Analyzing Latin Square Difficulty

Fernando Javier López Cerezo

January 13, 2022

1 Objectives

The objective of this project is to build a backtracking algorithm that solves an arbitrary instance of the Latin Squares problem. In this problem a n x n numerical puzzle is given with some squares having fixed values. The algorithm should fill the puzzle with numbers from 1 to n in a way such that no number is repeated in any column or row. We are then going to use this algorithm to find the critical complexity point of this problem.

2 Experimental Setup

For this project we used:

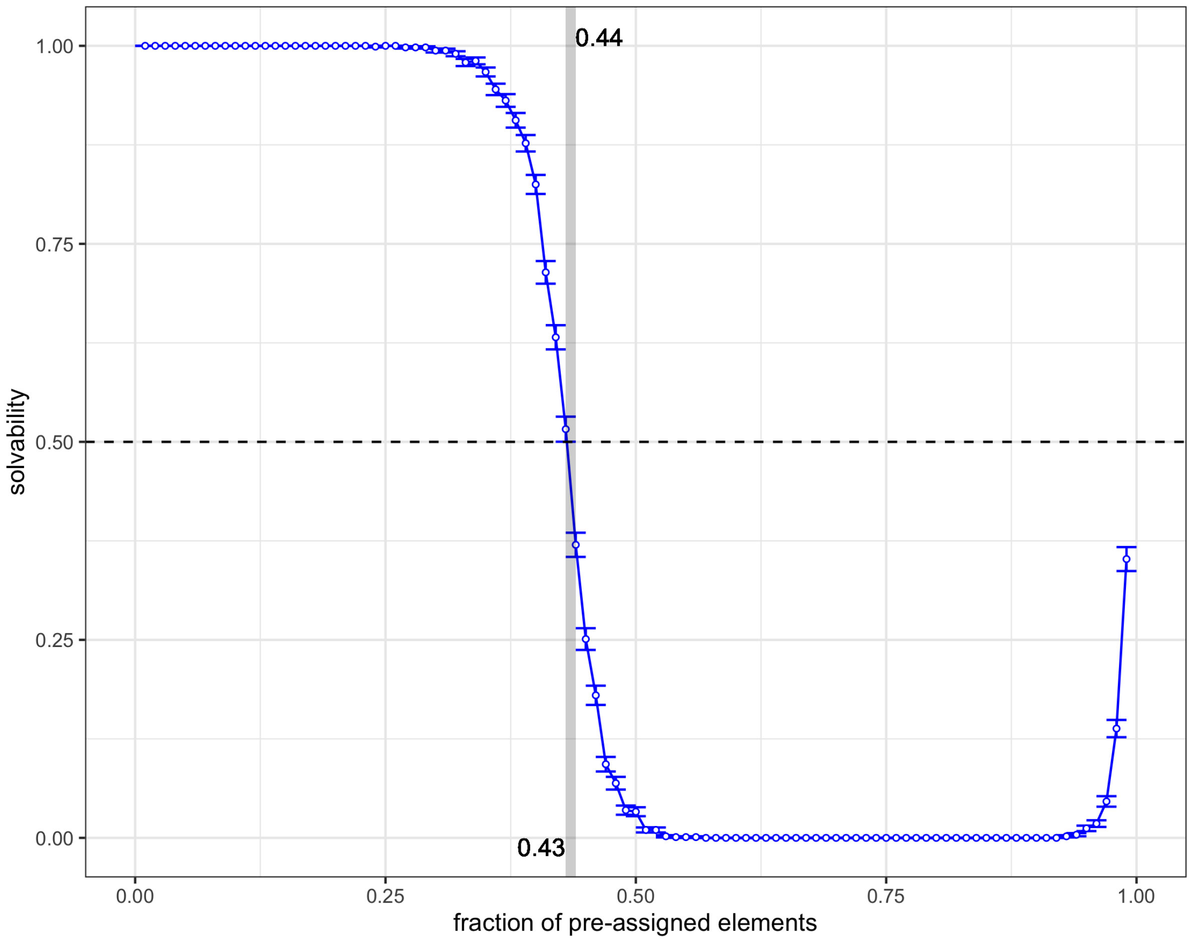
- MacBook Air 13 (1,8 GHz Dual-Core Intel Core i5 CPU, 8 GB 1600 MHz DDR3 RAM and macOS Catalina 10.15.5 OS)
- Java 15.0.2 and R 4.1.1

The method we used to obtain our data was the following:
The backtracking algorithm fills the puzzle row by row, from left to right. If a position is empty it checks which values are feasible for that position and continues recursively. If the position is fixed (can't be modified) the algorithm will still check if the value is feasible, in case the initial instance is unsolvable. When there are no feasible values left to put on a certain squares the algorithm will backtrack to the previous square and try with new values. Our algorithm also counts the number of recursive steps performed.

This algorithm was tested with more than 100,000 10x10 Latin Squares using different values for the control parameter γ (proportion of clues). At last all of this data was anaylzed using R and studied to assess the complexity of the algorithm as a function of the control parameter.

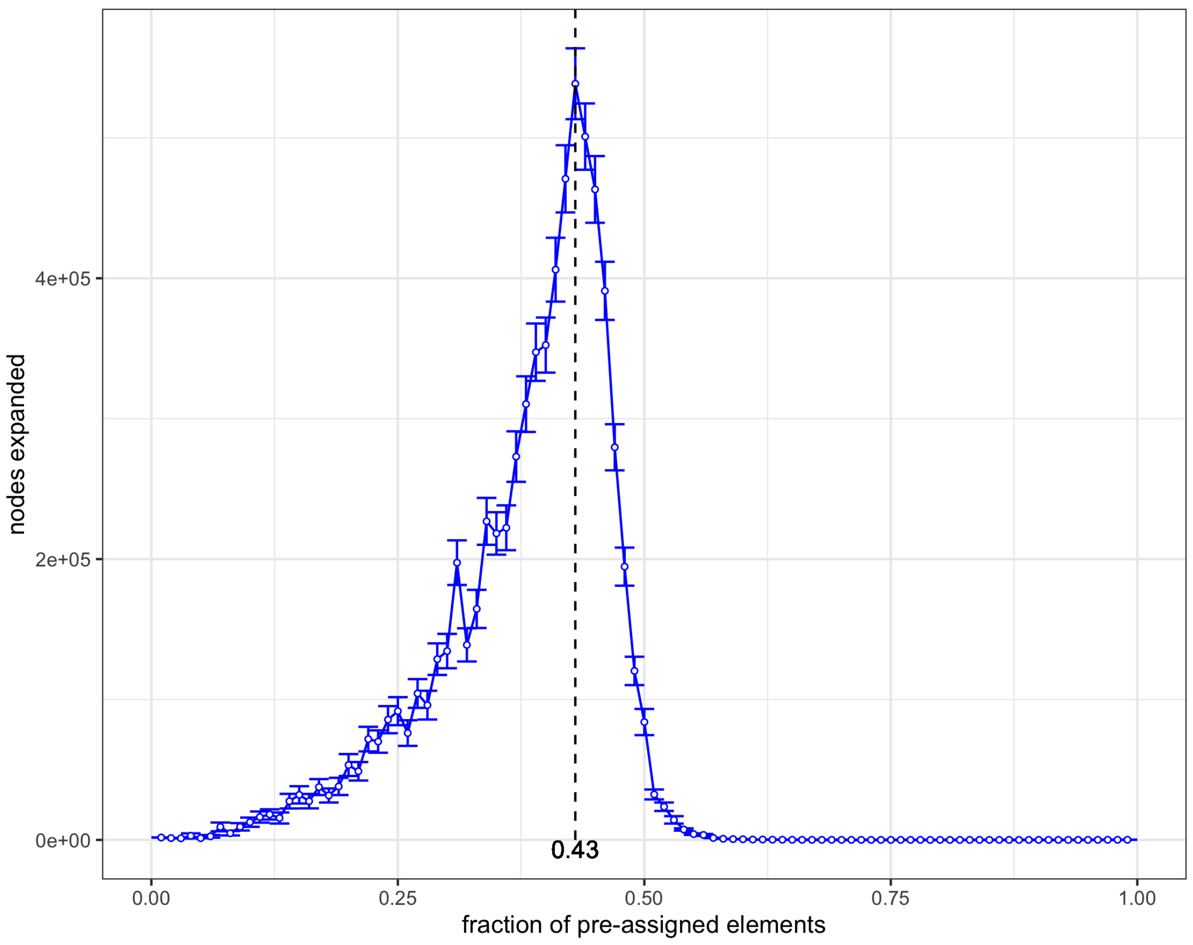
3 Empirical Results

Using R we plotted the following graph showing how the solvability of the algorithm varies with the fraction of pre-assigned elements.



We can see that instances with $\gamma < 0.43$ are under-constrained (the constraints are not very limiting and hence these instances have many solutions). Instances with $\gamma > 0.44$ are over-constrained (the constraints are very limiting and hence these instances do not have any solution).

We also plotted the following graph showing how the number of nodes expanded varies with the fraction of pre-assigned elements.



The peak of difficulty coincides with the transition between these two regions. The maximum is around $\gamma = 0.43$.

4 Discussion

As we can see the critically-constrained instances are found for $0.43 < \gamma < 0.44$. These instances may or may not have a solution. If they do, they may have very few solutions, hence why they are so time-consuming. If they don't, going through the search space and finding out that there is no solution can also explain why they are so time-consuming.

As a last comment, since Latin squares can be regarded as a simplified version of Sudoku it'd be interesting to study this same problem using Sudokus intead of Latin Squares and compare the results.