Trường đại học bách khoa hà nội

Báo cáo môn học:

Trí tuệ nhân tạo

Game đánh cờ Caro sử dụng thuật toán cắt tỉa Alpha-Beta

Sinh viên thực hiện:
Tạ Quang Tùng
Đỗ Tiến Đạt

Giáo viên hướng dẫn:
TS. Lê Thanh Hương

Hà Nội, Ngày 8 tháng 5 năm 2018

Mục lục

1 Các thuật toán tìm kiếm trong game đối kháng

1.1 Thuật toán minimax

Trong Lý thuyết trò chơi (game theory), giá trị maximin của một người chơi là giá trị cao nhất mà người chơi chắc chắn có thể đạt được mà chưa cần biết những lượt chơi của những người chơi khác. Được định nghĩa là:

$$\underline{v_i} = \max_{a_i} \min_{a_{-i}} v_i(a_i, a_{-i})$$

Trong đó:

- i là thứ tự người chơi.
- $-i$ là tất cả các người chơi ngoại trừ người chơi i.
- a_i là hành động được tạo bởi người chơi i.
- a_{-i} là những hành động được tạo bởi tất cả những người chơi khác.
- v_i là hàm giá trị của người chơi i.

Ngược lại, giá trị minimax của một người chơi là giá trị nhỏ nhất mà những người chơi khác có thể đạt được, mà không cần biết đến lượt đi của người chơi đang xét. Nó được định nghĩa là:

$$\overline{v_i} = \min_{a_{-i}} \max_{a_i} v_i(a_i, a_{-i})$$

Thuật toán Minimax là thuật toán đệ quy cho việc tìm kiếm nước đi tiếp theo của một người chơi trong một trò chơi n người chơi, thông thường là 2. Một giá trị được gán cho mỗi trạng thái trong game. Giá trị này được tính toán bằng hàm ước lượng vị trí các đối tượng trong game và có ý nghĩa chỉ định mức độ tốt của một người chơi khi mà đi tới được trạng thái đó. Người chơi sẽ tạo một nước đi mà tối đa hóa giá trị tối thiểu của trạng thái mà nước đi của những người chơi khác tạo ra ngay sau đó. Thuật toán có thể được mô tả bằng các node trong một cây trạng thái.

Mã giả của thuật toán minimax trong trường hợp giới hạn độ sâu như sau:

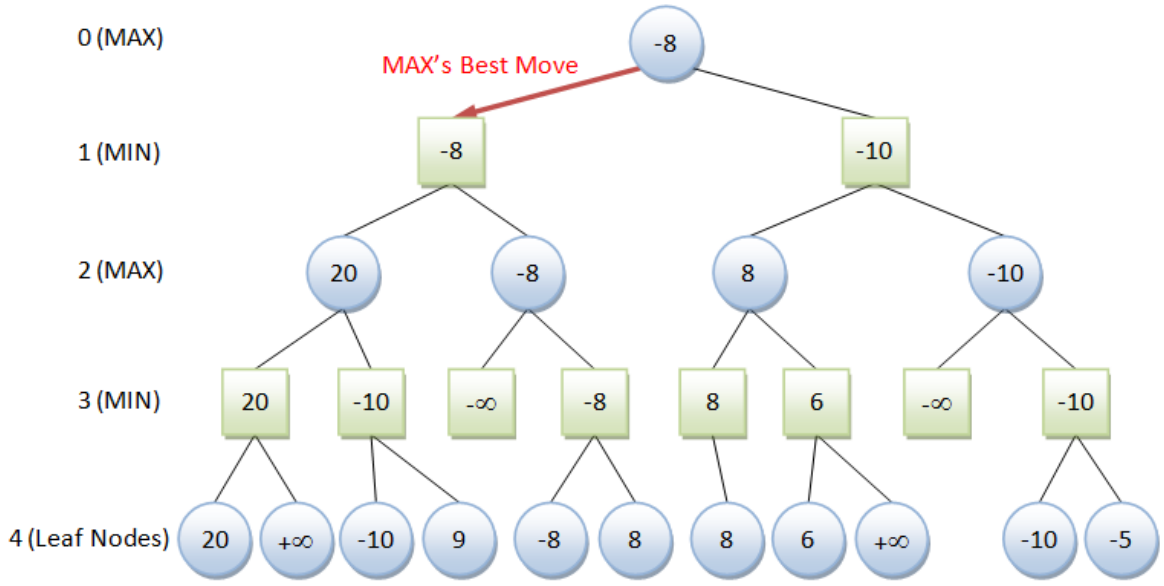
Input:

- node: Một object của lớp biểu diễn trạng thái của game.
- depth: Độ sâu sẽ tìm kiếm.
- maximizing: Biến boolen xác định xem là người chơi max (True) hay min (False).

Output: Giá trị heurisic nhỏ nhất / lớn nhất.

```
def minimax(node, depth, maximizing):
    if depth == 0 or node.is_terminal():
        return node.heuristic_value()
    if maximizing:
        best_value = -infinity
        for child in node.children():
            v = minimax(child, depth - 1, False)
            best_value = max(best_value, v)
        return best_value
    else:
        best_value = +infinity
        for child in node.children():
            v = minimax(child, depth - 1, True)
            best_value = min(best_value, v)
        return best_value
```

Hình 1: Cây trạng thái của thuật toán minimax



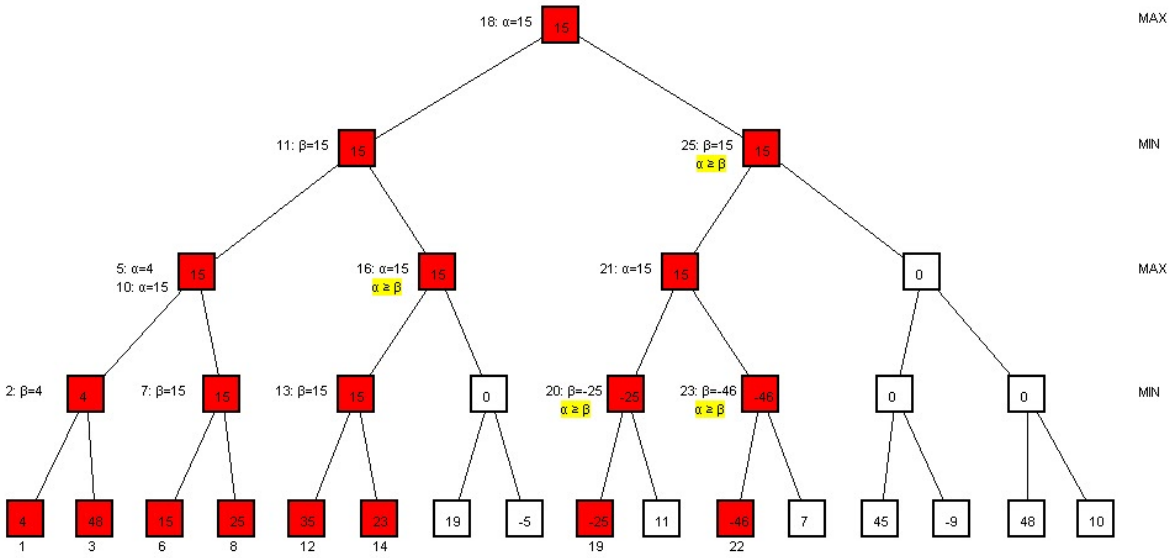
Thuật toán minimax có độ phức tạp lũy thừa theo độ sâu và số lượng nhánh của mỗi node. Do đó nó không khả thi khi áp dụng vào thực tế. Tuy nhiên, hiệu năng của thuật toán có thể tăng lên đáng kể nhờ sử dụng cắt tỉa Alpha-Beta.

1.2 Thuật toán cắt tỉa Alpha-Beta

Là thuật toán tìm kiếm cải tiến thuật toán tìm kiếm minimax bằng cách giảm số node phải đánh giá. Nó sẽ dừng hoàn toàn đánh giá một nước đi nếu như nó có thể chứng minh được rằng nước đi đó là tồi hơn những nước đi đã được xét.

Ý tưởng của thuật toán là duy trì 2 giá trị: alpha và beta. Trong đó alpha đại diện cho giá trị nhỏ nhất của người chơi max được bảo đảm sẽ đạt được và beta đại diện cho giá trị lớn nhất của người chơi min được bảo đảm sẽ đạt được. Nếu như giá trị lớn nhất của người chơi min (beta) được bảo đảm đạt được nhỏ hơn giá trị nhỏ nhất của người chơi max (alpha) được bảo đảm đạt được (beta <= alpha) thì người chơi max sẽ không cần xét đến những node con cháu của node hiện tại đang xét - tương tự với người chơi min.

Hình 2: Cắt tỉa Alpha-Beta



Lợi ích của thuật toán thuật toán cắt tỉa Alpha-Beta dựa vào việc các nhánh tìm kiếm trong thuật toán minimax có thể được loại bỏ. Từ đó, thời gian tìm kiếm có thể được dành cho những nhánh con "hứa hẹn" hơn. Với độ phức tạp thuật toán:

- Trong trường hợp tồi nhất: $O(b^d)$
- Trong trường hợp tốt nhất: $O(\sqrt{b^d})$

Trong đó b là số lượng node con của một node và d là độ sâu tìm kiếm.

Mã giả của thuật toán:
Input:

- node: Một object của lớp biểu diễn trạng thái của game.
- depth: Độ sâu sẽ tìm kiếm.
- alpha: Giá trị alpha.
- beta: Giá trị beta.
- maximizing: Biến boolen xác định xem là người chơi max (True) hay min (False).

Output: Giá trị heurisic nhỏ nhất / lớn nhất.

```
def alphabeta(node, depth, alpha, beta, maximizing):
    if depth == 0 or node.is_terminal():
        return node.heuristic_value()

    if maximizing:
        v = -infinity
        for child in node.children():
            res = alphabeta(child, depth-1, alpha, beta, False)
            v = max(v, res)
            alpha = max(alpha, v)
            if beta <= alpha:
                break
        return v
    else:
        v = +infinity
        for child in node.children():
            res = alphabeta(child, depth-1, alpha, beta, True)
            v = min(v, res)
            beta = min(beta, v)
            if beta <= alpha:
                break
        return v
```

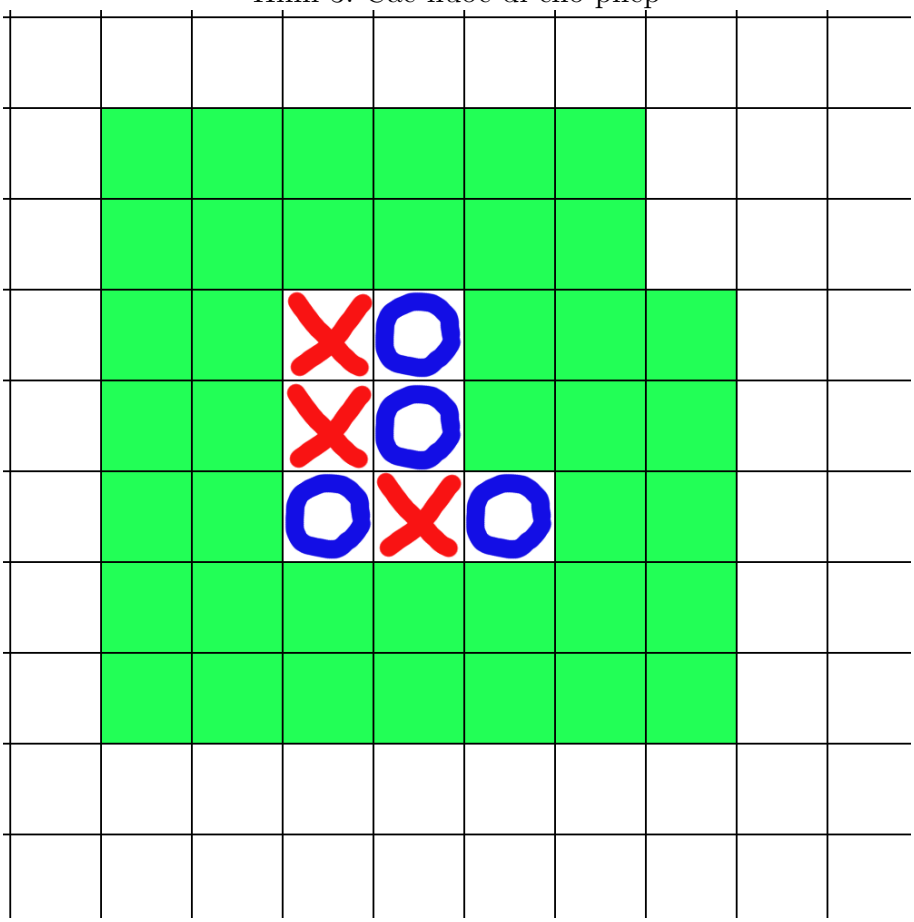
2 Các thuật toán áp dụng trong game cờ caro

Mô hình bàn cờ được áp dụng là sử dụng ma trận có kích thước lớn tùy ý.

2.1 Xác định các nước đi cho phép

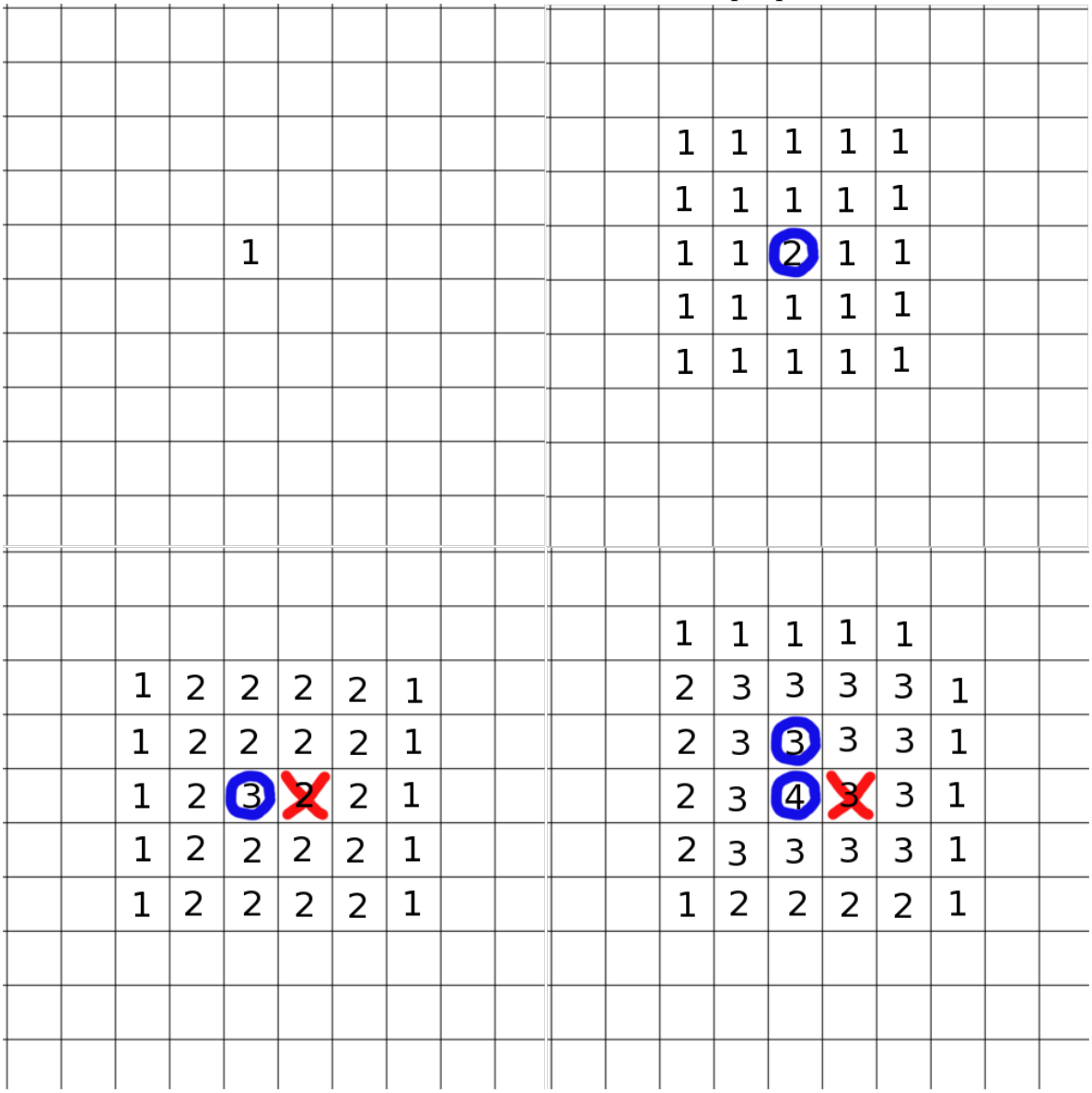
Ý tưởng: Các nước đi cho phép ở một trạng thái bất kì của game được xác định là khoảng cách tối đa là 2 đến ít nhất một quân cờ đã đánh trước đó trong game (Quân cờ đầu tiên chỉ có 1 nước có thể đi duy nhất là tại trung tâm bàn cờ).

Hình 3: Các nước đi cho phép



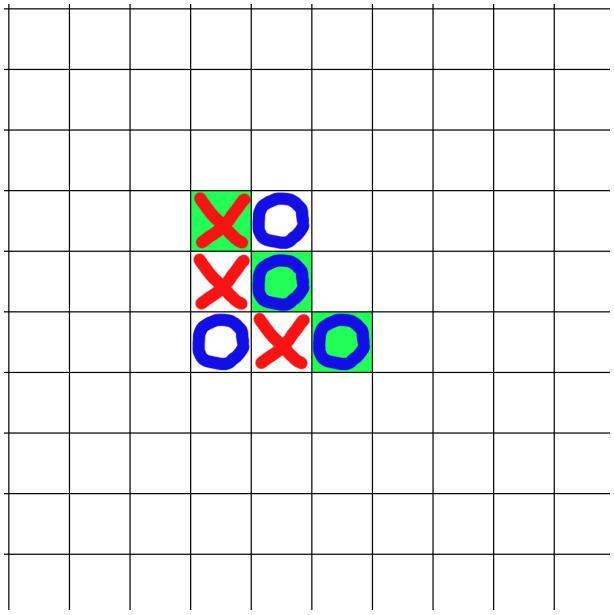
Cách xác định: Sử dụng một ma trận khác lưu các giá trị số với giá trị ban đầu là 0. Với mỗi ví trị được đánh, 5x5 các ô xung quanh (tính cả tâm) được tăng giá trị lên 1. Ban đầu chỉ có vị trí trung tâm là được khởi tạo bằng 1. Tất cả những ô mà có số khác 0 và không phải là vị trí đã được đánh, thì nó là vị trí cho phép được đánh ở lượt chơi kế tiếp.

Hình 4: Xác định các nước đi cho phép

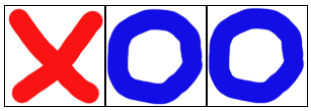


2.2 Hàm tính giá trị heuristic

Từ ma trận các nước đã đi, ta trích ra tất cả những đường (thẳng, ngang hoặc chéo). Ví dụ đường chéo:

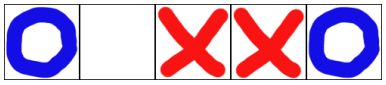


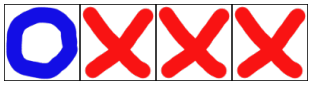
Sẽ được trích thành một vector:



Từ các đường đã trích ra, ta sẽ tính điểm số trên các đường này rồi cộng tổng các điểm sổ lại. Các đường sau đó được trích ra thành các đoạn để có thể dễ dàng tính trọng số. Việc trích thành các đoạn ra sao cũng phụ thuộc vào người chơi (quân X hay O) đang được xét.
Ví dụ đường:

Sẽ được trích thành 2 đoạn (nếu được xét theo X):





Ta sẽ tính điểm dựa trên từng đoạn, cộng tổng sẽ được điểm của một đường. Từ đó ta tính được giá trị heuristic của một trạng thái trong game.