Report: P2 – Constraint Solver Implementation

Overview

In this coursework I have created a constraint solver to demonstrate my understanding of how constraint solvers operate. This involved implementing two search algorithms: forward checking and maintaining arc consistency. Forward checking makes use of concepts such as domain pruning and look-ahead to solve a CSP. Whereas maintaining arc consistency builds on top of this by using the concept of global arc consistency, which it enforces throughout the entire search. In addition, my implementation of both these algorithms have been evaluated by comparing their performances when solving instances of various problem classes.

Design and Implementation

Forward Checking:

The implementation of forward checking employs 2-way branching which has two recursive functions: one for the left branch which involves assigning a value to a variable and one for the right branch which involves removing this value for the variable's domain. Both these functions are called by a function which acts as the starting point to facilitate the switching from one branch to another. This can be thought of as the parent node for the right and left child nodes. The underlying mechanism for this is backtracking which is based on a condition at each child node and involves checking whether the any of the pruned domains have been emptied, and if so, any changes that were made are undone. Additionally, dynamic variable order is used to improve the efficiency of the search. Smallest-domain first is used here where the smallest domain amongst all the unassigned variable is identified and this variable is the next variable to assign. The value from this domain to assign is chosen in an ascending order which involves ordering the domain list and selecting the first element.

The last and most important component in this algorithm is domain pruning which results in backtracking as mentioned above. This makes use of the *BinaryCSP* class which contains the list of binary constraints. Each time a variable has been assigned a value, arc revision involves extracting the binary constraint that contains this variable and going through the list of tuples. Since this list of tuples contains all the valid values that can be assigned for each of the variables within the constraint, if there are any values in the domain of the other variable in the constraint that doesn't follow this, that value is removed. This process ends either when all the relevant constraints have been extracted and all the invalid values have been pruned, or when a domain has been emptied. Therefore, this results in either the domains of all the variables to being updated when the revision is successful or restored when unsuccessful.

Design decisions (choice of data structures):

- Each decision variable is stored as an object of the *Variable* class where, within the class, the variable is represented as an integer, and it also contains the domain of the variable which is a list of values.
    - Each problem instance therefore consists of a series of objects which is stored as a list (*List<Variable>)* and represents the state of the search.
    - Domain pruning involved modifying this domain.
- To facilitate the functionality of undoing the pruning, a stack is used where each element in the stack represents the state of the search every time a branch has been entered.
    - The state as mentioned above consists of the list of variable objects.
    - During arc revision, every time a domain in one of the variable objects has been emptied, an element from the stack is popped and assigned to the current state of the search. This popped element represents the state of the search before a change has been made (either a value being assigned to a variable, or a value being removed from the domain of the variable).
    - So before revising future arcs, the current state of the search is pushed into the stack in case an empty domain is detected during the revision.
- Assigned variables are stored in an array where each element is the variable integer from the *Variable* class.
    - This is used to keep track of variables that have been assigned or unassigned a value.
    - It is also used in the conditional check for whether a solution has been found and if so, this array is printed.

Maintaining Arc Consistency:

The implementation of maintaining arc consistency also employs 2-way branching. The core structure of the algorithm is mostly the same as forward checking since maintaining arc consistency builds on top of it. The main difference is in how the arcs are revised. It uses AC3 which is an algorithm used for maintaining global arc consistency, therefore the function for domain pruning is different. Each time a change occurs in one of the variable's domains, all the relevant arcs which involve this variable are added to the queue. So, this involves initially setting up the queue before performing arc revision. Each arc is then revised which may result in more domains being pruned and so more arcs are added to the queue. This process repeats until the queue is empty, which occurs when either all the arcs have been revised or a domain has been emptied. This results in backtracking and the pruning being undone which follows the same approach described above. An important thing to notice is that the AC3 algorithm is also initiated before the search begins so that the search space is reduced. Therefore, the initial queue contains all the arcs added to it. Note that node consistency is a precondition for maintaining arc consistency, but this isn't implemented in the code since there aren't any unary constraints involving just one node.

Design decisions (choice of data structures):

- Each arc is stored as an object of the *Arc* class which contains the two decision variables that are within the scope of the constraint.
- The queue is implemented as a linked-list and is used to store all the arc objects when maintaining global arc consistency.
- (The rest of the design decisions are the same as forward checking).

A note on performance

Both algorithms are noticeably slow especially when it comes to larger problem instances. This is particularly true for maintaining arc consistency, where sometimes the solver doesn't produce a result. This performance issue is due to two key design decisions that were made. Firstly, the choice of storing the variables as a list of objects resulted in several nested *for* and *if* statements when accessing the correct variable domain. This is because the entire list requires to be searched to find the relevant variable object and results in poorly structure code. Therefore, for large instances which involve many decision variables, this results in more time being spent searching through the variable list. Secondly, the choice of using a stack to store all the search states resulted in a lot of memory being utilized just to support one functionality., there is scope to drastically improve the performance of these algorithms by making better design decisions.

Empirical Evaluation

The experiment to compare the merits of forward checking and maintaining arc consistency involves using problem instances from three problems classes: n-Queens, Langford's problem, and Sudoku. For each problem class, 8 instances are used to test the solver, where each contains an increasing amount of complexity. The independent variable in this case is the number of decision variables which indicates the complexity of the instances, and the dependent variables are the time taken, nodes in the search tree and arc revisions. The control variables are the three problem classes chosen for this experiment. As part of the experiment, both algorithms are given all the problem instances as input and the output produced is recorded. The result of conducting this experiment is given below where each instance file is stated, and the result produced by both algorithms for each instance is provided.

N-Queens

- 4Queens.csp (4 decision variables):
    - FC results:
        - Time taken: **1ms.**
        - Nodes visited: **8.**
        - Arcs revised: **18.**

    - MAC result:
        - Time taken: **2ms.**
        - Nodes visited: **5.**
        - Arcs revised: **46.**
- 6Queens.csp (6 decision variables):
    - FC results:
        - Time taken: **3ms.**
        - Nodes visited: **49.**
        - Arcs revised: **197.**
    - MAC result:
        - Time taken: **9ms.**
        - Nodes visited: **27.**
        - Arcs revised: **405.**
- 8Queens.csp (8 decision variables):
    - FC results:
        - Time taken: **5ms.**
        - Nodes visited: **80.**
        - Arcs revised: **366.**
    - MAC result:
        - Time taken: **18ms.**
        - Nodes visited: **25.**
        - Arcs revised: **635.**
- 10Queens.csp (10 decision variables):
    - FC results:
        - Time taken: **7ms.**
        - Nodes visited: **80.**
        - Arcs revised: **415.**
    - MAC result:
        - Time taken: **34ms.**
        - Nodes visited: **45.**
        - Arcs revised: **1588.**
- 14Queens.csp (14 decision variables):
    - FC results:
        - Time taken: **73ms.**
        - Nodes visited: **1329.**
        - Arcs revised: **8735.**
    - MAC result:
        - Time taken: **294ms.**
        - Nodes visited: **273.**
        - Arcs revised: **16860.**
- 16Queens.csp (16 decision variables):
    - FC results:
        - Time taken: **218ms.**
        - Nodes visited: **6873.**

        - Arcs revised: **49333.**
    - MAC result:
        - Time taken: **437ms.**
        - Nodes visited: **696.**
        - Arcs revised: **46164.**
- 18Queens.csp (18 decision variables):
    - FC results:
        - Time taken: **768ms.**
        - Nodes visited: **27463.**
        - Arcs revised: **211828.**
    - MAC result:
        - Time taken: **1071ms.**
        - Nodes visited: **1927.**
        - Arcs revised: **151716.**
- 20Queens.csp (20 decision variables):
    - FC results:
        - Time taken: **3585ms.**
        - Nodes visited: **131944.**
        - Arcs revised: **1082961.**
    - MAC result:
        - Time taken: **1898ms.**
        - Nodes visited: **3016.**
        - Arcs revised: **287226.**

Measurements table for FC:

Instance file	Number of decision variables	Time taken to solve (ms)	Number of nodes visited	Number of arcs revised
4Queens.csp	4	1	8	18
6Queens.csp	6	3	49	197
8Queens.csp	8	5	80	366
10Queens.csp	10	7	80	415
14Queens.csp	14	73	1329	8735
16Queens.csp	16	218	6873	49444
18Queens.csp	18	768	27463	211828
20Queens.csp	20	3585	131944	1082961

Measurements table for MAC:

Instance file	Number of decision variables	Time taken to solve (ms)	Number of nodes visited	Number of arcs revised
4Queens.csp	4	2	5	46
6Queens.csp	6	9	27	405
8Queens.csp	8	18	25	635
10Queens.csp	10	34	45	1588
14Queens.csp	14	294	273	16860
16Queens.csp	16	437	696	46164
18Queens.csp	18	1071	1927	151716
20Queens.csp	20	1898	3061	287226

Langford's problem

- langfords2_3.csp (6 variables):
  - FC results:
    - Time taken: **1ms.**
    - Nodes visited: **6.**
    - Arcs revised: **15.**
  - MAC result:
    - Time taken: **6ms.**
    - Nodes visited: **7.**
    - Arcs revised: **113.**
- langfords2_4.csp (8 variables):
  - FC results:
    - Time taken: **1ms.**
    - Nodes visited: **8.**
    - Arcs revised: **28.**
  - MAC result:
    - Time taken: **15ms.**
    - Nodes visited: **15.**
    - Arcs revised: **329.**
- langfords2_9.csp (18 variables):
  - FC results:
    - Time taken: **11ms.**
    - Nodes visited: **18.**
    - Arcs revised: **153.**
  - MAC result:
    - Time taken: _
    - Nodes visited: _
    - Arcs revised: _
- langfords2_10.csp (20 variables):
  - FC results:

        - Time taken: **13ms.**
        - Nodes visited: **20.**
        - Arcs revised: **190.**
    - MAC result:
        - Time taken: _
        - Nodes visited: _
        - Arcs revised: _
- langfords3_3.csp (9 variables):
    - FC results:
        - Time taken: **3ms.**
        - Nodes visited: **14.**
        - Arcs revised: **58.**
    - MAC result:
        - Time taken: _
        - Nodes visited: _
        - Arcs revised: _
- langfords3_4.csp (12 variables):
    - FC results:
        - Time taken: **6ms.**
        - Nodes visited: **20.**
        - Arcs revised: **120.**
    - MAC result:
        - Time taken: _
        - Nodes visited: _
        - Arcs revised: _
- langfords3_9.csp (27 variables):
    - FC results:
        - Time taken: **60ms.**
        - Nodes visited: **50.**
        - Arcs revised: **696.**
    - MAC result:
        - Time taken: **76973ms.**
        - Nodes visited: **9484.**
        - Arcs revised: **931678.**
- langfords3_10.csp (30 variables):
    - FC results:
        - Time taken: **101ms.**
        - Nodes visited: **56.**
        - Arcs revised: **858.**
    - MAC result:
        - Time taken: **431787.**
        - Nodes visited: **34676.**
        - Arcs revised: **3816555.**

Measurements table for FC:

Instance file	Number of decision variables	Time taken to solve (ms)	Number of nodes visited	Number of arcs revised
langfords2_3.csp	6	1	6	15
langfords2_4.csp	8	1	8	28
langfords2_9.csp	18	11	18	153
langfords2_10.csp	20	13	20	190
langfords3_3.csp	9	3	14	58
langfords3_4.csp	12	6	20	120
langfords3_9.csp	27	60	50	696
langfords3_10.csp	30	101	56	858

Measurements table for MAC:

Instance file	Number of decision variables	Time taken to solve (ms)	Number of nodes visited	Number of arcs revised
langfords2_3.csp	6	6	7	113
langfords2_4.csp	8	15	15	329
langfords2_9.csp	18	_	_	_
langfords2_10.csp	20	_	_	_
langfords3_3.csp	9	_	_	_
langfords3_4.csp	12	_	_	_
langfords3_9.csp	27	76973	9484	931678
langfords3_10.csp	30	431787	34676	3816555

Sudoku

- Sudoku1.csp (beginner):
  - FC results:
    - Time taken: **64ms.**
    - Nodes visited: **117.**
    - Arcs revised: **4359.**
  - MAC result:
    - Time taken: **6385ms.**
    - Nodes visited: **81.**
    - Arcs revised: **7413.**
- Sudoku2.csp (easy):
  - FC results:
    - Time taken: **254ms.**

        - Nodes visited: **913.**
        - Arcs revised: **27946.**
    - MAC result:
        - Time taken: **6909ms.**
        - Nodes visited: **114.**
        - Arcs revised: **10517.**
- Sudoku3.csp (medium):
    - FC results:
        - Time taken: _
        - Nodes visited: _
        - Arcs revised: _
    - MAC result:
        - Time taken: _
        - Nodes visited: _
        - Arcs revised: _
- Sudoku4.csp (hard):
    - FC results:
        - Time taken: **3971ms.**
        - Nodes visited: **24321.**
        - Arcs revised: **905888.**
    - MAC result:
        - Time taken: **7438ms.**
        - Nodes visited: **416.**
        - Arcs revised: **36801.**
- Sudoku5.csp (very hard):
    - FC results:
        - Time taken: **252ms.**
        - Nodes visited: **827.**
        - Arcs revised: **29114.**
    - MAC result:
        - Time taken: **7361ms.**
        - Nodes visited: **129.**
        - Arcs revised: **13253.**
- Sudoku6.csp (extremely hard):
    - FC results:
        - Time taken: **228ms.**
        - Nodes visited: **730.**
        - Arcs revised: **23980.**
    - MAC result:
        - Time taken: **6798ms.**
        - Nodes visited: **85.**
        - Arcs revised: **8357.**
- SimonisSudoku.csp:
    - FC results:

        - Time taken: **54ms.**
        - Nodes visited: **98.**
        - Arcs revised: **3808.**
    - MAC result:
        - Time taken: **8410ms.**
        - Nodes visited: **81.**
        - Arcs revised: **7870.**
- FinnishSudoku.csp:
    - FC results:
        - Time taken: **19939ms.**
        - Nodes visited: **129785.**
        - Arcs revised: **4730650.**
    - MAC result:
        - Time taken: **62797ms.**
        - Nodes visited: **29085.**
        - Arcs revised: **3404886.**

Measurements table for FC:

Instance file	Number of decision variables	Time taken to solve (ms)	Number of nodes visited	Number of arcs revised
Sudoku1.csp	81 (beginner)	64	117	4359
Sudoku2.csp	81 (easy)	254	913	27946
Sudoku3.csp	81 (medium)	_	_	_
Sudoku4.csp	81 (hard)	3971	24321	905888
Sudoku5.csp	81 (very hard)	252	827	29114
Sudoku6.csp	81 (extremely hard)	228	730	23980
SimonisSudoku.csp	81	54	98	3808
FinnishSudoku.csp	81	19939	129785	4730650

Measurements table for MAC:

Instance file	Number of decision variables	Time taken to solve (ms)	Number of nodes visited	Number of arcs revised
Sudoku1.csp	81 (beginner)	6381	81	7413
Sudoku2.csp	81 (easy)	6909	114	10517
Sudoku3.csp	81 (medium)	_	_	_
Sudoku4.csp	81 (hard)	7438	416	36801
Sudoku5.csp	81 (very hard)	7361	129	13253
Sudoku6.csp	81 (extremely hard)	6798	85	8357
SimonisSudoku.csp	81	8410	81	7870
FinnishSudoku.csp	81	62797	29085	3404886

Comparing the results of both algorithms

Overall, there are three key observations that can be made from all the experiment results. Firstly, forward checking does fewer arc revisions than maintaining arc consistency. This observation is consistent with how both algorithms operate. Maintaining arc consistency maintains a queue of arcs and every time a domain of one variable is changed, further arcs are added to the queue which means that the number of arcs revised after each variable is assigned a value increases substantially. Whereas forward checking doesn't take this extra step of adding more arcs when pruning the domains. It simply revises the relevant arcs once per variable assignment. Secondly, forward checking visits more nodes than maintaining arc consistency which is a direct consequence of the first observation. Since maintaining arc consistency does more arc revisions per variable assignment, it infers more information every time a variable is assigned a value. Essentially, it can look ahead further than forward checking, and therefore can make more informed decision and do less backtracking. This results in a smaller search space, which is basically the number of nodes visited. Forward checking can't infer as much information since it doesn't visit as many arcs. Lastly, maintaining arc consistency takes noticeably longer to reach a solution. This makes sense considering how much more processing it does compared to forward checking. It employs AC3 which is a sophisticated algorithm for maintaining global arc consistency which can be a demanding process especially for larger problem instances since it would result in larger queue of arcs. Note however that for some instances, maintaining arc consistency seems to perform substantially worse than forward checking, and sometimes doesn't produce a result. This can be seen in the Langford's problem class for example.

The above observations look at the results itself, but there are also observations to be made in how the results change with respect to the instance size which is certainly the key focus of this experiment. There is an obvious trend which is that as the number of decision variables increases, all three measured values increase for both algorithms. This is to be expected since larger problems usually require more computation effort to solve. More interestingly, there seems to be a shift in the pattern for the number of arcs revised as the instance size

increases. For smaller instances, forward checking revises fewer arcs compared to maintaining arc consistency. But for larger instances, the opposite is true, suggesting that maintaining arc consistency seems to produce better results overall and is more suited for larger instances. This pattern in the measurements for nodes visited and arcs revised is true for all the problem classes and is certainly true for the time taken measurements, where again maintaining arc consistency shows better performance for larger instances (with a few exceptions). Most of these conclusions are drawn from the n-Queens and Sudoku problem classes since maintaining arc consistency solves as many of the instances as forward checking. This isn't true for the Langford's problem class as maintaining arc consistency doesn't solve half the instances and the results produced seems to contradict the pattern observed from the other two problem classes. For this reason, the results from that hold less weight in the evaluation.

Example output of running forward checking and maintaining arc consistency

- Result produced by the forward checking algorithm given 20Queens.csp as the input.

```
CS4402 Constraint Programming\Practicals\P2\src> javac Solver.java
PS C:\Users\user\OneDrive - University of St Andrews\University of St Andrews\Fourth Year\
CS4402 Constraint Programming\Practicals\P2\src> java Solver instances/20Queens.csp
Choose the search algorithm to run.
 1: Forward Checking
 2: Maintaining Arc Consistency
1
Initiating Forward Checking...
Solution:
Variable 0 = 0
Variable 1 = 2
Variable 2 = 4
Variable 3 = 1
Variable 4 = 3
Variable 5 = 12
Variable 6 = 14
Variable 7 = 11
Variable 8 = 17
Variable 9 = 19
Variable 10 = 16
Variable 11 = 8
Variable 12 = 15
Variable 13 = 18
Variable 14 = 7
Variable 15 = 9
Variable 16 = 6
Variable 17 = 13
Variable 18 = 5
Variable 19 = 10
Nodes visited: 131944
Arc revisions: 1082961
Time taken: 3643ms
```

- Result produced by the maintaining arc consistency algorithm given 20Queens.csp as the input.

```
CS4402 Constraint Programming\Practicals\P2\src> javac Solver.java
PS C:\Users\user\OneDrive - University of St Andrews\University of St Andrews\Fourth Year\
CS4402 Constraint Programming\Practicals\P2\src> java Solver instances/20Queens.csp
Choose the search algorithm to run.
 1: Forward Checking
 2: Maintaining Arc Consistency
2
Initiating Maintaining Arc Consistency...
Solution:
Variable 0 = 0
Variable 1 = 19
Variable 2 = 1
Variable 3 = 4
Variable 4 = 2
Variable 5 = 7
Variable 6 = 3
Variable 7 = 11
Variable 8 = 15
Variable 9 = 18
Variable 10 = 16
Variable 11 = 14
Variable 12 = 17
Variable 13 = 8
Variable 14 = 5
Variable 15 = 9
Variable 16 = 6
Variable 17 = 13
Variable 18 = 10
Variable 19 = 12
Nodes visited: 3016
Arc revisions: 287226
Time taken: 1877ms
```