Report on

Constraint Satisfaction Problem

Course ID: CSE 318

Course Title: Artificial Intelligence Sessional

Submitted By

Shehabul Islam Sawraz

1805088

Date of Submission: 09-01-2023

Introduction:

Latin Squares are N*N Matrices where no row or column can have any of the numbers from 1 to N more than once. Latin Square Completion (**LSC**) problems are where a partially filled Latin Square would be given and the rest of the cells would be left for the system to fill according to the constraints.

LSC as CSP:

LSC problems can be formulated as Constraint Satisfaction Problems (CSP), as we have to rely on Heuristics rather than definitive approaches.

Constraints or Arcs: No row or column can have the same number more than once.

Nodes or Variables: The cells.

Values: Numbers from 1 to N.

Domain: Subset of {1 to N} depending on the current values of that row and column the variable is in.

Variable Order Heuristic:

VAH1: The variable chosen is the one with the smallest domain.

VAH2: The variable chosen is the one with the maximum degree to unassigned variables. Also, called max-forward-degree.

VAH3: The variable chosen by VAH1. Ties are broken by VAH2.

VAH4: The variable chosen is the one that minimizes the VAH1 or VAH2.

VAH5: A random unassigned variable is chosen.

Value Order Heuristic: *Least Constraining Value First*

After a variable has been selected using a Variable-Order-Heuristic, to decide on the order in which to examine its values, “**Least-Constraining- Value**” heuristic has been chosen. It prefers the value that rules out the fewest choices for the neighboring variables (**in this problem, the neighboring variables are the ones that are either in same row or same column**) in the constraint graph. As it tries to leave the **maximum flexibility for subsequent variable assignments**, it is more likely to go through the path which will provide a solution faster. As we need only one solution in this assignment, this heuristic provides better performance than choosing the values from the domain of a variable randomly or serially. The runtime decreases by a factor of around 10 for this heuristic than using the first value in the domain. If we need to enumerate all solutions rather than just fine one, then value ordering would be irrelevant.

Results:

We ran 6 different data with our implementation of Backtracking and Forward Checking with all the Variable Heuristics stated above.

Test Case	Method	Variable Heuristic	Nodes	Backtracks	Runtime (ms)
d-10-01	Backtracking	VAH1	291	233	0
		VAH2	595430785	595430727	782241
		VAH3	77	19	1
		VAH4	111	53	2
		VAH5	116527304	116527246	148233
	Forward Checking	VAH1	271	213	0
		VAH2	749199	749141	1612
		VAH3	76	18	1
		VAH4	107	49	1
		VAH5	229180	229122	484
d-10-06	Backtracking	VAH1	58	0	0
		VAH2	473513336	473513278	609926
		VAH3	58	0	0
		VAH4	58	0	0
		VAH5	4590184	4590242	5775
	Forward Checking	VAH1	58	0	0
		VAH2	3674242	3674184	7309
		VAH3	58	0	0
		VAH4	58	0	0
		VAH5	78190	78132	156

d-10-07	Backtracking	VAH1	268	210	0
		VAH2	26641951	26641893	32356
		VAH3	58	0	0
		VAH4	58	0	0
		VAH5	557591138	557591080	712737
	Forward Checking	VAH1	246	188	0
		VAH2	32971	32913	62
		VAH3	58	0	0
		VAH4	58	0	0
		VAH5	8257	8199	16
d-10-08	Backtracking	VAH1	102	44	0
		VAH2	*	*	*
		VAH3	382	324	1
		VAH4	536	478	3
		VAH5	641068367	641068309	814229
	Forward Checking	VAH1	99	41	0
		VAH2	45443578	45443520	101638
		VAH3	352	294	2
		VAH4	484	426	3
		VAH5	13446	13388	32
d-10-09	Backtracking	VAH1	67	9	0
		VAH2	13597542	13597484	17767
		VAH3	58	0	1
		VAH4	58	0	0
		VAH5	2628968	2628910	3610
	Forward Checking	VAH1	66	8	0
		VAH2	96171	96113	242

		VAH3	58	0	1
		VAH4	58	0	0
		VAH5	2004144	2004202	2977
d-15-01	Backtracking	VAH1	81883	81776	344
		VAH2	*	*	*
		VAH3	81027	80920	467
		VAH4	134173	134066	529
		VAH5	*	*	*
	Forward Checking	VAH1	74026	73919	267
		VAH2	*	*	*
		VAH3	72615	72508	347
		VAH4	118969	118862	402
		VAH5	*	*	*

NB: Tests that took more than 25 minutes are marked as *

Analysis:

To apply the heuristics given, a **modified Backtrack** is used, where the modification is that when a value of a variable is successfully assigned, then the domain of its neighboring variables are being updated.

The difference between the Forward Checking and the modified Backtrack is that in forward checking, after updating the domain of neighboring variables, if any domain becomes empty, then immediately backtracked from that node, whereas no such decision is made based on the size of the domain in the modified backtrack. The modified backtrack is done when the possible domain size of the chosen variable becomes 0.

Reason to use a Modified Backtrack is that, if a pure backtrack solver is used, then the heuristics will not get the updated domain. And so, it will work on the initial domain every time and this will not be able to provide the power of the heuristics and the number of visited node with be much bigger that we can’t get the solve in short time.

Observations:

From the data table above, it is clear that the **Forward Checking performs better than Backtrack** for all the test cases as it detects failure earlier and reduces the number of nodes in the search tree by pruning larger parts of the tree earlier. For example, for test case "**d-10-07**", the **Backtrack (BT)** with heuristic **VAH2** searches total **26641951** nodes where the **Forward Checking (FC)** searches only **32971** nodes (A factor of **800** improvement).

Now, if we compare the performance among the 5 Variable Order Heuristics, we see that if we use FC using VAH3 we get the best performance in terms of the number of nodes in almost all cases. But **using VAH3 has a higher overhead than VAH1**. When the size of the Latin Square increases, the runtime for FC with VAH1 becomes much lower than FC with VAH3. If we use the **number of nodes as the performance metric, FC with VAH3 is the best scheme**. Otherwise, if **runtime is our performance metric, we will get better performance with FC with VAH1.**

In VAH1 we take the variable with the smallest domain size. It chooses a variable that is most likely to cause a failure. Early detection of failure, reduces the number of nodes and runtime. In VAH3 we use this approach and to break ties if any we take the variable involved in the largest number of constraints with unassigned variables. It reduces the branching factor on future choices .

After VAH1 & VAH3, VAH4 gives better performance. As, VAH4 tries to combine heuristic VAH1 and VAH2 by minimizing the ratio. It gives similar performance to VAH3 in some test cases.

The VAH2 heuristic is mainly used to break tie of VAH1 heuristic. It alone does not provide much improvement in performance which is evident from the data table. Finally, the VAH5 heuristic picks variables randomly and its performance changes in different runs but on average, it provides poor performance than VAH1, VAH3 and VAH4.

Conclusion:

We observed that Forward Checking usually works better than just Backtracking and some of the Heuristics works better with FC and some with BT.

Lastly, we can see that clearly Forward Checking with VAH3 outperforms all the other combinations