Bangladesh University of Engineering and Technology

Department of Computer Science and Engineering

Academic Year 2022 - 2023

CSE 318
-Artificial Intelligence Sessional-

Offline No. 2
Constraint Satisfaction Problem (Latin Square)

Name: Anwarul Bashir Shuaib

Roll: 1805010

Section: A1

Date of Submission: January 09, 2023

Evaluating Heuristics for Solving the Latin Square Problem

Abstract

In this report, we present a constraint satisfaction approach to solving the Latin square problem. We implemented simple backtracking and backtracking with forward checking, and tested five different heuristics to guide the search. We also included the least constraining value ordering heuristic in our comparison. The performance of the different heuristics was measured and compared. The best performance was achieved with backtracking and forward checking using the heuristic of selecting variables by minimum domain size.

Problem Description

The Latin square problem is a combinatorial problem where the goal is to fill in a grid of size n×n with n different symbols (usually the numbers 1 to n), such that no symbol appears more than once in any row or column. It is a generalization of the Sudoku puzzle, and is often used as a test case for constraint satisfaction algorithms. The problem can be represented as a constraint satisfaction problem, where each cell in the grid is a variable, and the constraints specify that each row and column must contain each symbol exactly once.

$$\begin{bmatrix} 5 & 2 & 3 & 4 & 1 \\ 1 & 4 & 5 & 3 & 2 \\ 3 & 1 & 4 & 2 & 5 \\ 2 & 5 & 1 & 4 & 3 \\ 4 & 3 & 2 & 1 & 5 \end{bmatrix}$$

Heuristics

1. `VAH1`: Select the variable with the smallest domain. This heuristic aims to choose variables that are most constrained, as they have fewer options for legal values.

2. `VAH2`: Select the variable with the maximum degree to unassigned variables. This heuristic aims to choose variables that are connected to the largest number of unassigned variables, as this may allow for more forward pruning.

3. `VAH3`: Select the variable using VAH1, with ties broken using VAH2. This heuristic combines the two previous heuristics, prioritizing variables with small domains but also considering the number of connections to unassigned variables.

4. `VAH4`: Select the variable that minimizes the ratio of VAH1 to VAH2. This heuristic aims to balance the conflicting goals of selecting variables with small domains and selecting variables with many connections to unassigned variables.

5. `VAH5`: Select a random unassigned variable. This heuristic does not consider any specific properties of the variables or the state of the search. It simply chooses a variable at random.

Simulation Report

We conducted several tests to compare the performance of different heuristics and solvers. The results are summarized in the following table, which shows the number of nodes explored, the number of backtracks, the time taken, and the solver used for each test case.

Table 1: Summary of the Simulation

Test Case	Heuristic	Explored	Backtracks	Time	Solver
d-10-01	VAH1	24379	19088	103	FC
	VAH2	93671906	115662652	106584	FC
	VAH3	665	296	3	FC
	VAH4	1002	382	4	FC
	VAH5	120250533	120653001	147631	FC
	VAH1	450473	1096627	299	BT
	VAH2	2928097	7424685	1815	BT
	VAH3	217384	588202	137	BT
	VAH4	217384	588202	144	BT
	VAH5	69455201	156923151	51293	BT
d-10-06	VAH1	451284	362926	498	FC
	VAH2	2214587	3129324	2425	FC
	VAH3	84364	50933	88	FC
	VAH4	83083	51488	87	FC
	VAH5	7044944	8096524	8360	FC
	VAH1	119067	355076	71	BT
	VAH2	200637	519527	122	BT
	VAH3	20774	61923	15	BT
	VAH4	20769	61906	17	BT
	VAH5	578643	1485599	435	BT
d-10-07	VAH1	143859	105117	156	FC
	VAH2	46987974	55151903	52095	FC
	VAH3	10424	6717	15	FC
	VAH4	15372	11173	17	FC
	VAH5	45195402	45973700	53809	FC
	VAH1	100189	267501	68	BT
	VAH2	3467200	8341173	2145	BT
	VAH3	48868	110819	32	BT
	VAH4	48868	110819	36	BT
	VAH5	135783001	304518847	99503	BT

Table 2: Summary of the Simulation (Continued)

Test Case	Heuristic	Explored	Backtracks	Time	Solver
d-10-08	VAH1	524365	481008	582	FC
	VAH2	2219501	3071439	2475	FC
	VAH3	2318987	1815075	2438	FC
	VAH4	2322561	1840567	2457	FC
	VAH5	255641	302907	301	FC
	VAH1	424230	1230159	261	BT
	VAH2	270678388	679271441	151577	BT
	VAH3	30045815	86260528	20177	BT
	VAH4	30045815	86260528	20751	BT
	VAH5	217316866	536891604	164085	BT
d-10-09	VAH1	30875	27011	33	FC
	VAH2	2387247067	3868210293	2641196	FC
	VAH3	1328349	1004392	1371	FC
	VAH4	1711481	1319629	1796	FC
	VAH5	569883266	749575720	668438	FC
	VAH1	1225880	3680281	703	BT
	VAH2	203984	590622	124	BT
	VAH3	16177286	48534935	10209	BT
	VAH4	16287885	49055830	10547	BT
	VAH5	72185277	211931801	56123	BT

Machine Configuration

- `CPU`: Intel(R) Xeon(R) Platinum 8272CL CPU @ 2.60GHz
- `Cache Size`: 36608 KB
- `Java JDK`: 11.0.14.1
- `Operating System`: Ubuntu 20.04.5 LTS (Focal Fossa)
- `System Memory`: 4GB

Visualization

In order to visualize the performance of the different heuristics, we plotted three clustered stacked bar charts for the time taken, explored nodes count, and backtracked nodes count for the simulations. These charts provide a clear comparison of the relative effectiveness of the heuristics across different metrics. **Due to the large variations in the values for these metrics, we took the logarithm of the values before plotting the charts**. By analyzing the results of these charts, we can gain insights into the strengths and weaknesses of each heuristic and how they compare to one another.

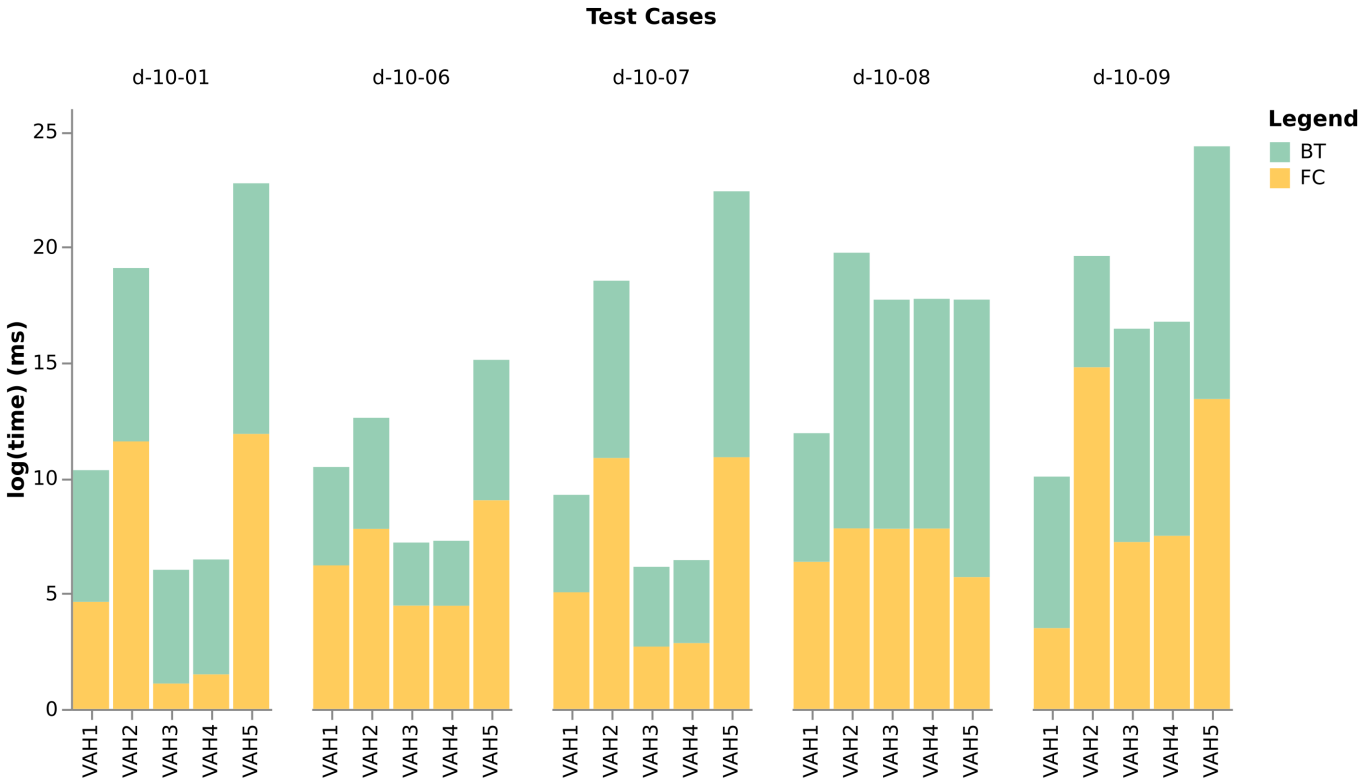

Figure 1: Comparison of the time taken using different heuristics

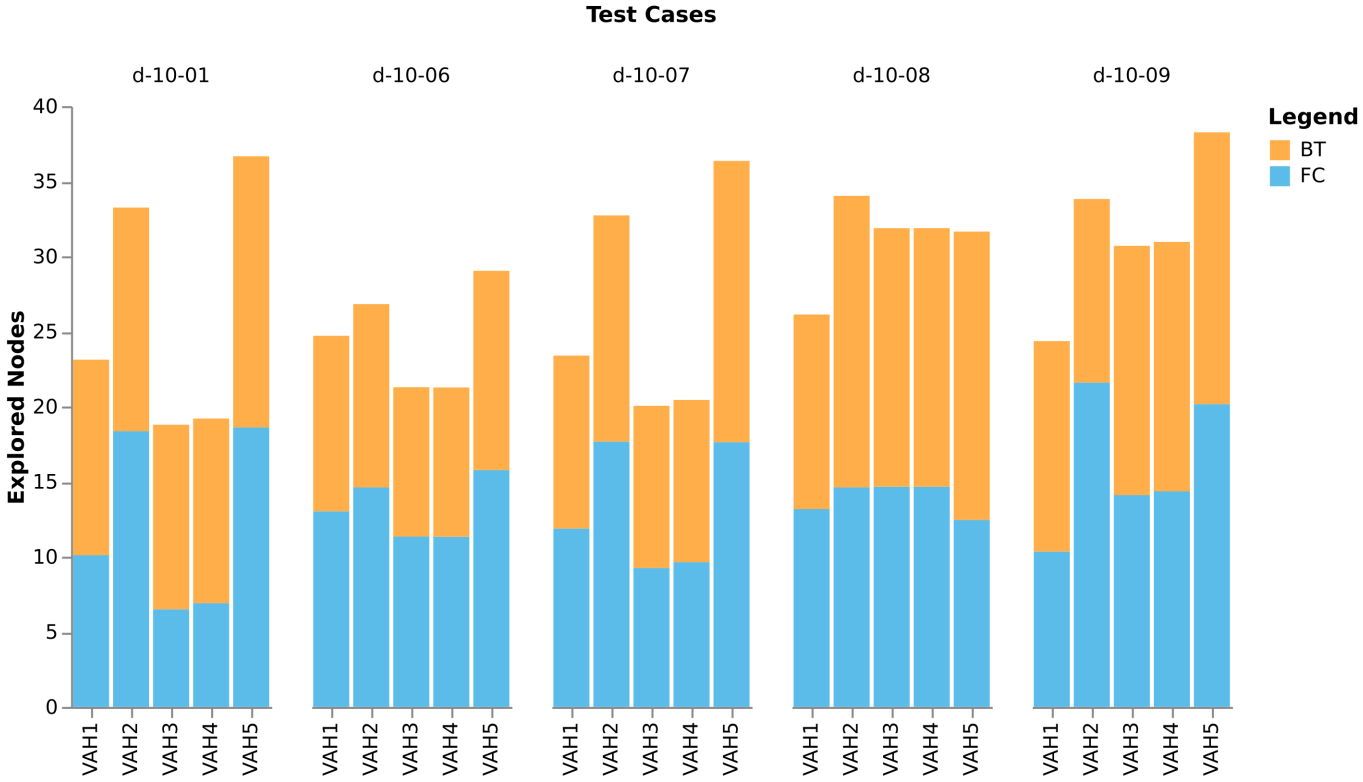

Figure 2: Comparison of the number of explored nodes using different heuristics

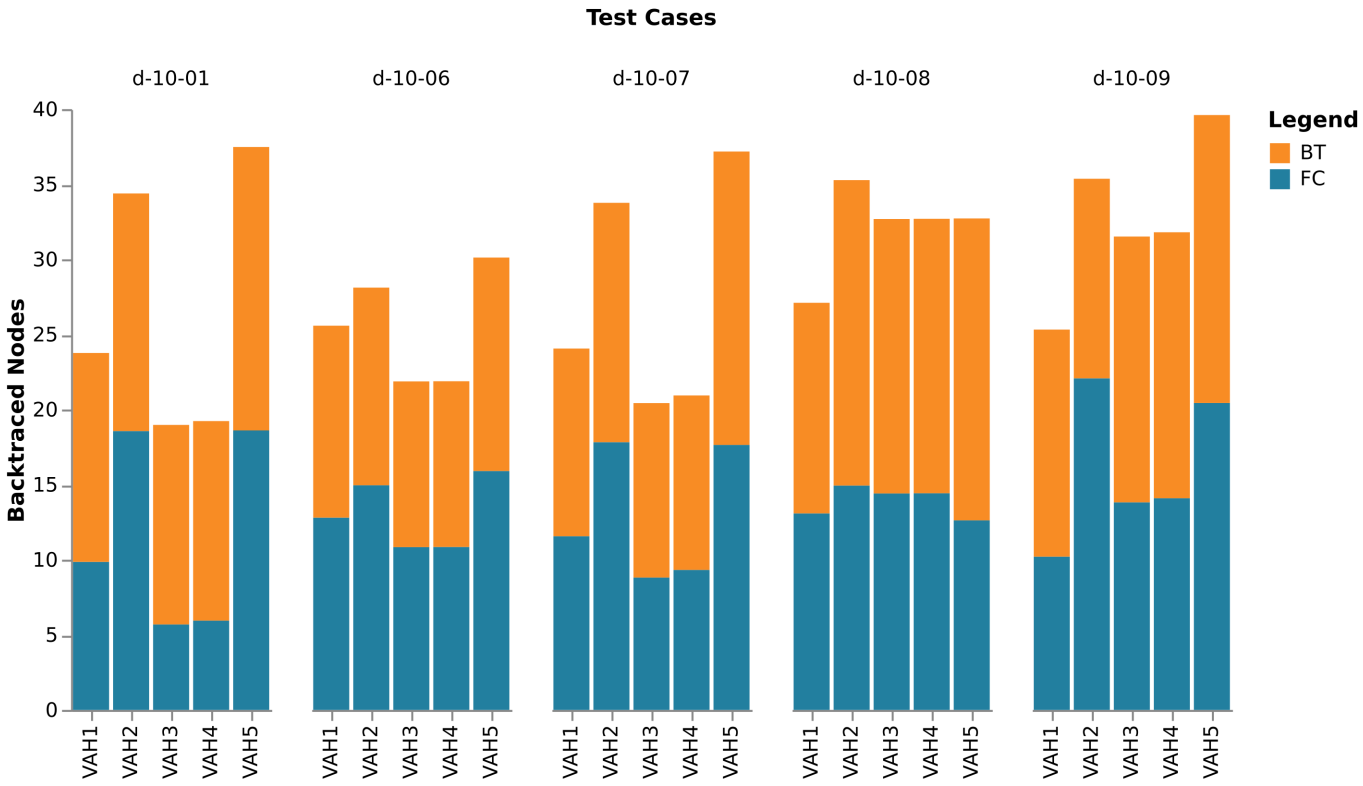

Figure 3: Comparison of the number of backtracked nodes using different heuristics

Justification for Variable Ordering Heuristics

Based on the results of our simulations, it appears that the VAH3 heuristic performed the best, followed by VAH1. The VAH5 heuristic performed the worst, with VAH2 also performing poorly.

One possible reason for these results is that the VAH3 heuristic is able to effectively balance the conflicting goals of selecting variables with small domains (as in VAH1) and selecting variables with many connections to unassigned variables (as in VAH2). By combining these two factors, VAH3 is able to make more informed choices about which variables to assign first, leading to faster convergence and fewer explored and backtracked nodes.
The VAH1 heuristic, which simply selects the variable with the smallest domain, performed well but was outperformed by VAH3. This suggests that while choosing variables with small domains can be effective, additional considerations such as the number of connections to unassigned variables can further improve performance.
On the other hand, the VAH5 heuristic, which selects a random unassigned variable, performed poorly compared to the other heuristics. This is likely because it does not consider any specific properties of the variables or the state of the search, leading to suboptimal choices and slower convergence.

Overall, the results of our simulations suggest that the VAH3 heuristic is the most effective for solving the Latin square problem using constraint satisfaction techniques, followed by VAH1. The VAH2 and VAH5 heuristics performed relatively poorly in comparison.

Justification for Value Ordering Heuristic

Selecting the least constraining value from the domain for each variable is a common heuristic used in constraint satisfaction problems. This heuristic aims to select values for variables that will have the least impact on the remaining variables and constraints in the problem.

There are several benefits to using the least constraining value heuristic. First, it can help to reduce the number of backtracks and improve the overall efficiency of the search. By choosing values that are less likely to cause conflicts with other variables or constraints, the algorithm is able to make progress more quickly and avoid the need to backtrack as frequently.
Second, the least constraining value heuristic can also help to improve the quality of the solution. By choosing values that are less likely to cause conflicts, the algorithm is more likely to find a solution that satisfies all of the constraints in the problem.

Overall, the least constraining value heuristic is a useful tool for improving the efficiency and effectiveness of constraint satisfaction algorithms. It can be particularly useful in problems with many variables and constraints, where the search space is large and the risk of backtracking is high.

Conclusion

In conclusion, the Latin square problem was solved using constraint satisfaction techniques, with two algorithms (simple backtracking and backtracking with forward checking) implemented and tested using five different heuristics. The best performance was achieved with backtracking with forward checking. The least constraining value ordering heuristic was also utilized. The results of the simulations showed that the VAH3 heuristic was the most effective, followed by VAH1. The VAH2 and VAH5 heuristics performed relatively poorly in comparison. These findings suggest that carefully selecting variables and values using informed heuristics can significantly improve the efficiency and effectiveness of constraint satisfaction algorithms for solving the Latin square problem.