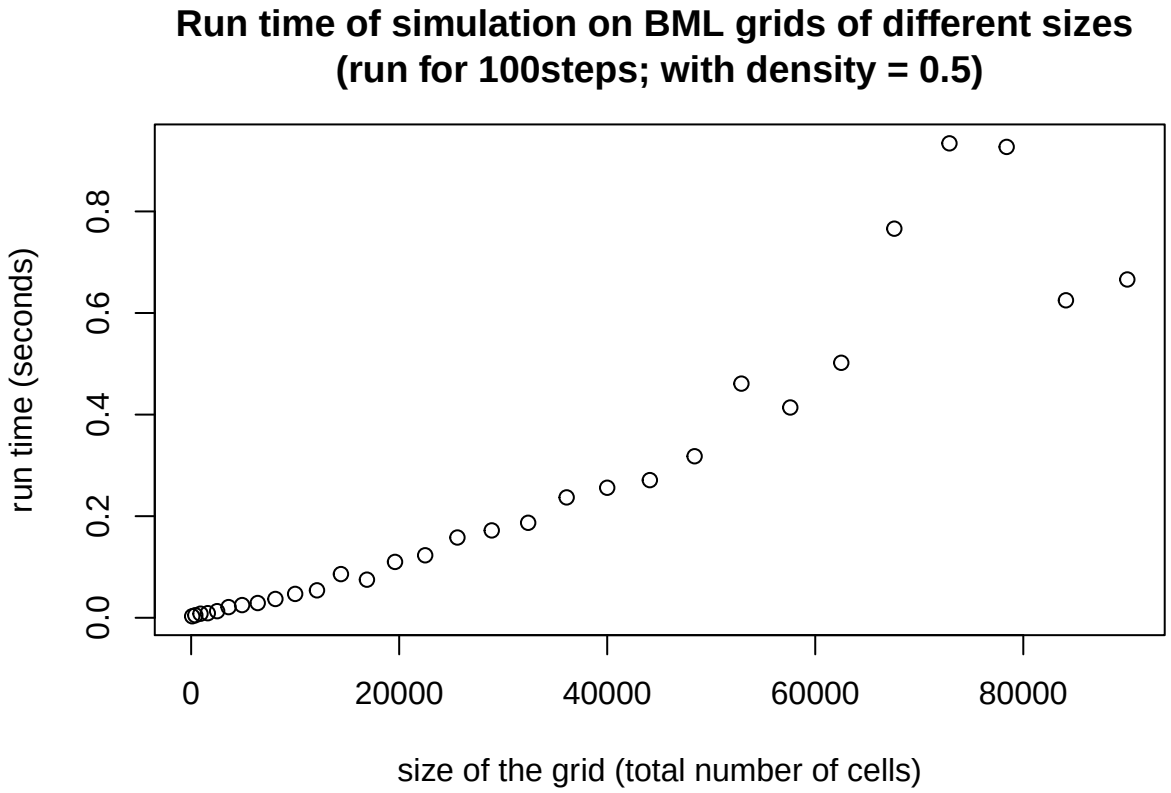
Performance of the "runBMLGrid()" function

The "**runBMLGrid()**" function is the vectorized version of the function to run simulation. In this version I change the representation of the colors in the grid. I choose to represent the colors ("white", "red" and "blue") with integers 0, 1 and 2, rather than with characters. Hence the function is improved and is slightly faster than the one I used in assignment 2.

Run time by different grid sizes ("runBMLGrid()")

In this test I do the speed test multiple times. In each iteration I initialize a square BML grid and run the simulation for 100 steps. I change the size of the grid in each iteration (from 10×10 to 30×30), but keep the density of the grid fixed (at 0.5). Then I record the run time in each iteration and plot the run time against the size of the grid. From the plot below we can see as the size of the grid increases, the run time increases. It seems like a linear relationship between the size of the grid and the run time.

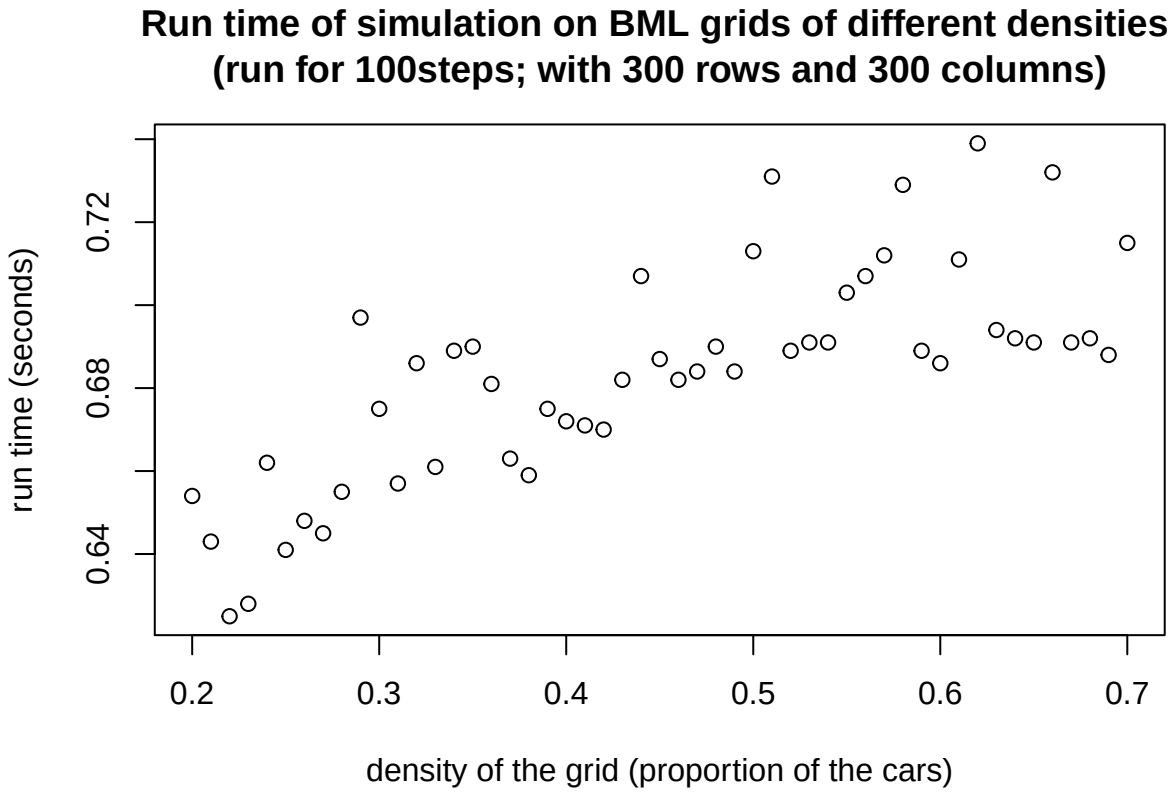
```
test.r1 = SimulateBML::plotPerformanceDim(run.function = SimulateBML:::runBMLGrid)
```



Run time by different densities ("runBMLGrid()")

In this test I do the speed test multiple times. In each iteration I initialize a square BML grid and run the simulation for 100 steps. I change the density of the grid in each iteration (from 0.2 to 0.7), but keep the size of the grid fixed (at 300×300). Then I record the run time in each iteration and plot the run time against the density of the grid. From the plot below we can see as the density of the grid increases, the run time increases. We probably can fit a linear line but the relationship between the density of the grid and the run time is not as clear as the relationship between the size of the grid and the run time. We can see there are a lot of noise in the plot.

```
test.r2 = SimulateBML::plotPerformanceDen(run.function = SimulateBML:::runBMLGrid)
```



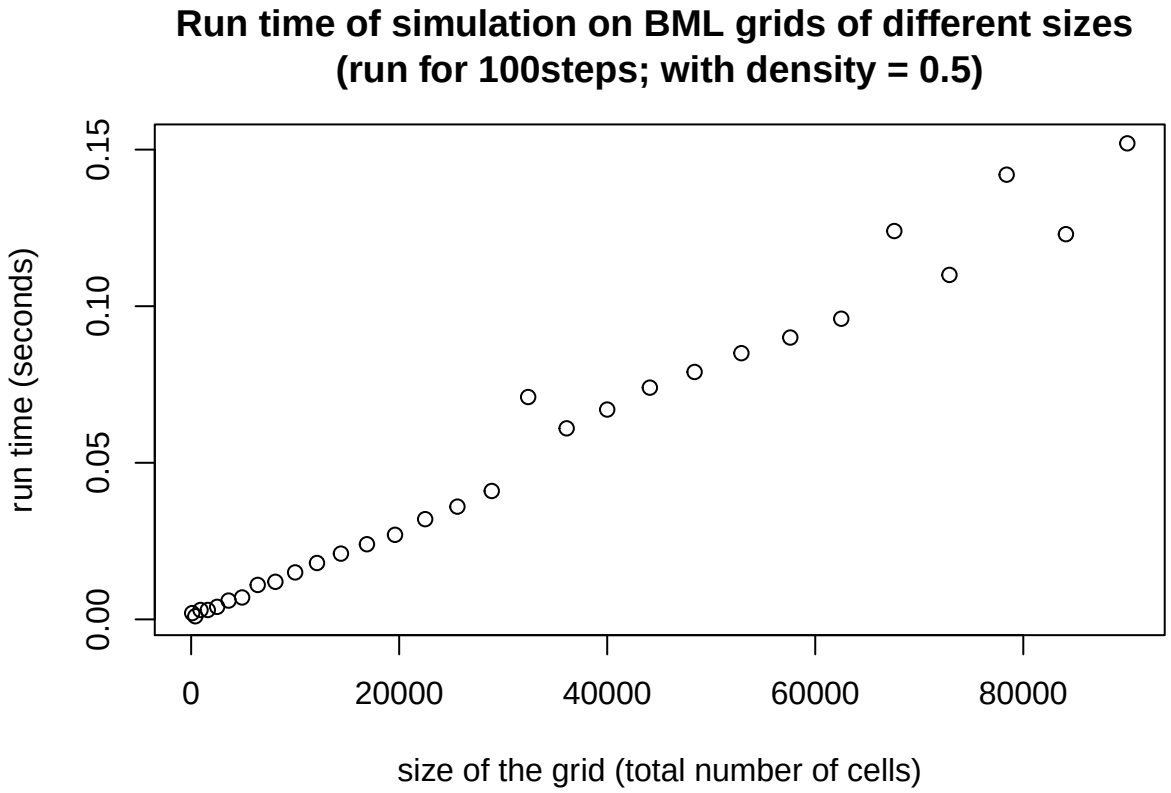
Performance of the "crunBMLGrid()" function

The "**crunBMLGrid()**" function is the C version of the function to run simulation. I implement C routines to move the cars and use wrapper functions in R to call it. The C code is actually implementing the algorithm of the "slower version" function I wrote in Assignment 2.

Run time by different grid sizes ("crunBMLGrid()")

The way I perform the test are exactly the same as I do in testing the "**runBMLGrid()**" function. The trend and qualitative result is similar. Below is the plot.

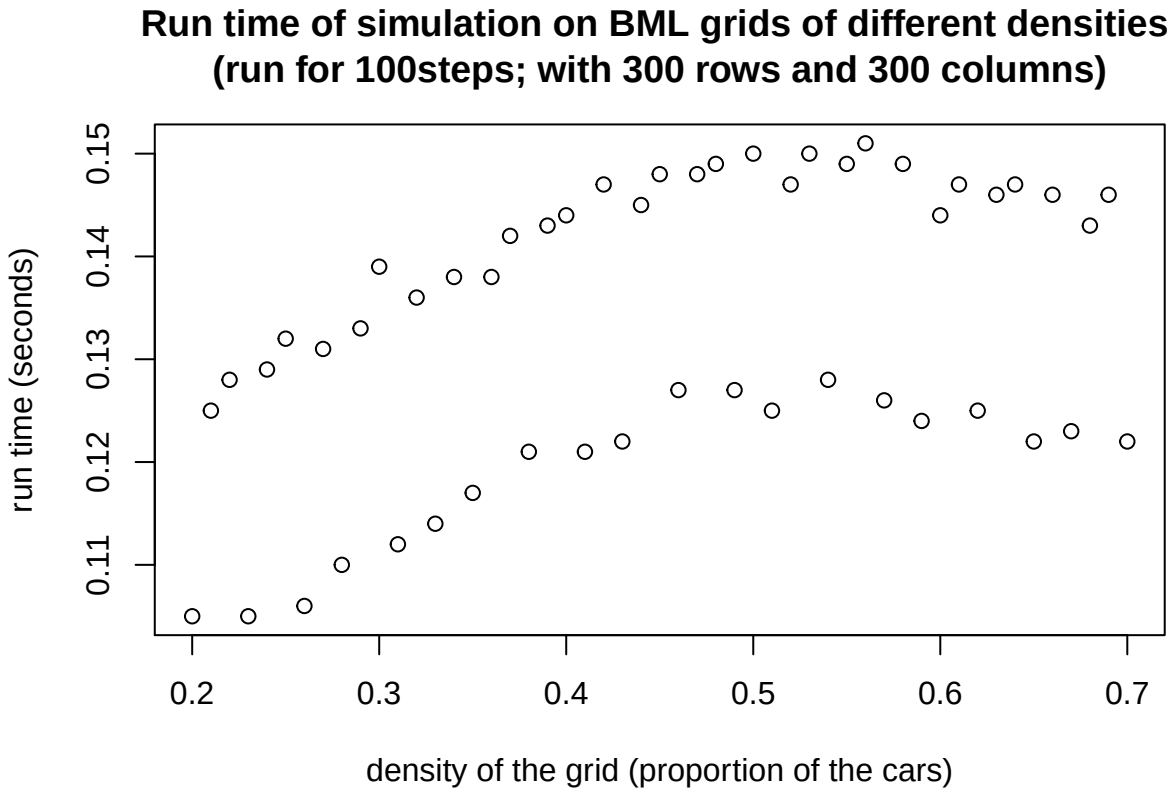
```
test.c1 = SimulateBML::plotPerformanceDim(run.function = SimulateBML::crunBMLGrid)
```



Run time by different densities ("crunBMLGrid()")

The way I perform the test are exactly the same as I do in testing the "**runBMLGrid()**" function. The trend and qualitative result is similar. Below is the plot.

```
test.c2 = SimulateBML::plotPerformanceDen(run.function = SimulateBML::crunBMLGrid)
```



Comparing the run time of the two functions

From the scale of the y axis of the plots above we already observe that the "**crunBMLGrid()**" function runs much faster than the "**runBMLGrid()**" function. Since I record the run time, I calculate the ratio and find out the "**crunBMLGrid()**" function reduces the run time by a factor of about 5 from the "**runBMLGrid()**" function.

```
mean(test.r1/test.c1)
```

```
## [1] 4.066756
```

```
mean(test.r2/test.c2)
```

```
## [1] 5.177301
```