AI 1: Lab Assignment 2

Constraint Satisfaction Problems

Instructions

1. This is a ***team assignment*** to be completed in a team of two or three students and consists of a Python coding element and a report.
2. Answer all questions and ***formulate your answers concisely and clearly***.
3. Remember ***it is not allowed to use code not developed by your team***. Doing so constitutes plagiarism. If you use external sources (videos, websites, discussion fora, etc.) to develop your code make sure to ***clearly refer to them in your report and in your code***. Failing to disclose these sources constitutes again an instance of plagiarism.
4. ***Reports need to be written in LaTeX*** and submitted as a pdf ***separately from the code***. A template is available on Brightspace (under 'Course overview'). The text should be ***authored by your team and your team only***.
5. Comply with the deadline provided on Brightspace. ***Deadlines are strict*** (penalties apply for late submissions, see below).
6. The code should be submitted in a ***separate zip file***.

Assessment The grade of this lab assignment will counts for ***10% of your final grade***. We ***subtract*** 2^{n-1} grade points for a submission that is between $n-1$ and n days late ($n \geq 1$).

Introduction

In this lab you will work with a constraint satisfaction solver that is already largely implemented except for its heuristics. You can download it from Brightspace. In order to solve a CSP with it, the variables, domains and constraints must be specified in a separate file that imports the solver. Constraints can be any Python interpretable strings. There are examples provided.

The basic outline of the solver is shown in pseudocode 1 below. An intuitive way to think about the solver is to imagine a search tree that is explored depth first in which a new branch is generated each time a variable gets assigned a value. The solver is able to print this search tree when setting the `verbose` variable to true.

Keep in mind that the program was designed for this course to be easily understandable and not to be as efficient as possible. In particular, the number of times the functions `node_consistency()` and `arc_consistency()` have to evaluate constraints may not be representative for other solvers.

⚠ **Warning:** For efficiency reasons the `keep_node` method only resolves node-inconsistencies that were caused by assigning a value to the current variable. It may not make the whole problem node consistent if there already were node inconsistencies before. If you want to make sure that the problem stays node consistent at all times you must set `init_node` and `keep_node` to true. The same holds for `keep_arc`.

Algorithm 1 The basic structure of the csp solver .

```
procedure SolveCSPRec(csp)
    if all variables got assigned a value then
        print the current solution
        return
    if csp.heuristic = mrv then
        var ⟵ choose a variable using the minimal remaining value heuristic
    else if csp.heuristic = deg then
        var ⟵ choose a variable using the degree heuristic
    else
        var ⟵ the next variable
    for all d in the domain of var do
        assign d to var
        if there is a constraint that evaluates to False then
            skip to next iteration
        if csp.keep_node and makeNodeConsistent(csp) results in empty-domain then
            skip to next iteration
        if csp.keep_arc and makeArcConsistent(csp) results in empty-domain then
            skip to next iteration
        SolveCSPRec(csp)
        backtrack()  /*unassigned var and reverse any domain reductions done in this iteration*/

procedure solveCSP(csp)
    Input: A constraint satisfaction problem csp = ⟨V, D, C⟩ with a set of variables V, domains D
and constraints C. As well as the heuristic and propagation techniques to be used.
    if csp.init_node = True then
        make the csp node-consistent
    if csp.init_arc = True then
        make the csp arc-consistent
    SolveCSPRec(csp)
```

1 Forward-checking [10 pts]

To help you understand the workings of the solver in depth, search tree printing can be enabled with the `verbose` variable. By adding `init_node`, `init_arc`, `keep_node` and `keep_arc` you will be able to use the different forward-checking mechanisms and compare their respective search trees. In this exercise, you will run the problem specified in the `equations.py` file. The simplicity of this problem will allow easy comparison of the different mechanisms. Printing will be harder to follow for more complex exercises.

The problem asks you to consider a set of three variables A, B and C with the domains $D_A = \{1, 2\}$, $D_B = \{1, 2, 3\}$ and $D_C = \{1, 2, 3, 4\}$. The variables are constrained by the following equations:

$$\begin{aligned} 2 * A &= C \\ A &\neq B \\ A + B &\leq C \end{aligned}$$

Questions:

a) Run the solver with and without keeping the problem node consistent. Use the search tree to explain how node-consistency is helping to reduce the amount of visited states for this problem.

b) Run the solver again while making the problem initially arc consistent and keeping it node and arc consistent. Use the search tree to explain how arc-consistency is helping to reduce the amount of visited states for this problem.

2 Implementing Heuristics [30 pts]

Questions:

a) Extend the solver with a minimal remaining value heuristic (mrv) by completing the function "mrv_heuristic()". Bear in mind that it might reach a point in which the mrv has to evaluate multiple variables with the same lowest domain size. In this case, return the first within the list of variables.

b) Similarly, extend the solver with a degree heuristic (deg) by completing the function "degree_heuristic()". Bear in mind that it might reach a point in which the mrv has to evaluate multiple variables with the same lowest domain size. In this case, return the first within the list of variables.

Run the CSP described in `testing_problem.py` and check your results against Table 1.

heuristic	keep_node	keep_arc	states visited	constraints evaluated
mrv	False	False	440	3640
mrv	True	False	133	3980
mrv	True	True	53	7755
deg	False	False	137	919
deg	True	False	54	1194
deg	True	True	29	668

Table 1: The number of states visited and constraints evaluated your solver should have for `testing_problem.py` after implementing the heuristics.

3 Analysing Problems [20 pts]

3.1 N queens [10 pts]

Read the implementation of the N Queens problem found in `queen.py`.

Questions:

a) Analyze each of the different heuristics and propagation techniques for this problem in isolation and describe for each of them why they are (not) helpful. Also consider 3 combinations of propagation techniques and heuristics you expect to be most efficient and reflect on their performance.

b) If you implement Python's shuffle function in the list of variables or constraints, you may notice the results differ slightly between runs. Why doesn't the solver give the exact same results even though the problem has not changed? Does this effect the best possible heuristic? Why?

3.2 Harry Potter [10 pts]

In the book "Harry Potter and the Philosopher's Stone" (Rowling, 1998), we encounter what is called "the potion puzzle", in which you have to decipher a riddle to discover which of the potions is poisonous and which is not. Several constraints that will aid you to solve the puzzle are hidden within the riddle. This riddle is implemented in the file `harry_potter.py`.

Questions:

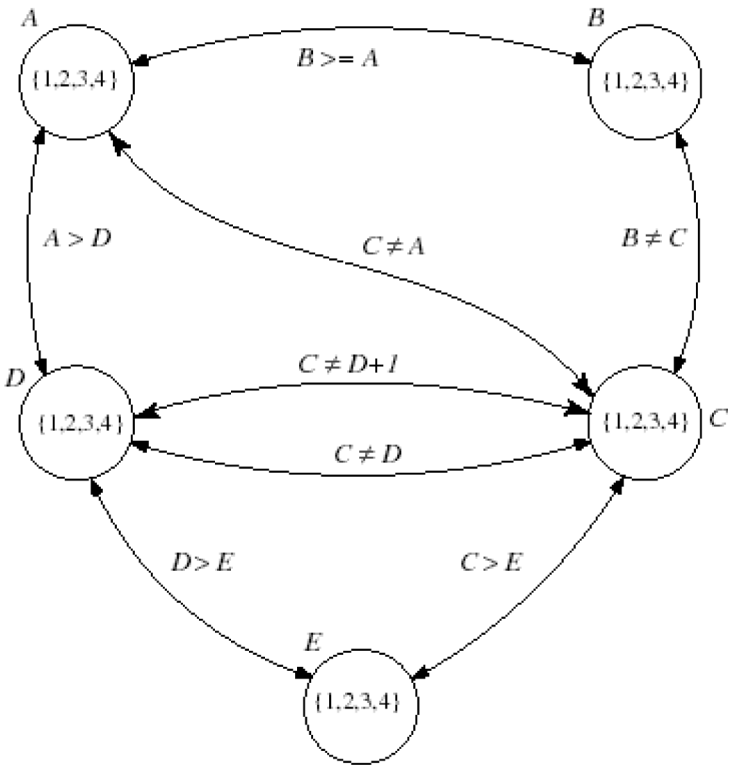
a) Analyze each of the different heuristics and propagation techniques for this problem in isolation and describe for each of them why they are (not) helpful. Also consider 3 combinations of propagation techniques and heuristics you expect to be most efficient and reflect on their performance.

b) Consider the constraints that specify how many of each potion-types exist. Why do they not prevent the solver from keep exploring states in which there are for example more than two wines? How is that influencing the efficiency of the solver?

4 Writing CSP descriptions [30 pts]

4.1 Constraint graph [15 pts]

Consider the following constraint graph:



Questions:

a) Create a working file for this problem and determine all solutions.

b) Analyze each of the different heuristics and propagation techniques for this problem in isolation and describe for each of them why they are (not) helpful. Also consider 3 combinations of propagation techniques and heuristics you expect to be most efficient and reflect on their performance.

4.2 Cryptarithmetic puzzles [15 pts]

In *cryptarithmetic puzzles* some words (consisting of at most 10 letters in total) are used in an arithmetic expression. Each letter stands for a unique digit. The same letter stands for the same digit throughout the puzzle. Different digits are different letters. There are no leading zeros.

Consider the *cryptarithmetic puzzle* `SEND+MORE=MONEY`, which has the unique solution `D=7, E=5, M=1, N=6, O=0, R=8, S=9, Y=2`.

Questions:

a) Create working file for one of the following examples of cryptarithmetic puzzles and determine all solutions.

```
UN + UN + NEUF = ONZE
ONE + NINE + TWENTY + FIFTY = EIGHTY
I + GUESS + THE + TRUTH = HURTS
```

b) Analyze each of the different heuristics and propagation techniques for this problem in isolation and describe for each of them why they are (not) helpful. Also consider 3 combinations of propagation techniques and heuristics you expect to be most efficient and reflect on their performance.

5 Bonus [0 pts *but* mystery prize]

As a bonus exercise you can specify your own csp. This could be any problem that interests you, real world or not. Have a look at the other csp's as inspirations. You can be as creative as you want! :)

Submit the .py file before Monday the 29th on the discussion board on Brightspace. Also add a short .pdf in which you motivate your design choices and explain the result. Mention your group number in the title.

This exercise wont be graded but after the deadline students will be able to look at the submission and vote for the one they like most. There will be a prize for the winners!

References

Rowling, J. K. (1998). *Harry potter and the philosopher's stone* (Vol. 1). Arthur A. Levine Books.