Assignment 2: Constraint Satisfaction Problems

Ole C. Eidheim[*]

8th September 2025

Deadline: 26.09.2025, 23:59 hrs

1 Overview

In this assignment, you will implement backtracking search and AC-3 for Constraint Satisfaction Problems. You will then use these algorithms to solve Sudoku boards of varying difficulty.

You are required to implement the algorithms, properly document the source code, and write a short PDF report. However, to make sure you start on the right track, we provide you with a *code handout* for Python. This code suggests a code structure, and populates the necessary data structures for two example CSPs, namely the map coloring problem and Sudoku. You are welcome to use and adapt this code in your submission, but it is not mandatory.

We recommend you implement the backtracking search algorithm first, and test this algorithm on the map coloring problem.

2 CSPs, Backtracking and AC-3

Using backtracking search to solve CSPs is described in **Chapter 5.3** of the textbook (AIMA 4th Ed., Global edition), and the pseudocode is shown in **Figure 5.5 (p. 176)**. The pseudocode references three functions that are not concretely specified in the textbook, namely: Select-Unassigned-Variable, Order-Domain-Values, and Inference.

For this assignment, it is sufficient to implement the following:

1. Select-Unassigned-Variable: return any unassigned variable.
2. Order-Domain-Values: any ordering for the given variable is fine.
3. Inference: no inference needs to be performed during the backtracking search. You can instead apply the AC-3 algorithm (described **Figure 5.3, (p. 171)**) prior to running the backtracking search. For simpler Sudoku problems this will reduce the variable domains to one value each.

2.1 Code handout

The *code handout* suggests which data structures to use to represent the CSP. As mentioned in **Chapter 5.1**, a Constraint Satisfaction Problem can be represented as three components X, D, and C:

- X: set of *variables*.
- D: set of domains, one for each variable. Each domain defines which values a variable can be assigned.

[*]updated from the work of Måløy and Sánchez-Díaz

- *C*: set of *constraints*, where a constraint can be a set of legal pairs of values for two given variables.

In the *code handout*, the following data structures are used:

- *variables*: list of variable names.
- *domains*: dictionary that associates a variable name with a set of legal values.
- *binary_constraints*: dictionary that associates constrained variable pairs to a set of legal value pairs. In the *code handout*, *binary_constraints* is populated from a list of *edges* containing variable pairs that must not be assigned the same value. You will not need to use non-binary constraints in this assignment.

2.1.1 An example

The construction of the map colouring CSP from the textbook (Chapter 5.1.1, p. 165) is shown below as demonstrated in the file *map_coloring.py* in the *code handout*. The contents of the data structures are also presented, where dictionaries and sets use curly brackets {}, lists use square brackets [], and tuples use parentheses ().

```
>>> from csp import CSP
>>> variables = ['WA', 'NT', 'Q', 'NSW', 'V', 'SA', 'T']
>>> csp = CSP(
...     variables=variables,
...     domains={variable: {'red', 'green', 'blue'}
...              for variable in variables},
...     edges=[
...         ('SA', 'WA'),
...         ('SA', 'NT'),
...         ('SA', 'Q'),
...         ('SA', 'NSW'),
...         ('SA', 'V'),
...         ('WA', 'NT'),
...         ('NT', 'Q'),
...         ('Q', 'NSW'),
...         ('NSW', 'V')
...     ]
... )

>>> print(csp.variables)
['WA', 'NT', 'Q', 'NSW', 'V', 'SA', 'T']

>>> print(csp.domains)
{'WA': {'blue', 'red', 'green'},
 'NT': {'blue', 'red', 'green'},
 'Q': {'blue', 'red', 'green'},
 'NSW': {'blue', 'red', 'green'},
 'V': {'blue', 'red', 'green'},
 'SA': {'blue', 'red', 'green'},
 'T': {'blue', 'red', 'green'}}

>>> print(csp.binary_constraints)
{('SA', 'WA'): {('blue', 'green'),
                ('blue', 'red'),
                ('green', 'blue'),
                ('green', 'red'),
                ('red', 'blue'),
                  ('red', 'green')},
('SA', 'NT'): {('blue', 'green'),
                ('blue', 'red'),
                ('green', 'blue'),
                ('green', 'red'),
                ('red', 'blue'),
                ('red', 'green')},
('SA', 'Q'): {('blue', 'green'),
               ('blue', 'red'),
               ('green', 'blue'),
               ('green', 'red'),
               ('red', 'blue'),
               ('red', 'green')},
('SA', 'NSW'): {('blue', 'green'),
                 ('blue', 'red'),
                 ('green', 'blue'),
                 ('green', 'red'),
                 ('red', 'blue'),
                 ('red', 'green')},
('SA', 'V'): {('blue', 'green'),
               ('blue', 'red'),
               ('green', 'blue'),
               ('green', 'red'),
               ('red', 'blue'),
               ('red', 'green')},
('WA', 'NT'): {('blue', 'green'),
                ('blue', 'red'),
                ('green', 'blue'),
                ('green', 'red'),
                ('red', 'blue'),
                ('red', 'green')},
('NT', 'Q'): {('blue', 'green'),
               ('blue', 'red'),
               ('green', 'blue'),
               ('green', 'red'),
               ('red', 'blue'),
               ('red', 'green')},
('Q', 'NSW'): {('blue', 'green'),
                ('blue', 'red'),
                ('green', 'blue'),
                ('green', 'red'),
                ('red', 'blue'),
                ('red', 'green')},
('NSW', 'V'): {('blue', 'green'),
                ('blue', 'red'),
                ('green', 'blue'),
                ('green', 'red'),
                ('red', 'blue'),
                ('red', 'green')}}
```

The variables are stored as a list of strings in the `csp` object. All the seven variables have the same domain: *red, green* and *blue*. The printout of `csp.binary_constraints` shows the legal value pairs for the edges, but the binary constraints are directional, same as is presented in the book. The constraint for `('WA', 'SA')` is not stored in `csp.binary_constraint` even though it is the same as `('SA', 'WA')`. If you decide to use the *code handout*, you have to take this into account or modify `csp.binary_constraints` to your liking.

3 Sudoku boards as CSPs

The *code handout* already includes a file, `sudoku.py`, which reads a Sudoku board from a text file and constructs a CSP for solving the Sudoku board. There is also a function to output the solution as a Sudoku board called `print_solution(solution)`.

3.1 What you should implement

If you choose to use the *code handout*, your task is to implement backtracking search and AC-3 in the methods `CSP.backtracking_search()` and `CSP.ac_3()` in the `csp.py` file. After this, you can run the files `map_coloring.py` and `sudoku.py` to output the solutions. It is suggested that you complete the backtracking search first and test your implementation on the `map_coloring.py` file.

In `sudoku.py`, AC-3 is used to reduce the domains prior to running the backtracking search. It is not expected that you use the AC-3 algorithm during the backtracking search.

You must solve the four Sudoku boards illustrated in Figures 1a-1d, and the results should be included in your PDF report.

4 Deliverables

1. Source files containing well-commented code for backtracking search and AC-3 that are able to solve the Sudoku boards.
2. A **single** PDF report containing:
   (a) Your program's solution for each of the Sudoku boards shown in Figures 1a-1d.
   (b) The domains after running the AC-3 algorithm for each of the four boards.
   (c) The number of times your `backtrack()` function was *called*, and the number of times your `backtrack()` function returned failure, for each of the four boards.
   (d) The runtime of the backtracking search algorithm for each of the four boards.
   (e) The total runtime of the AC-3 and backtracking search algorithms for each of the four boards.
   (f) Brief discussion of your results. For example, explain why the AC-3 algorithm drastically decrease the total runtime when solving some Sudoku problems.

Recommendations

Make sure that your code is well documented. **Deliver the code and the PDF report as two separate files. If your code project consists of many files, the project should be archived in a ZIP file.**

		4		3			5	
6		9	4					
		5	1			4	8	9
				6		9	3	
3			8		7			2
	2	6		4				
4	5	3			9	6		
					4	7		5
	9			5		2		

(a) Easy board (`easy.txt`)

				3			4	
1		9	7					
			8	5	1		7	
		2	6		7	8	3	
9		6		1		2		7
	3	1	5		2	9		
	1		3	6	9			
					5	7		3
	9			7				

(b) Medium board (`medium.txt`)

1		2		4				7
				8				
		9	5			3		4
			6		7	9		
5	4						2	6
		6	4		5			
7		8			3	4		
				1				
2				6		5		9

(c) Hard board (`hard.txt`)

		1			7			
6			4			3		
				3			6	4
3	8			7	6			
							3	6
2	7			1	5			
				2			5	1
7			1			2		
		8			9			

(d) Very hard board (`veryhard.txt`)

Figure 1: Sudoku boards you will solve