Assignment 1
Solving Sudoku as a Constraint Satisfaction Problem (CSP)

T-768-SMAI, Informed Search Methods in AI

Reykjavik University - School of Computer Science, Menntavegi 1, IS-101 Reykjavík, Iceland

Filippo Lampa
filippo23@ru.is

Damiano Pasquini
damiano23@ru.is

05. September 2023

1 Context

The objective of this first assignment is to familiarise with Constraint Satisfaction Problems (CSPs). A CSP is a type of computational problem, whose main idea is to find a solution that satisfies a set of constraints or conditions while adhering to specific variables and their domains. The core task was to implement a Simple-CSP solver which, given in input any kind of problem represented as a CSP, is able to solve it by exploiting at least one of the following backtracking-based algorithms [1], whose implementation was left as part of the assignment:

- Chronological Backtracking (**BT**)
- Backjumping search (**BJ**)
- Conflict-directed Backjumping search (**CBJ**)

In addition, also the AC-3 algorithm to force arc consistency had to be implemented, in order to see the effect on the performances of the three search algorithms. Lastly, in order to get hands-on an example of CSP, the other main task was to use those algorithms to solve some Sudoku problems. Given a 9x9 Sudoku, the constraints and the domains of the variables had to be extracted, to allow the problem to be interpreted and solved by the aforesaid CSP solver.

2 Tasks

In this section, our findings and opinions with respect to each part of the assignment will be discussed. For every section implemented, our thoughts about the approach and the algorithm will be drafted to summarise our take-home messages from this first assignment.

2.1 Domain and constraint generation

This part is strictly related to the problem being solved. Extracting the variable's domains and constraints from the original instances of the problem is essential to make it possible to approach the problem as a CSP. In this case, the problem was Sudoku, which has some well-defined rules making therefore variables and domains describable as follows

$$V = \{V_0, V_1, ..., V_{80}\}$$

$$D = \{D_0, D_1, ..., D_{80}\}$$

$$D_i = \{1, 2, ..., 9\} \; for \; i = 0, 1, ..., 80$$

where V is the set of variables and D is the domains associated to each cell. For what concerns the constraints, essential to ensure that the correct values are simultaneously assigned to the variables, those are the following:

- For each row i, all values in that row must be different.
- For each column j, all values in that column must be different.
- For each 3×3 block b, all values must be different.

For this kind of problem, finding and extracting domain values and constraints to convert the puzzles into constraint networks happens to be quite natural and straightforward, since the possible values of a specific cell are well-defined (whether it has a starting value or not) and the condition to respect is always the same (diff), applied to rows, columns and blocks. Moreover, the project came with two example files containing the constraints and domains of the benchmark puzzles. They were helpful in getting an idea about the correctness of the implemented constraint network generator, comparing the example files with the ones obtained by its execution.

2.2 Search Algorithms implementation

The majority of the code written while implementing the search algorithm within this assignment is based on the pseudo-code which can be found in [1]. No heavy modifications and optimisations have been applied during the implementation, in order to get a clearer idea about their time and space complexity as defined in literature. To ease the implementation process, the utility functions provided with the skeleton of the code have been used, such as “consistent_upto_level", useful to check the last level to break a constraint.

2.2.1 Chronological Backtracking

Chronological Backtracking (also known as Backtracking) is a first example of backtracking Search Algorithm. In its simplicity, since it may be viewed as a tree traversal, it still manages to outperform Generate-and-Test backtracking by considering partial solutions despite suffering from Thrashing. It has been implemented in a recursive fashion, trying all the values in a variable's domain and moving through the variables using recursion, until the base case is met meaning that a completely consistent branch has been found.

2.2.2 Backjumping search

The main difference with the backtracking previously described is its ability to avoid thrashing by reducing the size of the search space. The structure is similar to the backtracking one, still following a recursive approach until the same base case is reached. The main difference is the management of the index related to the current variable, which can be updated to the index of the last variable breaking the consistency (“max_check_level") in order to reduce the number of visited nodes when an inconsistent value is found.

2.2.3 Conflict-directed Backjumping

This algorithm drastically improves the performance of the search by introducing the possibility to perform multiple jumps while looking for the variable that created the inconsistency. It's possible thanks to the introduction of the concept of conflict sets, which store the indexes of the variables breaking the constraint currently taken into account. As for the other two, this was implemented in a recursive way, looping through the values of the domain and checking if a consistent one can be found. If it's found, it simply assigns it and moves to the next variable by making the recursive call, otherwise it checks within the conflict set all the variables that broke the constraint and jumps to the one deeper in the tree, bringing with it the conflict set to keep the information gained at the lower level which could be useful to execute multiple jumps. This algorithm, among the three, was arguably the most difficult one to implement, given the lack of literature and the rather high level of abstraction shown by the pseudo-code provided in [1].

2.2.4 AC-3 algorithm

This constraint propagation algorithm made it possible to impressively lower both the computation time and the number of nodes visited. The underlying idea is simple, since all it does is reducing the domain of the variables by examining each constraint one by one and eliminating values from the domains of variables that can't satisfy the constraint. It basically narrows down the possible values for each variable so that you can solve the problem more effectively. This optimisation can be activated by passing the corresponding option through the command line and the impact on the execution time is drastic.

3 Results

Let's start by taking a look at the comparison of the performances between the implemented algorithms, both with and without applying the AC-3 algorithm.

Puzzle N°	Chronological Backtracking		Backjumping		Conflict-Directed Backtracking	
	Nodes N°	Time	Nodes N°	Time	Nodes N°	Time
0	10353	0.1559	1477	0.0178	2289	0.0207
1	95776	1.4205	1864	0.0155	15615	0.1422
2	185359	2.8627	1376	0.0194	7827	0.0964
3	210449	2.6924	27265	0.3333	16793	0.2225
4	228101	2.5443	14754	0.1419	13940	0.1314
5	274481	3.2834	41638	0.4824	22841	0.2554
6	284080	2.6677	5085	0.0384	4741	0.0424
7	302347	4.6342	16896	0.2716	40228	0.5255
8	339700	8.6233	3414	0.0772	3271	0.0640
9	372342	5.9160	25138	0.4511	24661	0.4010
10	486885	11.8758	7006	0.1162	6934	0.1114
11	1123919	21.9206	37669	0.6401	32283	0.5394
12	1181915	19.2619	15282	0.1939	12776	0.1620
13	1247725	18.5327	87519	1.2554	68286	0.9775
14	1962927	27.0267	87573	1.1863	85047	1.1296
15	2320446	38.6457	75455	1.1076	45979	0.6906
16	2744247	26.6153	122346	1.3500	423337	7.3222
17	3494454	54.6960	272559	3.6846	116352	1.6673
18	8370964	161.7544	29106	0.4630	21900	0.3438
19	12632762	191.7520	27061	0.2909	34810	0.3743

Figure 1: Comparison between the three implemented algorithms before applying AC-3

Puzzle N°	Chronological Backtracking		Backjumping		Conflict-Directed Backtracking	
	Nodes N°	Time	Nodes N°	Time	Nodes N°	Time
0	82	0.0006	82	0.0007	82	0.0007
1	82	0.0006	82	0.0008	82	0.0006
2	82	0.0006	82	0.0006	82	0.0006
3	82	0.0006	82	0.0007	82	0.0007
4	82	0.0006	82	0.0007	82	0.0006
5	82	0.0006	82	0.0007	82	0.0006
6	82	0.0006	82	0.0007	82	0.0006
7	82	0.0006	82	0.0008	82	0.0007
8	82	0.0007	82	0.0007	82	0.0007
9	82	0.0006	82	0.0007	82	0.0007
10	82	0.0006	82	0.0007	82	0.0008
11	418	0.0034	344	0.0031	307	0.0026
12	82	0.0007	82	0.0007	82	0.0007
13	82	0.0006	82	0.0007	82	0.0006
14	82	0.0006	82	0.0007	82	0.0006
15	82	0.0006	82	0.0007	82	0.0006
16	82	0.0006	82	0.0008	82	0.0006
17	82	0.0006	82	0.0007	82	0.0007
18	82	0.0006	82	0.0006	82	0.0007
19	82	0.0006	82	0.0006	82	0.0007

Figure 2: Comparison between the three implemented algorithms after applying AC-3

As we can notice from the tables 1 and 4, the implemented algorithms were able to solve all the puzzles in the benchmark, all of the three returning the same solution for each puzzle, somehow proving the correctness of the output. The solution was omitted from the tables above but is shown in the appendix A. The results improve moving from Backtracking towards the CBJ algorithm as expected and as stated in literature, drastically reducing the average number of nodes visited and the computational time when passing from Backtracking to Backjumping, and furtherly improving the results when using CBJ. Moreover, if we toggle the AC-3 algorithm through command line, we cut out the majority of nodes visited and make the time needed to complete the computation almost trifling.

4 Future Works

As mentioned in Section 2, the provided implementation is almost faithful to the pseudo-code shown in [1]. The results shown in Section 3 could therefore be further improved and optimised with some shrewdnesses. Despite the CBJ algorithm seems to have already introduced many mechanisms and structures to take care of the possible optimisations and underwent huge improvements compared to the other two, there is still room for improvement. For instance, all three algorithms could introduce a kind of variable and value ordering heuristics. By applying an ordering, it could be possible to choose the most promising variable and value at each step, which could significantly reduce the search space and improve the algorithm's efficiency. This would lead to a sort of dynamic variable ordering strategy that adapts during the search based on conflict information, allowing to focus on the most relevant variables. Some other improvements like the use of different data structures (e.g. hash maps) to keep track of more info and further reduce the computational time could be adopted, but may lead to a higher memory consumption.

A Outcomes

A.1 Standard

Puzzle N°	Solver	Visited Nodes	Search Time (sec)	Result
Chronological Backtracking				
0	SolverType.BT	10353	0.1559	[1,6,5,4,3,8,7,....,1,2,3,7,4,9,8,6]
1	SolverType.BT	95776	1.4205	[3,1,2,6,7,5,4,....,9,6,4,2,8,1,5,3]
2	SolverType.BT	185359	2.8627	[4,6,3,1,9,8,5,....,4,2,3,5,9,1,6,7]
3	SolverType.BT	210449	2.6924	[5,3,1,7,2,6,4,....,9,4,6,7,8,5,1,3]
4	SolverType.BT	228101	2.5443	[8,7,4,6,9,3,1,....,5,1,4,8,2,6,9,7]
5	SolverType.BT	274481	3.2834	[8,4,7,6,1,5,3,....,9,4,5,2,3,7,8,1]
6	SolverType.BT	284080	2.6677	[9,6,7,3,8,1,4,....,8,2,6,1,5,7,9,4]
7	SolverType.BT	302347	4.6342	[2,9,3,5,4,1,6,....,8,7,6,2,5,1,3,9]
8	SolverType.BT	339700	8.6233	[2,3,4,8,6,5,9,....,7,8,1,3,6,5,2,9]
9	SolverType.BT	372342	5.9160	[3,7,1,6,5,4,8,....,8,7,5,6,9,3,2,1]
10	SolverType.BT	486885	11.8758	[5,2,4,7,8,1,6,....,4,5,6,9,3,2,1,8]
11	SolverType.BT	1123919	21.9206	[2,9,3,8,5,1,7,....,3,6,9,7,8,5,1,2]
12	SolverType.BT	1181915	19.2619	[8,4,5,6,3,2,1,....,2,8,3,5,7,4,6,1]
13	SolverType.BT	1247725	18.5327	[3,6,5,4,2,9,7,....,2,8,7,1,6,3,9,4]
14	SolverType.BT	1962927	27.0267	[6,4,8,7,3,2,9,....,9,3,4,5,6,2,7,8]
15	SolverType.BT	2320446	38.6457	[3,2,4,9,6,1,5,....,6,1,3,2,7,4,5,8]
16	SolverType.BT	2744247	26.6153	[1,7,8,2,4,3,6,....,9,6,3,7,2,1,8,5]
17	SolverType.BT	3494454	54.6960	[9,2,1,6,7,4,8,....,4,2,9,1,6,7,5,8]
18	SolverType.BT	8370964	161.7544	[4,8,1,7,3,2,6,....,6,3,4,5,7,2,9,8]
19	SolverType.BT	12632762	191.7520	[7,8,3,4,2,6,9,....,9,2,7,6,1,4,5,3]
0	SolverType.BJ	1477	0.0178	[1,6,5,4,3,8,7,....,1,2,3,7,4,9,8,6]
1	SolverType.BJ	1864	0.0155	[3,1,2,6,7,5,4,....,9,6,4,2,8,1,5,3]
2	SolverType.BJ	1376	0.0194	[4,6,3,1,9,8,5,....,4,2,3,5,9,1,6,7]
3	SolverType.BJ	27265	0.3333	[5,3,1,7,2,6,4,....,9,4,6,7,8,5,1,3]
4	SolverType.BJ	14754	0.1419	[8,7,4,6,9,3,1,....,5,1,4,8,2,6,9,7]
5	SolverType.BJ	41638	0.4824	[8,4,7,6,1,5,3,....,9,4,5,2,3,7,8,1]
6	SolverType.BJ	5085	0.0384	[9,6,7,3,8,1,4,....,8,2,6,1,5,7,9,4]
7	SolverType.BJ	16896	0.2716	[2,9,3,5,4,1,6,....,8,7,6,2,5,1,3,9]
8	SolverType.BJ	3414	0.0772	[2,3,4,8,6,5,9,....,7,8,1,3,6,5,2,9]
9	SolverType.BJ	25138	0.4511	[3,7,1,6,5,4,8,....,8,7,5,6,9,3,2,1]
10	SolverType.BJ	7006	0.1162	[5,2,4,7,8,1,6,....,4,5,6,9,3,2,1,8]
11	SolverType.BJ	37669	0.6401	[2,9,3,8,5,1,7,....,3,6,9,7,8,5,1,2]
12	SolverType.BJ	15282	0.1939	[8,4,5,6,3,2,1,....,2,8,3,5,7,4,6,1]
13	SolverType.BJ	87519	1.2554	[3,6,5,4,2,9,7,....,2,8,7,1,6,3,9,4]
14	SolverType.BJ	87573	1.1863	[6,4,8,7,3,2,9,....,9,3,4,5,6,2,7,8]
15	SolverType.BJ	75455	1.1076	[3,2,4,9,6,1,5,....,6,1,3,2,7,4,5,8]
16	SolverType.BJ	122346	1.3500	[1,7,8,2,4,3,6,....,9,6,3,7,2,1,8,5]
17	SolverType.BJ	272559	3.6846	[9,2,1,6,7,4,8,....,4,2,9,1,6,7,5,8]
18	SolverType.BJ	29106	0.4630	[4,8,1,7,3,2,6,....,6,3,4,5,7,2,9,8]
19	SolverType.BJ	27061	0.2909	[7,8,3,4,2,6,9,....,9,2,7,6,1,4,5,3]
0	SolverType.CBJ	2289	0.0207	[1,6,5,4,3,8,7,....,1,2,3,7,4,9,8,6]
1	SolverType.CBJ	15615	0.1422	[3,1,2,6,7,5,4,....,9,6,4,2,8,1,5,3]
2	SolverType.CBJ	7827	0.0964	[4,6,3,1,9,8,5,....,4,2,3,5,9,1,6,7]
3	SolverType.CBJ	16793	0.2225	[5,3,1,7,2,6,4,....,9,4,6,7,8,5,1,3]
4	SolverType.CBJ	13940	0.1314	[8,7,4,6,9,3,1,....,5,1,4,8,2,6,9,7]
5	SolverType.CBJ	22841	0.2554	[8,4,7,6,1,5,3,....,9,4,5,2,3,7,8,1]
6	SolverType.CBJ	4741	0.0424	[9,6,7,3,8,1,4,....,8,2,6,1,5,7,9,4]
7	SolverType.CBJ	40228	0.5255	[2,9,3,5,4,1,6,....,8,7,6,2,5,1,3,9]
8	SolverType.CBJ	3271	0.0640	[2,3,4,8,6,5,9,....,7,8,1,3,6,5,2,9]
9	SolverType.CBJ	24661	0.4010	[3,7,1,6,5,4,8,....,8,7,5,6,9,3,2,1]
10	SolverType.CBJ	6934	0.1114	[5,2,4,7,8,1,6,....,4,5,6,9,3,2,1,8]
11	SolverType.CBJ	32283	0.5394	[2,9,3,8,5,1,7,....,3,6,9,7,8,5,1,2]
12	SolverType.CBJ	12776	0.1620	[8,4,5,6,3,2,1,....,2,8,3,5,7,4,6,1]
13	SolverType.CBJ	68286	0.9775	[3,6,5,4,2,9,7,....,2,8,7,1,6,3,9,4]
14	SolverType.CBJ	85047	1.1296	[6,4,8,7,3,2,9,....,9,3,4,5,6,2,7,8]
15	SolverType.CBJ	45979	0.6906	[3,2,4,9,6,1,5,....,6,1,3,2,7,4,5,8]
16	SolverType.CBJ	423337	7.3222	[1,7,8,2,4,3,6,....,9,6,3,7,2,1,8,5]
17	SolverType.CBJ	116352	1.6673	[9,2,1,6,7,4,8,....,4,2,9,1,6,7,5,8]
18	SolverType.CBJ	21900	0.3438	[4,8,1,7,3,2,6,....,6,3,4,5,7,2,9,8]
19	SolverType.CBJ	34810	0.3743	[7,8,3,4,2,6,9,....,9,2,7,6,1,4,5,3]

Figure 3: Results of the algorithms without using the AC-3 for constraint propagation

A.2 AC-3

Puzzle N°	Solver	Visited Nodes	Search Time (sec)	Result
Chronological Backtracking				
0	SolverType.BT	82	0.0006	[1,6,5,4,3,8,7,....,1,2,3,7,4,9,8,6]
1	SolverType.BT	82	0.0006	[3,1,2,6,7,5,4,....,9,6,4,2,8,1,5,3]
2	SolverType.BT	82	0.0006	[4,6,3,1,9,8,5,....,4,2,3,5,9,1,6,7]
3	SolverType.BT	82	0.0006	[5,3,1,7,2,6,4,....,9,4,6,7,8,5,1,3]
4	SolverType.BT	82	0.0006	[8,7,4,6,9,3,1,....,5,1,4,8,2,6,9,7]
5	SolverType.BT	82	0.0006	[8,4,7,6,1,5,3,....,9,4,5,2,3,7,8,1]
6	SolverType.BT	82	0.0006	[9,6,7,3,8,1,4,....,8,2,6,1,5,7,9,4]
7	SolverType.BT	82	0.0006	[2,9,3,5,4,1,6,....,8,7,6,2,5,1,3,9]
8	SolverType.BT	82	0.0007	[2,3,4,8,6,5,9,....,7,8,1,3,6,5,2,9]
9	SolverType.BT	82	0.0006	[3,7,1,6,5,4,8,....,8,7,5,6,9,3,2,1]
10	SolverType.BT	82	0.0006	[5,2,4,7,8,1,6,....,4,5,6,9,3,2,1,8]
11	SolverType.BT	418	0.0034	[2,9,3,8,5,1,7,....,3,6,9,7,8,5,1,2]
12	SolverType.BT	82	0.0007	[8,4,5,6,3,2,1,....,2,8,3,5,7,4,6,1]
13	SolverType.BT	82	0.0006	[3,6,5,4,2,9,7,....,2,8,7,1,6,3,9,4]
14	SolverType.BT	82	0.0006	[6,4,8,7,3,2,9,....,9,3,4,5,6,2,7,8]
15	SolverType.BT	82	0.0006	[3,2,4,9,6,1,5,....,6,1,3,2,7,4,5,8]
16	SolverType.BT	82	0.0006	[1,7,8,2,4,3,6,....,9,6,3,7,2,1,8,5]
17	SolverType.BT	82	0.0006	[9,2,1,6,7,4,8,....,4,2,9,1,6,7,5,8]
18	SolverType.BT	82	0.0006	[4,8,1,7,3,2,6,....,6,3,4,5,7,2,9,8]
19	SolverType.BT	82	0.0006	[7,8,3,4,2,6,9,....,9,2,7,6,1,4,5,3]
0	SolverType.BJ	82	0.0007	[1,6,5,4,3,8,7,....,1,2,3,7,4,9,8,6]
1	SolverType.BJ	82	0.0008	[3,1,2,6,7,5,4,....,9,6,4,2,8,1,5,3]
2	SolverType.BJ	82	0.0006	[4,6,3,1,9,8,5,....,4,2,3,5,9,1,6,7]
3	SolverType.BJ	82	0.0007	[5,3,1,7,2,6,4,....,9,4,6,7,8,5,1,3]
4	SolverType.BJ	82	0.0007	[8,7,4,6,9,3,1,....,5,1,4,8,2,6,9,7]
5	SolverType.BJ	82	0.0007	[8,4,7,6,1,5,3,....,9,4,5,2,3,7,8,1]
6	SolverType.BJ	82	0.0007	[9,6,7,3,8,1,4,....,8,2,6,1,5,7,9,4]
7	SolverType.BJ	82	0.0008	[2,9,3,5,4,1,6,....,8,7,6,2,5,1,3,9]
8	SolverType.BJ	82	0.0007	[2,3,4,8,6,5,9,....,7,8,1,3,6,5,2,9]
9	SolverType.BJ	82	0.0007	[3,7,1,6,5,4,8,....,8,7,5,6,9,3,2,1]
10	SolverType.BJ	82	0.0007	[5,2,4,7,8,1,6,....,4,5,6,9,3,2,1,8]
11	SolverType.BJ	344	0.0031	[2,9,3,8,5,1,7,....,3,6,9,7,8,5,1,2]
12	SolverType.BJ	82	0.0007	[8,4,5,6,3,2,1,....,2,8,3,5,7,4,6,1]
13	SolverType.BJ	82	0.0007	[3,6,5,4,2,9,7,....,2,8,7,1,6,3,9,4]
14	SolverType.BJ	82	0.0007	[6,4,8,7,3,2,9,....,9,3,4,5,6,2,7,8]
15	SolverType.BJ	82	0.0007	[3,2,4,9,6,1,5,....,6,1,3,2,7,4,5,8]
16	SolverType.BJ	82	0.0008	[1,7,8,2,4,3,6,....,9,6,3,7,2,1,8,5]
17	SolverType.BJ	82	0.0007	[9,2,1,6,7,4,8,....,4,2,9,1,6,7,5,8]
18	SolverType.BJ	82	0.0006	[4,8,1,7,3,2,6,....,6,3,4,5,7,2,9,8]
19	SolverType.BJ	82	0.0006	[7,8,3,4,2,6,9,....,9,2,7,6,1,4,5,3]
0	SolverType.CBJ	82	0.0007	[1,6,5,4,3,8,7,....,1,2,3,7,4,9,8,6]
1	SolverType.CBJ	82	0.0006	[3,1,2,6,7,5,4,....,9,6,4,2,8,1,5,3]
2	SolverType.CBJ	82	0.0006	[4,6,3,1,9,8,5,....,4,2,3,5,9,1,6,7]
3	SolverType.CBJ	82	0.0007	[5,3,1,7,2,6,4,....,9,4,6,7,8,5,1,3]
4	SolverType.CBJ	82	0.0006	[8,7,4,6,9,3,1,....,5,1,4,8,2,6,9,7]
5	SolverType.CBJ	82	0.0006	[8,4,7,6,1,5,3,....,9,4,5,2,3,7,8,1]
6	SolverType.CBJ	82	0.0006	[9,6,7,3,8,1,4,....,8,2,6,1,5,7,9,4]
7	SolverType.CBJ	82	0.0007	[2,9,3,5,4,1,6,....,8,7,6,2,5,1,3,9]
8	SolverType.CBJ	82	0.0007	[2,3,4,8,6,5,9,....,7,8,1,3,6,5,2,9]
9	SolverType.CBJ	82	0.0007	[3,7,1,6,5,4,8,....,8,7,5,6,9,3,2,1]
10	SolverType.CBJ	82	0.0008	[5,2,4,7,8,1,6,....,4,5,6,9,3,2,1,8]
11	SolverType.CBJ	307	0.0026	[2,9,3,8,5,1,7,....,3,6,9,7,8,5,1,2]
12	SolverType.CBJ	82	0.0007	[8,4,5,6,3,2,1,....,2,8,3,5,7,4,6,1]
13	SolverType.CBJ	82	0.0006	[3,6,5,4,2,9,7,....,2,8,7,1,6,3,9,4]
14	SolverType.CBJ	82	0.0006	[6,4,8,7,3,2,9,....,9,3,4,5,6,2,7,8]
15	SolverType.CBJ	82	0.0006	[3,2,4,9,6,1,5,....,6,1,3,2,7,4,5,8]
16	SolverType.CBJ	82	0.0006	[1,7,8,2,4,3,6,....,9,6,3,7,2,1,8,5]
17	SolverType.CBJ	82	0.0007	[9,2,1,6,7,4,8,....,4,2,9,1,6,7,5,8]
18	SolverType.CBJ	82	0.0007	[4,8,1,7,3,2,6,....,6,3,4,5,7,2,9,8]
19	SolverType.CBJ	82	0.0007	[7,8,3,4,2,6,9,....,9,2,7,6,1,4,5,3]

Figure 4: Results of the algorithms using the AC-3 for constraint propagation

References

[1] Ian Miguel and Qiang Shen. Solution techniques for constraint satisfaction problems: Foundations. *Artificial Intelligence Review*, 15:243–267, 2001.