Games

Outline

I. Game as adversarial search

II. The minimax algorithm

III. Alpha-beta pruning

* Figures/images are from the textbook site (or by the instructor). Otherwise, the source is specifically cited unless citation would make little sense due to the triviality of generating such an image.
I. Games

Competitive environments: goals are in conflict.
Adversarial search problems (games)
I. Games

Competitive environments: goals are in conflict.
Adversarial search problems (games)
Why Study Games?

- Multiagent environment
  - Aggregate of a large number of agents for predictions (e.g., price rise).
  - Nondeterminism made by adversarial agents.
  - Introduction of new modeling techniques.
Why Study Games?

- Multiagent environment
  
  ◆ Aggregate of a large number of agents for predictions (e.g., price rise).
  
  ◆ Nondeterminism made by adversarial agents.
  
  ◆ Introduction of new modeling techniques.

- Mathematical game theory – an important branch of economics.
Why Study Games?

- Multiagent environment
  - Aggregate of a large number of agents for predictions (e.g., price rise).
  - Nondeterminism made by adversarial agents.
  - Introduction of new modeling techniques.

- Mathematical game theory – an important branch of economics.

  Nash equilibrium for a non-cooperative game:
  Each player has chosen a strategy and no player can increase own expected payoff by changing their strategy while the other players keep theirs unchanged.

John Nash (Princeton)
Nobel Prize in Economics (1994)

Why Study Games?

- Multiagent environment
  - Aggregate of a large number of agents for predictions (e.g., price rise).
  - Nondeterminism made by adversarial agents.
  - Introduction of new modeling techniques.

- Mathematical game theory – an important branch of economics.
  
  Nash equilibrium for a non-cooperative game:
  
  Each player has chosen a strategy and no player can increase own expected payoff by changing their strategy while the other players keep theirs unchanged.

- Appealing subject for study in AI.
  - Fun and entertaining.
  - Hard – engaging the intellectual faculties of humans.
  - Abstract nature – easy to represent with small number of actions.

History of Computer Games


1956  John McCarthy conceives alpha-beta search.

1982  BELLE becomes the first chess program to achieve master status.


1997  Deep Blue (IBM) defeats world chess champion Garry Kasparov.

2017  AlphaGo (Alphabet) defeats world’s no. 1 Go player Ke Jie.

- Visual pattern recognition
- Reinforcement learning
- Neural networks
- Monte Carlo tree search

2018  AlphaZero (Alphabet) defeats top programs in Go, chess, shogi.

2019  Pluribus (CMU) defeats top-ranked players in Texas hold’em games with six players.

Types of Games

- Games with deterministic, perfect information (e.g., chess, go, checkers)
- Stochastic games (e.g., backgammon)
- Partially observable games (e.g., bridge, poker)
II. Two-Player Game

- Perfect information – fully observable.
- Zero sum – what is good for one player is just as bad for the other.

\[
\text{move} \Leftrightarrow \text{action} \\
\text{position} \Leftrightarrow \text{state}
\]

MAX and MIN: two players.
Formal Definition of a Game

- $s_0$: initial state – game setup.
Formal Definition of a Game

- $s_0$: initial state – game setup.

At a state $s$:

- $\text{TO-MOVE}(s)$: the player to move in the state.
Formal Definition of a Game

• $s_0$: initial state – game setup.

At a state $s$:

• $\text{TO-MOVE}(s)$: the player to move in the state.

• $\text{ACTIONS}(s)$: the set of legal moves in the state.
Formal Definition of a Game

- $s_0$: initial state – game setup.

At a state $s$:

- $\text{TO-MOVE}(s)$: the player to move in the state.
- $\text{ACTIONS}(s)$: the set of legal moves in the state.
- $\text{RESULT}(s, a)$: the transition model defining the next state from taking action $a$. 
Formal Definition of a Game

• $s_0$: initial state – game setup.

At a state $s$:

• $\text{TO-MOVE}(s)$: the player to move in the state.

• $\text{ACTIONS}(s)$: the set of legal moves in the state.

• $\text{RESULT}(s, a)$: the transition model defining the next state from taking action $a$.

• $\text{IS-TERMINAL}(s)$: to test if the game is over, i.e., if $s$ is a terminal state.
Formal Definition of a Game

- \( s_0 \): initial state – game setup.

At a state \( s \):

- \( \text{TO-MOVE}(s) \): the player to move in the state.
- \( \text{ACTIONS}(s) \): the set of legal moves in the state.
- \( \text{RESULT}(s, a) \): the transition model defining the next state from taking action \( a \).
- \( \text{IS-TERMINAL}(s) \): to test if the game is over, i.e., if \( s \) is a terminal state.
- \( \text{UTILITY}(s, p) \): a utility function to return a value to the player \( p \) if the game ends in terminal state \( s \).
Formal Definition of a Game

- \( s_0 \): initial state – game setup.

At a state \( s \):
- TO-MOVE(\( s \)): the player to move in the state.
- ACTIONS(\( s \)): the set of legal moves in the state.
- RESULT(\( s, a \)): the transition model defining the next state from taking action \( a \).
- IS-TERMINAL(\( s \)): to test if the game is over, i.e., if \( s \) is a terminal state.
- UTILITY (\( s, p \)): a utility function to return a value to the player \( p \) if the game ends in terminal state \( s \).

  e.g., in chess, win (1), loss (0), draw (1/2)

  Total payoff for all players is constant (zero-sum game):
  \[
  1 + 0 = 0 + 1 = \frac{1}{2} + \frac{1}{2} = 1
  \]
State Space Graph (Tic-Tac-Toe)

Vertices $\leftrightarrow$ states and edges $\leftrightarrow$ moves

MAX (x)
State Space Graph (Tic-Tac-Toe)

Vertices $\leftrightarrow$ states and edges $\leftrightarrow$ moves

MAX (x)

MIN (o)
State Space Graph (Tic-Tac-Toe)

Vertices $\leftrightarrow$ states and edges $\leftrightarrow$ moves
State Space Graph (Tic-Tac-Toe)

Vertices ↔ states and edges ↔ moves

MAX (x)

MIN (o)

MAX (x)

MIN (o)

...
State Space Graph (Tic-Tac-Toe)

Vertices ↔ states and edges ↔ moves

MAX (x)

MIN (o)

MAX (x)

MIN (o)

TERMINAL

Utility

-1 0 +1
State Space Graph (Tic-Tac-Toe)

Vertices $\leftrightarrow$ states and edges $\leftrightarrow$ moves

$9! = 362,880$ terminal nodes
(5,478 distinct states)

$10^{40}$ for chess!
Two-Ply Game Tree

_Ply_: one move by a player
Optimal Strategy

Work out the minimax value of every state $s$ in the tree,

$$\text{MINIMAX}(s)$$

assuming both players play optimally:

- MAX moves to a state of maximum value at its turn;
- MIN moves to a state of minimum value at its turn.
Work out the minimax value of every state $s$ in the tree,

$$\text{MINIMAX}(s)$$

assuming both players play optimally:

- MAX moves to a state of maximum value at its turn;
- MIN moves to a state of minimum value at its turn.

$$\text{MINIMAX}(s) =$$
Optimal Strategy

Work out the minimax value of every state $s$ in the tree,

$$ \text{MINIMAX}(s) $$

assuming both players play optimally:

- MAX moves to a state of maximum value at its turn;
- MIN moves to a state of minimum value at its turn.

$$ \text{MINIMAX}(s) = \begin{cases} 
\text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\
\max_{a \in \text{Actions}(s)} \text{MINIMAX} \left( \text{RESULT}(s, a) \right) & \text{if TO-MOVE}(s) = \text{MAX} \\
\min_{a \in \text{Actions}(s)} \text{MINIMAX} \left( \text{RESULT}(s, a) \right) & \text{if TO-MOVE}(s) = \text{MIN} 
\end{cases} $$
Minimax Value at Min Nodes

MIN: choose a move to a MAX node with the lowest value.
Minimax Value at Min Nodes

MIN: choose a move to a MAX node with the lowest value.
Minimax Value at Min Nodes

**MIN**: choose a move to a MAX node with the lowest value.
Minimax Value at Min Nodes

MIN: choose a move to a MAX node with the lowest value.
Minimax Value at Min Nodes

MIN: choose a move to a MAX node with the lowest value.
Minimax Value at a Max Node

\[ \text{MAX: choose a move to a MIN node with the highest value.} \]
Solution of the Game

Best move for MAX: $a_1$
Best move for MIN in response: $b_1$
The Minimax Search Algorithm

function MINIMAX-SEARCH(game, state) returns an action
    player ← game.TO-MOVE(state)
    value, move ← MAX-VALUE(game, state)
    return move

function MAX-VALUE(game, state) returns a (utility, move) pair
    if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
    \( v \leftarrow -\infty \)
    for each \( a \) in game.ACTIONS(state) do
        \( v_2, a_2 \leftarrow \text{MIN-VALUE}(game, game.RESULT(state, a)) \)
        if \( v_2 > v \) then
            \( v, move \leftarrow v_2, a \)
    return \( v, move \)

function MIN-VALUE(game, state) returns a (utility, move) pair
    if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
    \( v \leftarrow +\infty \)
    for each \( a \) in game.ACTIONS(state) do
        \( v_2, a_2 \leftarrow \text{MAX-VALUE}(game, game.RESULT(state, a)) \)
        if \( v_2 < v \) then
            \( v, move \leftarrow v_2, a \)
    return \( v, move \)
Algorithm Execution

Depth-first search with backed-up value on return from a node.
Algorithm Execution

Depth-first search with backed-up value on return from a node.
Algorithm Execution

Depth-first search with backed-up value on return from a node.
Algorithm Execution

Depth-first search with backed-up value on return from a node.
Algorithm Execution

Depth-first search with backed-up value on return from a node.
Algorithm Execution

Depth-first search with backed-up value on return from a node.
Algorithm Execution

Depth-first search with backed-up value on return from a node.
Algorithm Execution

Depth-first search with backed-up value on return from a node.
Algorithm Execution

Depth-first search with backed-up value on return from a node.
Algorithm Execution

Depth-first search with backed-up value on return from a node.
Algorithm Execution

Depth-first search with backed-up value on return from a node.
Algorithm Execution

Depth-first search with backed-up value on return from a node.
Algorithm Execution

Depth-first search with backed-up value on return from a node.
Summary on Minimax

- Complete if the game tree is finite.
Summary on Minimax

- Complete if the game tree is finite.
- Optimal against an optimal opponent.
Summary on Minimax

♦ Complete if the game tree is finite.

♦ Optimal against an optimal opponent.

If MIN does not play optimally,
1) MAX will play at least as well as an optimal player;
2) but there may be a better strategy against the suboptimal MIN.
Summary on Minimax

- Complete if the game tree is finite.
- Optimal against an optimal opponent.
  
  If MIN does not play optimally,
  1) MAX will play at least as well as an optimal player;
  2) but there may be a better strategy against the suboptimal MIN.

- Complexities:

  Time: $O(b^m)$
Summary on Minimax

- Complete if the game tree is finite.
- Optimal against an optimal opponent.

If MIN does not play optimally,
1) MAX will play at least as well as an optimal player;
2) but there may be a better strategy against the suboptimal MIN.

- Complexities:

  \[ \text{Time: } O(b^m) \]
Summary on Minimax

♦ Complete if the game tree is finite.

♦ Optimal against an optimal opponent.

If MIN does not play optimally,
1) MAX will play at least as well as an optimal player;
2) but there may be a better strategy against the suboptimal MIN.

♦ Complexities:

\[ \text{Time: } O(b^m) \quad \text{Space: } O(bm) \]
Summary on Minimax

💎 Complete if the game tree is finite.

💎 Optimal against an optimal opponent.

If MIN does not play optimally,
1) MAX will play at least as well as an optimal player;
2) but there may be a better strategy against the suboptimal MIN.

💎 Complexities:

\[ \text{Time: } O(b^m) \quad \text{Space: } O(bm) \]

Chess: \( b \approx 35 \) and \( m \approx 100 \) for a reasonable game. Exact optimal solution infeasible!
Multiplayer Games

Extend the minimax algorithm:

- Every node now has a vector of values.

\[ \langle v_A, v_B, v_C \rangle \] for three players \( A, B, C \)

Utility vector
Extend the minimax algorithm:

- Every node now has a vector of values:
  \[ \langle v_A, v_B, v_C \rangle \text{ for three players } A, B, C \]

![Utility vector diagram](image)
Multiplayer Games

Extend the minimax algorithm:

- Every node now has a vector of values.

\[ \langle v_A, v_B, v_C \rangle \text{ for three players } A, B, C \]

Utility vector

```
 to move  
 A       
  \( (1, 2, 6) \)  \( (1, 2, 6) \)  \( (0, 5, 2) \)  \( (5, 4, 5) \)
  B       
  \( (1, 2, 6) \)  \( (1, 2, 6) \)  \( (0, 5, 2) \)  \( (5, 4, 5) \)
  C       
  \( (1, 2, 6) \)  \( (4, 2, 3) \)  \( (7, 4, 1) \)  \( (7, 7, 1) \)  \( (5, 4, 5) \)
  A       
```
Multiplayer Games

Extend the minimax algorithm:

- Every node now has a vector of values.

\[
\langle v_A, v_B, v_C \rangle
\]

for three players \( A, B, C \)

Utility vector

```
to move
A
B
C
A
(1, 2, 6)
(6, 1, 2)
(0, 5, 2)
(5, 4, 5)
(1, 2, 6)
(4, 2, 3)
(6, 1, 2)
(7, 4, 1)
(0, 5, 2)
(7, 7, 1)
(5, 4, 5)
(1, 2, 6)
(6, 1, 2)
(7, 4, 1)
(5, 1, 1)
(0, 5, 2)
(7, 7, 1)
(5, 4, 5)
```
Multiplayer Games

Extend the minimax algorithm:

- Every node now has a vector of values.

\[ \langle v_A, v_B, v_C \rangle \] for three players \( A, B, C \)

Utility vector

Diagram:

- At each node, the utility vector is displayed.
- The move that leads to the maximum utility is indicated by a black arrow.
- The move that leads to the minimum utility is indicated by a green arrow.
- The root node is at the top, with A, B, and C moving in order.
- The leaf nodes represent the final states of the game with their corresponding utility vectors.
Multiplayer Games

Extend the minimax algorithm:

- Every node now has a vector of values.

\[ \langle v_A, v_B, v_C \rangle \] for three players A, B, C

Utility vector

![Game Tree Diagram]

- Node values represent utility vectors.
- Moves proceed until a terminal state is reached.
Multiplayer Games

Extend the minimax algorithm:

- Every node now has a vector of values.

\[ \langle v_A, v_B, v_C \rangle \] for three players A, B, C

Utility vector

```
to move
A
B
C
A
```

\[ (1, 2, 6) \]

\[ (1, 2, 6) \]

\[ (6, 1, 2) \]

\[ (0, 5, 2) \]

\[ (5, 4, 5) \]

\[ (1, 2, 6) \]

\[ (4, 2, 3) \]

\[ (7, 4, 1) \]

\[ (5, 1, 1) \]

\[ (7, 7, 1) \]

\[ (5, 4, 5) \]
Multiplayer Games

Extend the minimax algorithm:

- Every node now has a vector of values.

\[ \langle v_A, v_B, v_C \rangle \] for three players \( A, B, C \)

Utility vector
Multiplayer Games

Extend the minimax algorithm:

- Every node now has a vector of values.

\[ \langle v_A, v_B, v_C \rangle \] for three players \( A, B, C \)

Utility vector

Backed-up value at a node \( n \) = utility vector of the successor state with the highest value for the player choosing at \( n \)
#III. Alpha-Beta Cutoff

- #states is exponential in the depth of the game tree.

- But we can often compute the correct minimax decision by pruning large parts of the tree that do not affect the outcome.
Re-examining the Game Tree

Fig. 5.5 in the textbook *incorrectly executes* the algorithm in Fig. 5.7.
Re-examining the Game Tree

Fig. 5.5 in the textbook *incorrectly executes* the algorithm in Fig. 5.7.

(a)

possible range of the returned value

$\alpha \beta$ 

$(-\infty, \infty)$
Re-examining the Game Tree

Fig. 5.5 in the textbook *incorrectly executes* the algorithm in Fig. 5.7.

possible range of the returned value

\( (-\infty, \infty) \)
Re-examining the Game Tree

Fig. 5.5 in the textbook *incorrectly executes* the algorithm in Fig. 5.7.
Re-examining the Game Tree

Fig. 5.5 in the textbook *incorrectly executes* the algorithm in Fig. 5.7.
Re-examining the Game Tree

Fig. 5.5 in the textbook *incorrectly executes* the algorithm in Fig. 5.7.

(a)

 posible range of the returned value

\(-\infty, \infty\)
Fig. 5.5 in the textbook incorrectly executes the algorithm in Fig. 5.7.
Re-examining the Game Tree

Fig. 5.5 in the textbook *incorrectly executes* the algorithm in Fig. 5.7.

Min node returns a value $\leq 3$

Possible range of the returned value

$(\alpha, \beta)$

$(-\infty, \infty)$

$(-\infty, 3]$
Re-examining the Game Tree

Fig. 5.5 in the textbook *incorrectly executes* the algorithm in Fig. 5.7.

Min node returns a value $\leq 3$

Possible range of the returned value

(a) $(-\infty, \infty)$

(b) $(-\infty, 3]$
Re-examining the Game Tree

Fig. 5.5 in the textbook *incorrectly executes* the algorithm in Fig. 5.7.

Min node returns a value ≤ 3

Possible range of the returned value

Min node returns $(-\infty, 3]$
Re-examining the Game Tree

Fig. 5.5 in the textbook *incorrectly executes* the algorithm in Fig. 5.7.

Min node returns a value $\leq 3$

$\alpha \beta
\langle -\infty, \infty \rangle$

possible range of the returned value

$\langle -\infty, 3 \rangle$

$\langle -\infty, \infty \rangle$

$\langle -\infty, 3 \rangle$
Cont’d

(c) \((-\infty, 3] \rightarrow (-\infty, \infty)\)
Cont’d

(c) $(-\infty, 3]$ $(-\infty, \infty)$
Continued...
Cont’d
Cont’d
Cont’d
Cont’d

(c) \((-\infty, 3]\) and [3, \(\infty\))

(d) \((-\infty, 3]\) and [3, \(\infty\)]

2 < 3 (no change in \(\beta\) value)
Cont’d

(c) $(-\infty, 3] \rightarrow [3, \infty)$

(d) $(-\infty, 3] \rightarrow [3, \infty)$

$2 < 3$

(no change in \(\beta\) value)

pruned
Cont’d
Cont’d
Cont’d
Cont’d
Cont'd
Cont'd
Cont’d
Cont’d
\[
\text{MINIMAX}(\text{root}) = \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2))
\]
\[
= \max(3, \min(2, x, y), 2)
\]
\[
= \max(3, z, 2) \quad \text{where } z = \min(2, x, y) \leq 2
\]
\[
= 3.
\]
A Larger Example (Wikipedia)
A Larger Example (Wikipedia)

Current min value (4) < current max value (5) at parent; no need for further exploration
A Larger Example (Wikipedia)

Current min value (4) < current max value (5) at parent; no need for further exploration
A Larger Example (Wikipedia)
A Larger Example (Wikipedia)

Current min (5) at node < current max (6) at parent
A Larger Example (Wikipedia)

Current min (5) at node < current max (6) at parent