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

**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:

* Chúng ta tăng cường Tìm kiếm Monte Carlo Thuần túy bằng cách sử dụng chính sách lựa chọn giới hạn tin cậy trên (upper confidence bound - UCB1). Nghĩa là, chúng ta sử dụng UCB1 để xác định hành động nào sẽ thực hiện lượt chơi tiếp theo. Điều này sẽ cho phép thuật toán tập trung vào các hành động cần thu thập thêm thông tin. Lưu ý rằng Giới hạn Tin cậy Trên áp dụng cho Cây (Upper Confidence Bounds applied to Trees - UCT) hoàn chỉnh tạo ra một cây và bước mở rộng trong mã cần được thêm vào. Nhánh cây cho nước đi đã chọn được bảo tồn và sử dụng cho các nước đi trong tương lai.

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 Monte Carlo với Giới hạn Tin cậy Trên (Upper Confidence Bound)**

Xem AIMA trang 163.

Chúng ta tăng cường tìm kiếm Monte Carlo thuần túy bằng cách sử dụng UCB1 làm **chính sách lựa chọn (selection policy)**. Điều này có thể được xem như một phiên bản hạn chế của UTC:

* Chỉ xây dựng một cây có độ sâu 1 và sử dụng chính sách lựa chọn UBC1.
* Sử dụng một chính sách chơi ngẫu nhiên.

**Lưu ý về chính sách chơi:** 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 một **chính sách chơi (playout policy)** tốt (ví dụ: heuristic được tạo thủ công hoặc một mạng nơ-ron [học được bằng cách tự chơi sử dụng học tăng cường](https://towardsdatascience.com/how-to-teach-an-ai-to-play-games-deep-reinforcement-learning-28f9b920440a)).

**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)

board = empty\_board()

print(playout(board, 0))

print(playout(board, 0))

print(playout(board, 0))

**Kết quả:**

-1

0

1

**Giới hạn Tin cậy Trên Áp dụng cho Cây (Hạn chế ở Độ sâu 1)**

**Mã nguồn:**

Python

import pandas as pd

import math

DEBUG = 1

def UCT\_depth1(board, N = 100, player = 'x'):

"""Upper Confidence bound applied to Trees for limited tree depth of 1.

Simulation budget is N playouts."""

global DEBUG

C = math.sqrt(2) # tradeoff constant

# the tree is 1 action deep

acts = actions(board)

u = [0] \* len(acts) # total utility through actions

n = [0] \* len(acts) # number of playouts through actions

n\_parent = 0 # total playouts so far (i.e., number of playouts through parent)

# make sure we try each action once

UCB1 = [+math.inf] \* len(acts)

for i in range(N):

# Select

action\_id = UCB1.index(max(UCB1))

# Expand

# UTC would expand the tree. We keep the tree at depth 1, essentially performing

# Pure Monte Carlo search with an added UCB1 selection policy.

# Simulate

p = playout(board, acts[action\_id], player = player)

# Back-Propagate (i.e., update counts and UCB1)

u[action\_id] += p

n[action\_id] += 1

n\_parent += 1

for action\_id in range(len(acts)):

if n[action\_id] > 0:

UCB1[action\_id] = u[action\_id] / n[action\_id] + C \* math.sqrt(math.log(n\_parent) / n[action\_id])

# return action with largest number of playouts

action = acts[n.index(max(n))]

if DEBUG >= 1:

print(pd.DataFrame({'action':acts,

'total utility':u,

'# of playouts':n,

'UCB1':UCB1}))

print()

print(f"Best action: {action}")

return action

**Mã nguồn (Thử nghiệm UCT\_depth1):**

Python

board = empty\_board()

display(board)

%timeit -n 1 -r 1 UCT\_depth1(board, N = 1000)

**Kết quả:**

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

action total utility # of playouts UCB1

0 0 38 110 0.699849

1 1 28 91 0.697332

2 2 26 87 0.697346

3 3 7 46 0.700204

4 4 238 448 0.706858

5 5 -5 14 0.636246

6 6 25 86 0.691504

7 7 1 33 0.677336

8 8 24 85 0.685510

Best action: 4

250 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)

**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()

%timeit -n1 -r1 UCT\_depth1(board)

**Kết quả:**

Board:

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

action total utility # of playouts UCB1

0 2 23 25 1.526971

1 5 12 16 1.508714

2 6 8 13 1.457102

3 7 9 13 1.534025

4 8 33 33 1.528300

Best action: 8

9.07 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)

**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()

%timeit -n1 -r1 UCT\_depth1(board, N = 1000)

**Kết quả:**

Board:

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

action total utility # of playouts UCB1

0 2 4 620 0.155727

1 5 -11 11 0.120694

2 6 -5 341 0.186620

3 7 -16 28 0.131004

Best action: 2

129 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)

**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()

%timeit -n1 -r1 UCT\_depth1(board, N = 1000)

**Kết quả:**

Board:

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

action total utility # of playouts UCB1

0 3 -12 42 0.287819

1 5 -9 75 0.309193

2 6 -11 26 0.305871

3 7 141 790 0.310723

4 8 -10 67 0.304841

Best action: 7

149 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)

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

**Mã nguồn:**

Python

board = empty\_board()

print("Board:")

show\_board(board)

print()

%timeit -n1 -r1 UCT\_depth1(board, N = 100)

%timeit -n1 -r1 UCT\_depth1(board, N = 5000)

**Kết quả:**

Board:

[Hình ảnh: Bàn cờ Tic-Tac-Toe trống]

action total utility # of playouts UCB1

0 0 11 23 1.111072

1 1 1 9 1.122729

2 2 -2 2 1.145966

3 3 3 12 1.126087

4 4 -2 6 0.905641

5 5 0 8 1.072983

6 6 4 15 1.050263

7 7 6 16 1.133714

8 8 0 9 1.011618

Best action: 0

33.6 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)

action total utility # of playouts UCB1

0 0 24 121 0.573554

1 1 30 135 0.577441

2 2 135 372 0.576892

3 3 28 131 0.574342

4 4 1834 3591 0.579595

5 5 3 63 0.567607

6 6 85 264 0.575986

7 7 15 97 0.573700

8 8 68 226 0.575427

Best action: 4

849 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)

**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()

display(UCT\_depth1(board))

**Kết quả:**

Board:

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

action total utility # of playouts UCB1

0 1 -4 12 0.542754

1 3 -4 4 0.517427

2 4 3 42 0.539717

3 5 -4 4 0.517427

4 6 -2 26 0.518261

5 7 -5 12 0.459420

Best action: 4

4

**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 UCB1):**

Python

def ucb1\_10\_player(board, player = 'x'):

action = UCT\_depth1(board, N = 10, player = player)

return action

def ucb1\_100\_player(board, player = 'x'):

action = UCT\_depth1(board, N = 100, player = player)

return action

**Mã nguồn (UCB1(10) vs Ngẫu nhiên):**

Python

DEBUG = 0

print("UCB1 (10) vs. random:")

%timeit -n1 -r1 display(play(ucb1\_10\_player, random\_player))

print()

print("random vs. UCB1 (10):")

%timeit -n1 -r1 display(play(random\_player, ucb1\_10\_player))

**Kết quả:**

UCB1 (10) vs. random:

{'x': 91, 'o': 6, 'd': 3}

427 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)

random vs. UCB1 (10):

{'x': 20, 'o': 71, 'd': 9}

339 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)

**Mã nguồn (UCB1(100) vs Ngẫu nhiên):**

Python

DEBUG = 0

print("UCB1 (100) vs. random:")

%timeit -n 1 -r 1 display(play(ucb1\_100\_player, random\_player))

print()

print("random vs. UCB1 (100):")

%timeit -n 1 -r 1 display(play(random\_player, ucb1\_100\_player))

**Kết quả:**

UCB1 (100) vs. random:

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

3.74 s ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)

random vs. UCB1 (100):

{'x': 4, 'o': 85, 'd': 11}

4.41 s ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)

**Mã nguồn (UCB1(100) vs UCB1(10)):**

Python

DEBUG = 0

print("UCB1 (100) vs. UCB1 (10):")

%timeit -n 1 -r 1 display(play(ucb1\_100\_player, ucb1\_10\_player))

print()

print("UCB1 (10) vs. UCB1 (100):")

%timeit -n 1 -r 1 display(play(ucb1\_10\_player, ucb1\_100\_player))

**Kết quả:**

UCB1 (100) vs. UCB1 (10):

{'x': 88, 'o': 1, 'd': 11}

6.09 s ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)

UCB1 (10) vs. UCB1 (100):

{'x': 32, 'o': 53, 'd': 15}

4.74 s ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)