Assignment 4: Constraint Satisfaction Problems

Håkon Måløy   Xavier F.C. Sánchez-Díaz[*]

28th September 2022

Deadline: 28.10.2022, 23:59 hrs

1 Overview

In this assignment, you will implement a general solver for Constraint Satisfaction Problems using *backtracking search* and the arc-consistency algorithm AC-3. You will then use this program to solve Sudoku boards of varying difficulty.

You will be required to implement the CSP solver yourselves, properly documenting your source code and attaching a report as well. However, to make sure you start on the right track, we provide you with a *skeleton code* for Python. This code suggests an interface for your CSP solver and initialises and populates the necessary data structures to represent each CSP instance. You are welcome to use and adapt this code in your submission, but it is not mandatory.

2 CSPs, Backtracking and AC-3

Using backtracking search to solve CSPs is described in **Chapter 5.3** of the textbook (AIMA 4th Ed.), and summarised as pseudocode in **Figure 5.5 (p. 176)**. The pseudocode references three functions that have not been specified any further in the textbook, namely: SELECT-UNASSIGNED-VARIABLE, ORDER-DOMAIN-VALUES, and INFERENCE.

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

1. SELECT-UNASSIGNED-VARIABLE: return any unassigned variable
2. ORDER-DOMAIN-VALUES: any ordering for the given variable is fine
3. INFERENCE: this is the **arc consistency algorithm AC-3** which is described in **Figure 5.3, (p. 171)**

2.1 Skeleton code

The python *skeleton code* suggests which data structures to use to represent the CSP. As mentioned in **Chapter 5.1**, a Constraint Satisfaction Problem can be represented as a tuple (X, D, C) where:

- X is a set of *variables*
- D is an assignment of a *domain* to each variable, where a domain is the set of legal values for the given variable,

[*]updated from the work of Håkon Måløy

- C is the set of *constraints*, where a constraint is a set of legal pairs of values for two given variables.

In the skeleton code, we use the following data structures:

- X is a list of variable names.
- D is a dictionary that associates a variable name with its corresponding list of legal values
- C is a dictionary that associates a variable name i with *another* dictionary, where this second dictionary contains the constraints affecting variable i. Specifically, the second dictionary associates the name of another variable j (given $i \neq j$) with an array of *legal pairs of values* for the pair of variables (i, j).

You can look at the code type-hinting for additional information.

2.1.1 An example

Consider the following printout from Python which shows the contents of these data structures when the map colouring CSP from the textbook (Chapter 5.1.1, p. 165) is implemented using the skeleton code:

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

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

>>> print(csp.constraints)
{'WA': {'SA': [('red', 'green'), ('red', 'blue'), ('green', 'red'),
('green', 'blue'), ('blue', 'red'), ('blue', 'green')],
'NT': [('red', 'green'), ... ... ...
```

And many more lines of output.

We can notice a few things. For example, that variables are strings, and they are contained as a list in the object `csp`. Notice also how all seven variables have the same domain: *red, green* and *blue*. The excerpt from `csp.constraints` shows that `WA` is connected to `SA` and `NT`, and the 2-level depth of the dictionaries.

You can differentiate between the containers by their delimiters: dictionaries use curly brackets $\{\}$, while lists use square brackets [] and tuples use parentheses $()$.

In the skeleton code there is some sample code that sets up the map colouring CSP shown above. The map colouring CSP can be a helpful way to test your code as you implement backtracking and AC-3.

3 Sudoku boards as CSPs

The skeleton code includes functions to read Sudoku boards from text files and to represent these boards as CSPs called `create_sudoku_csp(filename)`. There is also a function to print it in a readable format called `print_sudoku_solution(solution)`

3.1 What you should implement

If you choose to use the skeleton code, then your task is to implement backtracking search and AC-3 at the indicated locations in the skeleton code. After this, your Sudoku solver should be ready to go.

To demonstrate that your CSP solver works, your program should solve the four Sudoku boards illustrated in Figures 1a-1d, and the results should be included in your report.

		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

4 Deliverables

1. Source files containing well-commented code for a CSP solver based on backtracking search and AC-3 that is able to solve Sudoku boards.
2. A **single** PDF containing:
   (a) Your program's solution for each of the boards shown above.
   (b) 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.
   (c) Brief comments about the results in the consistency checks (point (b) just above). For example, you could compare the performance of the algorithm on the different boards, or try to relate the results to the theory, or something else that you find relevant. What is important here is that you are able to show understanding through reflection about these results.

Recommendations

Make sure that your code is well documented. **Deliver the code and the PDF as separate files in your hand in. If your project consists on many files, do not include the report in your ZIP (or other archive format) file; upload them separately.**