Nội dung chính

Game playing

- Game playing problem
- The minimax algorithm
- Resource limitations
  - Heuristics
  - alpha-beta pruning
Các loại trò chơi

- **Some examples:**
  - Chess
  - Tic-tac-toe
  - …

- **Accessible environments:** các đối thủ hoàn toàn biết thông tin đầy đủ hay không? chơi cờ (đầy đủ), chơi tú lơ khơ (không đầy đủ)

- **Search:** Choì cờ là tìm kiếm trong không gian các trạng thái của trò chơi (tim và lựa chọn đầy các cách đi đến cuối cùng là thắng (hoặc không thua hoặc “tốt nhất”) cho dù đối thủ đi theo cách nào).
Searching for the next move

- **Complexity:** many games have a huge search space
  
  **Chess:** \( b = 35, m=100 \Rightarrow \text{nodes} = 35^{100} \)
  
  if each node takes about 1 ns to explore then each move will take about \(10^{50}\) millennia to calculate.

- **Resource (e.g., time, memory) limit:** optimal solution not feasible/possible, thus must approximate

1. **Pruning:** makes the search more efficient by discarding portions of the search tree that cannot improve quality result.

2. **Evaluation functions:** heuristics to evaluate utility of a state without exhaustive search.
Two-player games

A game formulated as a search problem:

- Initial state: ?
- Operators: ?
- Terminal state: ?
- Utility function: ?
Two-player games

A game formulated as a search problem:

- **Initial state:** board position and turn
- **Operators:** definition of legal moves
- **Terminal state:** conditions for when game is over
- **Utility function:** a numeric value that describes the outcome of the game. E.g., -1, 0, 1 for loss, draw, win. (AKA **payoff function**)
Example: Tic-Tac-Toe
The minimax algorithm

- Perfect play for deterministic environments with perfect information
- **Basic idea:** choose move with highest minimax value
  = best achievable payoff against best play
- **Algorithm:**
  1. Generate game tree completely
  2. Determine utility of each terminal state
  3. Propagate the utility values upward in the tree by applying MIN and MAX operators on the nodes in the current level
  4. At the root node use minimax decision to select the move with the max (of the min) utility value

- Steps 2 and 3 in the algorithm assume that the opponent will play perfectly.
Generate Game Tree
Generate Game Tree
Generate Game Tree
Generate Game Tree

1 ply

1 move
A subtree

- win
- lose
- draw
What is a good move?
Minimax

- Minimize opponent’s chance
- Maximize your chance

CS 561, Sessions 8-9
Minimax

• Minimize opponent’s chance
• Maximize your chance
Minimax

- Minimize opponent’s chance
- Maximize your chance
Minimax

- Minimize opponent’s chance
- Maximize your chance
minimax = maximum of the minimum

MAX

MIN

1st ply

2nd ply
Minimax: Recursive implementation

function MINIMAX-DECISION(game) returns an operator
    for each op in OPERATORS[game] do
        Value[op] ← MINIMAX-VALUE(APPLY(op, game), game)
    end
    return the op with the highest Value[op]

function MINIMAX-VALUE(state, game) returns a utility value
    if TERMINAL-TEST(game)(state) then
        return UTILITY(game)(state)
    else if MAX is to move in state then
        return the highest MINIMAX-VALUE of SUCCESSORS(state)
    else
        return the lowest MINIMAX-VALUE of SUCCESSORS(state)

Complete: ?  Time complexity: ?
Optimal: ?    Space complexity: ?
Minimax: Recursive implementation

function MINIMAX-DECISION(game) returns an operator
    for each op in OPERATORS[game] do
        Value[op] ← MINIMAX-VALUE(APPLY(op, game), game)
    end
    return the op with the highest Value[op]

function MINIMAX-VALUE(state, game) returns a utility value
    if TERMINAL-TEST[game](state) then
        return UTILITY[game](state)
    else if MAX is to move in state then
        return the highest MINIMAX-VALUE of Successors(state)
    else
        return the lowest MINIMAX-VALUE of Successors(state)

Complete: Yes, for finite state-space
Optimal: Yes

Time complexity: $O(b^m)$
Space complexity: $O(bm)$ (= DFS Does not keep all nodes in memory.)
The minimax algorithm: PROBLEM

- **Perfect play for deterministic environments with perfect information**
- **Basic idea:** choose move with highest minimax value
  = best achievable payoff against best play
- **Algorithm:**
  1. Generate game tree completely
  2. Determine utility of each terminal state
  3. Propagate the utility values upward in the tree by applying MIN and MAX operators on the nodes in the current level
  4. At the root node use **minimax decision** to select the move with the max (of the min) utility value
- **Solutions:** (1) Heuristics, (2) alpha-beta pruning
1. Move evaluation without complete search

- Complete search is too complex and impractical

- **Evaluation function**: evaluates value of state using heuristics and cuts off search

- **New MINI MAX**:
  - **CUTOFF-TEST**: cutoff test to replace the termination condition (e.g., deadline, depth-limit, etc.)
  - **EVAL**: evaluation function to replace utility function (e.g., number of chess pieces taken)
Evaluation functions

- **Weighted linear evaluation function**: to combine $n$ heuristics

$$f = w_1f_1 + w_2f_2 + \ldots + w_nf_n$$

E.g., $w$’s could be the values of pieces (1 for prawn, 3 for bishop etc.)

$f$’s could be the number of type of pieces on the board
Minimax with cutoff: viable algorithm?

\textbf{MinimaxCutoff} is identical to \textbf{MinimaxValue} except
1. \texttt{Terminal?} is replaced by \texttt{Cutoff?}
2. \texttt{Utility} is replaced by \texttt{Eval}

Does it work in practice?

\[ b^m = 10^6, \quad b = 35 \quad \Rightarrow \quad m = 4 \]

4-ply lookahead is a hopeless chess player!

4-ply \approx \text{human novice}
8-ply \approx \text{typical PC, human master}
12-ply \approx \text{Deep Blue, Kasparov}

Assume we have 100 seconds, evaluate \(10^4\) nodes/s; can evaluate \(10^6\) nodes/move
2. $\alpha-\beta$ pruning: search cutoff

- **Pruning**: eliminating a branch of the search tree from consideration without exhaustive examination of each node.

- **$\alpha-\beta$ pruning**: the basic idea is to prune portions of the search tree that cannot improve the utility value of the max or min node, by just considering the values of nodes seen so far.

- Does it work? Yes, in roughly cuts the branching factor from $b$ to $\sqrt{b}$ resulting in double as far look-ahead than pure minimax.
$\alpha$-$\beta$ pruning: example

MAX

MIN

$\geq 6$
α-β pruning: example

MAX

MIN

≥ 6

≤ 2

6
12
8

2

×

×
\[ \alpha-\beta \text{ pruning: example} \]

MAX

MIN

\[
\begin{array}{c}
\geq 6 \\
\leq 2 \\
\leq 5 \\
\end{array}
\]

\[
\begin{array}{ccc}
6 & 12 & 8 \\
6 & 2 & 5 \\
\end{array}
\]
\( \alpha-\beta \) pruning: example

\[ \geq 6 \]

Selected move

\[ \leq 2 \]

\[ \leq 5 \]
**α-β pruning: general principle**

If $\alpha > v$ then MAX will choose $m$ so prune tree under $n$

*Similar for $\beta$ for MIN*
The $\alpha$-$\beta$ algorithm

```c
Int MaxValue(Board b, int depth, int alpha, int beta) {
    /* alpha: the best score for MAX along the path to b
       beta: the best score for MIN along the path to b */
    if ((GameOver(b) or depth>MaxDepth)
        return h(b)
    int max = -infinity
    for each legal move m in board b {
        copy b to c
        make move m in board c
        int x = MinValue(c, depth+1, alpha, beta)
        if (x>max) max = x
        if (x>alpha) alpha = x
        if (alpha>=beta) return alpha
    }
    return max
}

int MinValue(Board b, int depth, int alpha, int beta) {
    /* alpha: the best score for MAX along the path to b
       beta: the best score for MIN along the path to b */
    if ((GameOver(b) or depth>MaxDepth)
        return h(b)
    int min = infinity
    for each legal move m in board b {
        copy b to c
        make move m in board c
        int x = MaxValue(c, depth+1, alpha, beta)
        if (x<min) min = x
        if (x<beta) beta = x
        if (alpha>=beta) return beta
    }
    return min
}
```
The $\alpha$-$\beta$ algorithm

Note: These are both local variables. At the start of the algorithm, we initialize them to \( \alpha = -\infty \) and \( \beta = +\infty \).

```c
Int MaxValue(Board b, int depth, int alpha, int beta) {
    /* alpha: the best score for MAX along the path to b
    beta: the best score for MIN along the path to b */
    if ((GameOver(b) or depth>MaxDepth)
        return h(b)
    int max = -infinity
    for each legal move m in board b {
        copy b to c
        make move m in board c
        int x = MinValue(c, depth+1, alpha, beta)
        if (x>max) max = x
        if (x>alpha) alpha = x
        if (alpha>=beta) return alpha
    }
    return max
}

int MinValue(Board b, int depth, int alpha, int beta) {
    /* alpha: the best score for MAX along the path to b
    beta: the best score for MIN along the path to b */
    if ((GameOver(b) or depth>MaxDepth)
        return h(b)
    int min = infinity
    for each legal move m in board b {
        copy b to c
        make move m in board c
        int x = MaxValue(c, depth+1, alpha, beta)
        if (x<min) min = x
        if (x<beta) beta = x
        if (alpha>=beta) return beta
    }
    return min
}
```
More on the $\alpha$-$\beta$ algorithm

In Min-Value:

```c
int min = infinity
for each legal move m in board b {
    copy b to c
    make move m in board c
    int x = MaxValue(c, depth+1, alpha, beta)
    if (x>min) min = x
    if (x<beta) beta = x
    if (alpha>=beta) return beta
}
return min
```

Max-Value loops over these

Min-Value loops over these

\[ \alpha = -\infty \quad \beta = +\infty \]

\[ \alpha = -\infty \quad \beta = 5 \]

\[ \alpha = -\infty \quad \beta = 5 \]

\[ \alpha = -\infty \quad \beta = 5 \]

\[ \alpha = -\infty \quad \beta = 5 \]

\[ \alpha = -\infty \quad \beta = 5 \]

\[ \alpha = -\infty \quad \beta = 5 \]
More on the $\alpha$$\beta$ algorithm

In Max-Value:

int max = -infinity
for each legal move m in board b {
    copy b to c
    make move m in board c
    int x = MinValue(c, depth+1, alpha, beta)
    if (x>max) max = x
    if (x>alpha) alpha = x
    if (alpha>=beta) return alpha
}
return max

Max-Value loops over these

MAX

MIN

MAX

5
6
2
8
7

$\alpha = -\infty$
$\beta = +\infty$

$\alpha = -\infty$
$\beta = +\infty$

$\alpha = 5$
$\beta = +\infty$

...
More on the $\alpha$-$\beta$ algorithm

In Min-Value:

```
int min = infinity
for each legal move m in board b {
    copy b to c
    make move m in board c
    int x = MaxValue(c, depth+1, alpha, beta)
    if (x>min) min = x
    if (x<beta) beta = x
    if (alpha>=beta) return beta
}
return min
```

Min-Value loops over these

---

MAX

MIN

End loop and return 5
More on the $\alpha$-$\beta$ algorithm

In Max-Value:

```
int max = -infinity
for each legal move m in board b {
    copy b to c
    make move m in board c
    int x = MinValue(c, depth+1, alpha, beta)
    if (x>max) max = x
    if (x>alpha) alpha = x
    if (alpha>=beta) return alpha
}
return max
```

Max-Value loops over these

```
MAX

MIN

MAX

5 10 6 2 8 7

\alpha = -\infty
\beta = +\infty
\alpha = -\infty
\beta = +\infty
\alpha = -\infty
\beta = +\infty
\alpha = 5
\beta = +\infty
\alpha = 5
\beta = +\infty
\alpha = 5
\beta = +\infty
```

End loop and return 5
Example

MiniMax
+ Alpha–Beta
## Solution

<table>
<thead>
<tr>
<th>NODE</th>
<th>TYPE</th>
<th>ALPHA</th>
<th>BETA</th>
<th>SCORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Max</td>
<td>-I</td>
<td>+I</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Min</td>
<td>-I</td>
<td>+I</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>Max</td>
<td>-I</td>
<td>+I</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>Min</td>
<td>-I</td>
<td>+I</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>Max</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>D</td>
<td>Min</td>
<td>-I</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>Max</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>D</td>
<td>Min</td>
<td>-I</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>Max</td>
<td>10</td>
<td>+I</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>Min</td>
<td>10</td>
<td>+I</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>Max</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>G</td>
<td>Min</td>
<td>10</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>C</td>
<td>Max</td>
<td>10</td>
<td>+I</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>Min</td>
<td>-I</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>Max</td>
<td>-I</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>Min</td>
<td>-I</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>Max</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>K</td>
<td>Min</td>
<td>-I</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NODE</th>
<th>TYPE</th>
<th>ALPHA</th>
<th>BETA</th>
<th>SCORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>Max</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>Min</td>
<td>-I</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>A</td>
<td>Max</td>
<td>10</td>
<td>+I</td>
<td>10</td>
</tr>
<tr>
<td>Q</td>
<td>Min</td>
<td>10</td>
<td>+I</td>
<td>10</td>
</tr>
<tr>
<td>R</td>
<td>Max</td>
<td>10</td>
<td>+I</td>
<td>10</td>
</tr>
<tr>
<td>S</td>
<td>Min</td>
<td>10</td>
<td>+I</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>Max</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>S</td>
<td>Min</td>
<td>10</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>R</td>
<td>Max</td>
<td>10</td>
<td>+I</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>Min</td>
<td>10</td>
<td>+I</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>Max</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>V</td>
<td>Min</td>
<td>10</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>R</td>
<td>Max</td>
<td>10</td>
<td>+I</td>
<td>10</td>
</tr>
<tr>
<td>Q</td>
<td>Min</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>A</td>
<td>Max</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

CS 561, Sessions 8-9
State-of-the-art for deterministic games

Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.


Othello: human champions refuse to compete against computers, who are too good.

Go: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.
Summary

- **Game playing**
  - Game playing problem
  - The minimax algorithm
  - Resource limitations
    - Heuristics
    - alpha-beta pruning
Consider the following game tree in which the evaluation function values are shown below each leaf node. Assume that the root node corresponds to the maximizing player. Assume the search always visits children left-to-right.

(a) Compute the backed-up values computed by the minimax algorithm. Show your answer by writing values at the appropriate nodes in the above tree.

(b) Compute the backed-up values computed by the alpha-beta algorithm. What nodes will not be examined by the alpha-beta pruning algorithm?

(c) What move should Max choose once the values have been backed-up all the way?
Exercise 2: trò chơi bốc bi

Có 2 dòng bi có số bi lần lượt là m và n viên bi. Hai người chơi, mỗi người đến lượt mình để chỉ được phép bốc 1 - k viên bi trên 1 dòng.

(a) Xây dựng không gian tìm kiếm với m=10, n=8, k=5
(b) Cài đặt giải thuật minimax cho máy (người chơi thứ nhất là máy và người chơi thứ hai là người sử dụng chương trình)?
(c) Hãy tìm chiến lược chơi tốt nhất cho máy để khả năng thắng của máy là lớn nhất?
(d) Qua bài tập này, “Liệu có giải thuật để đảm bảo khi chơi là chắc thắng?”