National University of Computer and Emerging Sciences

FAST School of Computing Fall 2025

AI-2002-Artificial Intelligence

Assignment 03

Instructions for Submission:

Dear students, we will use auto-grading tools, so failure to follow the submission format below will result in zero marks.

1. Assignments must be completed individually.
2. Combine all your work into one folder and name it RollNumber-Section.zip.
3. Run and test your program on any machine before submission.
4. Submit the .ZIP file on Google Classroom before the deadline. Submissions through other means (e.g., email) will not be accepted.
5. Check your ZIP file for issues like corruption, viruses, or mistakenly including .exe files. If the file cannot be downloaded, you will receive zero marks.
6. Ensure displayed output is clear and well-presented. Use appropriate comments and indentation in your code.
7. Total Marks: 100.
8. Syntax errors in the code will result in zero marks for the corresponding part of the assignment.
9. Your code must be generic and reusable.
10. You cannot use built-in functions or import any libraries.
11. Submit your assignment 3 hours before the deadline to avoid last-minute issues (e.g., internet problems).
12. You may use any programming language of your choice.
13. Each assignment includes a demo and viva. Completing the assignment does not guarantee full marks. Poor viva performance can result in significant mark deductions or zero marks.

Deadline:

The deadline to submit the assignment is 11th October 2024, 11:59 PM. Submit your assignment on Google Classroom (Classroom Tab, not Lab). Only .ZIP files are acceptable; other formats will receive zero marks. Timely submission is your responsibility, and no exceptions will be made. Late submission policies will apply as outlined in the course syllabus.

Tip:

Start early to ensure timely completion of the assignment.

Plagiarism:

Plagiarism is strictly prohibited. If plagiarism is detected, you will receive zero marks. Copying from the internet is the easiest way to get caught.

Note:

Follow these instructions exactly. Failure to comply will result in zero marks.

Course Crisis

Finding the best solution is always challenging, especially when working with machine learning models. These models rarely provide an optimal solution, instead, they aim to find the closest possible outcome. Moreover, implementing such models can be expensive and resource-intensive.

Let's consider a relatable example at our university, the "course crisis":

- Hamza is registered for Cloud Computing but wants to switch to DDE (Data-Driven Engineering).
- Mahreen is enrolled in Marketing Management but prefers Psychology to create a 2-day class schedule.
- Noman is already in DDE Section A but wants to transfer to Section C because he prefers the instructor there.

Managing all these requests is quite a task, and it often falls on Sir Amir to figure it out.

To make this process simpler, we can record all the swap requests from students, including their roll numbers. However, due to certain constraints, the roll numbers in the records must be randomized.

Your task is to read the provided file of swap requests and create a set of instructions to perform swaps (or a series of swaps) that satisfies the maximum number of students. This approach ensures we resolve the "course crisis" efficiently while making life easier for those managing these requests.

```
22i-0263 CN-A A B
22i-0263 DBMS-C C DAA-B
22i-0263 AI-C C C
22i-0263 AI-A A OS-A
21i-9772 SE-B B CC-A
21i-9772 SE-A A A
21i-9772 OS-A A A
21i-9772 DBMS-A A A
```

The provided files are structured as follows:

- First column: The roll number of the student.
- Second column: The course they are currently enrolled in, along with their section.
- Third column: The section the student is currently placed in.
- Last column: The section or course the student wants to switch to.

If no specific section is mentioned in the last column, it means the student is willing to be placed in any available section of their desired course. Make sure to give the older students preferene.

You should create different functions for the each of the search strategies. You will be graded besed on your problem representation, problem decomposition and program hygiene as well as correctness. Did you provide a good structure to the program using functions? Did you minimize the scope of variables to the smallest necessary? Did you use meaningful identifiers? Did you provide comments for your functions? etc.

Graph Classification

Analyzing and understanding complex networks is a key task in various fields like social networking, biology, and computer networks. One of the most challenging aspects of graph analysis is identifying specific patterns or structures (subgraphs) within a larger graph. These subgraphs often represent critical information such as tightly-knit communities, functional components, or recurring patterns.

Let's consider a relatable example in our university network, the "study group dilemma":

- **Study Group A:** A tightly connected group of 4 students frequently collaborates on assignments for Cloud Computing.
- **Study Group B:** A smaller trio of students works exclusively on projects for Data-Driven Engineering (DDE).
- **Study Group C:** A scattered set of students collaborates irregularly but is part of a chain where each student is connected to at least one other.

Managing and identifying these groups is crucial for resource allocation, such as assigning group study rooms, facilitating mentoring sessions, or even forming effective project teams. However, manually detecting and categorizing these groups is tedious, especially as the graph (representing connections) grows in size and complexity.

Your task is to read a provided graph file, identify all the subgraphs that match specific patterns, and classify them accordingly. The provided graph file is structured as follows:

- First column: Source node of an edge.
- Second column: Target node of an edge.
- Third column: Weight of the edge (optional, representing the strength of the connection).

You need to create functions that detect and classify subgraphs based on:

- Cliques: Fully connected subgraphs where every node is connected to every other node.
- Chains: Linear structures where each node connects to exactly two other nodes (except endpoints).
- Stars: A central node connected to multiple outer nodes with no connections among the outer nodes.
- Cycles: Closed paths where each node is part of a loop.

Make sure to:

- Provide older or more critical subgraphs (based on size or importance) with higher priority in your classification.
- Use different search strategies like Depth-First Search (DFS), Breadth-First Search (BFS), and custom heuristics to identify the subgraphs.
- Create reusable and well-documented functions for each search strategy and subgraph classification.

Ultimate Tick Tack Toe

Ultimate Tic-Tac-Toe is a strategic extension of the classic Tic-Tac-Toe game, played on a 3×3 grid of smaller 3×3 Tic-Tac-Toe boards. The objective is to win three small boards in a row (horizontally, vertically, or diagonally) on the larger board.

Rules of Ultimate Tic-Tac-Toe:

- The game begins with an empty 3×3 grid of smaller Tic-Tac-Toe boards.
- Players take turns marking a spot ('X' or 'O') on a small board.
- The active small board for the next move is determined by the position of the last move:
- If a player places a mark in position (r, c) of a small board, their opponent must play in the (r, c) small board of the large grid.
- If the targeted small board is already won or full, the player can move to any empty position.
- A player wins a small board by forming a horizontal, vertical, or diagonal sequence.
- A player wins the game by winning three small boards in a row.

Task 1: Constraint Formulation

You must represent Ultimate Tic-Tac-Toe as a Constraint Satisfaction Problem (CSP) with:

- Variables: Representing possible moves on the 3×3 small boards.
- Domains: Available moves (X, O, or empty).

Constraints:

- Valid moves follow the active small board rule.
- A player cannot mark an occupied position.
- Winning conditions for small boards and the large board.
- If a small board is won, no further moves should be allowed on it.
- Provide a detailed explanation of your constraint formulation.

Task 2: Implementing a CSP Solver for Ultimate Tic-Tac-Toe

Implement an AI agent that uses Backtracking Search with:

- Forward Checking (to eliminate illegal moves early).
- Arc Consistency (AC-3) algorithm to ensure valid board states.

Your algo should:

- Make optimal moves following the CSP formulation.
- Prevent the opponent from winning whenever possible.
- Win in the fewest possible moves.

Task 3: Experimentation & Analysis

Play a game against your algo and analyze how it performs.

Test different heuristics (such as Minimum Remaining Values (MRV) for choosing the next move).

Compare results with a basic minimax agent to evaluate efficiency.

Enhance your AI using Constraint Optimization to prioritize moves that lead to faster wins.
Implement Alpha-Beta Pruning with CSP to create a hybrid solver.

Submission Guidelines

- Code: A well-documented Python program implementing the CSP solver for Ultimate Tic-Tac-Toe.
- Demo: Screenshots or a video showing your AI playing against a human or another AI.
- Your Tick Tack Toe should have a proper visual GUI.
- Reference:
    - https://bejofo.com/ttt
    - https://michaelxing.com/

Honor Policy

This assignment is a individual learning opportunity that will be evaluated based on your ability to think independently, work through a problem in a logical manner solve the problems on your own. You may however discuss verbally or via email the general nature of the conceptual problem to be solved with your classmates or the course instructor, but you are to complete the actual assignment without resorting to help from any other person or other resources that are not authorized as part of this course. If in doubt, ask the course instructor. You may not use the Internet to search for solutions to the problem.