Constraint Satisfaction Problem: Latin Square

Khondker Salman Sayeed, 1805050

January 2023

1 Value Ordering Heuristic

When a variable is chosen to be assigned, a value must be assigned from its domain set. As we are searching for that variable's consistent assignment, we are searching in the depths of that node's branch. Therefore successful value choices will save us some depth traversals and quickly reach a solution or failure.

1.1 Least Constraining Value

Values from a variable's domain set was chosen in the *least constraining order*. In other words, the value in a variable's domain that causes the least amount of domain reductions from other variables was chosen first.

$$\min_{val \in Domain(var)} |\{n : n \in Neighbor(var), val \in Domain(n)\}|$$

1.2 Justification

The least constraining value was chosen first because this allows the solver to try the value that will most likely lead to a solution first. Once a variable is chosen for assignment we have already attempted to traverse that branch of the search problem, therefore a value that causes the least amount of domain reductions is less likely to fail due to empty domain sets in other variables. In accordance to our desire to reach the solution faster we prefer the value that is more likely to avoid backtracking on values for a variable.

2 Results

The CSP solver was executed on 5 different incomplete Latin Square Boards, 4 of them being 10 x 10, and 1 being 15 x 15.

Two different backtracking solving strategy was used:

- Simple Backtracking (BT)
- Backtracking with Forward Cheking (FC)

Five different variable ordering strategy was used for unassigned variables:

1. Smallest Domain
2. Maximum Degree in constraint graph
3. Tuple of Domain Size, Constraint Degree
4. Minimum ratio of Domain Size and Constraint Degree
5. Random

2.1 Execution Time Table

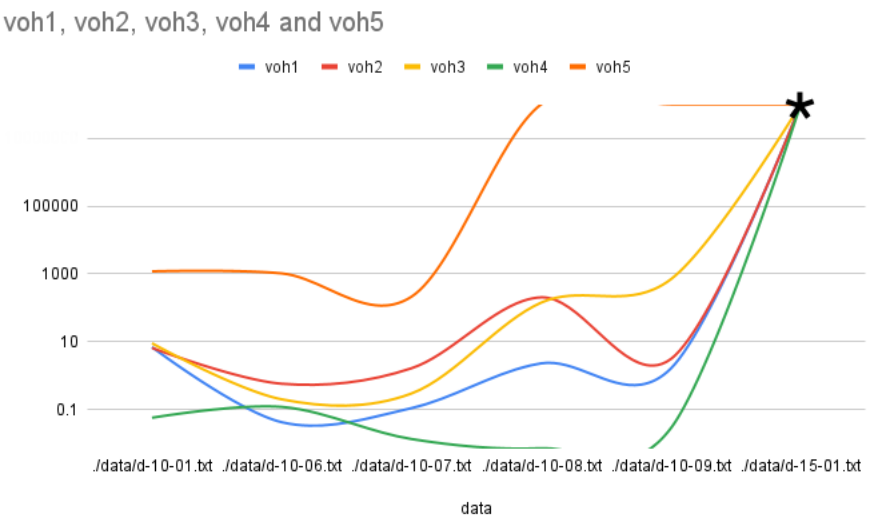
data	solver	voh	nodes	backtrack	runtime
./data/d-10-01.txt	BT	**1**	**599394**	**158072**	**2.5978**
		2	543820	138702	4.60792
		3	749843	194024	5.01375
		4	**4159**	**1160**	**0.042297**
		5	102831181	14690168	1191.46
	FC	**1**	**659**	**52**	**0.003798**
		2	12530	2679	0.141994
		3	**578**	**36**	**0.004028**
		4	1070	191	0.013995
		5	391277	100087	2.14648
./data/d-10-06.txt	BT	**1**	**4808**	**1241**	**0.023146**
		2	37371	10267	0.382735
		3	14459	3175	0.117616
		4	**7887**	**2447**	**0.089101**
		5	83462735	9273637	1036.96
	FC	**1**	**57**	**0**	**0.000273**
		2	6302	1357	0.078692
		3	**101**	**4**	**0.000681**
		4	2164	429	0.030431
		5	16832	4141	0.087441
./data/d-10-07.txt	BT	**1**	**8610**	**2487**	**0.036834**
		2	132435	39675	1.52887
		3	27975	8932	0.274266
		4	**918**	**267**	**0.010517**
		5	270411912	79831717	766.569
	FC	**1**	**114**	**4**	**0.00056**
		2	5533	1496	0.070484
		3	**64**	**2**	**0.000412**
		4	1133	220	0.016345
		5	1012	199	0.004576

Table 1 continued from previous page

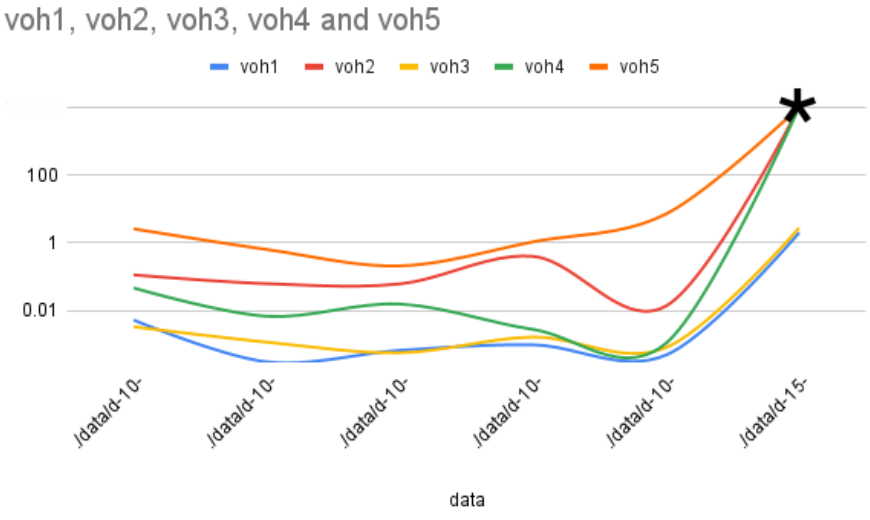
data	solver	voh	nodes	backtrack	runtime
./data/d-10-08.txt	BT	**1**	**245050**	**59624**	**1.17165**
		2	13208739	4591010	119.692
		3	11466299	3072735	87.0493
		4	**331**	**92**	**0.003976**
		5	64503159	19292055	193.189
	FC	**1**	**98**	**3**	**0.000464**
		2	37991	9205	0.458834
		3	**178**	**15**	**0.001147**
		4	268	47	0.003712
		5	203193	33865	1.05378
./data/d-10-09.txt	BT	**1**	**120896**	**30735**	**0.621182**
		2	230219	59539	1.96096
		3	49035362	13929474	326.368
		4	**1426**	**557**	**0.014787**
		5	*	*	*
	FC	**1**	**66**	**9**	**0.000478**
		2	2150	167	0.013378
		3	**69**	**12**	**0.000796**
		4	129	72	0.001065
		5	1311023	120457	7.00771
./data/d-15-01.txt	BT	1	*	*	*
		2	*	*	*
		3	*	*	*
		4	*	*	*
		5	*	*	*
	FC	**1**	**344858**	**24556**	**1.96675**
		2	*	*	*
		3	**381868**	**34567**	**2.70825**
		4	*	*	*
		5	*	*	*

Table 1

2.2 Variable Order Heuristic Figures



(a) Simple Backtracking



(b) Forward Checking

Figure 1: Runtimes for different Variable Order Heuristics (VOH)

2.3 Solver

From the runtime table **??** and figures 1a, 1b, it is clear that *forward checking* is the better solver among the two.

2.4 Justification For Forward Checking

Forward checking is better than Simple Backtracking because after each assignment, forward checking decreases the domain size of neighboring variable nodes in the constraint graph. So with each assignment, the search scope get narrower. This has two main benefits:

1. With fewer domains to try out, the problem branching on the lower search nodes become smaller. Which leads to *early failure* on branches and allows to find the correct solution quickly.

2. Appropriate variable ordering heuristics can quickly find the variable that is *most constrained*. Decreasing the domain size of variables allows this heuristic to dynamically find the variable whose domain has shrunk the most. That way successfull assignment of that variable requires less amount of branching, which reduces problem size – allowing for a quicker search.

2.5 Variable Order Heuristic

The contenders for the best variable ordering heuristic are *Smallest domain* (1) and *Minimum ratio of domain size and constraint degree* (4). I choose *Smallest Domain* to be the best heuristic choice considering the overall performance.

2.6 Justification For Smallest Domain

Smallest domain variable ordering heuristic is the problem-specific implementation of *most constrained variable* for this problem. The variable which has the smallest domain, has the most constrained search space because it has the smallest options to assign. So attempting to assign it first has two main benefits:

1. This makes the inconsistent values fail early – allowing efficient pruning of the search tree.

2. Assigning smaller domains early in the search problem means lower branching factor on the higher depths of the search tree. This potentially exponentially reduces the search space.

3 Conclusion

From the executions it is clear that the combination of *forward checking solver* with *smallest domain heuristic* for variable ordering is the best performer on average.