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

Project 2.1: Constraint Satisfaction Problems

Issued: 19 February 2024 Due: 10 March 2024

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 (eg: 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, but you are *allowed to consult P2ST (ChatGPT) App*.
- Python Version: $\geq$ **3.10**
- Deadline: **10 March 2024, 2359**
- Submission: Via **Coursemology**. Refer to Canvas → CS3243 → Files → Projects → coursemology_guide.pdf for details.

More details can be found in Section 4.

1.2 Academic Integrity and Late Submissions

Note that any material that does not originate from you (e.g., is taken from another source, with the exception of the P2ST (ChatGPT) App) should not be used directly. You should do up the solutions on your own. Failure to do so constitutes plagiarism. Sharing of materials between individuals is also strictly not allowed. Students found plagiarising or sharing their code 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

You will be given a `dict`. The dict 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 corresponds to names of the variables, and the values corresponds 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 2.1.5.

2.1.3 Requirements

You will define a python function, named `solve_CSP(input)`. This function takes in the input dictionary described above, and must return a `dict` with `[str : int]` elements, where the key `str` corresponds to the given variable symbols, and the value `int` corresponds to an assignment. This output 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 variables are strings, and all domain values are `ints`.

- 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 a 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 check for division by zero error when checking 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

The public test cases can be found in Canvas → Files → Projects → Project 2 → Project 2.1 → public_test_cases.py. Tips on how to use the public test cases can be found in 5.2.

4 Submission

4.1 Submission Details via Canvas

Refer to Canvas → CS3243 → Files → Projects → Coursemology guide.pdf for submission details.

4.2 P2ST (ChatGPT) App Usage

The P2ST (ChatGPT) app has been developed for CS3243 (this course). It is a tool that can be used to generate code from natural language descriptions. The objective is to help you better understand the contents of the course while skipping some of the more tedious parts of coding.

You are **permitted** to use the P2ST (ChatGPT) app to generate code to help you with this project.

No other app or AI-generated code may be used.

The plagiarism-checking phase will be conducted using a tool that can identify code that has been copied from other sources. If your code is flagged as duplicated, you may appeal to the teaching team only if the code was generated from or with the help of the P2ST (ChatGPT) app. If the teaching team finds that the code was not generated using the P2ST (ChatGPT) app and is instead generated via other means like using other AI assistance tools, plagiarising other people's work, and more, the appeal will not be successful.

Note that the correctness and efficiency of the code generated by the app are not guaranteed. You are responsible for any submissions made.

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 those 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 lecture.
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. AC3 is **not required** for this project. In fact, AC3 may even slow down your code in some of the test cases. (You may think about the time complexity of AC3, and determine in what situations would AC3 be a pessimisation for the backtracking algorithm.)