UC Davis STA 242 2015 Spring Assignment 4

Wenhao WU, 9987583

May 10, 2015

1 Implementation of `crunBMLGrid()`

In order to highlight the performance difference between R and C++ implementation of BML simulation, we implement the entire `crunBMLGrid()` in C++. To do so, we made use of package 'Rcpp' []. The overall algorithm design is very similar to the original R function `runBMLGrid()`. However, at each step when we move all blue or red cars, the main operations are executed in a for-loop instead of vectorized, as defined in our C++ function `moveCars()`. As suggested by the example "R vectorisation vs. C++ vectorisation" in [], one advantage of C++ for-loop implementation over the R vecotrized operation is that it might need to create less intermediate vector variables. In our design we embraced this idea by passing reference variables to functions whenever possible and modifing them in-place (which may not be possible in R), and the necessary intermediate variables (`buffer_loc_next` and `buffer_movable`) are also persistent.

2 Verification

We first verify qualitatively the behavior of BML model computed with our new `crunBMLGrid()` routine. As in our previous assignment, We pick a $r = 100$, $c = 99$ grid in which the number of blue cars and red cars are the same. After $N = 10000$ steps, the final states of the grid for $\rho = 0.2, 0.33, 0.38, 0.43, 0.5$ are plotted in Fig. 1, where we observe the same chaotic phenomenon as the results computed with our original `runBMLGrid()` routine.
We further verify that `crunBMLGrid()` and `runBMLGrid()` returns identical result given the same input with package 'testthat' []. Besides the degenerated cases, in a $r = 100$, $c = 99$ grid we test 6 different cases where there are different numbers of red and blue cars. For each case, we randomly generate 5 instances of initial grid `g`. Our tests make sure that the two routines returns identical result after $N = 10000$ steps for all 30 runs. We have also manually checked that, upon early break from the outer for-loop due to grid lock when ρ is large, the number of steps executed before breaking are the same for both routines.

3 Running Time Comparison with R's Vectorized Operation

To compare the performance of `crunBMLGrid()` and `runBMLGrid()`, we measure the running time of both functions for $r = c = 128, 256, 512, 1024$ and $\rho = 0.1, 0.2, 9.3, 0.4, 0.5, 0.6, 0.7$. For each of the $4 \times 7 = 28$ settings we randomly generate 10 initial grids and apply both `crunBMLGrid()` and `runBMLGrid()` on them and record the average running time. We also fix the number of steps to $N = 10000$ and have the same number of red and blue cars in the grid. Again we run this test on a Dell Precision T1700 workstation equipped with 16GB DDR3 RAM and a Core i7-4790K CPU in Ubuntu 14.04 OS. The average running time in seconds and the relative speed up from `runBMLGrid()` to `crunBMLGrid()` are plotted in Fig. 2 and Fig. 3, repectively. The original data of Fig. 2 is also provided in the Appendix. In our test cases, the speed up is from 3x to 10x. In general, the larger the edge length, the smaller the speed up is. Surprisingly, the speed up for cases where there is no grid lock detected is significantly higher than the cases where there is grid lock and early breaks. Our guess is that the initialization part of the routine takes a larger portion of time in `crunBMLGrid()` than in `runBMLGrid()`.

4 Conclusion

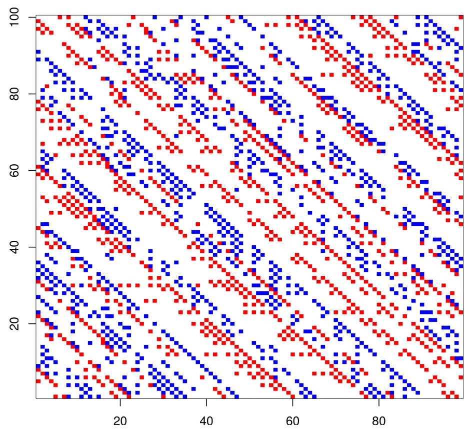
Based on the above results, our conclusion is

- There is a speed up from rewritting R's vectorization-based routines into C++ for-loop based routines in this assignment.
- However, in my opinion a C++ implementation is not worthwhile. In our case, the performance gain is not so significant, thus unless we are doing really time-consuming simulations, using R's vectorization-based routines is more advantageous in terms of agile development. Also, in order to maximize the speed up, we have rewritten the entire `runBMLGrid()` with C++. As a result, we cannot make the same function to return different types of values for different inputs and it is more difficult to add other functionalities (such as animation) to the function.

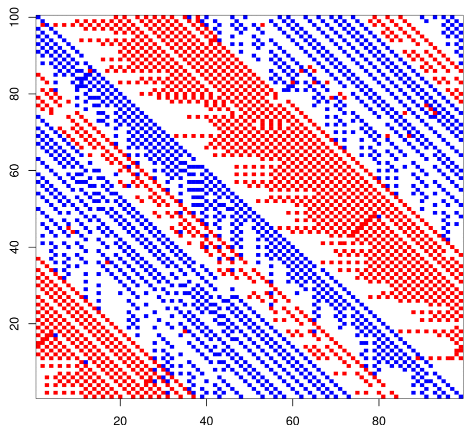
5 Appendix A: Running Time Results

6 Appendix B:

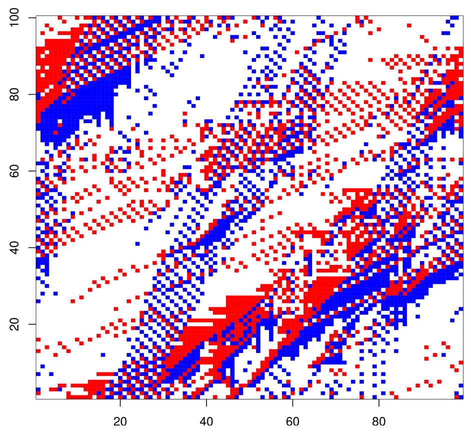
References



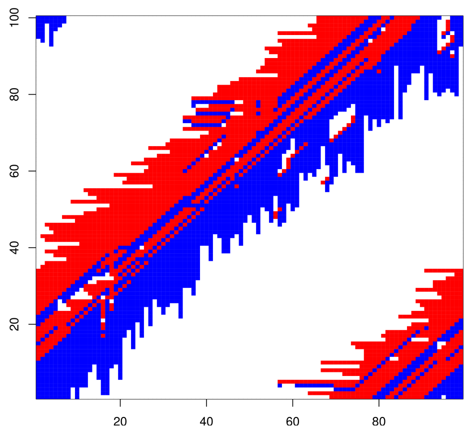
(a) $\rho = 0.2$



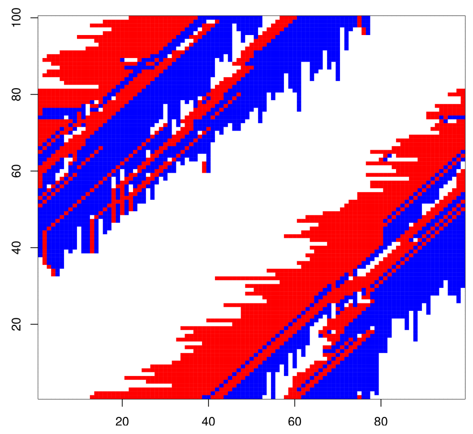
(b) $\rho = 0.33$



(c) $\rho = 0.38$



(d) $\rho = 0.43$



(e) $\rho = 0.5$

Figure 1: Final state of a 100×99 grid with equal number of blue and red cars after 10000 steps for different car density ρ.

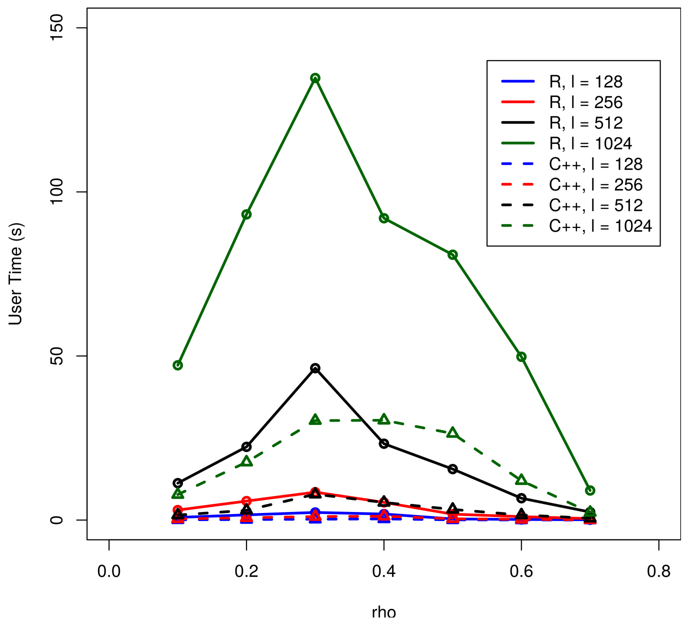


Figure 2: User time of `crunBMLGrid(g, 10000)` `runBMLGrid(g, 10000)` averaged over 10 repetitions for $\rho = 0.1, 0.2, 9.3, 0.4, 0.5, 0.6, 0.7$ and $r = c = 128, 256, 512, 1024$.

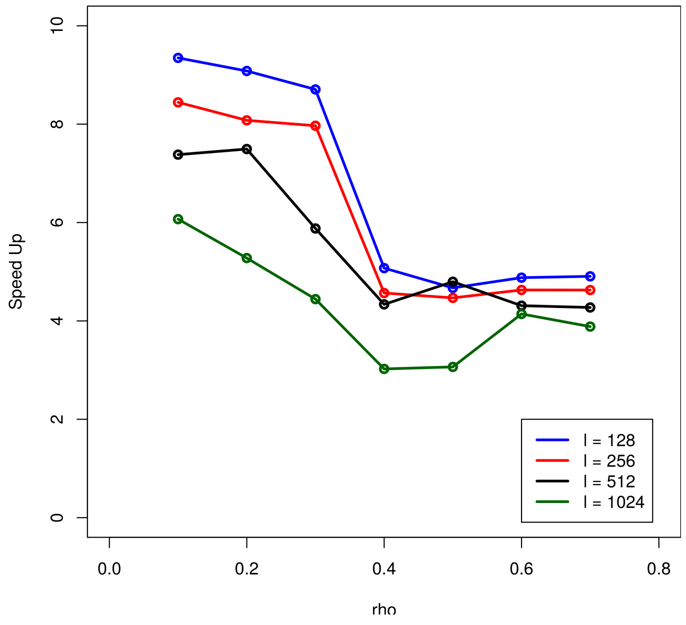


Figure 3: Relative speed up by rewriting the R routine with C++.