CS 561: Artificial Intelligence

Instructor: Sofus A. Macskassy, macskass@usc.edu
TAs: Nadeesha Ranashinghe (nadeeshr@usc.edu)
     William Yeoh (wyeoh@usc.edu)
     Harris Chiu (chiciu@usc.edu)

Lectures: MW 5:00-6:20pm, OHE 122 / DEN
Office hours: By appointment
Class page: http://www.rcf.usc.edu/~macskass/CS561-Spring2010/

This class will use http://www.uscden.net/ and class webpage
- Up to date information
- Lecture notes
- Relevant dates, links, etc.

Course material:
This time: Outline
(Adversarial Search – AIMA Ch. 6]

Game playing
- Perfect play
  - The minimax algorithm
  - alpha-beta pruning
- Resource limitations
- Elements of chance
- Imperfect information
What kind of games?

- **Abstraction**: To describe a game we must capture every relevant aspect of the game. Such as:
  - Chess
  - Tic-tac-toe
  - ...
- **Accessible environments**: Such games are characterized by perfect information
- **Search**: Game-playing then consists of a search through possible game positions
- **Unpredictable opponent**: Introduces uncertainty thus game-playing must deal with contingency problems
Searching for the next move

- **Complexity**: many games have a huge search space
  - **Chess**: \( b = 35, m=100 \Rightarrow nodes = 35^{100} \)
    - 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)
Games vs. search problems

- "Unpredictable" opponent → solution is a strategy specifying a move for every possible opponent reply

- Time limits → unlikely to find goal, must approximate

- Plan of attack:
  - Computer considers possible lines of play (Babbage, 1846)
  - Algorithm for perfect play (Zermelo, 1912; Von Neumann, 1944)
  - Finite horizon, approximate evaluation (Zuse, 1945; Wiener, 1948; Shannon, 1950)
  - First chess program (Turing, 1951)
  - Machine learning to improve evaluation accuracy (Samuel, 1952-57)
  - Pruning to allow deeper search (McCarthy, 1956)
Example: Tic-Tac-Toe

MAX (X)

MIN (O)

MAX (X)

MIN (O)

TERMINAL

Utility

-1 0 +1
## Type of games

<table>
<thead>
<tr>
<th></th>
<th>Deterministic</th>
<th>Chance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perfect information</td>
<td>chess, checkers, go, othello</td>
<td>backgammon monopoly</td>
</tr>
<tr>
<td>Imperfect information</td>
<td>battleships, blind tictactoe</td>
<td>bridge, poker, scrabble nuclear war</td>
</tr>
</tbody>
</table>
Type of games

- **Perfect Information**
  - Deterministic: chess, checkers, go, othello
  - Imperfect Information: battleships, blind tictactoe

- **Chance**
  - Backgammon
  - Monopoly
  - Bridge, poker, scrabble, nuclear war
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

X

X

X

X
Generate Game Tree

CS561 - Lecture 7 - Macskassy - Spring 2010
Generate Game Tree

1 ply

1 move
A subtree
What is a good move?
Minimax

- Minimize opponent’s chance
- Maximize your chance
Minimax: Recursive implementation

function **MINIMAX-DECISION**\((state)\) returns an action

inputs: \(state\), current state in game

return the \(a\) in **ACTIONS**\((state)\) maximizing **MIN-VALUE**\(\left(RESULT\left(a, state\right)\right)\)

function **MAX-VALUE**\((state)\) returns a utility value

if **TERMINAL-TEST**\((state)\) then return **UTILITY**\((state)\)

\(v \leftarrow -\infty\)

for \(a, s\) in **SUCCESSORS**\((state)\) do \(v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))\)

return \(v\)

function **MIN-VALUE**\((state)\) returns a utility value

if **TERMINAL-TEST**\((state)\) then return **UTILITY**\((state)\)

\(v \leftarrow \infty\)

for \(a, s\) in **SUCCESSORS**\((state)\) do \(v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s))\)

return \(v\)
Minimax: Recursive implementation

Complete??

Optimal??

Time complexity??

Space complexity??
α-β pruning: example

MAX

MIN

≥ 3

3 12 8
α-β pruning: example

MAX

MIN

3 12 8

3 ≥ 3

2 ≤ 2

x  x
α-β pruning: example

MAX

MIN

3 12 8

3 ≥ 3

2 ≤ 2

14 ≤ 14

-
α-β pruning: example

MAX

MIN

3 12 8

3

2

14 5

≥ 3

≤ 2

≤ 4 ≤ 5

CS561 - Lecture 7 - Macskassy - Spring 2010
\(\alpha-\beta\) pruning: example

\[
\begin{aligned}
\text{MAX} & \quad \geq 3 \\
\text{MIN} & \quad \leq 2
\end{aligned}
\]

```
3
12
8
2
```

```
14
5
2
```

```
\times
\times
```

CS561 - Lecture 7 - Macskassy - Spring 2010
\( \alpha - \beta \) pruning: example

MAX

MIN

Selected move
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

**function** ALPHA-BETA-DECISION(*state*) **returns** an action

return the $a$ in ACTIONS(*state*) maximizing MIN-VALUE(Result(*a*, *state*))

**function** MAX-VALUE(*state*, $\alpha$, $\beta$) **returns** a utility value

inputs: *state*, current state in game

$\alpha$, the value of the best alternative for MAX along the path to *state*

$\beta$, the value of the best alternative for MIN along the path to *state*

if TERMINAL-TEST(*state*) then return UTILITY(*state*)

$v \leftarrow -\infty$

for $a$, $s$ in SUCCESSORS(*state*) do

$v \leftarrow \max(v, \text{MIN-VALUE}(s, \alpha, \beta))$

if $v \geq \beta$ then return $v$

$\alpha \leftarrow \max(\alpha, v)$

return $v$

**function** MIN-VALUE(*state*, $\alpha$, $\beta$) **returns** a utility value

same as MAX-VALUE but with roles of $\alpha$, $\beta$ reversed
Properties of $\alpha-\beta$

Pruning does not affect final result.

Good move ordering improves effectiveness of pruning.

With “perfect ordering,” time complexity $= O(b^{m/2})$

$\Rightarrow$ doubles solvable depth.

A simple example of the value of reasoning about which computations are relevant (a form of metareasoning).

Unfortunately, $35^{50}$ is still impossible!
Resource limits

- Standard approach:
  - Use `CUTOFF-TEST` instead of `TERMINAL-TEST`
    - e.g., depth limit (perhaps add quiescence search)
  - Use `EVAL` instead of `UTILITY`
    - i.e., evaluation function that estimates desirability of position

- Suppose we have 100 seconds, and can explore $10^4$ nodes/second
  - $10^6$ nodes per move $\approx 35^{8/2}$
  - $\alpha$-$\beta$ reaches depth 8 $\Rightarrow$ pretty good chess program
Evaluation functions

For chess, typically linear weighted sum of features

\[ \text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) \]

- e.g., \( w_1 = 9 \) with
  - \( f_1(s) = \) (num white queens) - (num black queens), etc.
Digression: Exact values don't matter

- Behaviour is preserved under any **monotonic** transformation of \( \text{EVAL} \)

- Only the order matters:
  - Payoff in deterministic games acts as an **ordinal utility** function
Deterministic games in practice

- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database dening 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.
Nondeterministic games: backgammon
Nondeterministic games

- In nondeterministic games, chance introduced by dice, card-shuffling
- Simplified example with coin-flipping:
Algorithm for nondeterministic games

- **EXPECTIMINIMAX** gives perfect play
  - Just like **MINIMAX**, except we must also handle chance nodes:

```plaintext
... if state is a Max node then
    return the highest EXPECTIMINIMAX-Value of SUCCESSORS(state)
if state is a Min node then
    return the lowest EXPECTIMINIMAX-Value of SUCCESSORS(state)
if state is a chance node then
    return average of EXPECTIMINIMAX-Value of SUCCESSORS(state)
...```
Nondeterministic games in practice

Dice rolls increase \( b \): 21 possible rolls with 2 dice

Backgammon \( \approx 20 \) legal moves (can be 6,000 with 1-1 roll)
- depth 4 = 20 \((21 \times 20)^3 \approx 1:2 \times 10^9\)

As depth increases, probability of reaching a given node shrinks
- value of lookahead is diminished

\( \alpha-\beta \) pruning is much less effective

TDGAMMON uses depth-2 search + very good EVAL
- \( \approx \) world-champion level
Nondeterministic games: the element of chance

**expectimax** and **expectimin**, expected values over all possible outcomes
Nondeterministic games: the element of chance

Expectimax

MAX

Expectimin

MIN

\[ 4 = 0.5 \times 3 + 0.5 \times 5 \]
Order-preserving transformation do not necessarily behave the same!

Behaviour is preserved only by positive linear transformation of $EVAL$
Hence $EVAL$ should be proportional to the expected payoff
Games of imperfect information

E.g., card games, where opponent's initial cards are unknown

Typically we can calculate a probability for each possible deal

Seems just like having one big dice roll at the beginning of the game
Idea: compute the minimax value of each action in each deal, then choose the action with highest expected value over all deals

Special case: if an action is optimal for all deals, it's optimal.

GIB, current best bridge program, approximates this idea by
1) generating 100 deals consistent with bidding information
2) picking the action that wins most tricks on average
Commonsense Example

- Road A leads to a small heap of gold pieces
- Road B leads to a fork:
  - take the left fork and you'll find a mound of jewels;
  - take the right fork and you'll be run over by a bus.

- Road A leads to a small heap of gold pieces
- Road B leads to a fork:
  - take the left fork and you'll be run over by a bus;
  - take the right fork and you'll find a mound of jewels.

- Road A leads to a small heap of gold pieces
- Road B leads to a fork:
  - guess correctly and you'll find a mound of jewels;
  - guess incorrectly and you'll be run over by a bus.
Proper analysis

• Intuition that the value of an action is the average of its values in all actual states is **WRONG**

• With partial observability, value of an action depends on the **information state** or **belief state** the agent is in

• Can generate and search a tree of information states

• Leads to rational behaviors such as
  ◦ Acting to obtain information
  ◦ Signalling to one's partner
  ◦ Acting randomly to minimize information disclosure
Games are fun to work on! (and dangerous)

They illustrate several important points about AI

- perfection is unattainable ➔ must approximate
- good idea to think about what to think about
- uncertainty constrains the assignment of values to states
- optimal decisions depend on information state, not real state

Games are to AI as grand prix racing is to automobile design
Exercise: Game Playing

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?

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?