Game Playing: Adversarial Search

Chapter 5
Outline

- Games
- Perfect play
  - minimax search
  - $\alpha-\beta$ pruning
- Resource limits and approximate evaluation
- Games of chance
- Games of imperfect information
Games vs. Search Problems

In games we have:

- “Unpredictable” opponent ⇒ solution is a *strategy*, specifying a move for every possible opponent reply
- Time limits: Unlikely to find goal; do the best that you can.
Games vs. Search Problems

In games we have:

- “Unpredictable” opponent ⇒ solution is a *strategy*, specifying a move for every possible opponent reply
- Time limits: Unlikely to find goal; do the best that you can.

Game playing goes back a long way:

- Computer considers possible lines of play (Babbage, 1846)
- Algorithm for perfect play (Zermelo, 1912; Von Neumann, 1944)
- Finite horizon, approx. evaluation (Zuse, 1945; Wiener, 1948; Shannon, 1950)
- First chess program (Turing, 1951)
- Machine learning to improve evaluation (Samuel, 1952–57)
- Pruning to allow deeper search (McCarthy, 1956)
<table>
<thead>
<tr>
<th></th>
<th>deterministic</th>
<th>chance</th>
</tr>
</thead>
<tbody>
<tr>
<td>perfect information</td>
<td></td>
<td></td>
</tr>
<tr>
<td>imperfect information</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Types of Games</td>
<td>Deterministic</td>
<td>Chance</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>Perfect Information</td>
<td>Chess, checkers, go, othello,</td>
<td>Backgammon</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Monopoly</td>
</tr>
<tr>
<td>Imperfect Information</td>
<td>Battleships, blind tictactoe</td>
<td>Bridge, poker, scrabble, poker, war</td>
</tr>
</tbody>
</table>
Two-Player Games

- Two players, MAX and MIN, who take turns playing.

Main game components:
- Initial state: Initial game position.
- Actions: The set of legal moves.
- Transition function: Returns a list of legal moves and the resulting state.
- Terminal test: Determines when the game is over.
- Utility function: Value of a terminal state.

- Also called an objective or payoff function.
- Generally we'll deal with zero-sum games.
- Later we'll talk about a static evaluation function, which gives a value to every game state.
Two-Player Games

- Two players, MAX and MIN, who take turns playing.
- Main game components:
  
  * **Initial state**: Initial game position.
Two-Player Games

- Two players, MAX and MIN, who take turns playing.
- Main game components:
  
  * **Initial state**: Initial game position.
  * **Actions**: The set of legal moves

- Also called an objective or payoff function.
- Generally we’ll deal with zero-sum games.
- Later we’ll talk about a static evaluation function, which gives a value to every game state.
Two-Player Games

• Two players, MAX and MIN, who take turns playing.
• Main game components:

  *Initial state:* Initial game position.
  *Actions:* The set of legal moves
  *Transition function:* Returns a list of legal moves and the resulting state
Two-Player Games

- Two players, MAX and MIN, who take turns playing.
- Main game components:
  
  * **Initial state**: Initial game position.
  * **Actions**: The set of legal moves
  * **Transition function**: Returns a list of legal moves and the resulting state
  * **Terminal test**: Determines when the game is over.

- Later we'll talk about a static evaluation function, which gives a value to every game state.

- Generally we'll deal with zero-sum games.
Two-Player Games

- Two players, MAX and MIN, who take turns playing.
- Main game components:
  
  * **Initial state:** Initial game position.
  * **Actions:** The set of legal moves
  * **Transition function:** Returns a list of legal moves and the resulting state
  * **Terminal test:** Determines when the game is over.
  * **Utility function:** Value of a terminal state.
    
    - Also called a *objective* or *payoff function*
    - Generally we’ll deal with *zero-sum* games.
  
- Later we’ll talk about a *static evaluation function*, which gives a value to every game state.
Game Tree (2-player, deterministic, turns)

MAX (X)

MIN (O)

MAX (X)

MIN (O)

TERMINAL

Utility

-1 0 +1
Minimax

• Perfect play for deterministic, perfect-information games
• Idea: choose move to position with highest minimax value
  = best achievable payoff against best play
• E.g., 2-ply game:
Minimax Value

\[ \text{Minimax Value} (n) = \begin{cases} 
\text{Utility} (n) & \text{if } n \text{ is a terminal node} \\
\max_{s \in \text{Successors} (n)} \text{Minimax Value} (s) & \text{if } n \text{ is a MAX node} \\
\min_{s \in \text{Successors} (n)} \text{Minimax Value} (s) & \text{if } n \text{ is a MIN node}
\end{cases} \]
Minimax Algorithm

Function Minimax-Decision(state) returns an action
inputs: state current state in game
return \( a \in \text{Actions}(state) \) maximizing Min-Value(Result(a, state))

Function Max-Value(state) returns a utility value
if Terminal-Test(state) then return Utility(state)
\( v \leftarrow -\infty \)
for \( s \) 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 \( s \) in Successors(state) do \( v \leftarrow \text{Min}(v, \text{Max-Value}(s)) \)
return \( v \)
Properties of Minimax

Complete: ??
Properties of Minimax

Complete: Yes, if tree is finite. (Chess has specific rules for this).
Optimal: ??
Properties of Minimax

**Complete:** Yes, if tree is finite.

**Optimal:** Yes, against a rational opponent. Otherwise??

**Time complexity:** ??
Properties of Minimax

Complete: Yes, if tree is finite.

Optimal: Yes, against an optimal opponent. Otherwise??

Time complexity: $O(b^m)$

Space complexity: ??
Properties of Minimax

Complete: Yes, if tree is finite.

Optimal: Yes, against an optimal opponent. Otherwise??

Time complexity: $O(b^m)$

Space complexity: $O(bm)$ (depth-first exploration)
Properties of Minimax

Complete: Yes, if tree is finite.

Optimal: Yes, against an optimal opponent. Otherwise??

Time complexity: $O(b^m)$

Space complexity: $O(bm)$ (depth-first exploration)

• For chess, $b \approx 35$, $m \approx 100$ for “reasonable” games
  Exact solution is completely infeasible

• But do we need to explore every path?
\[\alpha - \beta\] Pruning

- Game tree search is inherently exponential
- However we can speed things up by \textit{pruning} parts of the search space that are guaranteed to be inferior.
- \textit{\(\alpha-\beta\) pruning} returns the same move as minimax, but prunes branches that can’t affect the final outcome.
$\alpha-\beta$ Pruning Example

MAX

MIN

\[\begin{array}{c}
3 \\
12 \\
8 \\
\end{array}\]
α–β Pruning Example

MAX

MIN

\[ \begin{aligned}
&3 &12 &8 \\
&3 & &2 \\
& & &2
\end{aligned} \]
$\alpha-\beta$ Pruning Example

MAX

MIN

3
12
8

2
14

3
2
2

\text{X X}

3
\leq 2
\leq 14

\geq 3
**α–β Pruning Example**

![Pruning Example Diagram]
$\alpha-\beta$ Pruning Example
• $\alpha$ is the best value (to MAX) found so far.
• If $V$ is worse than $\alpha$, MAX will avoid it.
  • So this node won’t be reached in play.
  • So prune that branch
• Define $\beta$ similarly for MIN
The General Case

• $\alpha$ is the value of the best (i.e. maximum) choice we have found so far for MAX.

• $\beta$ is the value of the best (i.e. minimum) choice we have found so far for MIN.

• $\alpha-\beta$ search updates the values of $\alpha$ and $\beta$ as it progresses.
  • It prunes branches at a node if they are known to be worse than the current $\alpha$ (for MAX) or $\beta$ (for MIN) values.

Note:

• The $\alpha$ values of MAX nodes can never decrease.

• The $\beta$ values of MIN nodes can never increase.
**α–β Search**

Observe:

- Search can be discontinued below any MAX node where that node has $\alpha$ value $\geq$ the $\beta$ value of any of its MIN ancestors.
  - The final value of this MAX node can then be set to its $\alpha$ value.

- Search can be discontinued below any MIN node where that node has $\beta$ value $\leq$ the $\alpha$ value of any of its MAX ancestors.
  - The final value of this MIN node can then be set to its $\beta$ value.

Main point (again):

- The $\alpha$ value of a MAX node = the current largest final value of its successors.

- The $\beta$ value of a MIN node = the current smallest final value of its successors.
The $\alpha-\beta$ Algorithm

Function Alpha-Beta-Decision(state) returns an action

$v \leftarrow \text{Max-Value}(\text{state}, -\infty, \infty)$

return the $a$ in Actions(state) with value $v$
The $\alpha-\beta$ Algorithm

Function $\text{Max-Value}(\text{state}, \alpha, \beta)$ returns a utility value

**inputs:** state current state in game

$\alpha$, the value of the best alternative for $\text{MAX}$ along the path to state

$\beta$, the value of the best alternative for $\text{MIN}$ along the path to state
The $\alpha–\beta$ Algorithm

Function \textbf{Max-Value}(state, $\alpha$, $\beta$) returns a utility value

\textbf{inputs:} state current state in game
- $\alpha$, the value of the best alternative for $\text{MAX}$ along the path to state
- $\beta$, the value of the best alternative for $\text{MIN}$ along the path to state

if Terminal-Test(state) then return Utility(state)

$v \leftarrow -\infty$

for $s$ in Successors(state) do
    $v \leftarrow \text{Max}(v, \text{Min-Value}(s, \alpha, \beta))$
    if $v \geq \beta$ then return $v$ /* discontinue since Min can do better elsewhere */

$\alpha \leftarrow \text{Max}(\alpha, v)$

return $v$

Function \textbf{Min-Value}(state, $\alpha$, $\beta$) returns a utility value

same as Max-Value but with roles of $\alpha$, $\beta$ reversed

This is slightly simpler than the algorithm in the 3$^{rd}$ ed.
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
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
- **Q:** What if you “reverse” a perfect ordering?
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
  
  Q: What if you “reverse” a perfect ordering?
- A simple example of the value of reasoning about which computations are relevant (a form of *metareasoning*)
- Unfortunately, for chess, \( 35^{50} \) is still impossible!
Resource Limits

- Most games cannot be exhaustively searched.
  - Usually have to terminate search before hitting a goal state.

- Standard approach:
  - Use Cutoff-Test instead of Terminal-Test
    e.g., depth limit
  - Use Eval instead of Utility/Goal-Test
    i.e., evaluation function that estimates desirability of position

- Suppose we have 100 seconds, explore $10^4$ nodes/second
  $\Rightarrow 10^6$ nodes per move $\approx 35^{8/2}$
  $\Rightarrow \alpha-\beta$ reaches depth 8 $\Rightarrow$ pretty good chess program
    (if we have a good static evaluation function).
• 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
  \[ f_1(s) = (\# \text{ of white queens}) - (\# \text{ of black queens}), \text{ etc.} \]
Evaluation Functions: Issues

- Quiescence vs. non-quiescence
  - Search to a quiescent area (i.e. where the static evaluation function doesn’t change much between moves).
  - Or (pretty much the same thing):
    If the static evaluation function changes radically between moves, keep searching.

- Horizon effect
  - Problem if there is an unavoidable loss that can be pushed beyond the cutoff by other moves.
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

- Used an endgame database giving perfect play for all positions with \( \leq 8 \) pieces on the board, a total of 443,748,401,247 positions.
- Now totally solved (by computer)
Deterministic Games in Practice: Chess

- Deep Blue searched 200 million positions per second, used very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.
Deterministic Games in Practice: Othello

- Human champions refuse to compete against computers, which are too good.
- Makes a good AI assignment!
Deterministic Games in Practice: Go

- Until recently, human champions refused to compete against computers, which were too bad.
Deterministic Games in Practice: Go

- Until recently, human champions refused to compete against computers, which were too bad.
- In chess, there are something around $10^{40}$ positions, in Go there are $10^{170}$ positions.
Deterministic Games in Practice: Go

- Until recently, human champions refused to compete against computers, which were too bad.
- In chess, there are something around $10^{40}$ positions, in Go there are $10^{170}$ positions.
- Go was considered hard because
  - the search space is staggering and
  - it was extremely difficult to evaluate a board position.
Deterministic Games in Practice: Go

- Until recently, human champions refused to compete against computers, which were too bad.
- In chess, there are something around $10^{40}$ positions, in Go there are $10^{170}$ positions.
- Go was considered hard because
  - the search space is staggering and
  - it was extremely difficult to evaluate a board position.
- However, in March 2016, AlphaGo beat Lee Sedol (winner of 18 world titles) 4 games to 1
- AlphaGo combines learning via neural networks, along with Monte Carlo tree search.
Deterministic Games in Practice: DeepBlue vs. AlphaGo

Deep Blue
- Handcrafted chess knowledge
- Alpha-beta search guided by heuristic evaluation function
- 200 million positions / second

AlphaGo
- Knowledge learned from expert games and self-play
- Monte-Carlo search guided by policy and value networks
- 60,000 positions / second

Q: Which seems the more “human-like”?
Deterministic Games in Practice: DeepBlue vs. AlphaGo

Deep Blue

- Handcrafted chess knowledge
- Alpha-beta search guided by heuristic evaluation function
- 200 million positions / second

AlphaGo

- Knowledge learned from expert games and self-play
- Monte-Carlo search guided by policy and value networks
- 60,000 positions / second

Q: Which seems the more “human-like”? 
Nondeterministic Games: Backgammon
Nondeterministic Games in General

- In nondeterministic games, chance is introduced by dice, card-shuffling, etc.
- Simplified example with coin-flipping:

```
MIN
  2
MAX
  4
  0
  5
CHANCE
  7
  4
  6
  −2
0.5
0.5
0.5
0.5
3
−1
```
ExpectiMinimax Value

\[
\text{ExpectiMinimax Value}(n) =
\begin{cases} 
\text{Utility}(n) & \text{if } n \text{ is a terminal node} \\
\max_{s \in \text{Successors}(n)} \text{ExpectiMinimax Value}(s) & \text{if } n \text{ is a MAX node} \\
\min_{s \in \text{Successors}(n)} \text{ExpectiMinimax Value}(s) & \text{if } n \text{ is a MIN node} \\
\sum_{s \in \text{Successors}(n)} P(s) \cdot \text{ExpectiMinimax Value}(s) & \text{if } n \text{ is a chance node}
\end{cases}
\]
Algorithm for Nondeterministic Games

- `EXPECTIMINIMAX` gives perfect play
Algorithm for Nondeterministic Games

- **EXPECTIMINIMAX** gives perfect play.
- Given the chance nodes, MAX may not get the best outcome.
  - But MAX’s move gives the best *expected* outcome.
Algorithm for Nondeterministic Games

- **Expectiminimax** gives perfect play
- Given the chance nodes, MAX may not get the best outcome. But MAX’s move gives the best *expected* outcome.
- Algorithm is just like **Minimax**, except we must also handle chance nodes:

  ```
  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 $= 20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$
- 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
Digression: Exact Values DO Matter

- Behaviour is preserved only by *positive linear* transformation of $\text{Eval}$
- Hence $\text{Eval}$ 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

* but in fact this doesn’t quite work out (as discussed next)
Example

- Four-card bridge/whist/hearts hand, MAX to play first
Example

- Four-card bridge/whist/hearts hand, MAX to play first
Example

- Four-card bridge/whist/hearts hand, \( \text{MAX} \) to play first
Commonsense Example

1. 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.
Commonsense Example

1. 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.

2. 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.
Commonsense Example

1. 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.

2. 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.

3. 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

• The 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* that 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
Summary

• Games are fun to work on!
• 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