**Tìm kiếm Đối kháng: Giải quyết Tic-Tac-Toe bằng Tìm kiếm Monte Carlo Thuần túy**

**Giới thiệu**

Trò chơi nhiều người chơi có thể được triển khai như sau:

1. Hành động không xác định: Đối thủ được xem như một phần của môi trường với các hành động không xác định. Tính không xác định là kết quả của các nước đi không rõ của đối thủ.
2. Quyết định Tối ưu: Tìm kiếm Minimax (tìm kiếm toàn bộ cây trò chơi) và cắt tỉa alpha-beta.
3. Tìm kiếm Cây Alpha-Beta Heuristic: Cắt bớt cây tìm kiếm và sử dụng heuristic để ước tính giá trị trạng thái.
4. **Tìm kiếm Monte Carlo:** Mô phỏng các lượt chơi (playouts) để ước tính giá trị trạng thái.

Ở đây, chúng ta sẽ triển khai tìm kiếm cho Tic-Tac-Toe. Trò chơi này là một **trò chơi có tổng bằng không (zero-sum game)**: Thắng bởi x kết quả là +1, thắng bởi o là -1 và hòa có giá trị là 0. Max chơi x và cố gắng tối đa hóa kết quả trong khi Min chơi o và cố gắng tối thiểu hóa kết quả.

Chúng ta sẽ triển khai:

* Tìm kiếm Monte Carlo thuần túy

Mã cho các hàm cơ bản được sử dụng cho tìm kiếm được triển khai trong tictactoe.py.

**Mã nguồn:**

Python

from tictactoe import empty\_board, actions, result, terminal, utility, other, show\_board

from tictactoe import random\_player, play

%precision 3

**Kết quả:**

'%.3f'

**Tìm kiếm Monte Carlo Thuần túy**

Xem AIMA trang 161ff.

Chúng ta triển khai một phiên bản cực kỳ đơn giản.

Đối với trạng thái hiện tại:

1. Mô phỏng $N$ lượt chơi ngẫu nhiên cho mỗi hành động có thể và
2. chọn hành động có giá trị tiện ích trung bình cao nhất.

**Lưu ý quan trọng:** chúng ta sử dụng ở đây một chính sách chơi ngẫu nhiên, cuối cùng chỉ tạo ra một tìm kiếm ngẫu nhiên hoạt động tốt cho bài toán đồ chơi này. Đối với các ứng dụng thực tế, bạn cần mở rộng mã với:

1. một **chính sách chơi (playout policy)** tốt (ví dụ: học được bằng cách tự chơi) và
2. một **chiến lược lựa chọn (selection strategy)** (ví dụ: UCB1).

**Mô phỏng các lượt chơi (playouts)**

**Mã nguồn:**

Python

import numpy as np

def playout(state, action, player = 'x'):

"""Perform a random playout starting with the given action on the given board

and return the utility of the finished game."""

state = result(state, player, action)

current\_player = other(player)

while(True):

# reached terminal state?

u = utility(state, player)

if u is not None: return(u)

# we use a random playout policy

a = np.random.choice(actions(state))

state = result(state, current\_player, a)

#print(state)

# switch between players

current\_player = other(current\_player)

# Playout for action 0 (top-left corner)

board = empty\_board()

print(playout(board, 0))

print(playout(board, 0))

print(playout(board, 0))

print(playout(board, 0))

print(playout(board, 0))

**Kết quả:**

1

1

1

-1

1

**Mã nguồn:**

Python

def playouts(board, action, player = 'x', N = 100):

"""Perform N playouts following the given action for the given board."""

return [ playout(board, action, player) for i in range(N) ]

u = playouts(board, 0)

print("Playout results:", u)

print(f"mean utility: {np.mean(u)}")

p\_win = sum(np.array(u) == +1)/len(u)

p\_loss = sum(np.array(u) == -1)/len(u)

p\_draw = sum(np.array(u) == 0)/len(u)

print(f"win probability: {p\_win}")

print(f"loss probability: {p\_loss}")

print(f"draw probability: {p\_draw}")

**Kết quả:**

Playout results: [1, 1, 1, 1, 1, 1, -1, 1, -1, 0, 1, 1, 1, 1, 1, 1, 0, -1, 0, 1, 0, -1, 1, -1, 1, 1, 1, 1, 0, 1, 1, 1, 1, -1, -1, -1, -1, 1, 0, -1, 1, 1, 1, 1, 1, -1, 1, 1, 1, 1, 0, -1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, -1, 1, 0, -1, 1, 1, -1, -1, 1, 1, 1, 1, 0, 0, -1, -1, -1, -1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, -1]

mean utility: 0.4

win probability: 0.64

loss probability: 0.24

draw probability: 0.12

**Lưu ý:** Điều này cho thấy người chơi đi trước có lợi thế đáng kể trong **chơi hoàn toàn ngẫu nhiên.** Người chơi thực sự không chơi ngẫu nhiên, một chính sách chơi tốt hơn sẽ hữu ích.

**Tìm kiếm Monte Carlo Thuần túy**

**Mã nguồn:**

Python

import math

DEBUG = 1

def pmcs(board, N = 100, player = 'x'):

"""Pure Monte Carlo Search. Returns the action that has the largest average utility.

The N playouts are evenly divided between the possible actions."""

global DEBUG

acts = actions(board)

n = math.floor(N/len(acts))

if DEBUG >= 1: print(f"Actions: {acts} ({N} total playouts = {n} playouts per action)")

ps = { i : np.mean(playouts(board, i, player, N = n)) for i in acts }

if DEBUG >= 1: display(ps)

action = max(ps, key=ps.get)

return action

board = empty\_board()

display(board)

%time print(pmcs(board))

print()

print("10000 playouts give a better utility estimate.")

%time print(pmcs(board, N = 10000))

**Kết quả:**

[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']

Actions: [0, 1, 2, 3, 4, 5, 6, 7, 8] (100 total playouts = 11 playouts per action)

{0: -0.091,

1: 0.182,

2: 0.545,

3: 0.727,

4: 0.636,

5: 0.091,

6: 0.545,

7: 0.000,

8: 0.273}

3

CPU times: user 32.1 ms, sys: 30 µs, total: 32.2 ms

Wall time: 30.4 ms

10000 playouts give a better utility estimate.

Actions: [0, 1, 2, 3, 4, 5, 6, 7, 8] (10000 total playouts = 1111 playouts per action)

{0: 0.348,

1: 0.147,

2: 0.344,

3: 0.153,

4: 0.536,

5: 0.183,

6: 0.338,

7: 0.217,

8: 0.321}

4

CPU times: user 1.7 s, sys: 51.5 ms, total: 1.76 s

Wall time: 1.72 s

Có vẻ như ô trung tâm và các góc tốt hơn nhiều.

**Một số Thử nghiệm**

**x sắp thắng (chơi 8)**

**Mã nguồn:**

Python

board = empty\_board()

board[0] = 'x'

board[1] = 'o'

board[3] = 'o'

board[4] = 'x'

print("Board:")

show\_board(board)

print()

%time display(pmcs(board))

**Kết quả:**

Board:

[Hình ảnh: Bàn cờ Tic-Tac-Toe với x ở 0, 4 và o ở 1, 3]

Actions: [2, 5, 6, 7, 8] (100 total playouts = 20 playouts per action)

{2: 0.700, 5: 0.500, 6: 0.900, 7: 0.700, 8: 1.000}

8

CPU times: user 12.9 ms, sys: 17.2 ms, total: 30.2 ms

Wall time: 9.82 ms

**o sắp thắng**

**Mã nguồn:**

Python

board = empty\_board()

board[0] = 'o'

board[1] = 'o'

board[3] = 'o'

board[4] = 'x'

board[8] = 'x'

print("Board:")

show\_board(board)

print()

%time display(pmcs(board))

print()

%time display(pmcs(board, N = 1000))

**Kết quả:**

Board:

[Hình ảnh: Bàn cờ Tic-Tac-Toe với o ở 0, 1, 3 và x ở 4, 8]

Actions: [2, 5, 6, 7] (100 total playouts = 25 playouts per action)

{2: 0.120, 5: -0.760, 6: -0.040, 7: -0.680}

2

CPU times: user 17 ms, sys: 8.28 ms, total: 25.3 ms

Wall time: 8.24 ms

Actions: [2, 5, 6, 7] (1000 total playouts = 250 playouts per action)

{2: 0.000, 5: -0.648, 6: 0.056, 7: -0.632}

6

CPU times: user 133 ms, sys: 131 ms, total: 265 ms

Wall time: 87 ms

**x có thể hòa nếu chọn 7**

**Mã nguồn:**

Python

board = empty\_board()

board[0] = 'x'

board[1] = 'o'

board[2] = 'x'

board[4] = 'o'

print("Board:")

show\_board(board)

print()

%time display(pmcs(board))

**Kết quả:**

Board:

[Hình ảnh: Bàn cờ Tic-Tac-Toe với x ở 0, 2 và o ở 1, 4]

Actions: [3, 5, 6, 7, 8] (100 total playouts = 20 playouts per action)

{3: -0.400, 5: 0.100, 6: -0.350, 7: 0.050, 8: -0.300}

5

CPU times: user 22.6 ms, sys: 16.8 ms, total: 39.4 ms

Wall time: 12.8 ms

**Bàn cờ trống: Chỉ có thể đảm bảo hòa**

**Mã nguồn:**

Python

board = empty\_board()

print("Board:")

show\_board(board)

print()

%time display(pmcs(board, N = 5000))

**Kết quả:**

Board:

[Hình ảnh: Bàn cờ Tic-Tac-Toe trống]

Actions: [0, 1, 2, 3, 4, 5, 6, 7, 8] (5000 total playouts = 555 playouts per action)

{0: 0.378,

1: 0.164,

2: 0.373,

3: 0.276,

4: 0.517,

5: 0.277,

6: 0.332,

7: 0.196,

8: 0.305}

4

CPU times: user 938 ms, sys: 242 ms, total: 1.18 s

Wall time: 860 ms

**Một tình huống xấu**

**Mã nguồn:**

Python

board = empty\_board()

board[0] = 'o'

board[2] = 'x'

board[8] = 'o'

print("Board:")

show\_board(board)

print()

%time display(pmcs(board))

**Kết quả:**

Board:

[Hình ảnh: Bàn cờ Tic-Tac-Toe với o ở 0, 8 và x ở 2]

Actions: [1, 3, 4, 5, 6, 7] (100 total playouts = 16 playouts per action)

{1: -0.750, 3: -0.750, 4: 0.312, 5: -0.750, 6: -0.500, 7: -0.688}

4

CPU times: user 22.8 ms, sys: 16.9 ms, total: 39.7 ms

Wall time: 13.1 ms

**Lưu ý:** Có vẻ như người chơi o ngẫu nhiên rất ít khả năng chặn x và tận dụng bẫy bằng cách chơi ở góc dưới bên trái!

**Thử nghiệm**

**Tìm kiếm Monte Carlo Thuần túy vs. Ngẫu nhiên**

**Mã nguồn (Định nghĩa người chơi PMCS):**

Python

def pmcs10\_player(board, player = 'x'):

action = pmcs(board, N = 10, player = player)

return action

def pmcs100\_player(board, player = 'x'):

action = pmcs(board, N = 100, player = player)

return action

def pmcs1000\_player(board, player = 'x'):

action = pmcs(board, N = 1000, player = player)

return action

DEBUG = 1

print("PMCS vs. random:")

display(play(pmcs10\_player, random\_player, N = 1))

**Kết quả:**

PMCS vs. random:

Actions: [0, 1, 2, 3, 4, 5, 6, 7, 8] (10 total playouts = 1 playouts per action)

{0: 1.000,

1: -1.000,

2: 1.000,

3: 1.000,

4: 1.000,

5: 1.000,

6: 1.000,

7: -1.000,

8: 1.000}

Actions: [2, 3, 4, 5, 6, 7, 8] (10 total playouts = 1 playouts per action)

{2: 0.000, 3: 1.000, 4: 1.000, 5: 1.000, 6: 0.000, 7: 0.000, 8: 1.000}

Actions: [2, 4, 5, 6, 8] (10 total playouts = 2 playouts per action)

{2: 1.000, 4: 1.000, 5: 0.000, 6: 1.000, 8: 1.000}

Actions: [4, 5, 6] (10 total playouts = 3 playouts per action)

{4: 0.333, 5: -1.000, 6: 1.000}

{'x': 1, 'o': 0, 'd': 0}

**Mã nguồn (Chơi nhiều ván với PMCS(10)):**

Python

DEBUG = 0

print("PMCS (10) vs. random:")

%time display(play(pmcs10\_player, random\_player))

print()

print("random vs. PMCS (10)")

%time display(play(random\_player, pmcs10\_player))

**Kết quả:**

PMCS (10) vs. random:

{'x': 91, 'o': 5, 'd': 4}

CPU times: user 567 ms, sys: 150 ms, total: 717 ms

Wall time: 511 ms

random vs. PMCS (10)

{'x': 20, 'o': 68, 'd': 12}

CPU times: user 287 ms, sys: 2.22 ms, total: 289 ms

Wall time: 272 ms

**Mã nguồn (Chơi nhiều ván với PMCS(100)):**

Python

DEBUG = 0

print("PMCS (100) vs. random:")

%time display(play(pmcs100\_player, random\_player))

print()

print("random vs. PMCS (100)")

%time display(play(random\_player, pmcs100\_player))

**Kết quả:**

PMCS (100) vs. random:

{'x': 98, 'o': 0, 'd': 2}

CPU times: user 6.03 s, sys: 228 ms, total: 6.26 s

Wall time: 6 s

random vs. PMCS (100)

{'x': 7, 'o': 82, 'd': 11}

CPU times: user 4.29 s, sys: 50.7 ms, total: 4.34 s

Wall time: 4.25 s

**Mã nguồn (PMCS(100) vs PMCS(10)):**

Python

DEBUG = 0

print("PMCS (100) vs. PMCS (10):")

%time display(play(pmcs100\_player, pmcs10\_player))

print()

print("PMCS (10) vs. PMCS (100)")

%time display(play(pmcs10\_player, pmcs100\_player))

**Kết quả:**

PMCS (100) vs. PMCS (10):

{'x': 88, 'o': 4, 'd': 8}

CPU times: user 6.12 s, sys: 264 ms, total: 6.39 s

Wall time: 6.07 s

PMCS (10) vs. PMCS (100)

{'x': 29, 'o': 54, 'd': 17}

CPU times: user 5.04 s, sys: 135 ms, total: 5.18 s

Wall time: 5.01 s