Report on

Constraint Satisfaction Problem

Roll **1805093**

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. LSC or Latin Square Completion 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. One of the reasons is LSC is NP Complete.

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 / VAH2.

VAH5: A random unassigned variable is chosen.

Value Order Heuristic:

Linear: Values are taken upon linear search of current available domain.

Least Constraining Value First: Value that shrinks others' domain the least is taken first.

Justification:

To justify why we used these Value heuristics here, first we need to understand what is expected state of the domains after each iteration and can we predict that outcome to innovate a proper way to get the next value that would always give the optimum result.

The domains are being shrunk almost randomly here with no predictability whatsoever. One way could be to randomly take the next number. But that and the linear iteration almost acts same as the probability of each of the domain values are similar.

Another way to do things could be to take the most used values till now also known as hot takes. This is the way human operates when solving Sudoku problems cause the most used value has the least places it can take place. But to implement that we would have to alter our algorithm by a fair amount.

Not changing the algorithm, yet changing the Value Heuristic then next led us to taking least used value next as the current variable has the most chance of using it next as it would seem. A major problem with this is not considering the local domain in hand rather considering the global least taken variable.

After a lot of changes, we decided on taking the Least Constraining Value next as it clearly gives us most flexibility which is much needed to get to the bottom of the solution. If we keep constraining our neighbors by assigning values that would shrink their domain the most, we surely would deviate from the solution by a big amount. Giving the them the chance to keep the largest possible domain possible will result to us getting the solution early.

Results:

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

- Linear Value Heuristic

Test Case	Method	Variable Heuristic	Nodes	Backtracks	Runtime (ms)
d-10-01	Backtracking	VAH1	415	179	2
		VAH2	54077640703	27038820323	5362413
		VAH3	85	14	0
		VAH4	55483	27713	61
		VAH5	3864856169	1932428056	549317
	Forward Checking	VAH1	400	164	2
		VAH2	125049	47966	62
		VAH3	84	13	0
		VAH4	15443	6064	15
		VAH5	126861	46613	65
d-10-06	Backtracking	VAH1	57	0	0
		VAH2	15449840919	7724920431	1413617
		VAH3	547	245	3
		VAH4	25581	12762	26
		VAH5	3871597	1935770	597
	Forward Checking	VAH1	57	0	0
		VAH2	25676	9730	10
		VAH3	327	129	1
		VAH4	8400	3289	4
		VAH5	447196	160381	101
d-10-07	Backtracking	VAH1	2101	1022	3
		VAH2	4623151189	2311575566	426631
		VAH3	459	201	0
		VAH4	46663	23303	26
		VAH5	5649525313	2824762628	744161
	Forward Checking	VAH1	2024	945	3
		VAH2	113609	43017	16
		VAH3	325	131	1
		VAH4	10795	4241	3
		VAH5	11951106	4334840	1942
d-10-08	Backtracking	VAH1	927	435	1

		VAH2	16378169289	8189084616	1374056
		VAH3	81	12	0
		VAH4	32267	16105	13
		VAH5	2030494915	1015247429	266425
	Forward Checking	VAH1	898	406	1
		VAH2	53483	21109	8
		VAH3	156	47	0
		VAH4	17514	7145	8
		VAH5	2895762	1055035	464
d-10-09	Backtracking	VAH1	57	0	0
		VAH2	*	*	*
		VAH3	8837	4390	11
		VAH4	8739	4341	6
		VAH5	26225709203	13112854573	3369442
	Forward Checking	VAH1	57	0	0
		VAH2	6389806	2515207	856
		VAH3	5736	2623	5
		VAH4	4938	1897	2
		VAH5	123810360	44616186	20033
d-15-01	Backtracking	VAH1	3656462	1828178	2453
		VAH2	*	*	*
		VAH3	836738	418316	1085
		VAH4	927207674	463603784	885812
		VAH5	*	*	*
	Forward Checking	VAH1	3452260	1623976	1794
		VAH2	705328687	273261041	101485
		VAH3	119737	56446	119
		VAH4	212265424	81919043	98166
		VAH5	*	*	*

*NB: Tests that took more than 2 hours are marked as **

- Least Constraining Value Heuristic

Test Case	Method	Variable Heuristic	Nodes	Backtracks	Runtime (ms)
d-10-01	Backtracking	VAH1	249	96	2
		VAH2	1358266597	679133270	210781
		VAH3	57	0	1
		VAH4	1505	724	5
		VAH5	*	*	*
	Forward Checking	VAH1	240	87	3
		VAH2	14157	5482	13
		VAH3	57	0	1
		VAH4	607	213	5
		VAH5	758476	274174	302
d-10-06	Backtracking	VAH1	57	0	0
		VAH2	3740420359	1870210151	465896
		VAH3	509	226	2
		VAH4	1983	963	2
		VAH5	*	*	*
	Forward Checking	VAH1	57	0	0
		VAH2	19197	7441	7
		VAH3	349	140	2
		VAH4	787	299	2
		VAH5	23190	8424	6
d-10-07	Backtracking	VAH1	399	171	1
		VAH2	3391412765	1695706354	433952
		VAH3	229	86	1
		VAH4	11583	5763	9
		VAH5	*	*	*
	Forward Checking	VAH1	379	151	1
		VAH2	28133	10881	18
		VAH3	213	73	1
		VAH4	2569	966	4
		VAH5	5543477	2022440	1621
d-10-08	Backtracking	VAH1	457	200	1

		VAH2	*	*	*
		VAH3	647	295	3
		VAH4	33021	16482	25
		VAH5	*	*	*
	Forward Checking	VAH1	441	184	0
		VAH2	115776	45656	25
		VAH3	597	256	2
		VAH4	17381	6785	19
		VAH5	99787	36954	22
d-10-09	Backtracking	VAH1	57	0	0
		VAH2	*	*	*
		VAH3	65	4	0
		VAH4	1277	610	0
		VAH5	26225709203	13112854573	3369442
	Forward Checking	VAH1	57	0	0
		VAH2	310	93	0
		VAH3	64	3	0
		VAH4	262	80	0
		VAH5	916745173	333813175	226655
d-15-01	Backtracking	VAH1	689604	344749	451
		VAH2	*	*	*
		VAH3	162068	80981	206
		VAH4	*	*	*
		VAH5	*	*	*
	Forward Checking	VAH1	653600	308745	645
		VAH2	112774511	44845411	28029
		VAH3	37297	17420	69
		VAH4	1154321542	452652469	691029
		VAH5	*	*	*

*NB: Tests that took more than 1 hour are marked as **

Aggregated Results:

To find which Method (BT or FC) with which VAH works better, we aggregated data from our 6 test cases.

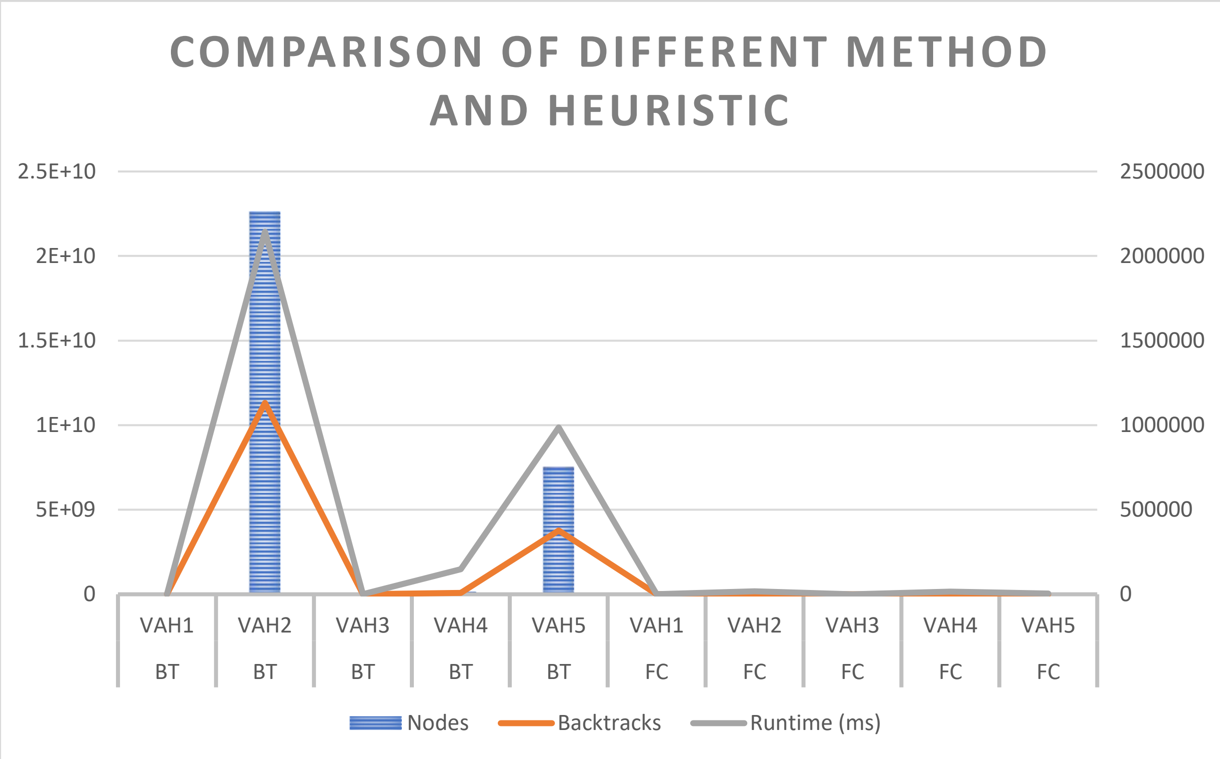
- Linear Value Heuristic

Method	*VAH*	*Nodes*	*Backtracks*	*Runtime (ms)*
BT	VAH1	610003	304969	409
BT	VAH2	22632200525	11316100234	2144179
BT	VAH3	141124	70529	183
BT	VAH4	154562734	77281334	147657
BT	VAH5	7554891439	3777445691	985988
FC	VAH1	575949	270915	300
FC	VAH2	118672718	45983011	17072
FC	VAH3	21060	9898	21
FC	VAH4	35387085	13656946	16366
FC	VAH5	27846257	10042611	4521

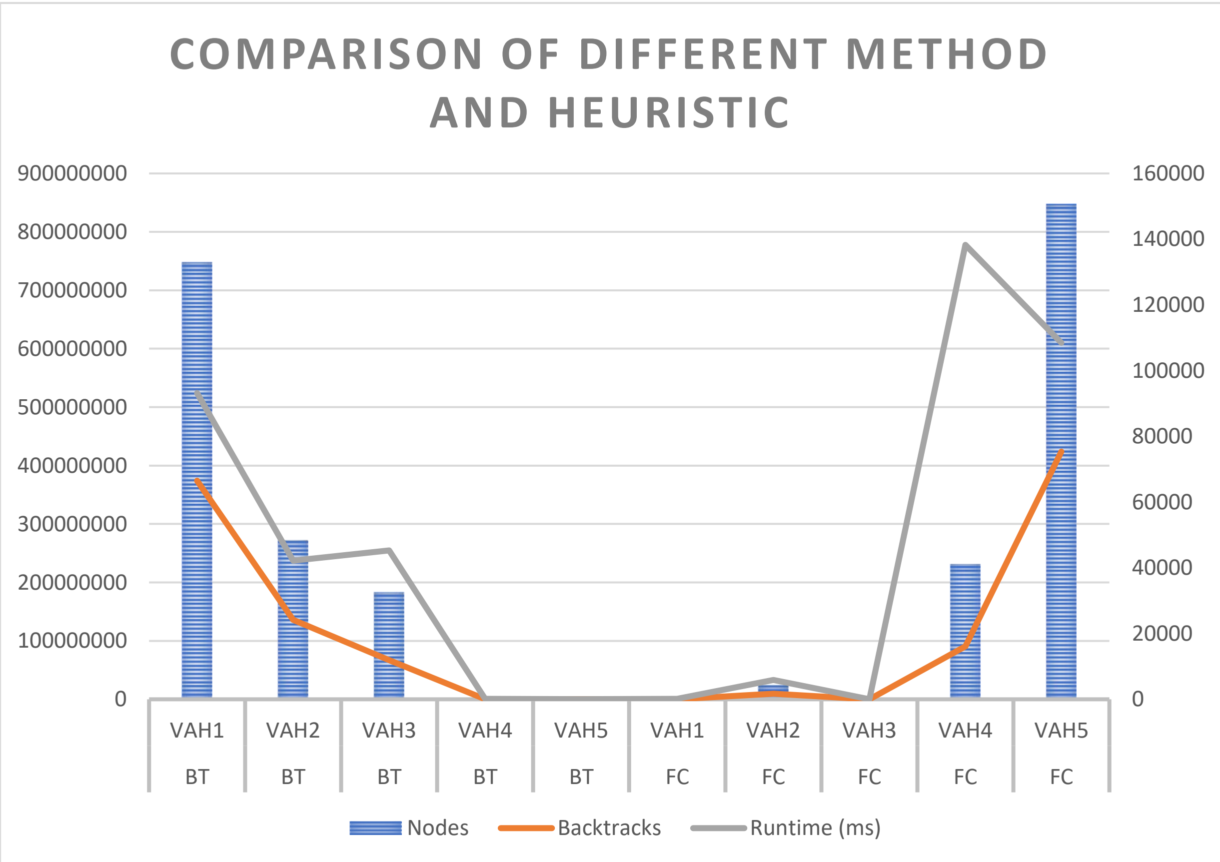
- Least Constraining Value Heuristic

Method	*VAH*	*Nodes*	*Backtracks*	*Runtime (ms)*
BT	VAH1	748084268	374042104	93179
BT	VAH2	271678945	135836999	42163
BT	VAH3	183349637	66762909	45332
BT	VAH4	147336	72627	98
BT	VAH5	56301	25107	47
FC	VAH1	134146	63066	132
FC	VAH2	23663779	9373630	5930
FC	VAH3	12565	5373	16
FC	VAH4	231016224	90585421	138267
FC	VAH5	847861538	423930732	108494

- Linear Value Heuristic



- Least Constraining Value Heuristic



Discussion:

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

We can see that clearly Forward Checking with VAH3 outperforms all the other combinations.