

LATIN SQUARE COMPLETION PROBLEM SOLVER OFFLINE 2 (CSP)

Fardin Anam Aungon - 1805087 Jan 9, 2023

Introduction

A Latin Square is an $n * n$ matrix that contains unique numbers from 1 to n along each column and row. In other words, each cell will have a value from 1 to n which is unique to all the other cells of its corresponding row and column. In Latin Square completion problem, an $n * n$ incomplete Latin Square is given initially. Our task is to complete the square.

We have written a program to find solutions to the Latin Square Completion problem using CSP. Here, we consider every cell as a variable. Every cell has a domain of integers from 1 to n . And every cell has a constraint that it can't have the value equal to any of the assigned cells that are present in its row and column. While our main goal is to find a solution for a given Latin Square, we also have to make it as efficient as possible. There are a number of heuristics on selecting a variable and its value. Selecting one rather than the other can make the process faster. So, finding out a good heuristic is necessary. Here, in this report, we have tried to figure out exactly that.

Value Order Heuristic

We have used **Least Constrained Value Heuristic** as our main value order heuristic to select a suitable value from the domain of a cell in our solver. This heuristic selects the value which is present in the domains of least number of unassigned cells that are in the corresponding row and column of the selected cell. The algorithm backtracks whenever it finds inconsistency with the assignments that have already been made. Inconsistency can occur whenever a cell has an empty domain and it has no value assigned. Or it can also occur whenever no value in the domain of a cell can be chosen to be set because all the

values in the domain have already been assigned to a cell in the corresponding row or column. It would be better if we could intelligently choose a value that has the least possibility of creating such an issue. If we always choose the value of a cell that is not present in most of the cells in the corresponding row or column, it will have the least possibility to create such a conflict that we have just discussed. This heuristic helps the solver to solve the Latin Square Completion Problem visiting lesser nodes or with lesser backtracks. But, it is required to calculate the degrees and then to sort the values after selecting every cell in the csp to implement least constraint value heuristic.

On the other hand, we have also tried selecting the values with a simple for each loop of the domain set. We have named it **No Order Heuristic**. Here, no overhead of calculation is required.

So, even though least constraint value heuristic requires to visit lesser nodes, time to number of nodes ratio becomes less than our no order heuristic. If we could come up with a better algorithm to calculate least constraint value heuristic, it would surely outperform the no order heuristic.

Variable Order Heuristics

We have used five variable order heuristics.

1. **Minimum Remaining Value (VAH1):** The variable chosen is the one with the smallest domain.
2. **Maximum Degree to Unassigned Variables (VAH2):** The variable chosen is the one with the maximum degree to unassigned variables. Also, called max-forward-degree.
3. **Minimum Remaining Value + Maximum Degree to Unassigned Variables (VAH3):** The variable chosen by VAH1, Ties are broken by VAH2.
4. **Minimum VAH1/VAH2 (VAH4):** The variable chosen is the one that minimizes the $VAH1 / VAH2$.
5. **Random Unassigned Variable (VAH5):** A random unassigned variable is chosen.

We have tried all possible combinations of heuristics with backtracking and forward checking with our given dataset. Here are the results of the tests:

Test Results For Least Constraint Value Heuristic

Value Order Heuristic	dataset	backtrack/ forward check	variable order heuristic	time (ms)	No. of Basktracks	No. of Nodes
Least Constrained Value	d-10-01	Backtrack	vah1	7	123	180
			vah2	215011	114742628	114742685
			vah3	6	25	82
			vah4	5	19	76
			vah5	1197154	765290116	765290173
		Forward checking	vah1	8	111	180
			vah2	6738	1719577	3167439
			vah3	5	23	82
			vah4	5	18	76
			vah5	72	9954	17726
	d-10-06	Backtrack	vah1	5	0	57
			vah2	55776	33573439	33573496
			vah3	5	0	57
			vah4	4	0	57
			vah5	5842	4279604	4279661
		Forward checking	vah1	4	0	57
			vah2	3199	844315	1573011
			vah3	5	0	57
			vah4	5	0	57

			vah5	2652	698843	1258089
	d-10-07	Backtrack	vah1	9	203	260
			vah2	1286	725455	725512
			vah3	5	0	57
			vah4	4	0	57
			vah5	1453407	982426264	982426321
		Forward checking	vah1	9	187	260
			vah2	104	15988	28344
			vah3	5	0	57
			vah4	5	0	57
			vah5	713	189489	325571
	d-10-08	Backtrack	vah1	5	22	79
			vah2	6253475	3696600662	3696600719
			vah3	5	0	57
			vah4	11	314	371
			vah5	8302	5943444	5943501
		Forward checking	vah1	5	20	79
			vah2	304586	81402789	15393388
			vah3	7	0	57
			vah4	10	277	371
			vah5	185	39889	69747
			vah1	4	9	66

	d-10-09	Backtrack	vah2	2967	1814800	1814857
			vah3	6	18	75
			vah4	4	0	57
			vah5	2264406	221899981	221900038
		Forward checking	vah1	5	8	66
			vah2	323	73184	134043
			vah3	6	16	75
			vah4	5	0	57
			vah5	4021	1167681	2036833
	d-15-01	Backtrack	vah1	266	41665	41771
			vah2	*	*	*
			vah3	22	1124	1230
			vah4	1445	268484	268590
			vah5	*	*	*
		Forward checking	vah1	267	37806	41771
			vah2	*	*	*
			vah3	23	992	1230
			vah4	1424	241033	268590
			vah5	*	*	*

Test Results For No Order Heuristic

Value Order Heuristic	dataset	backtrack/forward check	variable order heuristic	time (ms)	No. of Basktracks	No. of Nodes
No order	d-10-01	Backtrack	vah1	4	151	208
			vah2	516494	1060855659	1060855716
			vah3	5	132	189
			vah4	2	0	57
			vah5	839233	2375031830	2375031887
		Forward checking	vah1	5	136	208
			vah2	15429	15324876	28677475
			vah3	4	117	189
			vah4	3	0	57
			vah5	34	11464	19940
	d-10-06	Backtrack	vah1	1	0	57
			vah2	95474	200860854	200860911
			vah3	3	0	57
			vah4	2	0	57
			vah5	15323	39887213	39887270
		Forward checking	vah1	2	0	57
			vah2	4043	3723906	6936956
			vah3	3	0	57
			vah4	2	0	57

			vah5	195	157360	273096
	d-10-07	Backtrack	vah1	11	1498	1555
			vah2	131295	299113217	299113274
			vah3	2	4	61
			vah4	5	382	439
			vah5	664809	1873765411	1873765468
		Forward checking	vah1	13	1364	1555
			vah2	2819	2511432	4571958
			vah3	4	2	61
			vah4	6	350	439
			vah5	652	603712	1039371
	d-10-08	Backtrack	vah1	11	983	1040
			vah2	203842	409798183	409798240
			vah3	7	128	185
			vah4	3	77	134
			vah5	200900	551506471	551506728
		Forward checking	vah1	10	881	1040
			vah2	7115	6639045	12162652
			vah3	8	107	185
			vah4	3	65	134
			vah5	411	592005	592062
			vah1	3	0	57

	d-10-09	Backtrack	vah2	90396	192994992	192995049
			vah3	3	0	57
			vah4	66	35518	35575
			vah5	*	*	*
		Forward checking	vah1	2	0	57
			vah2	4032	4163973	7684384
			vah3	3	0	57
			vah4	68	31173	35575
			vah5	536	811435	811492
	d-15-01	Backtrack	vah1	1287	932566	932672
			vah2	*	*	*
			vah3	184	101816	101922
			vah4	747	597476	597582
			vah5	*	*	*
		Forward checking	vah1	1332	846384	932672
			vah2	*	*	*
			vah3	197	89287	101922
			vah4	768	535027	597582
			vah5	*	*	*

- *Green rows indicate the best scheme for a solver*
- *Yellow rows indicate the second best scheme for a solver*

Table: Benchmark of Latin Square Completion Problem solver with different heuristics

Conclusion:

The performance of a scheme depends vastly on the test data. But, in most of the cases VAH4 with forward checking performs better.

VAH4 uses the ratio of VAH1 and VAH2 where VAH1 selects the variable with the smallest domain and VAH2 selects the variable with the maximum forward degree. Choosing the cell with the smallest domain saves the time taken to find a suitable value. Again, as a cell with the smallest domain has the least possible children in the DFS, it also creates the chance of traveling fewer nodes. So, it minimizes the time and number of traversed nodes. On the other hand, a cell having the maximum degree to unassigned variable has the highest chance of creating conflict, hence has the highest chance of backtracking. Which means, it maximizes the time and number of traversed nodes. So, using VAH2 in inverse order can minimize time and number of traversed nodes.

Forward tracing detects inconsistency faster than simple backtracking because, after every assignment it checks if any of the domains of unassigned cells becomes empty or not. If so, it backtracks immediately. So, for any selected heuristic, forward checking performs better, if not the same as simple backtracking.

With all the discussions above, and the data we have found from running the tests, we can conclude that VAH4 with forward tracing performs the best in our solver.