CSE318

Artificial Intelligence Sessional

Offline-02

Constraint Satisfaction Problem - Latin Square

Bangladesh University of Engineering & Technology

Md. Sultanul Arifin

1805097

Section: B2

Level - 3, Term - 2

Department of Computer Science and Engineering

Submitted on January 06, 2023

Introduction:

In this assignment we are to solve a CSP (fill up a latin square) using various Variable Assignment Heuristics and compare their performance. Here we have used -

- **VAH1**: Choosing the variable with smallest domain first
- **VAH2**: Choosing the variable with most degree to unassigned variables
- **VAH3**: Choosing the variable with the smallest domain but tie breaking with the degrees to unassigned variables
- **VAH4**: Choosing the variable that has smallest ratio (size of domain)/(degrees to unassigned variables)
- **VAH5**: Choosing variables randomly

As a strategy for checking constraints, we used two methods-

- **Normal Backtrack**: As the name implies, here we checked all possible value assignment ***without shrinking the domain*** to get a solution
- **Forward Checking**: Here we shrink the domain of each related unassigned variable after each assignment. And not going forward if domain of any variable becomes empty

After choosing a variable a value is assigned using the **LCV (Least Constraining Value)** heuristics. Though it improved the node and backtrack count. It came with a cost of increased running time per node. However, in some VAH the improvement includes overall running time. In the following table the performance of various VAH is shown and the one with the lowest node and backtrack count is marked green. Here the ***node count is the total number of attempts to assign a value to a variable*** . However not all configurations of VAH and strategy chosen can solve within a tractable time (cutoff time 30min); they are marked with a star.

Observed Data:

Problem	Strategy	VAH	Number of Node	Number of Backtracks	Time (microsecond)
d-10-01.txt	Backtrack with Forward Checking	VAH1	229	15	27,102
	Backtrack with Forward Checking	VAH2	8,911,853	4,135,325	70,821,595
	Backtrack with Forward Checking	VAH3	58	0	5,528
	Backtrack with Forward Checking	VAH4	60	1	5,910
	Backtrack with Forward Checking	VAH5	4,197,488	1,834,431	38,267,292
	Normal Backtrack	VAH1	578,815	409,564	4,861,256
	Normal Backtrack	VAH2	*	*	*
	Normal Backtrack	VAH3	1,258,634	898,071	10,417,323
	Normal Backtrack	VAH4	14,652,997	10,691,070	128,887,565
	Normal Backtrack	VAH5	*	*	*

d-10-06.txt	Backtrack with Forward Checking	VAH1	58	0	9,471
	Backtrack with Forward Checking	VAH2	16,391,733	7,337,490	128,153,786
	Backtrack with Forward Checking	VAH3	58	0	7,545
	Backtrack with Forward Checking	VAH4	59	1	5,700
	Backtrack with Forward Checking	VAH5	214,933	91,802	2,047,078
	Normal Backtrack	VAH1	1,776,348	1,290,245	15,164,918
	Normal Backtrack	VAH2	*	*	*
	Normal Backtrack	VAH3	4,772,348	3,578,643	38,644,861
	Normal Backtrack	VAH4	3,589,081	2,686,539	28,682,709
	Normal Backtrack	VAH5	*	*	*

d-10-07.txt	Backtrack with Forward Checking	VAH1	575	37	27,577
	Backtrack with Forward Checking	VAH2	5,040,372	2110532	39,219,891
	Backtrack with Forward Checking	VAH3	377	23	32,952
	Backtrack with Forward Checking	VAH4	588	58	36,118
	Backtrack with Forward Checking	VAH5	114,433	47,456	1,212,713
	Normal Backtrack	VAH1	3,196,257	2,257,094	25,859,535
	Normal Backtrack	VAH2	*	*	*
	Normal Backtrack	VAH3	20,386,575	14,253,890	173,115,986
	Normal Backtrack	VAH4	238,988,321	172,590,536	1,795,978,237
	Normal Backtrack	VAH5	*	*	*

d-10-08.txt	Backtrack with Forward Checking	VAH1	301	22	47,356
	Backtrack with Forward Checking	VAH2	4,292,669	2,026,948	27,857,681
	Backtrack with Forward Checking	VAH3	82	3	5,404
	Backtrack with Forward Checking	VAH4	70	2	7,393
	Backtrack with Forward Checking	VAH5	231,705	98,570	1,814,387
	Normal Backtrack	VAH1	8,169,323	5,554,628	56,617,871
	Normal Backtrack	VAH2	*	*	*
	Normal Backtrack	VAH3	16,948,652	11,829,911	125,125,335
	Normal Backtrack	VAH4	*	*	*
	Normal Backtrack	VAH5	*	*	*

d-10-09.txt	Backtrack with Forward Checking	VAH1	58	0	5,096
	Backtrack with Forward Checking	VAH2	12,096,135	5,698,521	77,816,672
	Backtrack with Forward Checking	VAH3	5,591	613	106,152
	Backtrack with Forward Checking	VAH4	7,855	1,051	143,986
	Backtrack with Forward Checking	VAH5	3,856,801	1,674,179	26,432,419
	Normal Backtrack	VAH1	530,811	398,275	3,841,654
	Normal Backtrack	VAH2	*	*	*
	Normal Backtrack	VAH3	*	*	*
	Normal Backtrack	VAH4	*	*	*
	Normal Backtrack	VAH5	*	*	*

d-15-01.txt	Backtrack with Forward Checking	VAH1	1,828,285	204,202	30,380,407
	Backtrack with Forward Checking	VAH2	*	*	*
	Backtrack with Forward Checking	VAH3	418,423	39,615	8,133,390
	Backtrack with Forward Checking	VAH4	972,335	109,416	15,330,331
	Backtrack with Forward Checking	VAH5	*	*	*
	Normal Backtrack	VAH1	*	*	*
	Normal Backtrack	VAH2	*	*	*
	Normal Backtrack	VAH3	*	*	*
	Normal Backtrack	VAH4	*	*	*
	Normal Backtrack	VAH5	*	*	*

Analysis:

Normal Backtracking has the poorest overall performance as it blindly checks all “possible” values and doesn't make any further inference on the domains of other variables. Thus in many cases it is intractable. VAH1 performs better with normal backtracking. As there is no inference it cannot properly utilize other heuristics that involve forward-degree, that could potentially shrink the domain size. Thus VAH1 has the most efficient performance with normal backtrack.

Here we can see that VAH3 with forward checking provides the fastest solution in most cases. However VAH2 and VAH5 with normal backtracking gave the worst performance. This is due to the fact that

VAH3 uses MRV (Minimum Remaining Value). MRV heuristics picks up the most restrained variable for assignment and makes it easy for early failure detection. Backtrack early in the search removes the need for checking a lot of the nodes than that in the later in the search. However tie breaking with most degrees to unassigned variables shrinks the domain of many more variables, resulting in early failure in the strategy involving forward checking. Next in line is VAH4, which has the performance close to that of VAH3 with slightly more node count but less running time as it is “easier” than VAH3. The reason behind VAH4’s larger node count may be for the reason that it does not always choose the variable with the smallest domain eg. a variable can have more priority for its lower forward-degree count even if it has a larger domain.

VAH2 and VAH5 have the worst performance. As VAH2 only considers the degrees of relationship to unassigned variables, it can shrink the domain of most numbers of variables. But most often the variable that has larger forward-degree has larger domain size, which is responsible for the poor performance of VAH2. VAH5 faces the same difficulty. Though it has lower running time per node it has huge run time and node count as well, because it does not take into account any nicely guided measurements

Conclusion:

VAH3 with forward checking has the best performance in most cases