

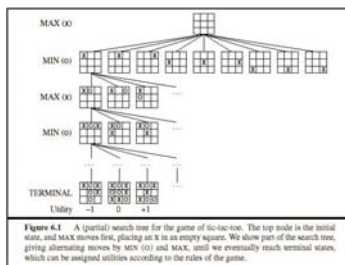
# CS 416 Artificial Intelligence

Lecture 9  
Adversarial Search  
Chapter 6

## Games

- "Shall we play a game?" 🎮
- Let's play tic-tac-toe

## Minimax



## What data do we need to play?

- Initial State
  - How does the game start?
- Successor Function
  - A list of legal (move, state) pairs for each state
- Terminal Test
  - Determines when game is over
- Utility Function
  - Provides numeric value for all terminal states

## Minimax Strategy

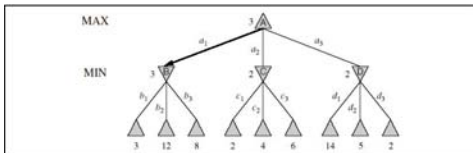
- Optimal Strategy
  - Leads to outcomes at least as good as any other strategy when playing an infallible opponent
  - Pick the option that most (max) minimizes the damage your opponent can do
    - maximize the worst-case outcome
    - because your skillful opponent will certainly find the most damaging move

## Minimax

- Algorithm
  - $\text{MinimaxValue}(n) =$ 

$\text{Utility}(n)$		if $n$ is a terminal state
$\max \text{MinimaxValue}(s)$	of all successors, $s$	if $n$ is a MAX node
$\min \text{MinimaxValue}(s)$	of all successors, $s$	if $n$ is a MIN node

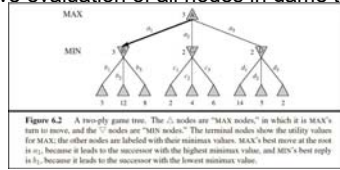
## Minimax



**Figure 6.2** A two-ply game tree. The  $\triangle$  nodes are "MAX nodes," in which it is MAX's turn to move, and the  $\nabla$  nodes are "MIN nodes." The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values. MAX's best move at the root is  $a_1$ , because it leads to the successor with the highest minimax value, and MIN's best reply is  $b_1$ , because it leads to the successor with the lowest minimax value.

## Minimax Algorithm

- We wish to identify minimax decision at the root
  - Recursive evaluation of all nodes in game tree



**Figure 6.2** A two-ply game tree. The  $\triangle$  nodes are "MAX nodes," in which it is MAX's turn to move, and the  $\nabla$  nodes are "MIN nodes." The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values. MAX's best move at the root is  $a_1$ , because it leads to the successor with the highest minimax value, and MIN's best reply is  $b_1$ , because it leads to the successor with the lowest minimax value.

- Time complexity =  $O(b^m)$

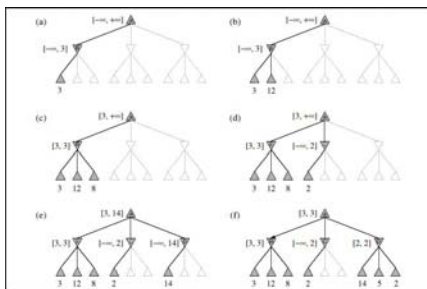
## Feasibility of minimax?

- How about a nice game of chess?
  - Avg branching = 35 and avg # moves = 50 for each player
    - $O(35^{100})$  time complexity =  $10^{154}$  nodes
      - $10^{40}$  distinct nodes
- Minimax is impractical if directly applied to chess

## Pruning minimax tree

- Are there times when you know you need not explore a particular move?
  - When the move is poor?
  - Poor compared to what?
  - Poor compared to what you have explored so far

## Alpha-beta pruning



## Alpha-beta pruning

- $\alpha$ 
  - the value of the best (highest) choice so far in search of MAX
- $\beta$ 
  - the value of the best (lowest) choice so far in search of MIN
- Order of considering successors matters (look at step f in previous slide)
  - If possible, consider best successors first

## Realtime decisions

- What if you don't have enough time to explore entire search tree?
  - We cannot search all the way down to terminal state for all decision sequences
  - Use a heuristic to approximate (guess) eventual terminal state

## Evaluation Function

- The heuristic that estimates expected utility
  - Cannot take too long (otherwise recurse to get answer)
  - It should preserve the ordering among terminal states
    - otherwise it can cause bad decision making
  - Define features of game state that assist in evaluation
    - what are features of chess?

## Truncating minimax search

- When do you recurse or use evaluation function?
  - Cutoff-Test (state, depth) returns 1 or 0
    - When 1 is returned, use evaluation function
  - Cutoff beyond a certain depth
  - Cutoff if state is stable (more predictable)
  - Cutoff moves you know are bad (forward pruning)

## Benefits of truncation

- Comparing Chess
  - Using minimax 5 ply
  - Average Human 6-8 ply
  - Using alpha-beta 10 ply
  - Intelligent pruning 14 ply

## Games with chance

- How to include chance in game tree?
  - Add chance nodes

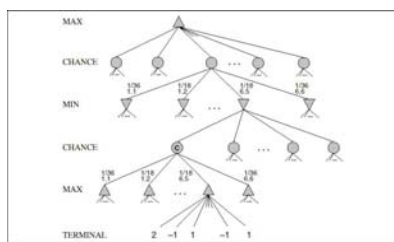


Figure 6.11 Schematic game tree for a backgammon position.

## Expectiminimax

- Expectiminimax (n) =
  - utility(n) if n is a terminal state
  - $\max_{s \in \text{Successors}(n)} \text{EXPECTIMINIMAX}(s)$  if n is a MAX node
  - $\min_{s \in \text{Successors}(n)} \text{EXPECTIMINIMAX}(s)$  if n is a MIN node
  - $\sum_{s \in \text{Successors}(n)} P(s) * \text{EXPECTIMINIMAX}(s)$  if n is a chance node

## Pruning

- Can we prune search in games of chance?
  - Think about alpha-beta pruning
    - don't explore nodes that you know are worse than what you have
    - we don't know what we have
      - chance node values are average of successors

## History of Games

- Chess, Deep Blue
  - IBM: 30 RS/6000 comps with 480 custom VLSI chess chips
  - Deep Thought design came from Campbell and Hsu at CMU
  - 126 mil nodes / s
  - 30 bil positions per move
  - routine reaching depth of 14
  - iterative deepening alpha-beta search

## Deep Blue

- evaluation function had 8000 features
- 4000 opening moves in memory
- 700,000 grandmaster games from which recommendations extracted
- many endgames solved for all five piece combos

## Checkers

- Arthur Samuel of IBM, 1952
  - program learned by playing against itself
  - beat a human in 1962 (but human clearly made error)
  - 19 KB of memory
  - 0.000001 Ghz processor

## Chinook, Jonathan Schaeffer, 1990

- Alpha-beta search on regular PCs
- database of all 444 billion endgame positions with 8 pieces
- Played against Marion Tinsley
  - world champion for over 40 years
  - lost only 3 games in 40 years
  - Chinook won two games, but lost match
- Rematch with Tinsley was incomplete for health reasons
  - Chinook became world champion

## Othello

- Smaller search space (5 to 15 legal moves)
- Humans are no match for computers

## Backgammon

- Garry Tesauro, TD-Gammon, 1992
  - Reliably ranked in top-three players of world

## Discussion

- How reasonable is minimax?
  - perfectly performing opponent
  - perfect knowledge of leaf node evaluations
- strong assumptions

## Building alpha-beta tree

- Can we restrict the size of game tree?
  - alpha-beta will blindly explore tree in depth-first fashion even if only one move is possible from root
  - even if multiple moves are possible, can we use a quick search to eliminate some entirely?
  - utility vs. time tradeoff to decide when to explore new branches or to stay with what you have

## Metareasoning

- Reasoning about reasoning
  - alpha-beta is one example
    - think before you think
    - think about utility of thinking about something before you think about it
    - don't think about choices you don't have to think about

## Goal-directed reasoning / planning

- Minimax starts from root and moves forward using combinatorial search
- What about starting at goal and working backward
  - We talked about difficulty of identifying goal states in bidirectional search
  - We do not know how to combine the two in practical way