Artificial Intelligence

BS (CS) _SUMMER_2024

Lab_07 Tasks

Learning Objectives:

1. Local Search Algorithm
2. Hill Climbing
3. Stochastic Hill Climbing
4. First Choice Hill Climbing
5. Local Beam Search

Lab Tasks:

Tic-Tac-Toe Local Search Optimization

Problem Statement:

In the classic game of Tic-Tac-Toe, two players take turns marking spaces on a 3x3 grid with their respective symbols, typically 'X' and 'O'. The objective is to achieve a row, column, or diagonal of three of their symbols in a row before the opponent does. The game ends in a draw if the grid is filled without a winner.

Design a local search algorithm to optimize the strategy for the 'X' player in Tic-Tac-Toe, aiming to increase the likelihood of winning or achieving a draw against an opponent following a simple rule-based strategy.

Objective: Develop a local search algorithm that iteratively refines the 'X' player's moves based on the current game state. The algorithm should explore neighboring states by considering possible moves and selecting the one that maximizes the chances of winning or preventing the opponent from winning.

Requirements:

1. The local search algorithm should start with an initial move strategy and iteratively explore nearby moves to improve its chances of winning.
2. The optimization process should consider the opponent's likely responses and adjust the strategy accordingly.
3. The algorithm should be simple and computationally efficient, suitable for real-time decision-making in a game scenario.

Pseudo Code: **(Optional Use this structure or made your own)**

```
import random

# Function to print the current state of the board
def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("-" * 9)

# Function to check if a player has won
def is_winner(board, player):
    pass

# Function to check if the board is full
def is_board_full(board):
    pass

# Function to get a list of empty cells on the board
def get_empty_cells(board):
```

```
    pass

# Function to make a move on the board
def make_move(board, player, position):
    pass

# Function for the opponent's move
def opponent_move(board):
    pass

# Function for the local search algorithm optimization
def local_search_algorithm(board):
    pass

# Function to play the Tic-Tac-Toe game
def play_tic_tac_toe():
    # Initialize the board
    board = [[' ' for _ in range(3)] for _ in range(3)]
    # Define the players
    player = 'X'
    opponent = 'O'

    # Print initial board
    print("Welcome to Tic-Tac-Toe!")
    print_board(board)

    # Main game loop
    while True:
        # Player's move
        print(f"\n{player}'s turn:")
        # Get input from the player
        # Update the board
        # Print the updated board
        # Check for win/draw
        # If game over, break the loop

        # Opponent's move
        print(f"\n{opponent}'s turn:")
        # Opponent makes a move
        # Update the board
        # Print the updated board
        # Check for win/draw
        # If game over, break the loop

        # Local search algorithm optimization for player's move
        print(f"\nOptimizing {player}'s move:")
        # Apply the local search algorithm
        # Update the board
        # Print the updated board
        # Check for win/draw
```

```
        # If game over, break the loop

# Entry point of the program
if __name__ == "__main__":
    # Start the game
    play_tic_tac_toe()
```