Pure Nash Equilibria Finder for N-Player Normal-Form Games

Git link:

https://github.com/razvanbaboiucs/BMDC-game-theory-assginment/tree/master/5-n-player-pure-ne

Algorithm Explanation:

Pseudocode:

```
FUNCTION FindPureNashEquilibria(game)
    equilibria ← empty list

    FOR EACH possible strategy profile in the game DO
        isEquilibrium ← true

        FOR EACH player in the game DO
            currentStrategy ← player's strategy in current profile
            currentPayoff ← player's payoff in current profile

            FOR EACH alternative strategy available to player DO
                IF player would get higher payoff by switching to alternative strategy THEN
                    isEquilibrium ← false
                    BREAK
                END IF
            END FOR

            IF not isEquilibrium THEN BREAK
        END FOR

        IF isEquilibrium THEN
            ADD current strategy profile to equilibria
        END IF
    END FOR

    RETURN equilibria
END FUNCTION
```

Explanation:

1. The algorithm uses a recursive approach to examine all possible strategy profiles in an n-player game.
2. For each complete strategy profile, it checks if it represents a Nash equilibrium by verifying:

    - No player can improve their payoff by unilaterally changing their strategy
    - This is done by comparing the current payoff with potential payoffs from all alternative strategies
3. The algorithm handles n-dimensional payoff matrices to represent the payoffs for each player in all possible strategy combinations.
4. Key features:
    - Supports any number of players (up to a defined maximum)
    - Each player can have a different number of strategies
    - Payoffs can be manually entered or randomly generated
    - Displays n-dimensional payoff matrices for each player
    - Returns all pure Nash equilibria found in the game

Data Structure:

- The algorithm uses n-dimensional arrays to represent payoff matrices
- For a game with n players, each payoff matrix has n dimensions
- Each dimension i corresponds to the number of strategies available to player i
- Access to specific payoffs is done using string-based indexing: `matrix[i][j][k]` becomes `matrix["i][j][k"]`

Time and Space Complexity:

- Time Complexity: O(s^n * n * s), where s is the maximum number of strategies per player and n is the number of players
    - O(s^n) to check all possible strategy profiles
    - O(n * s) to verify if each profile is a Nash equilibrium
- Space Complexity: O(n * s^n) to store the payoff matrices for all players

Example Games

Game 1: 2-Player Game (2x2)

Pure Nash Equilibria Finder for N-Player Normal-Form Games

Enter number of players (maximum 10): 2 Enter number of strategies for Player 1: 2 Enter number of strategies for Player 2: 2 Do you want random payoffs? (yes/no): yes

Payoff Matrix for Player 1: [[75, 42] [91, 12]]

Payoff Matrix for Player 2: [[18, 84] [53, 36]]

Pure Nash Equilibria: Found 1 pure Nash Equilibria: 1. Strategy profile: (1, 2) 2. Strategy profile: (2, 1)

Game 2: 3-Player Game (2x2x2)

Pure Nash Equilibria Finder for N-Player Normal-Form Games

Enter number of players (maximum 10): 3 Enter number of strategies for Player 1: 2 Enter number of strategies for Player 2: 2 Enter number of strategies for Player 3: 2 Do you want random payoffs? (yes/no): yes

Payoff Matrix for Player 1: [[[12, 48] [50, 91]] [[26, 44] [42, 89]]]

Payoff Matrix for Player 2: [[[28, 5] [17, 68]] [[13, 81] [24, 69]]]

Payoff Matrix for Player 3: [[[46, 43] [22, 97]] [[71, 24] [95, 81]]]

Pure Nash Equilibria: Found 1 pure Nash Equilibria: 1. Strategy profile: (1, 2, 2)

Game 3: 3-Player Game with Different Strategy Counts (2x3x2)

Pure Nash Equilibria Finder for N-Player Normal-Form Games

Enter number of players (maximum 10): 3 Enter number of strategies for Player 1: 2 Enter number of strategies for Player 2: 3 Enter number of strategies for Player 3: 2 Do you want random payoffs? (yes/no): yes

Payoff Matrix for Player 1: [[[35, 63] [32, 99] [92, 87]] [[65, 87] [3, 30] [19, 95]]]

Payoff Matrix for Player 2: [[[98, 22] [92, 28] [52, 48]] [[63, 50] [51, 35] [84, 11]]]

Payoff Matrix for Player 3: [[[21, 9] [27, 3] [43, 54]] [[83, 61] [69, 60] [68, 81]]]

Performance Considerations

For games with many players or strategies, the algorithm's performance can degrade exponentially due to the need to check all possible strategy combinations. Potential optimizations include:

1. Iterated elimination of strictly dominated strategies to reduce the search space
2. Parallelization of the search process