**Tìm kiếm Đối kháng: Giải quyết Tic-Tac-Toe bằng Tìm kiếm Cây Alpha-Beta Heuristic**

**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 Cây Monte Carlo: Mô phỏng các lượt chơi 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 Cây Alpha-Beta Heuristic

Các thuật toán tìm kiếm trên cây trò chơi và chúng ta có thể trả về một kế hoạch có điều kiện (hoặc kế hoạch một phần nếu sử dụng cắt bớt), nhưng việc triển khai ở đây chỉ xác định và trả về nước đi tối ưu tiếp theo.

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

**Tìm kiếm Cây Alpha-Beta Heuristic**

Xem AIMA trang 156ff.

**Hàm Đánh giá Heuristic**

**Mã nguồn:**

Python

import numpy as np

def eval\_fun(state, player = 'x'):

"""heuristic for utility of state. Returns score for a node:

1. For terminal states it returns the utility.

2. For non-terminal states, it calculates a weighted linear function using features of the state.

The features we look at are 2 in a row/col/diagonal where the 3rd square is empty. We assume that

the more of these positions we have, the higher the chance of winning.

We need to be careful that the utility of the heuristic stays between [-1,1].

Note that the largest possible number of these positions is 2. I weigh the count by 0.4,

guaranteeing that is in the needed range.

Function Returns: heuristic value, terminal?"""

# terminal state?

u = utility(state, player)

if u is not None: return u, True

score = 0

board = np.array(state).reshape((3,3))

diagonals = np.array([[board[i][i] for i in range(len(board))],

[board[i][len(board)-i-1] for i in range(len(board))]])

for a\_board in [board, np.transpose(board), diagonals]:

for row in a\_board:

if sum(row == player) == 2 and any(row ==' '): score += .4

if sum(row == other(player)) == 2 and any(row ==' '): score -= .4

return score, False

**Mã nguồn (Thử nghiệm hàm đánh giá):**

Python

board = empty\_board()

show\_board(board)

print(f"eval for x: {eval\_fun(board)}")

print(f"eval for o: {eval\_fun(board, 'o')}")

board = empty\_board()

board[0] = 'x'

board[1] = 'x'

board[2] = 'x'

show\_board(board)

print(f"eval for x: {eval\_fun(board)}")

print(f"eval for o: {eval\_fun(board, 'o')}")

board = empty\_board()

board[0] = 'x'

board[1] = 'x'

board[3] = 'x'

board[4] = 'o'

board[8] = 'o'

show\_board(board)

print(f"eval for x: {eval\_fun(board)}")

print(f"eval for o: {eval\_fun(board, 'o')}")

board = empty\_board()

board[0] = 'x'

board[1] = 'o'

board[3] = 'x'

board[4] = 'o'

show\_board(board)

print(f"eval for x: {eval\_fun(board)}")

print(f"eval for o: {eval\_fun(board, 'o')}")

Kết quả:

[Hình ảnh: Bàn cờ Tic-Tac-Toe trống]

eval for x: (0, False)

eval for o: (0, False)

[Hình ảnh: Bàn cờ Tic-Tac-Toe với x thắng ở hàng đầu]

eval for x: (1, True)

eval for o: (-1, True)

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

eval for x: (0.8, False)

eval for o: (-0.8, False)

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

eval for x: (0.0, False)

eval for o: (0.0, False)

**Tìm kiếm với Cắt bớt (Cutoff)**

Chúng ta thêm một điểm cắt bớt (cutoff) vào thuật toán DFS đệ quy cho Tìm kiếm Minimax với Cắt tỉa Alpha-Beta (xem AIMA trang 156ff). Chúng ta sử dụng hàm đánh giá heuristic và sao lưu giá trị bằng cách sử dụng tìm kiếm minimax với cắt tỉa alpha-beta để xác định nước đi tiếp theo.

**Mã nguồn:**

Python

import math

# global variables

DEBUG = 1 # 1 ... count nodes, 2 ... debug each node

COUNT = 0

def alpha\_beta\_search(board, cutoff = None, player = 'x'):

"""start the search. cutoff = None is minimax search with alpha-beta pruning."""

global DEBUG, COUNT

COUNT = 0

value, move = max\_value\_ab(board, player, -math.inf, +math.inf, 0, cutoff)

if DEBUG >= 1: print(f"Number of nodes searched (cutoff = {cutoff}): {COUNT}")

return {"move": move, "value": value}

def max\_value\_ab(state, player, alpha, beta, depth, cutoff):

"""player's best move."""

global DEBUG, COUNT

COUNT += 1

# cut off and terminal test

v, terminal = eval\_fun(state, player)

if((cutoff is not None and depth >= cutoff) or terminal):

if(terminal):

alpha, beta = v, v

if DEBUG >= 2: print(f"stopped at {depth}: {state} term: {terminal} eval: {v} [{alpha}, {beta}]" )

return v, None

v, move = -math.inf, None

# check all possible actions in the state, update alpha and return move with the largest value

for a in actions(state):

v2, a2 = min\_value\_ab(result(state, player, a), player, alpha, beta, depth + 1, cutoff)

if v2 > v:

v, move = v2, a

alpha = max(alpha, v)

if v >= beta: return v, move

return v, move

def min\_value\_ab(state, player, alpha, beta, depth, cutoff):

"""opponent's best response."""

global DEBUG, COUNT

COUNT += 1

# cut off and terminal test

v, terminal = eval\_fun(state, player)

if((cutoff is not None and depth >= cutoff) or terminal):

if(terminal):

alpha, beta = v, v

if DEBUG >= 2: print(f"stopped at {depth}: {state} term: {terminal} eval: {v} [{alpha}, {beta}]" )

return v, None

v, move = +math.inf, None

# check all possible actions in the state, update beta and return move with the smallest value

for a in actions(state):

v2, a2 = max\_value\_ab(result(state, other(player), a), player, alpha, beta, depth + 1, cutoff)

if v2 < v:

v, move = v2, a

beta = min(beta, v)

if v <= alpha: return v, move

return v, move

**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(alpha\_beta\_search(board, 2))

print()

%time display(alpha\_beta\_search(board, 4))

print()

%time display(alpha\_beta\_search(board))

**Kết quả:**

Board:

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

Number of nodes searched (cutoff = 2): 13

{'move': 8, 'value': 1}

CPU times: user 2.84 ms, sys: 2.62 ms, total: 5.46 ms

Wall time: 1.79 ms

Number of nodes searched (cutoff = 4): 47

{'move': 2, 'value': 1}

CPU times: user 7.77 ms, sys: 3.76 ms, total: 11.5 ms

Wall time: 3.82 ms

Number of nodes searched (cutoff = None): 61

{'move': 2, 'value': 1}

CPU times: user 7.86 ms, sys: 3.84 ms, total: 11.7 ms

Wall time: 3.92 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(alpha\_beta\_search(board, 2))

print()

%time display(alpha\_beta\_search(board))

**Kết quả:**

Board:

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

Number of nodes searched (cutoff = 2): 11

{'move': 2, 'value': -1}

CPU times: user 3.79 ms, sys: 7.83 ms, total: 11.6 ms

Wall time: 3.87 ms

Number of nodes searched (cutoff = None): 15

{'move': 2, 'value': -1}

CPU times: user 8.43 ms, sys: 4.38 ms, total: 12.8 ms

Wall time: 4.2 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(alpha\_beta\_search(board, 2))

print()

%time display(alpha\_beta\_search(board, 4))

print()

%time display(alpha\_beta\_search(board))

**Kết quả:**

Board:

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

Number of nodes searched (cutoff = 2): 21

{'move': 7, 'value': -0.4}

CPU times: user 21.4 ms, sys: 5.45 ms, total: 26.8 ms

Wall time: 8.81 ms

Number of nodes searched (cutoff = 4): 81

{'move': 7, 'value': 0}

CPU times: user 54.9 ms, sys: 26.7 ms, total: 81.5 ms

Wall time: 26.9 ms

Number of nodes searched (cutoff = None): 101

{'move': 7, 'value': 0}

CPU times: user 66 ms, sys: 67.7 ms, total: 134 ms

Wall time: 48.6 ms

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

**Mã nguồn:**

Python

board = empty\_board()

print("Board:")

show\_board(board)

print()

%time display(alpha\_beta\_search(board, 2))

print()

%time display(alpha\_beta\_search(board, 4))

print()

%time display(alpha\_beta\_search(board))

**Kết quả:**

Board:

[Hình ảnh: Bàn cờ Tic-Tac-Toe trống]

Number of nodes searched (cutoff = 2): 26

{'move': 0, 'value': 0}

CPU times: user 14.4 ms, sys: 13.8 ms, total: 28.1 ms

Wall time: 9.28 ms

Number of nodes searched (cutoff = 4): 541

{'move': 4, 'value': 0.0}

CPU times: user 301 ms, sys: 112 ms, total: 413 ms

Wall time: 182 ms

Number of nodes searched (cutoff = None): 18297

{'move': 0, 'value': 0}

CPU times: user 3.76 s, sys: 0 ns, total: 3.76 s

Wall time: 3.76 s

**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(alpha\_beta\_search(board, 2))

print()

%time display(alpha\_beta\_search(board, 4))

print()

%time display(alpha\_beta\_search(board))

**Kết quả:**

Board:

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

Number of nodes searched (cutoff = 2): 29

{'move': 4, 'value': -0.8}

CPU times: user 20.1 ms, sys: 11.1 ms, total: 31.2 ms

Wall time: 10.3 ms

Number of nodes searched (cutoff = 4): 148

{'move': 1, 'value': -1}

CPU times: user 81.4 ms, sys: 40.3 ms, total: 122 ms

Wall time: 40.5 ms

Number of nodes searched (cutoff = None): 238

{'move': 1, 'value': -1}

CPU times: user 126 ms, sys: 44.6 ms, total: 171 ms

Wall time: 56.8 ms

**Thử nghiệm**

**Minimax với Cắt tỉa Alpha-Beta vs. Ngẫu nhiên**

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

Python

def heuristic2\_player(board, player = 'x'):

return alpha\_beta\_search(board, cutoff = 2, player = player)["move"]

def heuristic4\_player(board, player = 'x'):

return alpha\_beta\_search(board, cutoff = 4, player = player)["move"]

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

return alpha\_beta\_search(board, cutoff = None, player = player)["move"]

DEBUG = 1

print("heuristic2 vs. random:")

display(play(heuristic2\_player, random\_player, N = 3))

**Kết quả:**

heuristic2 vs. random:

Number of nodes searched (cutoff = 2): 26

Number of nodes searched (cutoff = 2): 30

Number of nodes searched (cutoff = 2): 10

Number of nodes searched (cutoff = 2): 26

Number of nodes searched (cutoff = 2): 33

Number of nodes searched (cutoff = 2): 16

Number of nodes searched (cutoff = 2): 26

Number of nodes searched (cutoff = 2): 30

Number of nodes searched (cutoff = 2): 10

{'x': 3, 'o': 0, 'd': 0}

**Mã nguồn (Chơi nhiều ván):**

Python

DEBUG = 0

print("heuristic2 vs. random:")

%time display(play(heuristic2\_player, random\_player))

print("heuristic4 vs. random:")

%time display(play(heuristic4\_player, random\_player))

print()

print("random vs. heuristic2")

%time display(play(random\_player, heuristic2\_player))

print("random vs. heuristic4")

%time display(play(random\_player, heuristic4\_player))

**Kết quả:**

heuristic2 vs. random:

{'x': 93, 'o': 0, 'd': 7}

CPU times: user 393 ms, sys: 1.38 ms, total: 394 ms

Wall time: 387 ms

heuristic4 vs. random:

{'x': 99, 'o': 0, 'd': 1}

CPU times: user 4.92 s, sys: 0 ns, total: 4.92 s

Wall time: 4.92 s

random vs. heuristic2

{'x': 1, 'o': 87, 'd': 12}

CPU times: user 375 ms, sys: 0 ns, total: 375 ms

Wall time: 367 ms

random vs. heuristic4

{'x': 0, 'o': 81, 'd': 19}

CPU times: user 3.29 s, sys: 0 ns, total: 3.29 s

Wall time: 3.29 s

**Heuristic vs. Minimax với Cắt tỉa Alpha-Beta**

**Mã nguồn:**

Python

DEBUG = 0

# Note: No randomness -> play only once

print("heuristic2 vs. alpha\_beta")

%time display(play(heuristic2\_player, alpha\_beta\_player, N = 1))

print()

print("alpha\_beta vs. heuristic2")

%time display(play(alpha\_beta\_player, heuristic2\_player, N = 1))

print()

print("heuristic4 vs alpha\_beta")

%time display(play(heuristic4\_player, alpha\_beta\_player, N = 1))

print()

print("alpha\_beta vs. heuristic4")

%time display(play(alpha\_beta\_player, heuristic4\_player, N = 1))

**Kết quả:**

heuristic2 vs. alpha\_beta

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

CPU times: user 131 ms, sys: 0 ns, total: 131 ms

Wall time: 130 ms

alpha\_beta vs. heuristic2

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

CPU times: user 822 ms, sys: 0 ns, total: 822 ms

Wall time: 822 ms

heuristic4 vs alpha\_beta

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

CPU times: user 179 ms, sys: 0 ns, total: 179 ms

Wall time: 179 ms

alpha\_beta vs. heuristic4

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

CPU times: user 973 ms, sys: 0 ns, total: 973 ms

Wall time: 973 ms

**Heuristic vs. Heuristic**

**Mã nguồn:**

Python

DEBUG = 0

# Note: No randomness -> play only once

print("heuristic2 vs. heuristic4")

%time display(play(heuristic2\_player, heuristic4\_player, N = 1))

print()

print("heuristic4 vs. heuristic2")

%time display(play(heuristic4\_player, heuristic2\_player, N = 1))

**Kết quả:**

heuristic2 vs. heuristic4

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

CPU times: user 27.5 ms, sys: 0 ns, total: 27.5 ms

Wall time: 27.1 ms

heuristic4 vs. heuristic2

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

CPU times: user 51.1 ms, sys: 0 ns, total: 51.1 ms

Wall time: 50.7 ms