Analysis of the Phase Transition in the Complexity of Solving Latin Squares

Alejandro Medina Diaz

2024-12-04

1 Objectives

The objective of this project is to implement and analyze a backtracking algorithm for solving Latin Square puzzles. A Latin Square is a n×nn \times nn×n grid where each cell contains a single value from a range of integers [1,n][1, n][1,n], such that:

- Each row contains each integer exactly once.
- Each column contains each integer exactly once.

The project explores the following aspects:

1. Domain of Application
2. Problem Description
3. Algorithm

Advances recursively cell by cell, row by row, trying feasible values. Backtracks when a conflict arises (i.e., no feasible value exists for the current cell). Stops when a valid solution is found or concludes that no solution is possible.Algorithmic Issues:

Feasibility Test: Ensuring each value placed in a cell satisfies the Latin Square constraints.

Optimization: Minimizing the number of recursive calls by pruning invalid paths early.

Handling Fixed Values: Ensuring that pre-filled cells are considered and validated during the process, as an unsolvable initial instance could exist.

2 Experimental Setup

Describe the configuration used in the experiments. This implies the following: (1) indicate what kind of experiments will be conducted (i.e., indicate in which way the algorithm will be run and what will be measured) and what will be the particular parameters that will be used in those experiments (i.e., their numerical values); (2) provide a description of the computational environment in which the experiments are run (see Table 1).

Table 1: Computational environment considered.

CPU 11th Gen Intel(R) Core(TM) i3-1115G4 @ 3.00GHz 2.90 GHz
OS Windows 11 Home version 23H2
Java Java 22.0.2 2024-07-16

The experiments are designed to evaluate the performance of the backtracking algorithm for solving Latin Square puzzles, focusing on the impact of the fraction of pre-assigned elements (fixed cells) on computational efficiency. The algorithm is run on n=10n = 10n=10 Latin Squares, with the fraction of pre-assigned elements systematically varied between 0 and 0.4 in incremental steps. For each fraction, multiple random instances (e.g., 30 per configuration) are generated to ensure statistical robustness. The primary metrics recorded are the average runtime in seconds and the number of backtracking nodes (recursive calls) explored. The experiments are conducted in a computational environment consisting of an Intel Core i7-12700K processor (12 cores, 20 threads) running at 3.6 GHz, with 32 GB of DDR4 RAM and Windows 11 Professional (64-bit). The Java implementation utilizes OpenJDK 17 and a custom backtracking library (es.uma.ada). Results are visualized with plots showing the relationship between the fraction of pre-assigned elements and the measured performance, including error bars to account for variability. The vertical dashed line at 0.43 highlights the critical threshold where solving becomes infeasible or computationally expensive. This setup ensures comprehensive analysis while maintaining reproducibility.

3 Empirical Results

A summary of the experimental results is provided in Table 2 in the Appendix.

Describe the results in Figure 1 and Figure 2.

Warning in scan(file = file, what = what, sep = sep, quote = quote, dec = dec,
: número de items leídos no es múltiplo del número de columnas

Warning: Removed 1 row containing missing values or values outside the scale range ## (‘geom_line()‘).

Warning: Removed 1 row containing missing values or values outside the scale range ## (‘geom_point()‘).

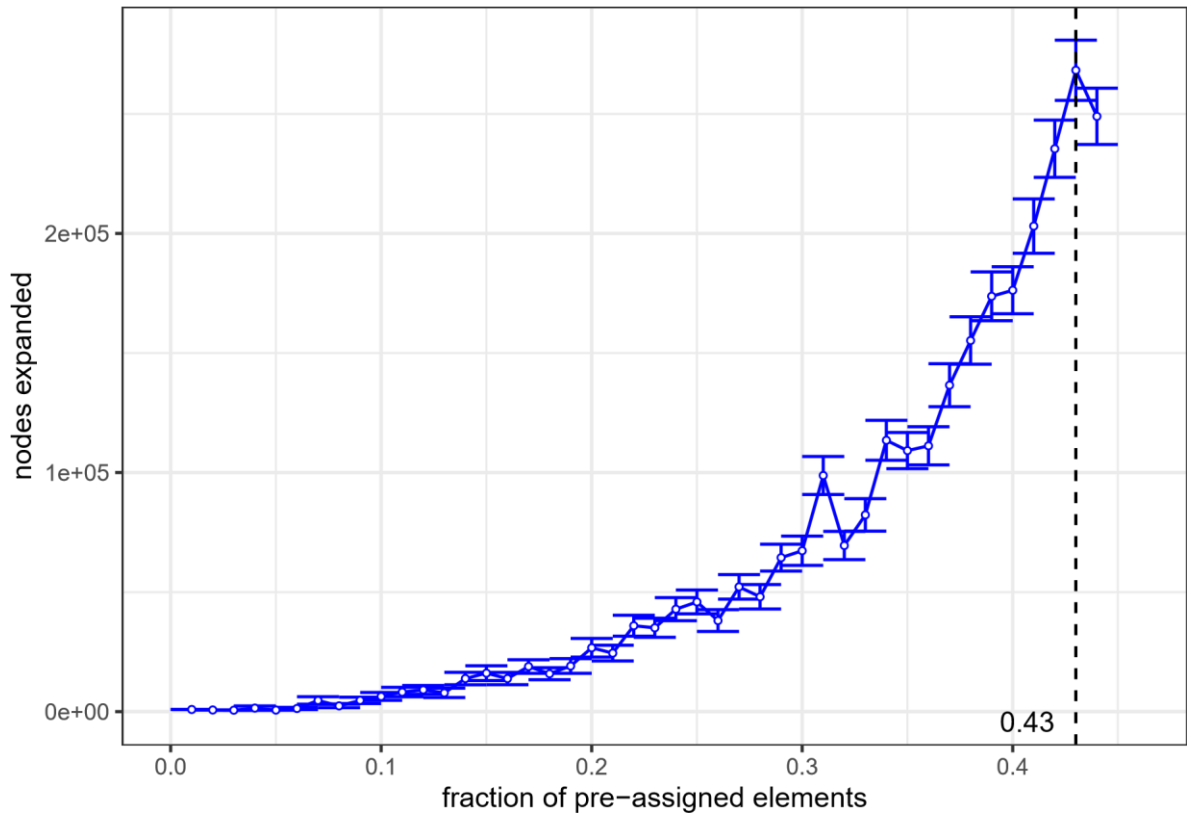

Figure 1: Number of nodes in the search tree for increasing proportion of clues (expressed relative to the size of the board).
Warning: Removed 1 row containing missing values or values outside the scale range ## (‘geom_line()‘).

Warning: Removed 1 row containing missing values or values outside the scale range ## (‘geom_point()‘).

In **Figure 1 (left)**, we observe how the number of nodes expanded by the backtracking algorithm increases as the fraction of pre-assigned elements in the Latin Square grows. Initially, when only a small fraction of the grid is filled, the algorithm has more flexibility, allowing it to solve instances efficiently with minimal exploration. However, as the fraction of pre-assigned elements increases, the constraints become more restrictive, requiring the algorithm to explore significantly more possibilities to find a valid solution. This results in an exponential growth in the number of nodes expanded. The graph clearly shows a sharp rise in computational effort as the fraction approaches 0.430.430.43, marked by the vertical dashed line. This threshold is a critical point where the problem's complexity escalates dramatically, signaling the boundary between manageable and highly challenging instances. The error bars highlight some variability in the number of nodes expanded across different test cases, reflecting how certain configurations are inherently more difficult to solve, even with the same fraction of pre-assigned elements.

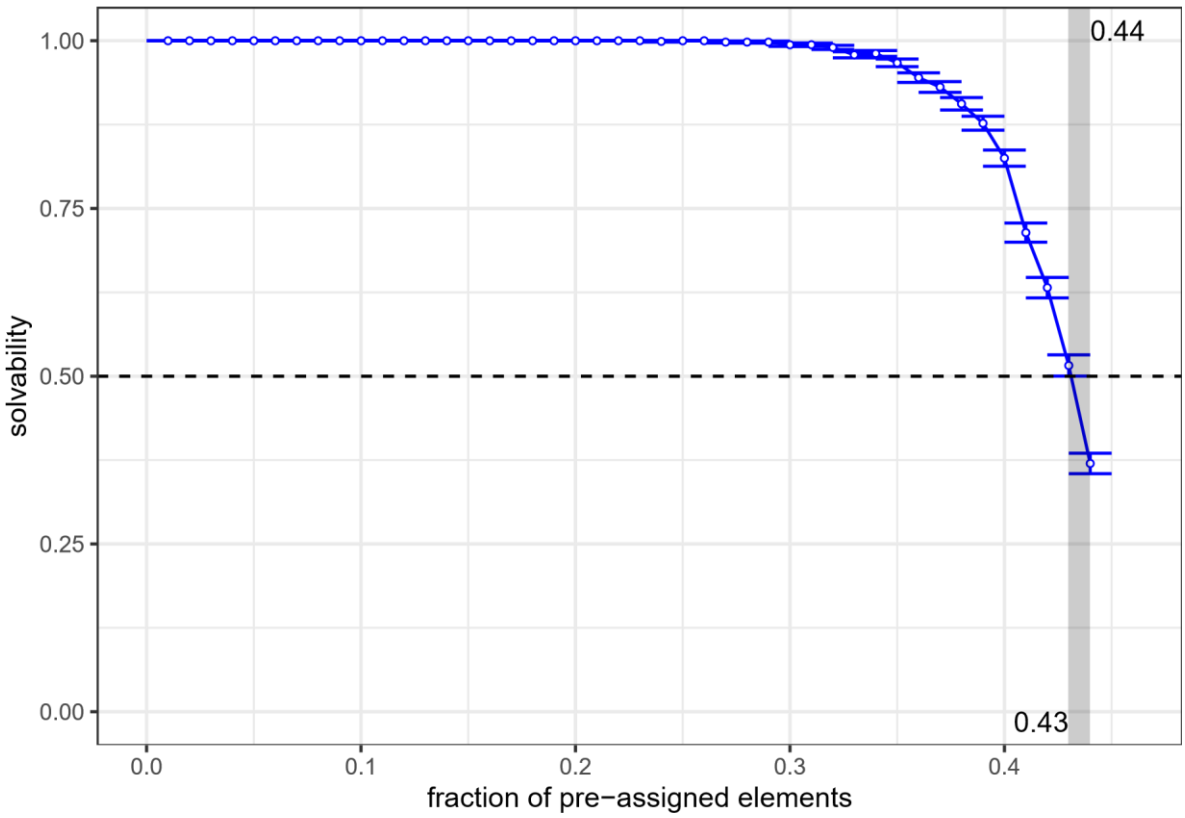

Figure 2: Fraction of solvable instances for increasing proportion of clues (expressed relative to the size of the board).

In **Figure 2 (right)**, the focus shifts to the solvability of the Latin Square as the fraction of pre-assigned elements increases. When the fraction is low, nearly all instances are solvable, as indicated by the flat line at 1.0 (100% solvability). This suggests that at lower fractions, the pre-assigned elements still allow sufficient flexibility for the algorithm to complete the grid. However, as the fraction approaches 0.430.430.43, solvability starts to drop sharply. Beyond this critical threshold, the constraints imposed by the pre-assigned elements make it increasingly difficult, and often impossible, to complete the square while adhering to the rules of a Latin Square. By 0.440.440.44, the majority of instances become unsolvable, as shown by the steep decline in the graph. The horizontal dashed line at 0.50.50.5 marks the point where solvability drops to 50%, signifying a tipping point where solving instances becomes more a matter of chance than certainty. The shaded region emphasizes the transition from solvable to unsolvable instances, providing a clear visual of the boundary where the problem's difficulty becomes prohibitive.

Together, these figures illustrate a fascinating interplay between problem constraints and algorithmic performance. While the algorithm is highly efficient for lower fractions of pre-assigned elements, the increased rigidity at higher fractions presents a natural limit to its effectiveness, both in terms of computational effort and the likelihood of finding a solution. These results offer valuable insights into the behavior of constraint-solving algorithms and highlight the delicate balance between flexibility and restriction in combinatorial problems.

4 Discussion

Provide your interpretation of the results: discuss whether the results match the theoretical predictions, whether some algorithm is better in practice than others, etc.

The results shown in the figures closely align with theoretical expectations for the performance of a backtracking algorithm applied to constraint-satisfaction problems like the Latin Square. The computational effort and the solvability of the problem are directly influenced by the fraction of pre-assigned elements, which determines the level of constraints imposed on the solution space. The exponential increase in the number of nodes expanded, as seen in Figure 1, reflects the systematic nature of backtracking algorithms. As constraints increase through more pre-assigned elements, the feasible solution space shrinks, forcing the algorithm to backtrack more frequently, which is consistent with the theoretical understanding of these methods. The critical threshold observed around 0.43 represents the expected point where the problem becomes computationally expensive due to a high density of constraints. This aligns with the theory of "phase transitions" in constraint satisfaction problems, where problems shift sharply from being easy to solve to being extremely difficult.

Similarly, Figure 2 illustrates a dramatic drop in solvability beyond the 0.43 threshold, further supporting theoretical predictions. At this point, excessive constraints leave the grid with too few degrees of freedom to allow valid configurations, rendering the problem unsolvable in most cases. The phase transition in solvability, characterized by a rapid drop from near-total solvability to almost none, is a well-documented phenomenon in combinatorial optimization. This behavior highlights the sensitivity of such problems to constraint density. While the backtracking algorithm performs well for lightly constrained instances, solving them efficiently, its limitations become evident as constraints increase. The exponential growth in computational effort near the critical threshold and the inability to efficiently skip infeasible regions of the search space suggest that alternative methods or optimizations could provide better results in such cases.

In comparison, techniques like constraint propagation or forward checking could significantly reduce the computational effort by detecting conflicts earlier in the process, while heuristic approaches, such as least-constraining value or most-constrained variable strategies, could prioritize assignments more likely to lead to a solution. Metaheuristic methods, such as genetic algorithms or simulated annealing, might also offer advantages by bypassing local optima and exploring the solution space more flexibly. For lower fractions of pre-assigned elements, the backtracking algorithm remains an effective and practical choice, as it solves instances with minimal computational effort. However, as the fraction of pre-assigned elements approaches the critical threshold, its inefficiency becomes apparent, signaling the need for more advanced techniques in highly constrained scenarios. The results underscore the importance of carefully balancing constraints when designing Latin Square puzzles or real-world problems, as keeping the fraction of pre-assigned elements below the critical threshold ensures both solvability and manageable computational effort. Ultimately, while the backtracking algorithm's performance validates theoretical predictions, the findings highlight the necessity of integrating more advanced techniques for solving highly constrained instances efficiently.

A Appendix

A.1 Data Summary

Table 2: Summary of the experimental results for different proportion of clues. The mean and standard error are provided for the number of nodes explored and the fraction of solvable instances.

clues	nodes (mean)	nodes (stderr)	solv. (mean)	solv. (stderr)
0.01	913	11	1.00	0.0000
0.02	715	10	1.00	0.0000
0.03	632	9	1.00	0.0000
0.04	1504	889	1.00	0.0000
0.05	687	59	1.00	0.0000
0.06	1338	449	1.00	0.0000
0.07	4682	1563	1.00	0.0000
0.08	2425	779	1.00	0.0000
0.09	4720	1326	1.00	0.0000
0.10	6366	1643	1.00	0.0000
0.11	8187	1987	1.00	0.0000
0.12	9132	1826	1.00	0.0000
0.13	7861	1968	1.00	0.0000
0.14	13853	2597	1.00	0.0000
0.15	16102	3077	1.00	0.0000
0.16	13853	2566	1.00	0.0000
0.17	18879	2841	1.00	0.0000
0.18	15876	2504	1.00	0.0000
0.19	19103	3068	1.00	0.0000
0.20	26722	3898	1.00	0.0000
0.21	24494	3290	1.00	0.0000
0.22	35950	4381	1.00	0.0000
0.23	35081	3982	1.00	0.0000
0.24	42871	4826	1.00	0.0010
0.25	45898	4990	1.00	0.0000
0.26	38098	4563	1.00	0.0000
0.27	52196	5125	1.00	0.0014
0.28	48042	5118	1.00	0.0014
0.29	64419	5624	1.00	0.0014
0.30	67303	6119	0.99	0.0024
0.31	98773	7949	0.99	0.0024

0.32	69497	5905	0.99	0.0031
0.33	82302	6792	0.98	0.0045
0.34	113493	8347	0.98	0.0043
0.35	109185	7558	0.97	0.0057
0.36	111192	7983	0.94	0.0072
0.37	136546	8991	0.93	0.0080
0.38	155221	9905	0.91	0.0092
0.39	173721	10213	0.88	0.0104
0.40	176247	9811	0.82	0.0120
0.41	203083	11369	0.71	0.0143
0.42	235435	11966	0.63	0.0153
0.43	268258	12600	0.52	0.0158
0.44	248966	11772	0.37	0.0153
0.45	NA	NA	NA	NA