Game Playing: Adversarial Search

Chapter 5

Outline

- Games
- Perfect play
  - minimax search
  - α–β pruning
- Resource limits and approximate evaluation
- Games of chance
- Games of imperfect information

Games vs. Search Problems

In games we have:

- "Unpredictable" opponent $\Rightarrow$ 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 $\Rightarrow$ 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)

Types of Games

	deterministic	chance
perfect information		
imperfect information		

Types of Games

	deterministic	chance
perfect information	chess, checkers, go, othello,	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble, poker, war

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 in a state

  Transition function: Returns a list of legal moves and the resulting state

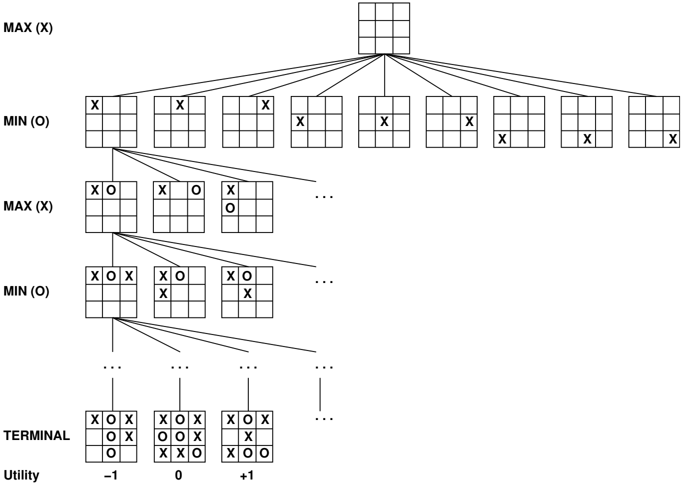
  Terminal test: Determines when the game is over.

  Utility function: Value of a terminal state.

    - Also called a *objective* or *payoff function*
    - Generally we'll deal with *zero-sum* games.

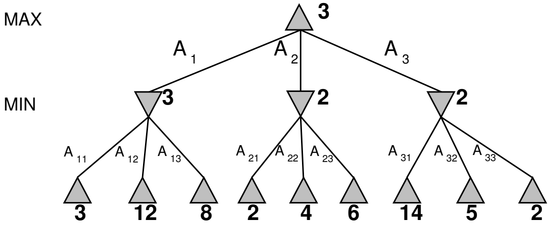
- Later we'll talk about a *static evaluation function*, which gives a value to every game state.

Game Tree (2-player, deterministic, turns)



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

$$MinimaxValue(n) =$$

$$\begin{cases} Utility(n) & \text{if } n \text{ is a terminal node} \\ \max_{s \in Successors(n)} MinimaxValue(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in Successors(n)} MinimaxValue(s) & \text{if } n \text{ is a MIN node} \end{cases}$$

Minimax Algorithm

```
Function Minimax-Decision(state) returns an action
   inputs: state current state in game
   return a ∈ 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 ← −∞
   for s in Successors(state) do v ←Max(v, Min-Value(s))
   return v

Function Min-Value(state) returns a utility value
   if Terminal-Test(state) then return Utility(state)
   v ←∞
   for s in Successors(state) do v ←Min(v, 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: $O(b^m)$

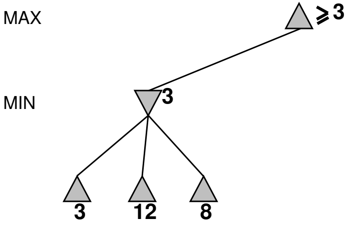
Space complexity: $O(bm)$ (depth-first exploration)

- For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
  - ☞ Exact solution is completely infeasible
- But do we need to explore every path?

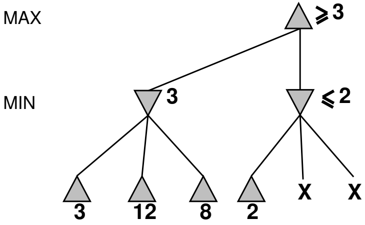
α–β Pruning

- Game tree search is inherently exponential
- However we can speed things up by *pruning* parts of the search space that are guaranteed to be inferior.
- α–β *pruning* returns the same move as minimax, but prunes branches that can't affect the final outcome.

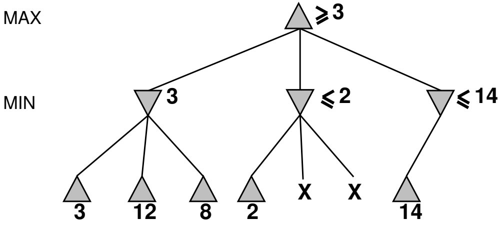
α–β Pruning Example



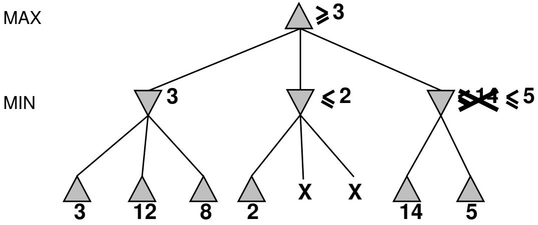
α–β Pruning Example



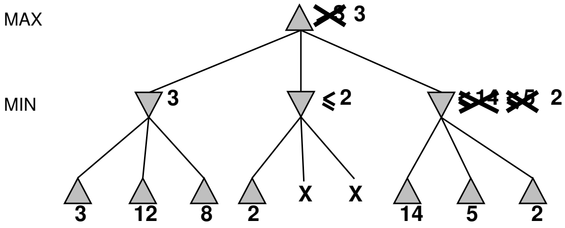
α–β Pruning Example



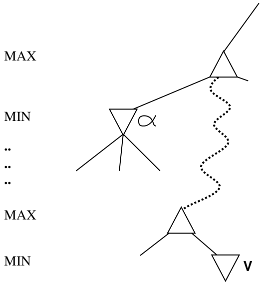
α–β Pruning Example



α–β Pruning Example



The General Case



- α is the best value (to MAX) found so far.
- If V is worse than α, MAX will avoid it.
  - So this node won't be reached in play.
  - So prune that branch
- Define β similarly for MIN

The General Case

- α is the value of the best (i.e. maximum) choice we have found so far for MAX.
- β is the value of the best (i.e. minimum) choice we have found so far for MIN.
- α–β search updates the values of α and β as it progresses.
  - It prunes branches at a node if they are known to be worse than the current α (for MAX) or β (for MIN) values.

Note:

- The α values of MAX nodes can never decrease.
- The β values of MIN nodes can never increase.

α–β Search

Observe:

- Search can be discontinued below any MIN node having β value $\leq$ the α value of any of its MAX node ancestors.
  - The final value of this MIN node can then be set to its β value.
- Search can be discontinued below any MAX node having α value $\geq$ the β value of any of its MIN node ancestors.
  - The final value of this MAX node can then be set to its α value.

Main point (again):

- The α value of a MAX node $=$ the current largest final value of its successors.
- The β value of a MIN node $=$ the current smallest final value of its successors.

The α–β Algorithm

Function Alpha-Beta-Decision(state) returns an action
  v $\leftarrow$Max-Value(state,$-\infty$,∞)
  return the a in Actions(state) with value v

The α–β Algorithm

```
Function Max-Value(state, α, β) returns a utility value
   inputs: state current state in game
      α, the value of the best alternative for MAX along the path to state
      β, the value of the best alternative for MIN along the path to state
   if Terminal-Test(state) then return Utility(state)
   v ← −∞
   for s in Successors(state) do
      v ← Max(v, Min-Value(s, α, β))
      if v ≥ β then return v
      α ← Max(α, v)
   return v

Function Min-Value(state, α, β) returns a utility value
   same as Max-Value but with roles of α, β reversed
```

☞ This is a bit simpler than the algorithm in the 3rd ed.

Properties of α–β

- 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 α–β

- 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 α–β

- Pruning *does not* affect final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering," time complexity $= O(b^{m/2})$

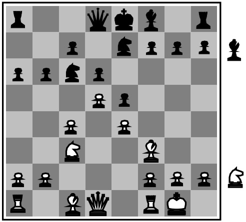
  $\Rightarrow$ *doubles* solvable depth

  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

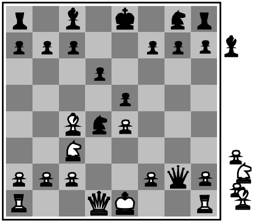
- Most games cannot be exhaustively searched.
- So have to terminate search before hitting a goal state (usually)
- Standard approach:
  - Use CUTOFF-TEST instead of TERMINAL-TEST
    e.g., depth limit (perhaps add *quiescence search*)
  - Use EVAL instead of UTILITY/GOAL-TEST
    i.e., *evaluation function* that estimates desirability of position
- Suppose we have 100 seconds, explore 10^4 nodes/second
  $\Rightarrow 10^6$ nodes per move $\approx 35^{8/2}$
  $\Rightarrow \alpha$–β reaches depth 8 $\Rightarrow$ pretty good chess program
  (if we have a good static evaluation function).

Evaluation Functions



Black to move

White slightly better



White to move

Black winning

- For chess, typically *linear* weighted sum of *features*

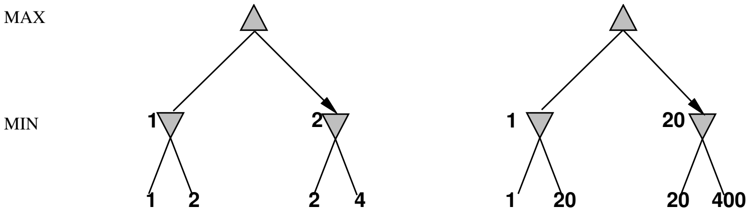
  $$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

- e.g., $w_1 = 9$ with
  $f_1(s) = (\#$ of white queens$) - (\#$ of black queens$)$, 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 searcing.
- Horizon effect

Digression: Exact Values Don't Matter



- Behaviour is preserved under any *monotonic* transformation of 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 giving perfect play for all positions with ≤ 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 defeated human world champion Gary Kasparov in a six-game match in 1997.
- 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.
- Would make a good AI assignment!

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

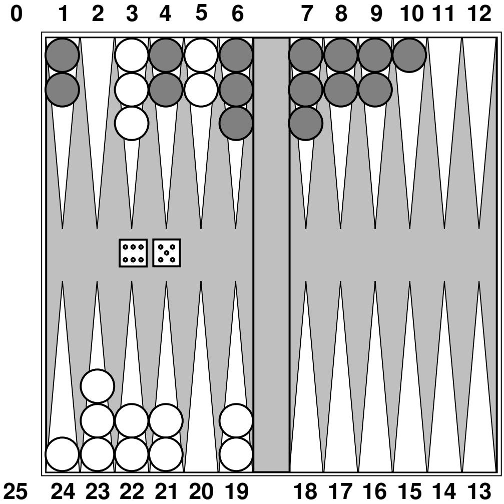
Deep Blue

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

AlphaGo

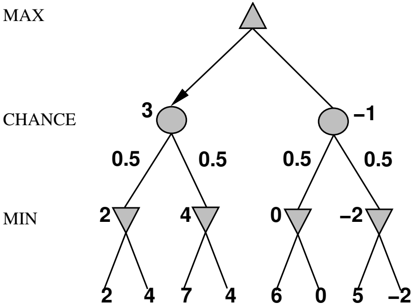
- Knowledge learned from expert games and self-play
- Monte-Carlo search guided by policy and value networks
- 60,000 positions / second

Nondeterministic Games: Backgammon



Nondeterministic Games in General

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



ExpectiMinimax Value

$$ExpectiMinimaxValue(n) =$$

$$\begin{cases} Utility(n) & \text{if } n \text{ is a terminal node} \\ \max_{s \in Successors(n)} ExpectiMinimaxValue(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in Successors(n)} ExpectiMinimaxValue(s) & \text{if } n \text{ is a MIN node} \\ \Sigma_{s \in Successors(n)} P(s).ExpectiMinimaxValue(s) & \text{if } n \text{ is a chance node} \end{cases}$$

Algorithm for Nondeterministic Games

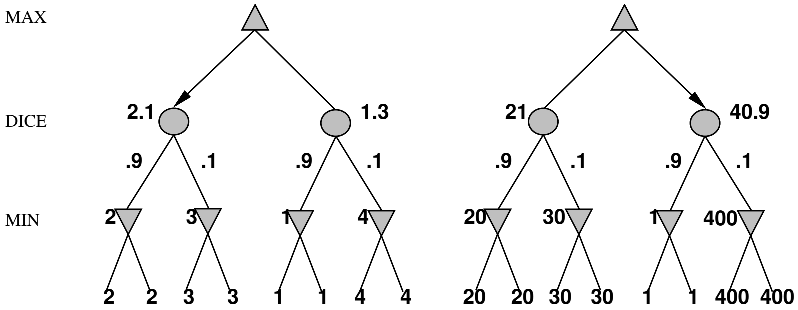
- EXPECTIMINIMAX gives perfect play
- Given the chance nodes, MAX may not get the best outcome.
- But MAX's move gives the best *expected* outcome.
- Algorithm is just like MINIMAX, except we must also handle chance nodes:

```
...
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)
  $\text{depth } 4 = 20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$
- As depth increases, probability of reaching a given node shrinks
  - value of lookahead is diminished
- α–β 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 EVAL
- Hence EVAL should be proportional to the expected payoff

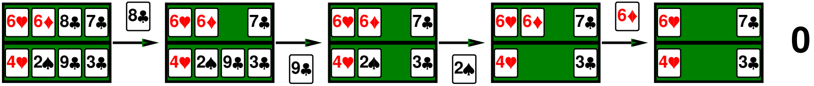
Games of Imperfect Information

- 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

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

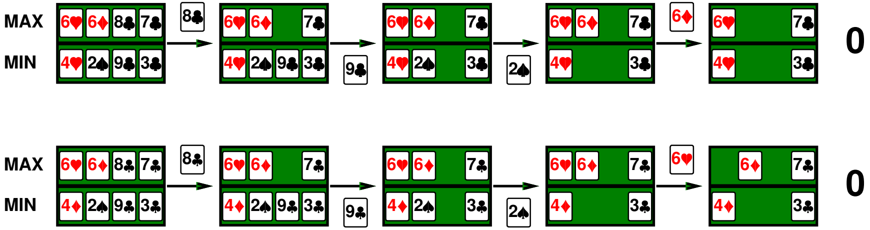
Example

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



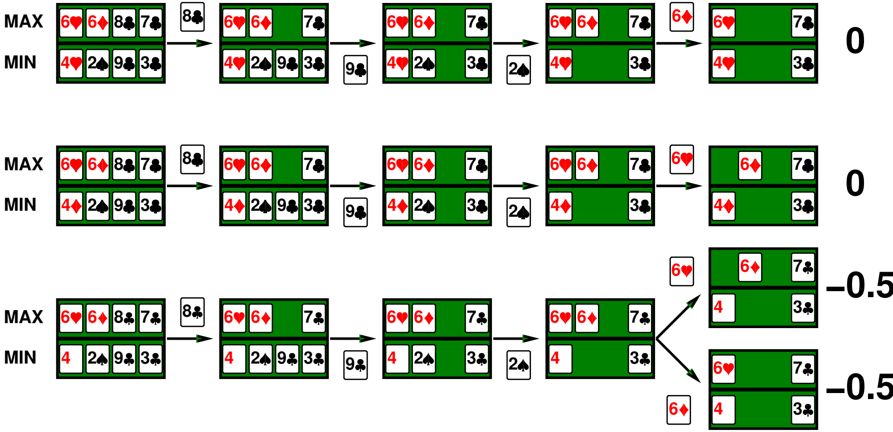
Example

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



Example

- Four-card bridge/whist/hearts hand, 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 $\Rightarrow$ 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