CS512– Artificial Intelligence

Instructor : Shashi Shekhar Jha (shashi@iitrpr.ac.in)

Lab Assignment - 1 | Due on 13/03/2021 2400 Hrs (100 Marks)

Submission Instructions:

All submission is through google classroom in one zip file. In case you face any trouble with the submission, please contact the TAs:

- Namrata Lodhi, 2019aim1007@iitrpr.ac.in
- Surbhi Madan, surbhi.19csz0011@iitrpr.ac.in
- Armaan Garg, 2019csz0002@iitrpr.ac.in

Your submission must be your original work. Do not indulge in any kind of plagiarism or copying. Abide by the honour and integrity code to do your assignment.

As mentioned in the class, late submissions will attract penalties.

Penalty Policy: There will be a penalty of 5% for every 24 Hr delay in the submission. E.g. For 1st 24 Hr delay the penalty will be 5%, for submission with a delay of >24 Hr and < 48 Hr, the penalty will be 10% and so on.

You submission must include:

- A legible PDF document with all your answers to the assignment problems, stating the reasoning and output.
- A folder named as 'code' containing the scripts for the assignment along with the other necessary files to run yourcode.
- A README file explaining how to execute your code.

Naming Convention:
Name the ZIP file submission as follows: ***YourName_rollnumber_Assignmentnumber.zip***
E.g. if your name is ABC, roll number is 2020csx1234 and submission is for
lab1 then you should name the zip file as: ABC_2020csx1234_lab1.zip

Section - I: Lion Tale

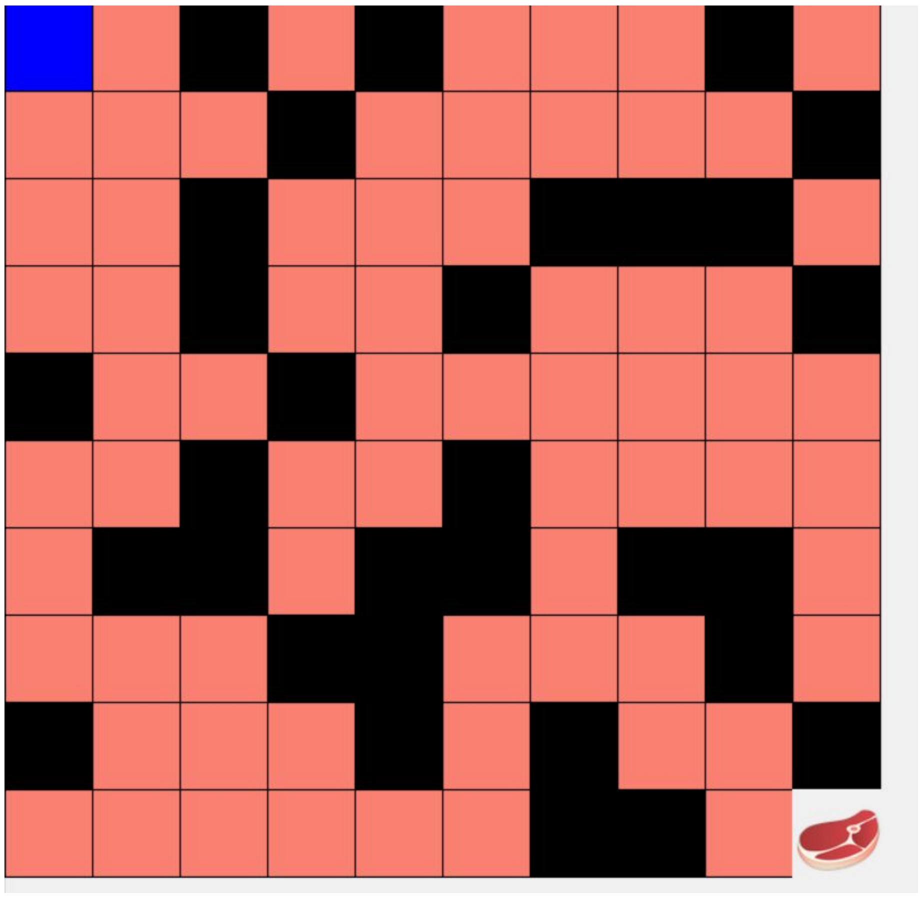
A Lion lives in a world with twisted and molded corridors and rounded treats. Navigation to the world is not straightforward due to the twisting nature of surroundings. Efficient navigation is Lion's first and foremost goal to find food (meat).

We provide a game model environment to implement and understand the working of different types of search algorithms. In this assignment, you need to build search algorithms so that Lion navigates through the world efficiently to quickly find the food.

You need to build your search algorithms on the maze world environment shown below.

If the Lion gets stuck at some locations during the search, you can forcefully quit the environment by typing CTRL-c into your terminal.

The Lion starts at the starting point (blue cell) of the grid while the meat is at the bottom right corner, as shown in the figure below.



For this section, use the template code provided alongside the assignment document.

Functions to edit:	
Lion_maze/lion_in_maze.py/search_algo()	Where all of your search algorithms will reside.
Files/Functions you might want to look at:	
Lion_maze/lion_in_maze.py	All functions to create the environment and GUI is available in this file only.
Lion_maze/color.py	You can change or customize the grid colors as per your choice by using this file.

Files to Edit and Submit: You will fill in a portion of search_algo() available in lion_in_maze.py. You should submit this file with your code and comments (kindly mention it specifically). Please *do not* change the other files/functions in this distribution.

In lion_in_maze.py, you'll find a fully implemented SearchAgent, which plans out a path through Maze world and then executes that path step-by-step. The search algorithms for formulating a plan are not implemented -- **that's your job**.
It's time to write full-fledged generic search functions to help the Lion. Remember that a search node must contain a state and the information necessary to reconstruct the path (plan) that gets to that state.

Important note: All of your search functions need to return a list of *actions* that will lead the agent (in our case Lion) from the start to the goal. These actions all have to be legal moves (right directions, no moving through walls).
Helping note: Make sure to **use** the Stack, Queue, and PriorityQueue data structures. These data structure implementations have particular properties.

Evaluation:
Each submission will be checked for plagiarism. If any submission is found to be plagiarized either with another fellow classmate or from other sources, then no further evaluations will be done and an "F" grade will be awarded.

NOTE: In each question, 2 Marks are for calculating the computing time and memory required by the system to implement each search algorithm. Show and compare the results using graphical representations in the PDF.

Question 1 (5 points): Finding Food using Depth First Search

Implement the depth-first search (DFS) algorithm in the search_algo function in lion_in_maze.py. To make your algorithm *complete*, ***write the graph search version of DFS***, which avoids expanding any already visited states.

Considering each step incurs a cost of 3 units, find the total cost for the Lion in the case of DFS. Does DFS follow completeness and optimality in this case? Analyze the time complexity of the search in proper steps.

Question 2 (5 points): Breadth-First Search

Implement the breadth-first search (BFS) algorithm in the search_algo function in lion_in_maze.py. To make your algorithm *complete*, **write the graph search version of BFS**, which avoids expanding any already visited states.

Is BFS successful in finding the solution? Justify your statement.

Consider each step incurs **5** units cost, find the total cost for BFS implementation. Also, discuss the space complexity of the approach in detail.

Question 3 (5 points): Varying the Cost Function

BFS may or may not find the best solution, but our task is to find the best or optimal solution. For this, implement a Varying Cost Search (VCS) algorithm.

<u>*Use the following cost function:*</u> For each step, the cost of vertical movement towards the target is 3 units otherwise the step cost is always 2 units. Implement the approach in search_algo function in lion_in_maze.py (obviously the Lion would always try to save its energy by taking a least cost path).

What is the total cost of search using VCS? Comment on the completeness of VCS. Also, discuss its time complexity.

Question 4 (8 points): A* search

Implement A* search in the search_algo function in lion_in_maze.py. A* takes a heuristic function as an argument. The heuristics function here takes two arguments: a state in the search problem (the main argument) and the search problem itself (for reference information).

You can test your A* implementation on the problem of finding a path to a fixed position in the environment using the Manhattan distance heuristic or Diagonal distance heuristic.

Use the cost function same as VCS and find the cost of the A* approach. Compare the complexity of A* with VCS and state whether A* is optimal and complete.

Question 5 (10 points): Working with a new environment [*Bonus Question]

(The total accumulated bonus marks will be used to help adjust your final total aggregate**)**

Implement A* search in a new OpenAI Gym Environment of your choice.
You can choose the environment from the below link:
https://gym.openai.com/envs/#classic_control

Helping note:
OpenAI Gym is a toolkit for developing and comparing various learning algorithms. It supports teaching agents everything from walking to playing games like pong or pinball. Gym provides an environment and it is upto the developer to implement any algorithms to solve the same. Developers can write agents using existing numerical computation libraries, such as TensorFlow or Theano.

Instructions according to UBUNTU OS : Work with Python3

Optional: To study about OpenAI Gym
https://towardsdatascience.com/reinforcement-learning-with-openai-d445c2c687d2

Install Gym
Reference link for installation of gym
http://gym.openai.com/docs/

Section II: The 8-queen puzzle

Remember the **8-queens puzzle**. It is a chess problem of placing eight queens on an 8×8 chessboard so that no two queens cross each other, thus a solution requires that no two queens share the same row, column, or diagonal. The 8-queens puzzle is an example of the more general ***n*-queens puzzle** of placing *n* non-attacking queens on an *n*×*n* chessboard, for which solutions exist for all natural numbers *n* with the exception of *n* = 2 and *n* = 3.

For Section II, download the template code provided alongside the assignment document.

Files you'll edit:	
GA/GA.py	To implement the code for local search based on Genetic Algorithm.

Files to Edit and Submit: You will fill in the section of GA.py **(comments are provided on where to add which functionality)** for the assignment. You should submit this file with your code and comments.

Note: You are not allowed to delete any given functions in the template but you can add more custom functions. The comments and template is there to provide you with surrounding structure and guidance.

Question 1: Implement GA [15 marks]

- Solve the n-queen problem using Genetic Algorithm.
- Take 5 different values for the hyperparameters viz. **number of queens, population size, crossover probability and mutation probability** and observe the changes in the best fitness value in the population across different generations for each combination of number of queens, crossover and mutation probability.

  (So there should be 5 outputs relating to 5 different setting of hyperparameters, each hyperparameter should be varied)

NOTE: Write all your observations supplementing them with graphs in the PDF file.

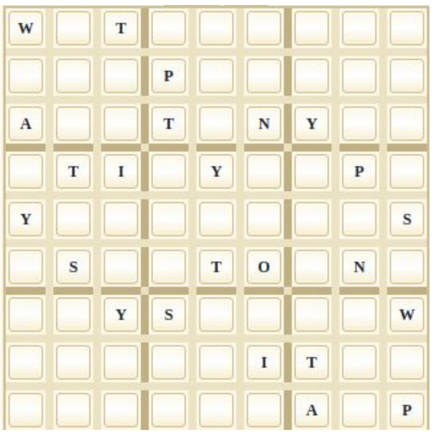
Section III: Solving Wordoku

In this section, you will design an AI agent to solve the Wordoku puzzle game. The core technique to focus on is the Constraint Satisfaction Problem (CSP). You will be implementing two methods to solve the Wordoku puzzle and compare the implementations. Wordoku is an extension of Sudoku where we have alphabets in place of digits. Some basics about the CSP approach for solving the same are given below.

Puzzle Background

The Wordoku puzzle has a 9x9 grid and a set of possible alphabets with some of the positions filled with those alphabets to ensure a solution can be reached. The goal is to find and fill remaining cells with alphabets such that each row, column and the 3x3 square (sub-grid) all must contain the alphabets, exactly once.

Note: There are exactly nine alphabets among which one of the characters is needed to be filled in blank space.



W		T						
			P					
A			T		N	Y		
	T	I		Y			P	
Y								S
	S			T	O		N	
		Y	S					W
					I	T		
						A		P

Helping note:

Solving Sudoku as Constraint Satisfaction Problems

A constraint satisfaction problem (CSP) consists of

- a set of variables,
- a domain for each variable, and
- a set of constraints.

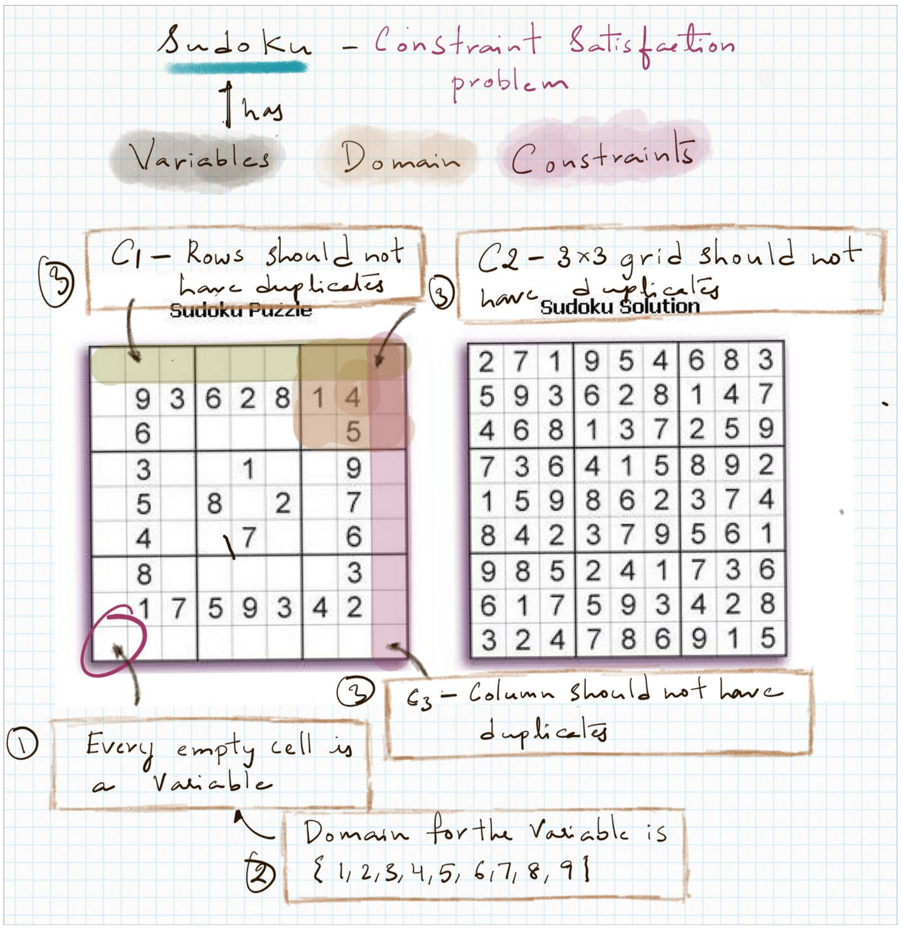
The aim is to choose a value for each variable such that the resulting possible world satisfies the constraints; we want a model of the constraints. The method in CSP should use the **constraint propagation** approaches as the inference to reduce the domain of each variable while using the **Backtracking search** with other suitable heuristics in order to find a solution.

Another approach that you have to consider is the Local Search based method - The MIN_CONFLIT algorithm. You can consider improving the Local Search method by incorporating Tabu Search.

Reference Explanation
Sudoku as CSP:

- Every empty cell is a variable
- Domain of each variable is {1,2,3,4,5,6,7,8,9}
- Constraints:
    - C1: Rows should not have duplicates
    - C2: Columns should not have duplicates
    - C3: 3x3 sub-grid should not have duplicates

Illustrations


Sudoku - Constraint Satisfaction problem
has
Variables
Domain
Constraints
③ C1 – Rows should not have duplicates
③ C2 – 3×3 grid should not have duplicates
Sudoku Puzzle
Sudoku Solution
③ C3 – Column should not have duplicates
① Every empty cell is a Variable
② Domain for the Variable is {1, 2, 3, 4, 5, 6, 7, 8, 9}

Sudoku Puzzle

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

Sudoku Solution

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

IMPLEMENTATION INSTRUCTIONS

Grid Layout

Your program will read the layout for each Wordoku puzzle from the input file that contains a 9x9 matrix of single characters. The characters will be the alphabets forming a possible set, plus the '*' character for any unassigned cell, where all of the cells are separated by space characters. For example, see below:

Hint: Each test case contains all the alphabets that form the possible set of characters.

All the possible characters of the set can be deduced from the filled cells.

```
* D * * * * A G I
* * B * * H * E D
* * F * * E * * H
* H * * G * I A *
* E * * I * * C *
* A I * F * * D *
C * * D * * G * *
E G * A * * B * *
I B H * * * * F *
```

FILES TO SUBMIT

(i) WordokuSolver_Backtracking.py

(ii) WordokuSolver_minconflict.py

The Python files contain the program that completes the objectives.

- 'input.txt' will be the name of the file containing the Wordoku puzzle.
- 'solution.txt' will be the name of the file wherein the solved Wordoku puzzle must be written.

NOTE:

- Calculate results for 5 different puzzles in the input.txt file. Code should be generic as it would be tested on various grid puzzles during evaluation.
- Find the 5 test cases in the appendix at the bottom of the document.
- Other Python files may be submitted as well, if needed. Be sure to document your code thoroughly, as marks will be deducted for poor or unreadable designs. Graphical interface design is encouraged but not required.

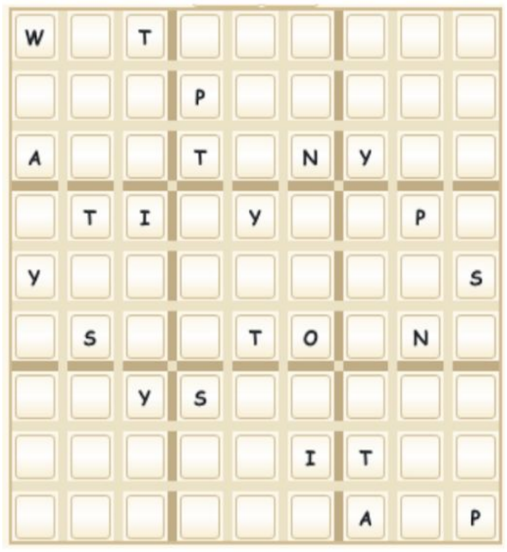
Question 1: Implement the *Backtracking Search* with Constraint Propagation which can solve any Wordoku puzzle given as input. **[20 marks]**

Question 2: Implement the Min_Conflict Search which can solve any Wordoku puzzle given as input. **[20 marks]**

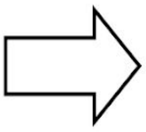
Question 3: State the time and space complexity for each algorithm in your report. Also, provide a table that compares performance of the above two algorithms for each 5 input cases.
Performance metrics: The total clock time, the search clock time, the number of nodes generated, N. Analyze your results to see if the behaviour you expected has been achieved or not, and why.
[7 marks]

Question 4: Find all the possible meaningful words formed in any row, column or diagonal in each test case and list them in the output. (If none possible, mention **'None'** in output) **[7 marks]**

Example of input matrix representation:



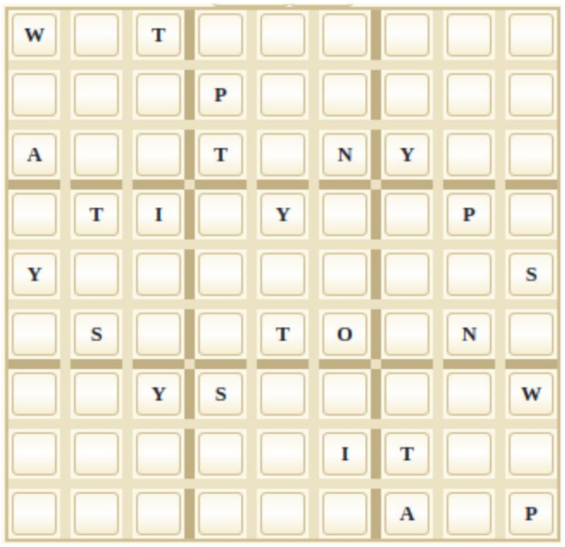
W		T						
			P					
A			T		N	Y		
	T	I		Y			P	
Y								S
	S			T	O		N	
		Y	S					
					I	T		
						A		P



W	*	T	*	*	*	*	*	*
*	*	*	P	*	*	*	*	*
A	*	*	T	*	N	Y	*	*
*	T	I	*	Y	*	*	P	*
Y	*	*	*	*	*	*	*	S
*	S	*	*	T	O	*	N	*
*	*	Y	S	*	*	*	*	*
*	*	*	*	*	I	T	*	*
*	*	*	*	*	*	A	*	P

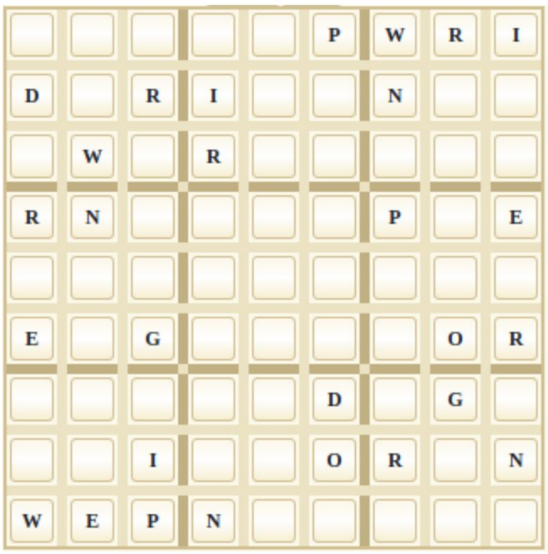
Test cases:

TEST CASE 1



W		T						
			P					
A			T		N	Y		
	T	I		Y			P	
Y								S
	S			T	O		N	
		Y	S					W
					I	T		
						A		P

TEST CASE 2



					P	W	R	I
D		R	I			N		
	W		R					
R	N					P		E
E		G					O	R
					D		G	
		I			O	R		N
W	E	P	N					

TEST CASE 3

	A						D	
		S				B		
				N				
	B		T		S		N	
				B				
	T		N		I		W	
			B		W			
A								
T				L		S		

TEST CASE 4

	R			N		B		
N				B		R		
			O					
			M					
	S	N						
E					U			
					A			
		B		S				M
		O					S	

TEST CASE 5

	P					I		D
D								
	B					P	A	
P					W			B
							K	
	A							
B								
E	R				K	B		

Best of Luck!!