CS4203 P2 - Report

190022658

November 25, 2022

1 Overview

In this practical, we were asked to design and implement constraint solvers for binary constraints using

1. Forward Checking and;
2. MAC (maintaining arc consistency)

2 Design

There are variables which keep track of state of domain, variable assignments and unassigned variables.

1. ArrayList<Integer>[] domainArr
    - used array because the number of variables are constant
2. int[] assignmentVal
    - same as domainArr
3. ArrayList<Integer> varlist
    - could use a boolean[] but ArrayList was easier to use

2.1 Forward Checking

The implementation is as the pseudo-code in lecture. The use of array for domainArr and assignmentVal made access easier.

For forward checking, the algorithm first goes to the left branch (when you assign value to the variable), prune the domains if the arcs revised are consistent then continue to branch out. Otherwise go to the right branch (when you don't assign value to variable), then remove value from variable's domain and continue if the domain of variable is not empty.

Domain pruning of every unassigned variable was done by the reviseArcs method. The revise method, selected the Binary Constraint applicable to the pair of present and future variables. It then removed any values from the domain of future that were not a part of the constraint given the assignment that was made to the present variable. If the size of the new domain was empty, then the method would return false. The new domain replaced the previous one in the domain map otherwise.

To undo the pruning, I copied the domains before calling revise arcs and restored it when undo needed.

The solution is found once varlist is empty, it then prints out the solution and exits.

2.2 Maintaining Arc Consistency

The implementation is as the pseudo-code in lecture.

To manage the queue, I used an array list. At the start, the queue is initialised using the getVariableConstraints method which return constraints applicable. These are the arcs that incident on the current variable. In a while loop, arcs are removed in a FIFO way and the domains are pruned by reviseDomain. This method is similar to the revise method but the revise method for forward checking utilises the variable assignments that are made. The method calls the revise method to remove domains in the case of assignment. Otherwise, a list of tuples for the constraint is filtered based on the domain of the variable. From the list of filtered tuples, values are added to the new domain of the second variable of the constraint. If no values in the current domain are valid, nothing is added to the new domain and the method returns false. Otherwise, the domain is updated.

The solution is again found if varlist is empty, it then prints out the solution and exits.

3 Evaluation

The results are shown as below. All of the test cases run and within 200s. The trend is that mac has fewer nodes and more arc revisions, this can be seen in most of the cases. mac also tends to take more time which is because there are a lot of constraints which lead to alot of arc revisions. SDF takes less time for bigger problems than ASC, this can be seen for FinnishSudoku, langfords3_10 and more. The time difference between smaller ones are not significant.

3.1 ASC

Result	Time, Node count, Arc count (fc)	Time, Node count, Arc count (mac)
4 Queens	0.11, 14, 16	0.10, 6, 21
6 Queens	0.20, 56, 91	0.19, 26, 169
8 Queens	0.29, 203, 367	0.41, 235, 1933
10 Queens	0.40, 186, 429	0.87, 1427, 14621
langfords2_3	0.19, 32 ,55	0.18, 13, 86
langfords2_4	0.29, 119, 224	0.27, 26,293
langfords3_9	1.69, 13946, 86346	3.01, 353, 47565
langfords3_10	7.20, 61798, 429417	9.17, 1608, 244534
FinnishSudoku	34.77, 319314, 7066671	193.68, 86347, 43494423
SimonisSudoku	0.83, 385, 9639	9.10, 1085, 533355

3.2 SDF

Result	Time, Node count, Arc count (fc)	Time, Node count, Arc count (mac)
4 Queens	0.12, 14, 16	0.11, 10, 29
6 Queens	0.18, 51, 86	0.21, 48, 190
8 Queens	0.27, 170, 323	0.29, 12, 108
10 Queens	0.41, 69, 182	0.56, 75, 630
langfords2_3	0.18, 38, 57	0.17, 6, 35
langfords2_4	0.30, 109, 195	0.30, 8, 84
langfords3_9	1.76, 7386, 43291	1.74, 59, 6251
langfords3_10	3.10, 24448, 160982	1.81, 66, 9233
FinnishSudoku	3.63, 27906, 516806	2.17, 238, 141921
SimonisSudoku	0.75, 81, 3240	10.58, 2696, 2647279

4 How to-s

```
make # out P2.jar
java -jar P2.jar ../instances/4Queens.csp fc asc asc
make clean
```

5 Conclusion

This practical reinforced my understanding of constraint solving algorithms. Given more time, I would create more instances of test cases and cleaned my algorithm for it to be faster. I would also enhanced the mac algorithm and used better datastructures where possible.