COMP3211: Fundamentals of Artificial Intelligence

Homework Assignment 3

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Release Date: Nov. 16, 2023

The HKUST Academic Honor Code

Honesty and integrity are central to the academic work of HKUST. Students of the University must observe and uphold the highest standards of academic integrity and honesty in all the work they do throughout their program of study. As members of the University community, students have the responsibility to help maintain the academic reputation of HKUST in its academic endeavors.

1 Introduction

In this assignment, there are three problems corresponding to:

- MRV and LCV Implementation in CSPs for Map Coloring (Problem I).
- Simulated Annealing Implementation for 8-queens problem (Problem II).
- Value Iteration Implementation (Problem III).

The deadline for submitting this homework is **23:59:59, Dec 3rd, 2023**. Please start as early as possible, and feel free to discuss with us if you have any questions about your design and implementation. The late penalty is 5% per day.

This assignment should be solved individually. No collaboration, sharing of solutions, or exchange of models is allowed. Please, do not directly copy existing code from anywhere other than your previous solutions, or the previous master solution. We will check assignments for duplicates. See below for more details about the homework.

2 Preliminaries

2.1 Environment Setup

Students are required to do the following programming assignments with the anaconda environment, as introduced in tutorial1.ipynb on Canvas. Before running the code, dependencies are required to install. You should run

```
pip3 install numpy
```

or

```
conda install numpy
```

The code is written by the jupyter notebook, you can open the assignment using

```
jupyter notebook csp_heuristic.ipynb
jupyter notebook 8_queen.ipynb
jupyter notebook value_iteration.ipynb
```

2.2 Additional Notes

In addition to the supplied code, you have the option to import the *built-in* `decimal` package in order to guarantee precise and accurate management of decimal numbers in your Python program. This is especially beneficial when adjusting the parameter to ensure it remains sufficiently small to prevent erroneous outcomes. The following is a code snippet for the usage of this package:

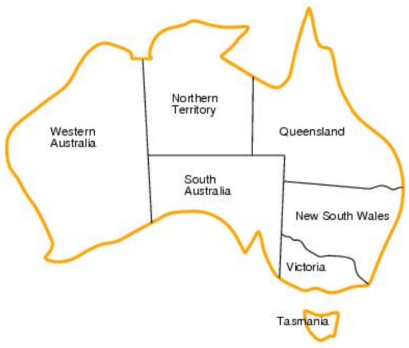
```python
# Creating and Using Decimal objects
from decimal import Decimal # import the package
num1 = Decimal('123.456')
num2 = Decimal('987.654')

# Addition
result_addition = num1 + num2
# Subtraction
result_subtraction = num1 - num2
# Multiplication
result_multiplication = num1 * num2
# Division
result_division = num1 / num2
```

Algorithm 1: Creating Decimal Objects

NOTE: the anaconda environment should contain all the packages that are required for this assignment if you CORRECTLY FOLLOW the steps from tutorial1.pptx. If you encounter a reported error like "matplotlib/numpy not found", just refer to Sec. 2.1 and install the "matplotlib/numpy" package.

3 Problem 1: MRV and LCV Implementation: Map Coloring (Total: 30 points)



Assignment Description

In this problem, you will work with a Jupyter Notebook script that models a map coloring problem as a Constraint Satisfaction Problem (CSP). In this task, you will implement based on the provided starter code to understand how Minimum Remaining Values (MRV) and Least Constraining Value (LCV) heuristics work.

Variables and Domains

Variables: The states of Australia (WA, NT, SA, Q, NSW, V, T).

Domains: A set of colors {Red, Green, Blue}.

Restrictions and Heuristics

a) **No Adjacent Color**: No two neighboring states can have the same color. This constraint is fundamental to the map coloring problem and ensures that the solution is valid. The adjacency of states is determined by the provided matrix, where a value of '1' at position (i, j) indicates that states i and j are neighbors.

b) **Minimum-Remaining-Values (MRV) Heuristic for Choosing States**: This heuristic chooses the variable (i.e., the state) with the fewest remaining legal colors to be assigned.

c) **Least Constraining Value (LCV) Heuristic for Choosing Colors**: The LCV heuristic chooses the color that rules out the fewest number of color choices for neighboring states.

Requirements

a) Complete the MRV Heuristic and LCV Heuristic functions `calcMRV()` to choose the states, and `calcLCV()` for choosing the color respectively in the provided Jupyter Notebook `csp_heuristic.ipynb`, respectively. (25 points)

b) The program will generate the solution of coloring as shown in a sample solution below:

```
Solution found:
State1 = color1
State2 = color2
State3 = color3
State4 = color4
State5 = color5
State6 = color6
T = Any color (Since Tasmania is not adjacent to any other state)
```

Document the produced solution of your program in the report. (5 points)

NOTE: Our code is tested on several OS and environment runtimes and is guaranteed to be executed.

Submission Guidelines

- Submit the completed Jupyter Notebook `csp_heuristic.ipynb`
- Submit the report containing the program output.
- Ensure that your code is executable with the expected output.

Assessment Criteria

Your submission will be evaluated based on:

- The correctness of your implementation of MRV and LCV Heuristics.
- The completeness of the result in the report.

4 Problem 2: The 8-Queens Problem using Simulated Annealing (Total: 30 points)

Assignment Description

Implement the Simulated Annealing algorithm and analyze the outcome in the setting of the 8-Queens problem. Explore different temperature schedules to study their impact on the algorithm's effectiveness. The 8-Queens problem is a classic puzzle where the goal is to place eight queens on an 8×8 chessboard in such a way that no two queens threaten each other. This means no two queens can occupy the same row, column, or diagonal.

There are three user-defined parameters in the program, namely, 'temperature', 'epochs' and 'decay'. The usages are the following:

- temperature: This is the starting temperature for the simulated annealing process. A higher initial temperature allows the algorithm to explore the solution space more freely at the beginning, potentially avoiding local minima. As the algorithm progresses, this temperature will gradually decrease.

- epochs: This is the total number of iterations or steps the algorithm will run. Each epoch represents a single update of the algorithm, where it tries a new configuration of queens on the board. More epochs allow more thorough exploration but take longer to run.

- decay ratio: This parameter determines how quickly the temperature decreases in each iteration of the algorithm. It is usually a value between 0 and 1 (not inclusive). A slower decay rate (closer to 1) means the temperature decreases more gradually, giving the algorithm more time to explore; a faster decay rate makes the temperature drop more quickly.

Requirements

- Implement the `SimulatedAnnealing.run()` function in the provided `8_queens.ipynb` code to solve the 8-Queens problem. Complete the function within the code. (20 points)

- The program uses a random initialization process for the positions of the 8 queens. To better observe how the program runs under different settings, please run the program 10 times for each of the following parameter combinations of temperature T, epochs e, and decay ratio d:

  - Setting I: $T = 4000, e = 10000, d = 0.9$

  - Setting II: $T = 2000, e = 10000, d = 0.9$

  - Setting III: $T = 1000, e = 10000, d = 0.9$

  If you implement your program correctly, a typical successful sample output of the program is (the program uses the letter 'Q' to denote the queens):

```
Solution:
(0, 3)
(1, 5)
(2, 0)
(3, 4)
(4, 1)
(5, 7)
(6, 2)
(7, 6)

Success, Elapsed Time: 540.795ms

Initial Board:

. . Q . . . . .
. . . Q . . . .
. . . Q . . . .
. . . . Q . . .
. . . . . . . Q
. . . . . Q . .
. . . . . . . Q
. . . Q . . . .

Final Board:

. . . Q . . . .
. . . . . Q . .
Q . . . . . . .
. . . . Q . . .
. Q . . . . . .
. . . . . . . Q
. . Q . . . . .
. . . . . Q . .

Number of attacks: 0
Elapsed Time: 540.795ms
```

  A typical unsuccessful sample output of the program is:

```
Unsuccessful, Elapsed Time: 221.006ms

```

```
Initial Board:

. . . . . . . Q
. . . . . . . Q
. . . . Q . . .
. . Q . . . . .
. . . Q . . . .
Q . . . . . . .
. . . . . . . Q
Q . . . . . . .

Final Board:

. . . . . Q . .
. . . . Q . . .
. . . . Q . . .
Q . . . . . . .
Q . . . . . . .
. Q . . . . . .
Q . . . . . . .
. . . Q . . . .

Number of attacks:  11
Elapsed Time: 221.006ms
```

Present the performance of your algorithm for each experiment setting considering the success rate and the average time to convergence in the following table. You should include this table in your report. (10 points)

Settings	Average Time to Covergence	Success Rate
Setting I		
Setting II		
Setting III		

Table 1: Performance of the Simulated Annealing Algorithm under Difference Settings

NOTE: Our code is tested on several OS and environment runtimes and is guaranteed to be executed.

Submission Guidelines

- Submit the Jupyter Notebook `8_queens.ipynb` with your implementation of the simulated annealing function.
- Submit a report with Table 1.
- Ensure that your code is executable with the expected output.

Assessment Criteria

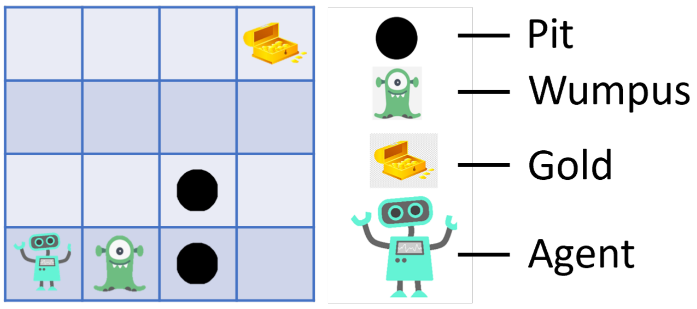
Your submission will be evaluated based on:

- The correctness of your simulated annealing implementation.
- The completeness of the result presented in the report.

5 Problem 3: Value Iteration in the Wumpus World (Total: 40 points)

Assignment Description

In this problem, we will delve into the application of Markov Decision Processes (MDPs) and the Value Iteration algorithm in the context of the Wumpus World. The Wumpus World is a well-known problem in the field of artificial intelligence, where the objective is to control an agent as it navigates through a grid-like environment in search of gold, all while avoiding deadly pits and the formidable Wumpus creature.



As you observed, the agent starts at grid coordinate (0,0) and its objectives are:

- Finding the gold, which provides a significant positive reward (+10).

- Avoiding the pits and the Wumpus, which are associated with negative penalties (-5 for each pit and -10 for the Wumpus).

- Minimizing the incurred movement penalty (-0.4 for each non-goal cell). Due to the noise of the control signal, the movements are stochastic: There is an 80% chance that the agent moves in the intended direction. To be more specific, there is a 10% chance that the agent moves in one of the orthogonal directions. For example, if the agent intends to move UP, there's an 80% chance it will move UP, a 10% chance it will move LEFT, and a 10% chance it will move RIGHT.

There are three user-defined parameters in the program, namely, 'gamma', 'eta' and 'max_iter'. The usages are the following:

- gamma: sets a discount factor of future rewards. It represents how much future rewards are valued compared to immediate rewards.

- eta: sets a threshold to determine whether the value iteration algorithm has converged.

- max_iter: sets the maximum number of iterations that the value iteration loop will run.

Requirement

- Your task is to implement the Value Iteration related functions: `MDP_value_iteration()` and `MDP_policy()` for the aforementioned 4×4 grid Wumpus World in order to control the agent to achieve the aforementioned objectives. You are not required to build the Wumpus World. It is in the provided Jupyter Notebook `value_iteration.ipynb` (30 points)

- Based on your implementation, you will use the program to calculate the utilities of each state and derive optimal policies for each grid as output. Experiment the program for each of the following parameter combinations of 'gamma' γ, 'eta' η, and 'max_iter' e:

  - Setting I: $\gamma = 0.3, \eta = 0.1, e = 10000$

  - Setting II: $\gamma = 0.6, \eta = 0.1, e = 10000$

  - Setting III: $\gamma = 0.9, \eta = 0.1, e = 10000$

  and document the output of the program into the report file (each parameter setting generates an output). (10 points)

  The program will generate an output like a sample solution output shown below if the implementation is correct:

```
Utilities and Policy for the Given Wumpus World:
111.00 ↓ | 222.11 ↑ | 33.22 ← | 44.33 → |
55.44 → | 66.55 ← | 7.66 ↓ | 8.77 ↑ |
0.99 ← | 10.10 ↑ | 11.10 → | 12.11 ↓ |
13.12 ↑ | 14.13 ↓ | 0.15 ← | 0.16 → |

```

Submission Guidelines

- Submit your code implementation along with a report.

- The report should include the results of the program for the three parameter settings specified in the requirements.

- Ensure that your code is executable.

NOTE: Again, our code is tested on several OS and environment runtimes and is guaranteed to be executed.

Assessment Criteria

- Correctness of the implementation.

- The completeness of your report, which should contain the program results under three different parameter settings.

6 Submission

Upon completion, students are required **combine three individual reports for problem 1, 2, and 3 into a combined PDF file.** Required results should be clearly shown. You should name the combined PDF file as **StudentID_report.pdf**.

Students should consolidate their code in the form of Jupyter Notebook (.ipynb) and make sure their notebooks can be executed without errors. You should name the code file for each problem as **StudentID_problem*.pdf**

If there are special requirements for running your code, please also specify them in a separate ReadMe file for each problem as **StudentID_README_problem*.md**

Finally, the combined pdf report, three jupyter notebooks, and the optional readme files should be **packaged together in a zip file** as **StudentID_Assignment3.zip** to be uploaded for evaluation.

In summary, a student with student ID 20000000 should submit the following in the zip file 20000000_Assignment3.zip:

- 20000000_report.pdf – The combined PDF report file
- 20000000_problem1.ipynb – Code implementation for problem 1
- 20000000_problem2.ipynb – Code implementation for problem 2
- 20000000_problem3.ipynb – Code implementation for problem 3

  (The following files are optional to submit)
- 20000000_README_problem1.md – ReadMe File for the code in problem 1
- 20000000_README_problem2.md – ReadMe File for the code in problem 2
- 20000000_README_problem3.md – ReadMe File for the code in problem 3