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

# Project 2: Local Search and Constraints

Issued: 23 September 2020 Due: 21 October 2020, 2359hrs

## Objectives

The objectives of this part of the project are to:

1. Solving Sudoku Puzzles as a Local Search or using Constraints
2. Become familiar with the implementation of the backtracking search algorithm.

**This project is worth 10% of your module grade.**

## General Project Requirements

The general project requirements are as follows:

- Group size: **2 students**
- Submission Deadline: **21 October 2020** (Wednesday), **2359 hours** (local time)
- Submission Platform: **LumiNUS** > **CS3243** > **Projects** > **Project 2 Submission Folder**
- Submission Format: One standard (non-encrypted) **zip file**[1] containing only the necessary project files. In particular, it should contain exactly one `.py` file. Make only one submission per group.

As to the specific project requirements, you must complete and submit the following:

- A file containing an implementation of a variant of the backtracking search algorithm or a local search-based strategy or a combination of both, that solves Sudoku puzzles.

[1]Note that it is the responsibility of the students in the project group to ensure that this file may be accessed by conventional means.

## Academic Integrity and Late Submissions

Do note that any material used 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. In any case, you may not share materials between groups.

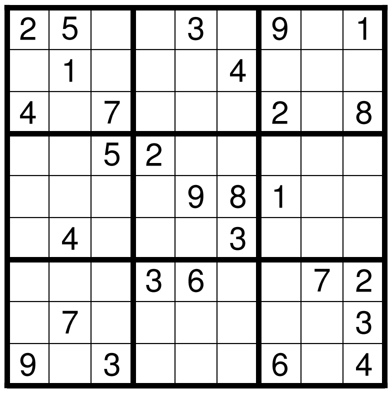
For projects submitted beyond the submission deadline, there will be a 20% penalty for submitting after the deadline and within 24 hours, 50% penalty for submitting after 24 hours past the deadline but within 48 hours, and 100% penalty for submitting more than 48 hours after the deadline For example, if you submit the project 30 hours after the deadline, and obtain 92%, a 50% penalty applies and you will only be awarded 46%.

---

### Background: The Sudoku Puzzle

Suduko is a logic-based, combinatorial number-placement puzzle. The objective is to fill a $9 \times 9$ grid with digits so that each column, each row, and each of the nine $3 \times 3$ sub-grids that compose the grid contain all digits from $1$ to $9$. The puzzle setter provides a partially completed grid, which for a well-posed puzzle, has a single solution. Figure 1 shows a sample Sudoku puzzle and Figure 2 shows the solution.

Remember, when you were a kid and you felt intelligent if you solved a hard sudoku puzzle. Well, if we have to ever build a really advanced AI, it better be able to solve a Sudoku puzzle. And now, you are going to indeed design a program that should be able to solve Sudoku faster than average 1st Grader/5th Grader/high school student/college graduate (Well, you get to decide your target).



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

Figure 1: A simple Sudoku Puzzle

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

Figure 2: Solution to the Sudoku puzzle in Figure 1.

The task of this project is to design an efficient Sudko solver: You have freedom to design your own solver based on either use local search technique (such as but not limited to hill climbing, variants of simulated annealing) or use constraints followed by backtracking search and inference. And for the adventurous ones, you can even combine the two.

## Submission Specifications

You need to submit one file, your python code file for this part. File names:

- The code filename should be `CS3243_P2_Sudoku_XX.py`

For example, Group 3 should submit two files: `CS3243_P2_Sudoku_03.py` (note that it is `03` and not `3`). Points will be deducted for not following the naming convention; please follow it closely.

**Input:** The Sudoku puzzle input is a text file containing $9$ lines (1 line per row). Each line contains $9$ digits, with $0$-values representing an empty cell.

Each given Sudoku puzzle will be:

- A $9 \times 9$ puzzle
- Valid: i.e., each row, column and subgrid will not contain any duplicate non-zero digits
- Well-formed: has a unique solution (and is thus, in particular, solvable)

For example, the puzzle from Figure 1 would be encoded as:

```
2 5 0 0 3 0 9 0 1
0 1 0 0 0 4 0 0 0
4 0 7 0 0 0 2 0 8
0 0 5 2 0 0 0 0 0
0 0 0 0 9 8 1 0 0
0 4 0 0 0 3 0 0 0
0 0 0 3 6 0 0 7 2
0 7 0 0 0 0 0 0 3
0 0 3 0 0 0 6 0 4
```

**Output:** The expected output should have the same format as the input. Your input should be valid, and only contain the digits 1-9. Thus, the solution for the puzzle given in Figure 1 should be:

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

**Code:** Please use Python 2.6 or 2.7 (the default Python version on SoC's Sunfire) to do this assignment. The template has been provided to you (`CS3243_P2_Sudoku_XX.py`). You may import Python Standard libraries if necessary. However, you may not use any external libraries. In addition, please note the following:

- You may create new classes or functions. However, all additional classes/functions should be defined within the same file.
- The solution to the given Sudoku puzzle should be returned by the `solve()` function.
- You SHOULD NOT modify the main function.
- Your submission must be executable. If your program can not be executed, you will get 0 for the code component.
- Your submission should compute correct solutions. Please ensure that your algorithm is correct; marks will be lost if your submission is unable to pass our test cases.

## Testing on our Grader

Note that running the code on your own and running on our grader may lead to different output (e.g. using global variables would work when running on your own, but not on our grader). It is imperative that you test out your code on our grader. If your code does not work on our grader, you will get a 0 for the corresponding section. Thus, we will be utilizing a platform that allows you view the output by running your code on our grader.

Register for the CS3243 course on CodePost.io to gain access to the autograder.

Logging in for the first time - Invite link: `https://codepost.io/signup/join?code=JAKZPL57LX` (Remember to check your spam/junk mail for the activation email (it'll most likely go there). Contact the course staff if you don't receive it after 30 minutes.)

Subsequent access: `https://codepost.io/student/CS3243%20Introduction%20to%20AI/AY20%2F21%20Semester%201/`

1. Rename your file to `CS3243_P2_Sudoku_01.py` for all testing purposes only on CodePost (This is IMPORTANT! Otherwise it will not work)
2. For the section you wish to test on our grader, click on "Upload assignment" and upload
3. Refresh the page
4. Select "View feedback" after the submission have been processed
5. You may check your output for each test case of the corresponding section (number of moves, time taken) under `_tests.txt` option (this is also the default view)

You may submit your files as many times as you like. Note that the marks awarded by the autograder is not at all reflective of the actual marks you will get. We are merely using the platform as a means for you to be able to view the number of moves and timings when tested on our grader.

## Marking Rubrics (Sudoku - 10 marks)

| Requirements (Marks Allocated) | Total Marks |
|---|---|
| • Correctly implement a variant of the backtracking search algorithm or a local search-based strategy or a combination of both (3)<br>• Pass all 4 public test cases within time threshold (2)<br>• Pass the 1 private test case within time threshold (1)<br>• Pass hidden set of test cases within reasonable time (4) | 10 |

## Test Cases and Thresholds

The first 4 cases are public. The next 1 is private, but you will be able to view the timings on CodePost. Here are the timings for your reference.

| Test Cases | Thresholds |
|---|---|
| • `input_1`: | • Time: $\leq 0.5s$ |
| • `input_2`: | • Time: $\leq 0.5s$ |
| • `input_3`: | • Time: $\leq 0.5s$ |
| • `input_4`: | • Time: $\leq 10s$ |
| • `input_5`: | • Time: $\leq 5s$ |

As a guide, the above timings are obtained on CodePost. Sunfire generally is much slower (approx. 4-5$\times$) compared to CodePost and your local machine (MacBook Pro 2.8 GHz Intel Core i7 as a reference). For consistency, we will be running your code and compare timings on Sunfire. Obviously, as you do not have access to the private test cases to run on Sunfire, you may take the timings on CodePost as a rough estimate (compare the difference in timings for the public test cases and you should have an idea about whether your private test cases timings are way off or not). So do not compare the exact timings above with that on Sunfire.