National University of Singapore
School of Computing
CS3243 Introduction to Artificial Intelligence

Project 2.1: Constraint Satisfaction Problems

Issued: 16 September 2024 Due: 13 October 2024, 2359hrs

1 Overview

In this project, you will **implement a solver for a general constraint satisfaction problem (CSP)**. You are to use the **backtracking algorithm**, along with any other optimisations you deem necessary (e.g., forward checking, AC-3, etc.).

This project is worth 3% of your module grade.

1.1 General Project Requirements

The general project requirements are as follows:

- **Individual** project: Discussion within a team is allowed, but **no code should be shared**
- Python Version: ≥ **3.12**
- Deadline: **13 October 2024, 2359hrs**
- Submission: Via **Coursemology** – for details, refer to the Coursemology Guide on Canvas

1.2 Academic Integrity and Late Submissions

Note that any material that does not originate from you (e.g., is taken from another source) should not be used directly. You should do up the solutions on your own. Failure to do so constitutes plagiarism. Sharing of code between individuals is also strictly not allowed. Students found plagiarising will be dealt with seriously.

For late submissions, there will be a 20% penalty for submissions received within 24 hours after the deadline, 50% penalty for submissions received between 24-48 hours after the deadline, and 100% penalty for submissions received after 48 hours after the deadline. For example, if you submit the project 30 hours after the deadline and obtain a score of 92%, a 50% penalty applies, and you will only be awarded 46%.

2 Project 2.1: CSP Solver

2.1 Task: Backtracking algorithm

2.1.1 Functionality

You will be given a set of variables, and their associated domains. You will also be given some constraints between the variables. The objective is to find a set of variable assignments such that all constraints are satisfied.

2.1.2 Input

The input is a `dict`, which is guaranteed to have two keys: `domains` and `constraints`.

The value corresponding to `domains` will be another `dict` containing `str: list[int]` elements. The keys of this dictionary correspond to names of the variables, and the values correspond to the domain of each variable.

The value corresponding to `constraints` will be yet another `dict` containing `(str, str): <function>` elements, where each element corresponds to a binary constraint. Thus, the keys for each element in this `constraints` dictionary corresponds to a pair of variables defining the scope of a constraint, while its value corresponds to the relationship binding the two variables specified in the key, which is specified in the form of a `lambda` function, `lambda x, y: <logical expression>`. This function will take in two integers, and return **`True`** if the constraint is satisfied, and **`False`** otherwise.

An example input is given below in Section 2.1.5.

2.1.3 Requirements

You must define a Python function, `solve_CSP(input)`. This function takes in the input dictionary described above and must return a `dict` with `[str: int]` elements, where each key, `str`, corresponds to the given variable symbol, and the value `int` corresponds to the assigned value. The output of `solve_CSP` must correspond to a complete and consistent solution to the given CSP. When no such solution exists, then the output should be **`None`**. You may define any other convenience functions or classes you require.

2.1.4 Assumptions

You may assume the following:

- All variable symbols have type `str`, and all domain values have type `int`.
- All constraints will be binary constraints.
- There will be **at most one constraint between each pair of variables**. That means that if `("A", "B")` appears as a key in the `constraints` dictionary, `("B", "A")` will NOT be included as another key.
- No exceptions will be thrown by any constraint function, as long as all function inputs satisfy their associated variable domains. More specifically, you do not have to perform a division by zero exception check when evaluating the constraints.
- If a key `("A", "B")` appears in the constraints dictionary, the variable `"A"` will be the first argument of the constraint function, and the variable `"B"` will be the second argument.
- If the CSP has no solution, the function should return `None`.

2.1.5 Example

An example input is the following:

```
input = {

'domains' : {
                'A' : [ 1,  2,  3,  4,  5 ],
                'B' : [ 2,  3,  4,  5,  6 ],
                'C' : [ 3,  4,  5,  6,  7 ],
                'D' : [ 5,  7,  9, 11, 13 ]
            },

'constraints'  : {
                   ('A','B') : lambda a,b : a+b == 8 and a >= b,
                   ('B','C') : lambda b,c : b <= c/2,
                   ('C','D') : lambda c,d : (c+d) % 2 == 0
                 }

}
```

One possible solution to the above CSP is the following:

```
output = { 'A' : 5,  'B' : 3,  'C' : 7,  'D' : 5 }
```

3 Grading

3.1 Grading Rubrics (Total: 3 marks)

Your code will be graded based on the following criteria.

- Correct implementation of backtracking algorithm by passing test cases correctly. (0.6m)
- Efficient implementation of backtracking algorithm by passing all test cases within the time limit. (2.4m)

3.2 Grading Details

There are a total of sixteen (16) test cases. Seven of the test cases are public, while nine test cases are private.

The marks distribution of the test cases are as follows:

- Public test cases 1 - 4: 0.15m each
- Public test cases 5 - 7: 0.2m each
- Private test cases 1 - 9: 0.2m each

Refer to Canvas > CS3243 > Files > Project > **Coursemology_guide.pdf** for submission details.

Refer to Canvas > CS3243 > Files > Projects > Project 2 > Project 2.1 > **p2.1_public_testcases.py** for the public test cases.

For tips on how to use the public test cases, please refer to Appendix Section 4.2.

---

4 Submission

4.1 Details for Submission via Coursemology

Refer to **Canvas** > **CS3243** > **Files** > **Projects** > **Coursemology_guide.pdf** for submission details.

---

5 Appendix

5.1 Allowed Libraries

The following libraries are allowed:

- Data structures: queue, collections, heapq, array, copy, enum, string
- Math: numbers, math, decimal, fractions, random, numpy
- Functional: itertools, functools, operators
- Types: types, typing

5.2 Tips and Hints

1. Public test cases 1 – 4 test for correctness. Minimal optimisations are required in order to pass these test cases.
2. The remaining test cases check for optimisations in your backtracking code. You are advised to implement optimisations such as inference and use some heuristics that were taught in the lectures on CSP.
3. Public test cases 5 – 7, and the private test cases, are adversarial test cases. If you are failing many of the private test cases, you may find it helpful to examine public test cases 5 – 7, then determine why those test cases run quickly ($<$ 0.1s) when certain optimisations are implemented but take a long time to run ($>$ 30s) otherwise.
4. AC-3 is **unnecessary** for this project. In fact, AC-3 may even slow down your code in some of the test cases. (You may think about the time complexity of AC-3 algorithm and determine the situations where AC-3 would correspond to a pessimisation for the backtracking algorithm.)