Department of Computer Science and Engineering, BUET

Report on Comparing types of Variable and Value Heuristic of CSP(Latin Square)

Ruhul Azgor

May 17, 2023

1 INTRODUCTION

Latin squares are a type of mathematical structure that have a number of interesting properties and applications. In this report, we will be exploring the use of constraint satisfaction problems (CSPs) to analyze and solve problems involving Latin squares.
A constraint satisfaction problem is a type of problem in which we are trying to find a solution that satisfies a set of constraints or requirements. In the context of Latin squares, these constraints might include requirements on the structure of the square or on the values that can be placed in the cells of the square.
In this report, we will first provide an overview of Latin squares and the basics of CSPs. We will then delve into the use of CSPs to solve problems involving Latin squares, including some examples and case studies.
Overall, this report aims to provide a comprehensive overview of the use of CSPs in the context of Latin squares, compare various variable and value order heuristics and to demonstrate the potential of this approach for solving a wide range of problems involving these structures.

2 VALUE ORDER HEURISTIC

2.1 Minimum number of Used Value

Let, There is a **4*4** Latin Square. After completion, each number from 1 to 4 will exactly used 4 times. The idea is to use the least used value from the available domain.If a value has already assigned more time then it is less likely to be assigned again. For this example, 4 is used 2 times, both 2 and 1 are used 2 times also and 3 is used once. As 3 is used only once, this heuristic would select **3 from { 2,3,4}** in 4^{th} row and column. But to implement this , the overhead become heavy. so this doesn't work so good. At least not better than the minimum value heuristic.

1	2	4	
	4	3	1
			2
3			

2.2 Minimum value from the domain

This heuristic doesn't track about any information. It select the minimum value from the available domain of values. Though it seems nothing fantastic but it doesn't add so much overhead.For the above example it will select **2 from { 2,3,4}** in 4^{th} row and column.

1	2	4	
	4	3	1
			2
2			

There are a few different insights that motivate the use of this heuristic. One insight is that values that are closer to the lower end of the domain may be "simpler" or "more constrained" than values that are further from the lower end. This means that the value at the lower end of the domain is more likely to be determined by the constraints of the problem, and therefore choosing such a value can help to reduce the number of remaining possibilities and guide the search in a more directed manner.
Another insight is that values that are closer to the lower end of the domain may be "more central" to the constraints of the problem, in the sense that they may have a greater influence on the values of other variables in the problem. Choosing such values may allow us to make more progress towards finding a complete solution, since they may be more closely tied to the constraints of the problem.

REMARK It doesn't matter if I choose minimum or maximum value or a random value from the domain. As it doesn't remember what values have been assigned, every value of the domain has the same probability of being correct.

3 Table: Results

Remark: Green: Most Optimal Yellow: Second Most Optimal

Problem	Solver	VAH	Value Heuristic	#Node	#Backtrack	Runtime(ms)
d-10-01	BT	VAH1	Minimum # of used value	110	57	6.2222
		VAH2		18455143746	18455143689	2H 54 m
		VAH3		460	403	11.05
		VAH4		507	450	4.69
		VAH5		25089	25032	43
		VAH1	Minimum value from the domain	240	183	7.2396
		VAH2		33291399579	33291399522	3 H 5 min
		VAH3		57	0	6.3201
		VAH4		57	0	3.8891
		VAH5		150383	150326	254
	FC	VAH1	Minimum number of used value	110	49	4.4977
		VAH2		11014034	6253749	4816.6785
		VAH3		460	376	6.40
		VAH4		507	422	4.25
		VAH5		1035	658	6.5878
		VAH1	Minimum value from the domain	250	178	4.7609
		VAH2		12850411	7337113	3656
		VAH3		57	0	4.3574
		VAH4		57	0	3.8279
		VAH5		6732	4667	8

Table 3.1: d-10-01

4 Conclusion

In this report, we have explored the use of constraint satisfaction problems (CSPs) to analyze and solve problems involving Latin squares. We have seen that CSPs can be a powerful tool for finding solutions to a wide range of problems involving these structures, including problems involving the construction and completion of Latin squares.

One of the key challenges in using CSPs for solving problems involving Latin squares is finding effective heuristics for guiding the search for solutions. In general, the best heuristic will depend on the specific problem being solved and the characteristics of the Latin square in question. Some heuristics that have been found to be effective in certain cases include backtracking search, forward checking.

In my opinion, The best heuristic for variable order is VH4.The heuristic of choosing the variable with the minimum of the ratio of the domain size to the "forward degree" (also known as the "constraint count") can be used in constraint satisfaction problems (CSPs) to guide the search for a solution. The idea behind this heuristic is to prioritize variables

Problem	Solver	VAH	Value Heuristic	#Node	#Backtrack	Runtime(ms)
d-10-06	BT	VAH1	Minimum # of used value	109	52	3.8557
		VAH2		5332579461	5332579404	53m
		VAH3		204	147	4.80
		VAH4		208	151	4.06
		VAH5		114392	114335	140
		VAH1		76	19	3.6998
		VAH2		1941574058	1941574001	11 m 9
		VAH3		57	0	3.69
		VAH4		57	0	2.904
		VAH5		66119	66062	99
	FC	VAH1	Minimum number of used value	109	49	3.613
		VAH2		7106153	4093832	3211.5512
		VAH3		204	137	3.04
		VAH4		208	139	2.11
		VAH5		6801	4696	8.7521
		VAH1		76	16	3.7812
		VAH2		4622405	2728220	1336
		VAH3		57	0	2.7121
		VAH4		57	0	2.6681
		VAH5		6735	4758	8

Table 3.2: d-10-06

that have a relatively small domain size compared to the number of constraints they participate in.
There are a few different insights that motivate the use of this heuristic. One insight is that variables with a smaller ratio of domain size to forward degree are often "more constrained" than variables with a larger ratio. This means that the value of a variable with a small ratio is more likely to be determined by the constraints of the problem, and therefore choosing such a variable can help to reduce the number of remaining possibilities and guide the search in a more directed manner.
Another insight is that variables with a smaller ratio may be "more central" to the constraints of the problem, in the sense that they participate in a relatively large number of constraints given their domain size. Choosing such variables may allow us to make more progress towards finding a complete solution, since they may have a greater influence on the values of other variables in the problem.
The heuristic of choosing the variable with the minimum of the ratio of the domain size to the forward degree can be a useful tool for guiding the search for a solution in CSPs, and it can help to reduce the complexity of the problem and improve the efficiency of the search. However, it is important to note that this heuristic may not always be the most effective choice, and it may be necessary to consider other factors or use different heuristics depending on the specific problem being solved.

Problem	Solver	VAH	Value Heuristic	#Node	#Backtrack	Runtime(ms)
		VAH1		72	15	3.3151
		VAH2		11207278061	11207278004	105 m
		VAH3	Minimum # of used value	966	909	4.26
		VAH4		1014	957	3.85
	BT	VAH5		20316	20259	31
		VAH1		91	34	3.465
		VAH2		9801720211	9801720154	52 m 20s
		VAH3		119	62	2.5869
		VAH4		146	89	2.2241
d-10-07		VAH5		156825	156768	216
		VAH1		72	11	3.0221
		VAH2		4651246	2507196	2004.9773
		VAH3	Minimum number of used value	966	820	3.66
		VAH4		1014	862	3.25
		VAH5		752	493	2.150
	FC	VAH1		91	29	3.0844
		VAH2		4443376	2398873	1179
		VAH3	Minimum value from the domain	119	69	2.3833
		VAH4		146	84	1.7212
		VAH5		11901	8302	7

Table 3.3: d-10-07

Also, Forward checking performs better than Backtracking. One reason why forward checking may be considered "better" than backtracking is that it can help to identify inconsistencies or conflicts in the problem more quickly, and eliminate them from the search space. This can reduce the amount of backtracking that is required, and make the search for a solution more efficient.
Another reason why forward checking may be considered "better" than backtracking is that it can help to reduce the size of the search space more effectively than backtracking alone. By eliminating values that are incompatible with the constraints of the problem as soon as they are assigned, forward checking can prevent the search from exploring large portions of the search space that are known to be infeasible. This can make the search for a solution significantly faster. Overall, the use of CSPs for solving problems involving Latin squares is an active area of research, and there is still much to be learned about the best approaches and algorithms for tackling these types of problems. However, the results obtained so far suggest that CSPs are a promising approach for solving a wide range of problems involving Latin squares, and that they have the potential to lead to new insights and solutions in this area.

Problem	Solver	VAH	Value Heuristic	#Node	#Backtrack	Runtime(ms)
d-10-08	BT	VAH1	Minimum # of used value	120	63	3.3525
		VAH2		706236573	706236516	7 m 18 s
		VAH3		92	35	1.34
		VAH4		92	35	1.3999
		VAH5		21465	21408	15
		VAH1		120	63	2.95
		VAH2		202262511	202262454	1m 8s
		VAH3		62	5	1.417
		VAH4		62	5	1.321
		VAH5		481	424	4
	FC	VAH1		120	57	2.8463
		VAH2		1622597	892956	741.1479
		VAH3		92	27	1.31
		VAH4		92	27	1.35
		VAH5		4352	2934	4.920
		VAH1	Minimum value from the domain	120	57	2.4151
		VAH2		1069062	603636	294
		VAH3		62	3	1.9343
		VAH4		62	3	1.3425
		VAH5		213	122	8

Table 3.4: d-10-08

Problem	Solver	VAH	Value Heuristic	#Node	#Backtrack	Runtime(ms)
d-10-09	BT	VAH1	Minimum # of used value	78	21	2.4492
		VAH2		5509844600	5509844543	52 m
		VAH3		10513	10456	19.9
		VAH4		11131	11074	15.29
		VAH5		2329	2272	16
		VAH1	Minimum value from the domain	57	0	2.3582
		VAH2		5507279980	5507279923	29m
		VAH3		4980	4923	11.9524
		VAH4		5256	5199	9.1539
		VAH5		5501	5444	8
	FC	VAH1	Minimum number of used value	78	17	2.3881
		VAH2		3864020	2023142	1516.13
		VAH3		10512	9265	17
		VAH4		11129	9770	14.74
		VAH5		467	265	1.69
		VAH1	Minimum value from the domain	57	0	1.6314
		VAH2		6328947	3347032	1514
		VAH3		4981	4324	9.9903
		VAH4		5257	4547	7.1587
		VAH5		651	404	0.955

Table 3.5: d-10-09

Problem	Solver	VAH	Value Heuristic	#Node	#Backtrack	Runtime(ms)
d-15-01	BT	VAH1	Minimum # of used value	1103887	1103781	2268.8139
		VAH2		Run Over 6 Hours.		
		VAH3		352575	352469	1177.35
		VAH4		473681	473575	1141.52
		VAH5		Run Over 6 Hours.		
		VAH1		374812	374706	629.434
		VAH2		Run Over 6 Hours.		
		VAH3		190535	190429	547.1002
		VAH4		238006	237900	470.4704
		VAH5		698004	423703	3 H 42 m
	FC	VAH1	Minimum number of used value	1103563	1002237	2058.8688
		VAH2		Run Over 6 Hours.		
		VAH3		352547	318653	1062.56
		VAH4		473755	425003	1038.4997
		VAH5		Run Over 6 Hours.		
		VAH1		374726	335739	572.5855
		VAH2		Run Over 6 Hours.		
		VAH3		190537	171793	501.5806
		VAH4		238004	213703	434.7652
		VAH5		966214037	642833610	64 min

Table 3.6: d-15-01