AL-2002 PROJECT24

Note: Carefully read the following instructions (*Each instruction contains a weightage*)

1. There must be a block of comments at start of every question's code by students; the block should contain brief description about functionality of code.
2. Comment on every function and about its functionality.
3. Use understandable name of variables.
4. Write a code in PYTHON language and you may use any of IDE or Notebook environment.
5. First think about the problems statements then you may start your programming.
6. At the end when you done your tasks, attached .py or .ipynb files on google classroom. Paste your complete code in word file along with output **(Make sure your submission is completed)**. In case of missing any file, marks will be deducted.
7. The project can be completed **with maximum 4 memebers in group**, but the evaluation will be based upon individual effort and VIVA.
8. Please submit your file in this format **19F1234, 19F1235, 19F1236**.
9. Do not submit your project after deadline. Late and email submission is not accepted.
10. Do not copy code from any source otherwise you will be penalized with negative marks.
11. **YOUR MARKING WILL BE BASED ON PRIOR SUBMISSION OF YOUR CODE BEFORE DEADLINE AND VIVA.**

PROBLEM 1 Autonomous Delivery Robot

An autonomous delivery robot for your online grocery orders, needs to navigate through an area in the city to deliver packages to designated locations. Its environment is filled with obstacles like buildings, houses, and vehicles on roads. Your task is to guide the robot to deliver the order safely to the customer. The following is a detailed description of your task.

Environment Representation:

- Design a representation of some city area, possibly as a grid or a graph.
- Include information about buildings, houses, delivery points, and vehicles on roads.
- A grid size of around 15 x 15 should be created as a graph.
- The initial "start location" for robot should be fixed.

Algorithm Implementation:

- Implement informed search algorithms like best first and A* for robot motion planning, which you should think is the best for this problem. For this, you have to make a comparison between both algorithms and choose the best.
- Develop a heuristic function using Euclidean distance that considers the distance to the goal, and the presence of obstacles.
- If applicable, the cost from one location to another location should be randomly generated but should be between 1 to 20 and the cost from the current location to the goal should be calculated by Euclidean distance.

Dynamic Environment Handling:

- Simulate dynamic changes in the environment such as changing the start, goal state and vehicles positions after delivering an item.
- Ensure that the robot can adapt its path planning in real-time to handle these changes efficiently.

Path Execution and Control:

- Develop a path execution module that allows the robot to follow the planned path while avoiding collisions.
- The location of goal should be generated randomly within the grid. You must create 5 different locations for delivering items.
- In the start robot waits for delivery, then you should assign delivery items to it and destination.

After the robot has done its job the previous delivery location should be the start point and then assign the next delivery location and it will do the same for the remaining 4 deliveries.

User Interface and Visualization:

- Create a user interface to interact with the simulation environment.
- Visualize the robot's path, obstacles, and delivery points in real time.

Performance Evaluation:

- Evaluate the performance of the motion planning algorithm in terms of path optimality, execution time, and adaptability to dynamic changes.

Implementation Steps:

Environment Representation:

You can use matplotlib to visualize the city area, with obstacles and delivery points represented graphically.

Algorithm Implementation:

Implement informed search algorithms for an autonomous delivery robot in Python.
Use an animation library to animate the robot's movement and visualize the planning process.

Python Libraries for Animation and GUI:

- **matplotlib:** It provides functionality for creating animations using its animation module.
- **Pygame:** This library is well-suited for game development and includes features for creating animations and interactive simulations.
- **Tkinter:** It's Python's built-in library for creating simple GUI applications.
- **PyQt or PySide:** These libraries provide more advanced GUI capabilities with extensive widget sets and support for modern GUI design.

Note: You can use any other python library if required.

Project Deliverables:

- Source code implementing the motion planning algorithm, dynamic environment handling, path execution, and simulation components.
- A user-friendly interface allows users to interact with the environment, visualize the robot's movement, and control the simulation.
- You can code in .ipynb or .py format.

Bonus Task:

Multi-Robot Coordination:

Extend the project to handle multiple autonomous robots navigating in the same environment simultaneously. Implement coordination strategies to prevent collisions and optimize delivery routes.

PROBLEM 2 Sudoku Puzzle:

A Sudoku board consists of 81 squares (see textbook section 6.2.6), some of which are initially filled with digits from 1 to 9. The puzzle is to fill in all the remaining squares such that no digit appears twice in any row, column, or 3×3 box. A row, column, or box is called a **unit**.

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

(a)

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

(b)

Figure 6.4 (a) A Sudoku puzzle and (b) its solution.

CSP Formulation:

Whenever we formulate a problem into a CSP, we need to identify the following: set of variables *Xi*, set of domains *Di*, and set of constraints *C*:

X is a set of variables, $\{X_1, \ldots, X_n\}$.

D is a set of domains, $\{D_1, \ldots, D_n\}$, one for each variable.

C is a set of constraints that specify allowable combinations of values.

All sudoku puzzles can be formulated as CSP by considering each **cell** as a variable. The initial domain of all cells is {1, 2, 3, 4, 5, 6, 7, 8, 9}. The constraints are formulated by the fact that in the solution of a sudoku puzzle, no two cells in a row, column or block can have identical numbers.

Implementation Activities:

The Sudoku puzzle is a classic constraint satisfaction problem (CSP). Implement the Arc-Consistency 3 (AC-3) Algorithm and the Backtracking Algorithm to solve the Sudoku puzzle in Python, and compare their time complexities. The Sudoku puzzle board should feature a graphical user interface (GUI) using Tkinter. The dataset for the Sudoku puzzles is provided.

function AC-3(*csp*) **returns** false if an inconsistency is found and true otherwise
  inputs: *csp*, a binary CSP with components (*X*, *D*, *C*)
  local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

  while *queue* is not empty **do**
    (X_i, X_j) ← POP(*queue*)
    if REVISE(*csp*, X_i, X_j) **then**
      if size of D_i = 0 **then return** *false*
      for each X_k in X_i.NEIGHBORS − {X_j} **do**
        add(X_k, X_i) to *queue*
  return *true*

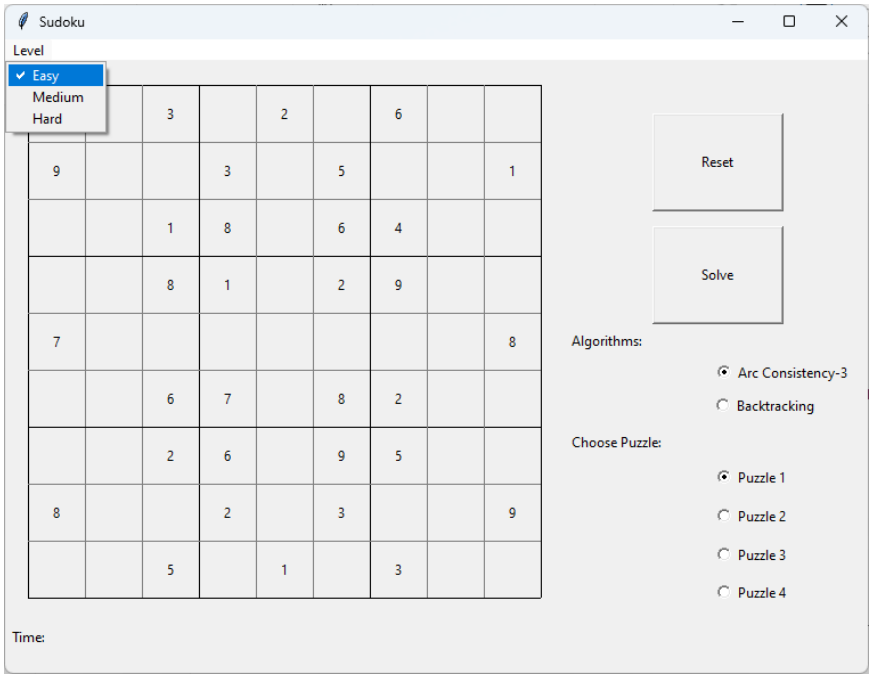
function REVISE(*csp*, X_i, X_j) **returns** true iff we revise the domain of X_i
  revised ← *false*
  for each *x* **in** D_i **do**
    if no value *y* in D_j allows (*x*, *y*) to satisfy the constraint between X_i and X_j **then**
      delete *x* from D_i
      revised ← *true*
  return *revised*

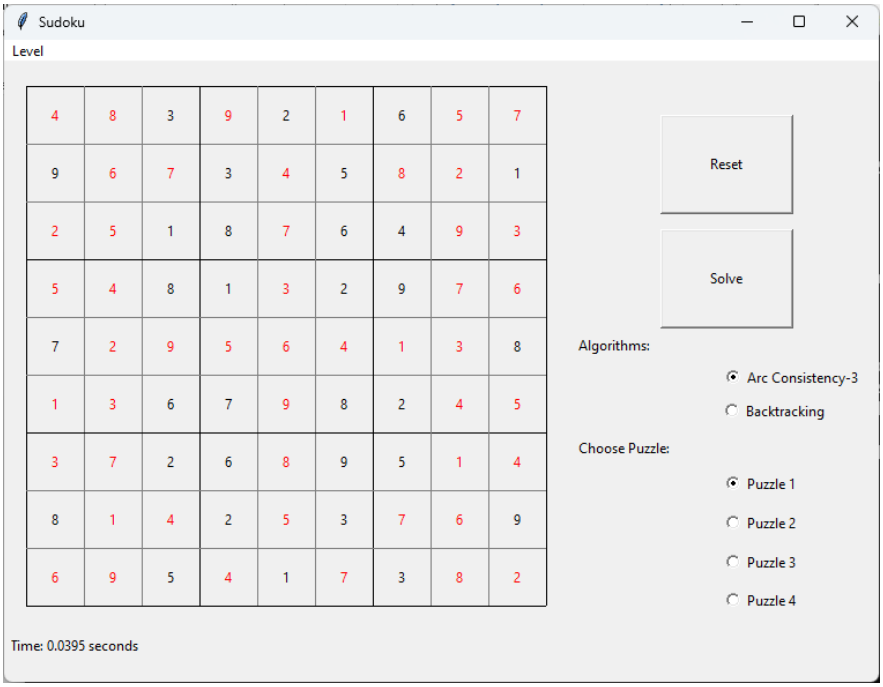
function BACKTRACKING-SEARCH(*csp*) **returns** a solution, or failure
  return BACKTRACK({}, *csp*)

function BACKTRACK(*assignment*, *csp*) **returns** a solution, or failure
  if *assignment* is complete **then return** *assignment*
  var ← SELECT-UNASSIGNED-VARIABLE(*csp*)
  for each *value* **in** ORDER-DOMAIN-VALUES(*var*, *assignment*, *csp*) **do**
    if *value* is consistent with *assignment* **then**
      add {*var* = *value*} to *assignment*
      inferences ← INFERENCE(*csp*, *var*, *value*)
      if *inferences* ≠ *failure* **then**
        add *inferences* to *assignment*
        result ← BACKTRACK(*assignment*, *csp*)
        if *result* ≠ *failure* **then**
          return *result*
      remove {*var* = *value*} and *inferences* from *assignment*
  return *failure*

Implementation Guidelines:

1. Stuart Russell and Peter Norvig. Artificial Intelligence: A Modern Approach, 4th edition. Pearson, 2022.
2. Code for the book "Artificial Intelligence: A Modern Approach" https://github.com/aimacode/aima-python (accessed April 15, 2024).
3. Wei-Meng Lee, "Programming Sudoku" www.apress.com/9781590596623 (accessed April 15, 2024).
4. Graphical User Interfaces, http://newcoder.io/gui/ (accessed April 15, 2024).





Evolution Criteria:

The project will be evaluated based on the following criteria:

1. Both algorithms should be correctly implemented and fully functional.
2. The Sudoku Board should offer the option to select either the Arc-3 algorithm or backtracking to solve the puzzle.
3. The Sudoku Board should allow users to select a difficulty level—easy, medium, or hard—and choose from one of four available puzzles, provide in dataset.
4. The time taken to solve the puzzle, also known as time complexity, should be displayed.
5. Quality of Code.

Dataset:

Easy Sudoku - 1

Puzzle #1

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

Puzzle #2

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

Puzzle #3

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

Puzzle #4

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

Medium Sudoku - 2

Puzzle #1

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

Puzzle #2

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

Puzzle #3

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

Puzzle #4

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

High Sudoku – 3

Puzzle #1

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

Puzzle #2

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

Puzzle #3

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

Puzzle #4

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

Dataset @sudokutodo.com

PROBLEM 3 Unsupervised Learning

you are required to design and implement an automated exam management system that can efficiently manage the seating plan and faculty allocation for the computer science, artificial intelligence, business analytics, software engineering, and electrical engineering students in batches 19, 20, 21, and 22. The project aims to efficiently assign duties to faculty members and create an optimal seating plan for students during exams. The system should use the k-means clustering technique to automate the seating plan and faculty allocation. The k-means clustering technique is a popular machine learning algorithm used to group similar data points together. In this case, the data points are the students and their respective domains.

Problem Statement:

The university has 30 rooms available for conducting exams, and there are approximately 2400 to 2500 students across batches 19, 20, 21, and 22 in various domains, including computer science, artificial intelligence, business analytics, software engineering, and electrical engineering. Each room has a capacity of 30 to 35 students, with a few rooms having a capacity of 25 seats. There are also faculty members available for each domain.

The challenge is to automate the process of assigning duties to faculty members and creating an efficient seating plan for the students during exams, taking into account the batch distribution, domain of study, room capacities, and faculty availability.

The system should be able to perform the following tasks:

1. **Data Collection:** Collect data about number of students from batches 19, 20, 21,22 and 23 in each domain (computer science, artificial intelligence, business analytics, software engineering, and electrical engineering). Collect information on the capacity of each room, including rooms with varying seat capacities.

2. **Data Preprocessing:** Clean and preprocess the data to ensure it is suitable for the k-means clustering algorithm.

3. **K-Means Clustering:** Use the k-means clustering algorithm to group the students based on their domains and batch numbers. Use the number of students from each domain and batch as features for clustering. Determine the optimal number of clusters based on domain and batch distribution.

4. **Seating Plan:** Generate a seating plan for each room based on the clusters formed. Ensure that the capacity of each room is not exceeded. Optimize seating arrangements to minimize disruptions and ensure a conducive exam environment. Account for any special requirements or accommodation for certain students.

5. **Faculty Allocation:** Allocate faculty members to each room based on the clusters formed. Ensure that each room has at least one faculty member from each domain. Assign faculty members to each cluster based on their domain expertise. Ensure that each faculty member is assigned to a cluster with students from relevant domains.

6. **Reporting:** Generate a report that summarizes the seating plan and faculty allocation for each exam.

Project Goals:

The goal of the project is to develop an automated system that:

1. Uses K-means clustering to group students based on their domains of study, ensuring that students from the same domain are seated together.

2. Group up the students based on batch distribution and room capacities, ensuring that each room is filled to its maximum capacity without exceeding the limit and groups should be based on different exams in same room.

3. Assigns faculty members to each exam room based on their expertise in the corresponding domain of study.

Project Deliverables:
The project should deliver the following components:

1. An algorithm or program that implements the K-means clustering technique to group students based on their domains of study.
2. An algorithm or program that generates an optimized seating plan, considering batch distribution, room capacities, and student clusters.
3. An algorithm or program that assigns faculty members to each exam room based on their domain expertise.
4. A user-friendly interface that allows administrators to input student and faculty data and view the generated seating plan and faculty assignments.

Evaluation Criteria:
The project will be evaluated based on the following criteria:

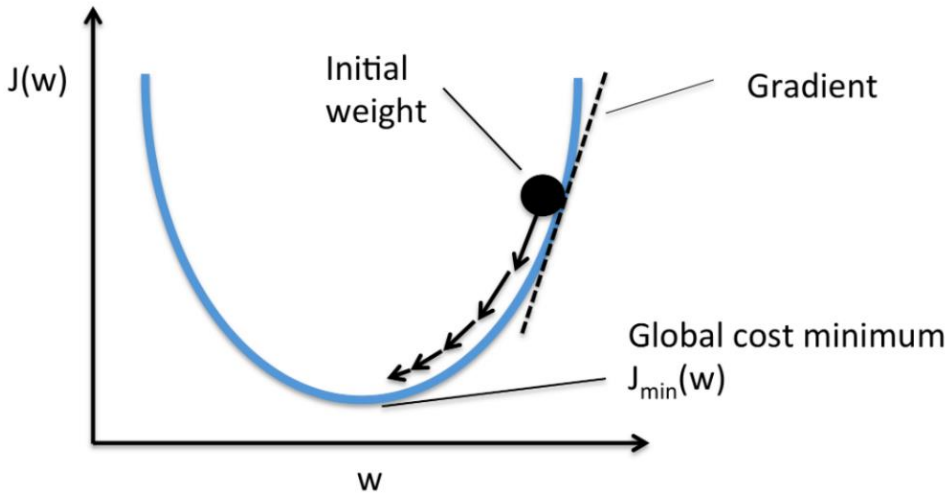
6. Correct implementation and functionality of the K-means clustering algorithm.
7. Accuracy and efficiency of the seating plan generation, considering batch distribution and room capacities.
8. Accuracy of faculty assignments based on domain expertise.
9. Code quality.

Implementation Guidelines:

- Utilize Python programming language along with libraries such as NumPy, Pandas, and Scikit-learn for data manipulation, clustering, and analysis.
- Design modular code for scalability and ease of maintenance.
- Document each step of the implementation process for clarity and reproducibility.
- Test the system with sample datasets and real-world scenarios to ensure accuracy and efficiency.

PROBLEM 4 Learning in AI

In this project task, you will explore and compare the performance of two fundamental learning algorithms, namely the Perceptron Learning Rule and the Gradient Descent Delta Rule, in the context of analyzing the classification of iris flowers species. The Perceptron Learning Rule and the Gradient Descent Delta Rule are both essential algorithms in the realm of machine learning, each with its own strengths and limitations. The project aims to understand how these algorithms behave when applied to a real-world dataset and to explore ways to improve their performance.



Objective:

- To analyze and compare the performance of the Perceptron Learning Rule and the Gradient Descent Delta Rule in classifying iris flowers species.
- To experiment with different activation functions and learning rates to optimize the accuracy of the models.

Tasks:

1. Implement the Perceptron Learning Rule and the Gradient Descent Delta Rule from scratch.
2. Utilize the provided dataset of iris flowers species for training and testing.
3. Experiment with different activation functions to observe their impact on model accuracy.
4. Divide the dataset into training and testing sets **using a 80/20 ratio** and evaluate the models' accuracy on both sets.
5. Adjust the learning rates of the models to minimize loss and achieve comparable accuracies.
6. Document the process, results, and observations for each algorithm.

Questions:

1. What are the key differences between the Perceptron Learning Rule and the Gradient Descent Delta Rule?
2. How does the choice of activation function influence the performance of the models?
3. What strategies can be employed to adjust the learning rate for optimal model training?
4. Discuss the implications of using different ratios for splitting the dataset into training and testing sets.
5. What challenges did you encounter while implementing the algorithms from scratch, and how did you overcome them?
6. Reflect on the strengths and limitations of each algorithm based on your experimentation.

Dataset Link:

https://archive.ics.uci.edu/dataset/53/iris