Vladimir Pchelin
Assignment #2
git@bitbucket.org:ph7vov/sta242.git

The content. First, I discuss the BML mode, then I discuss my code, profiling and performance issues.

Critical values. Speed vs Density. To find the critical value of density we do the following. We track average speed of cars between iterations N and $N+10$ with large enough N. It's expected that as N grows this average speed either goes to zero either becomes fairly close to one (this doesn't mean it converges to one). But, of course, the result also depends on the initial distribution of cars, which we can't control. The next figures represents dependence between the described speed and the corresponding density when $N = 800$.

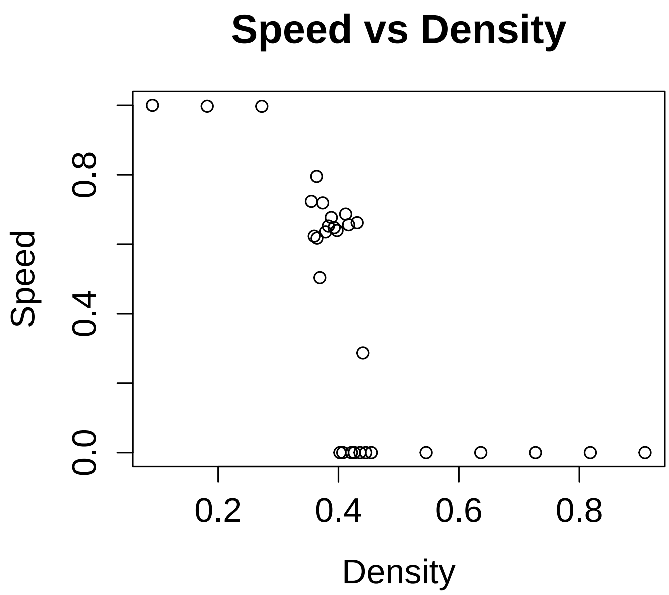

Figure 1

As one sees, it's reasonable to deduce that the critical point is between 0.37 and 0.41. At the same time it's hard to determine this point precisely because of the randomness of the initial placement of cars. We should also notice that the closer we are to the critical point the less predictable becomes car motion in the sense that it's asymptotic behavior strongly depends on the initial randomly chosen distribution.

Remark. Putting more points on the graph doesn't clarify the situation. It makes the picture dirty instead.

Speed vs Time. On the graph below we have lines representing average speed of cars of each color for two different densities: subcritical 0.3 and supercritical 0.5. This speed fluctuates a little bit from step to step. First of all, we see that the speed is stabilized staring from the step number 400. So we might conclude that our choice of $N = 800$ was a good choice at least for non-critical densities.

For the subcritical density the speed stabilizes faster and becomes close to one when time is large. We have almost free flow of cars. It takes more time for the supercritical speed to stabilize and it's exactly equal to zero when the number of steps if large enough. In the latter case cars don't move at all.

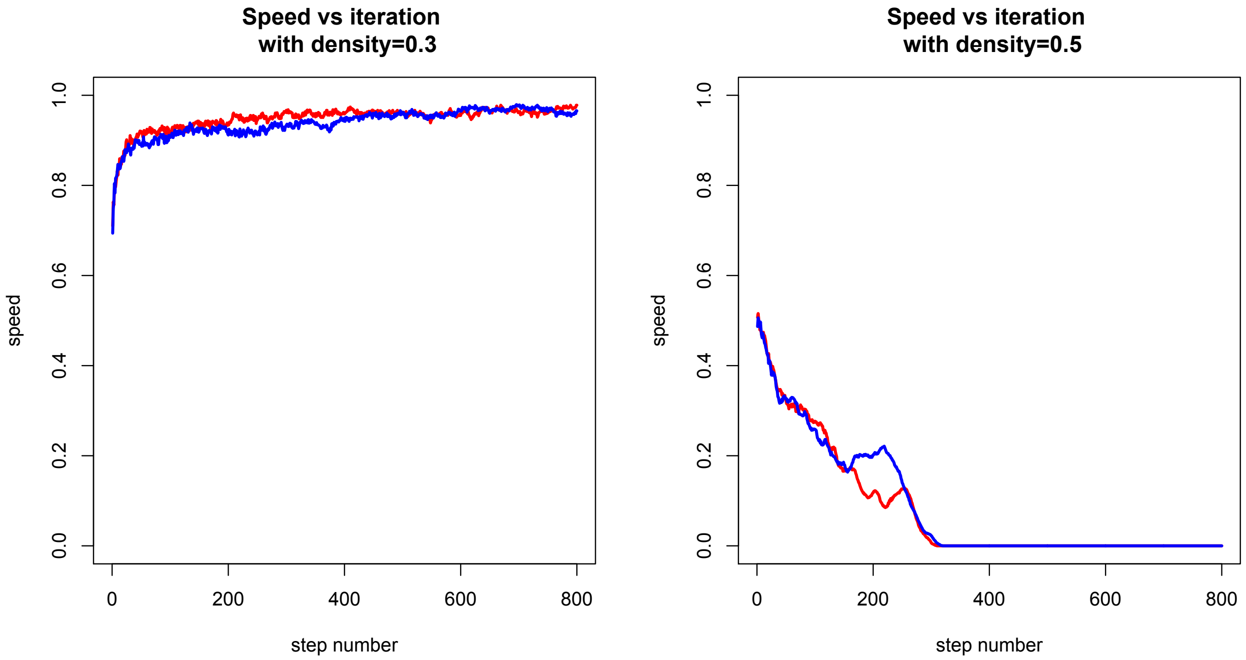

Figure 2

Matrix image. The next figure gives the car distributions after 1000 iterations.

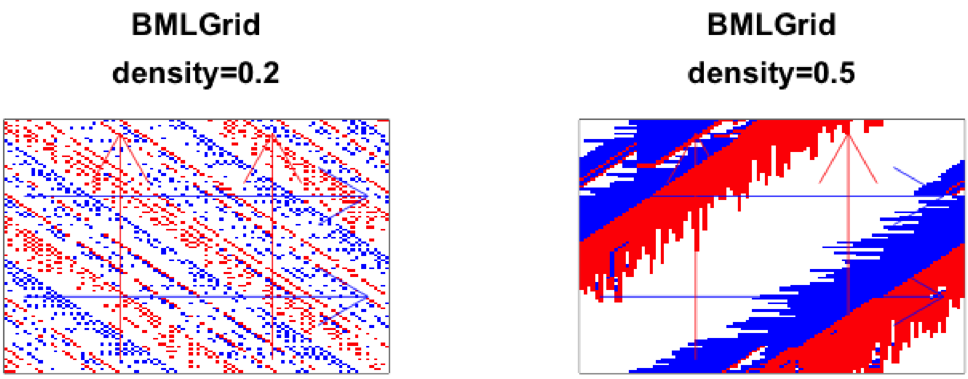

Figure 3

The above plots show the same picture: for subcritical density we have free flow, for supercritical density we have completely jammed regime.

Move function and performance. As advised in the assignment I made a code, function **runBMLGrid**, that is completely vectorized inside the loop (the loop over time, cars move N times) and I made a "naive" version of this code that uses **for** to loop over matrix entries, called **moveNaive**. Vectorization in **runBMLGrid** was made with help of **ifelse** and | functions.

Now we examine **system.time** for these two functions, see Appendix "Output". First of all, the time is almost proportional to the number of times cars move. That's why we will be only interested in **system.time** per N. The next table describes how the time depends on the size of matrix.

matrix size	10×10	100×100	1000×1000
system.time(moveNaive)/N	0.00060	0.03277	3.4974
system.time(runBMLGrid)/N	0.00029	0.01835	2.0459

Times is roughly proportional to the matrix size which is not quite fast. Also, time doesn't really depend on the density of cars (see Appendix).

I profiled the two algorithms (see Appendix) with a 100 by 100 matrix. It is a little surprising that profiling for the "naive" code indicates that **ifelse** took just few times less time than the **for** loop that made the major part of the work. Summarizing, at least for this particular assignment vectorization via **ifelse** didn't really make anything faster. The problem here is that I don't have a good understanding of how exactly **ifelse** works: how much it copies, allocates and so on. So for the significant performance enhancement we, probably, should exploit the parallel architecture of multicore/GPU in a clever way and/or interface R to other languages.

My package functionality. My package has the following functions: **createBMLGrid**, **runBMLGrid**, **plot.BMLGrid**, **summary.BMLGrid**, **speedBMLGrid**, **vectorDensitySpeed**. The first four have documentations. **speedBMLGrid** gives you a matrix each row of which contains information about the corresponding iteration step. **vectorDensitySpeed** allows you to construct a plot such as **Figure 1**.