Artificial Intelligence
Adversarial Search

Games

- Multiagent environment
- Cooperative vs. competitive
  - Competitive environment is where the agents' goals are in conflict
  - Adversarial Search
- Game Theory
  - A branch of economics
  - Views the impact of agents on others as significant rather than competitive (or cooperative).

Properties of Games

- Game Theorists
  - Deterministic, turn-taking, two-player, zero-sum games of perfect information
- AI
  - Deterministic
  - Fully-observable
  - Two agents whose actions must alternate
  - Utility values at the end of the game are equal and opposite
    - In chess, one player wins (+1), one player loses (-1)
    - It is this opposition between the agents' utility functions that makes the situation adversarial

Why Games?

- Small defined set of rules
- Well defined knowledge set
- Easy to evaluate performance
- Large search spaces
  - Too large for exhaustive search
- Fame and Fortune
  - e.g. Chess and Deep Blue

Games as Search Problems

- Games have a state space search
  - Each potential board or game position is a state
  - Each possible move is an operation to another state
  - The state space can be HUGE!!!!!!!
    - Large branching factor (about 35 for chess)
    - Terminal state could be deep (about 50 for chess)

Games vs. Search Problems

- Unpredictable opponent
- Solution is a strategy
  – Specifying a move for every possible opponent reply
- Time limits
  – Unlikely to find the goal…agent must approximate

Types of Games

	Deterministic	**Chance**
Perfect Information	*Chess, checkers, go, othello*	*Backgammon, monopoly*
Imperfect Information		*Bridge, poker, scabble, nuclear war*

Example Computer Games

- Chess – Deep Blue (World Champion 1997)
- Checkers – Chinook (World Champion 1994)
- Othello – Logistello
  - Beginning, middle, and ending strategy
  - Generally accepted that humans are no match for computers at Othello
- Backgammon – TD-Gammon (Top Three)
- Go – Goemate and Go4++ (Weak Amateur)
- Bridge (Bridge Barron 1997, GIB 2000)
  - Imperfect information
  - multiplayer with two teams of two

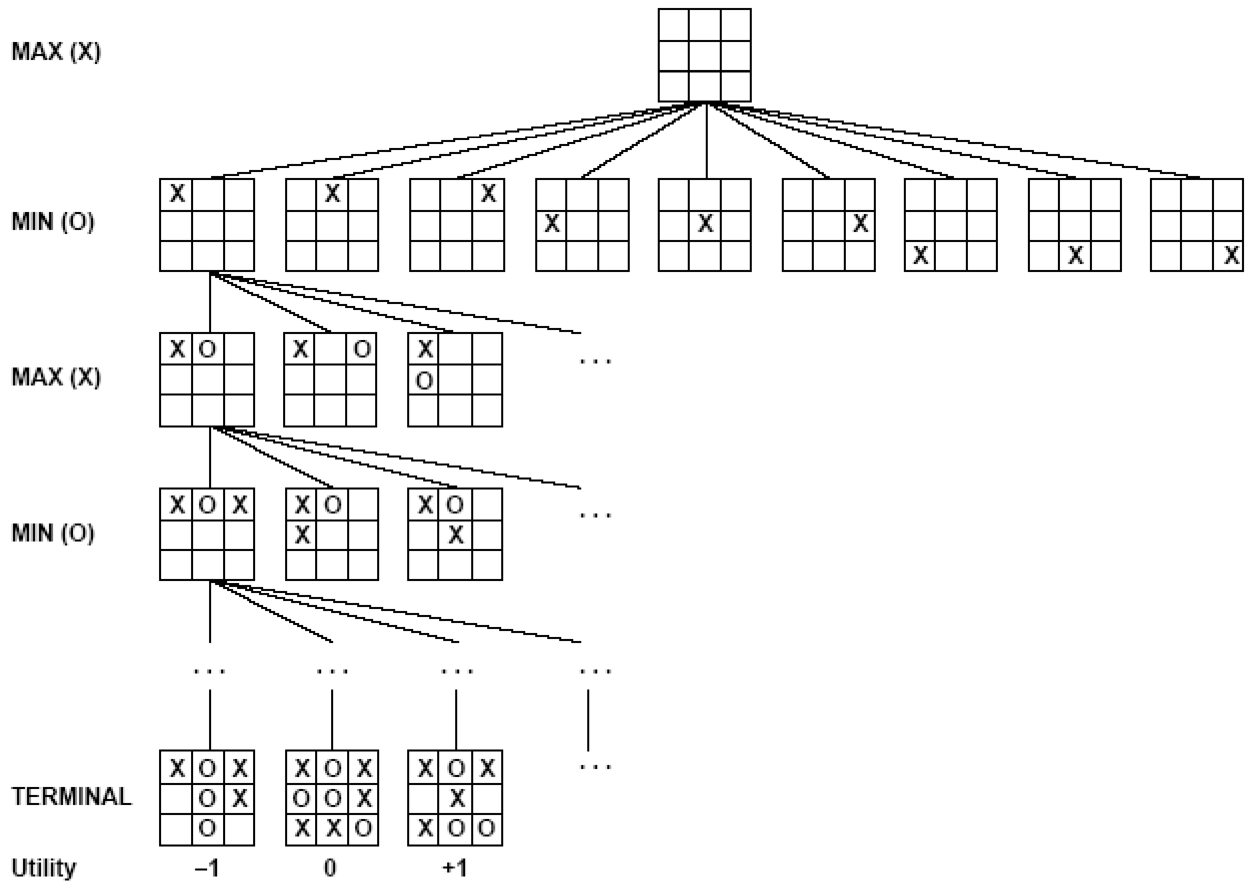
Optimal Decisions in Games

- Consider games with two players (MAX, MIN)
- Initial State
  - Board position and identifies the player to move
- Successor Function
  - Returns a list of (move, state) pairs; each a legal move and resulting state
- Terminal Test
  - Determines if the game is over (at terminal states)
- Utility Function
  - Objective function, payoff function, a numeric value for the terminal states (+1, -1) or (+192, -192)

Game Trees

- The root of the tree is the initial state
  - Next level is all of MAX's moves
  - Next level is all of MIN's moves
  - …
- Example: Tic-Tac-Toe
  - Root has 9 blank squares (MAX)
  - Level 1 has 8 blank squares (MIN)
  - Level 2 has 7 blank squares (MAX)
  - …
- Utility function:
  - win for X is +1
  - win for O is -1

Game Trees



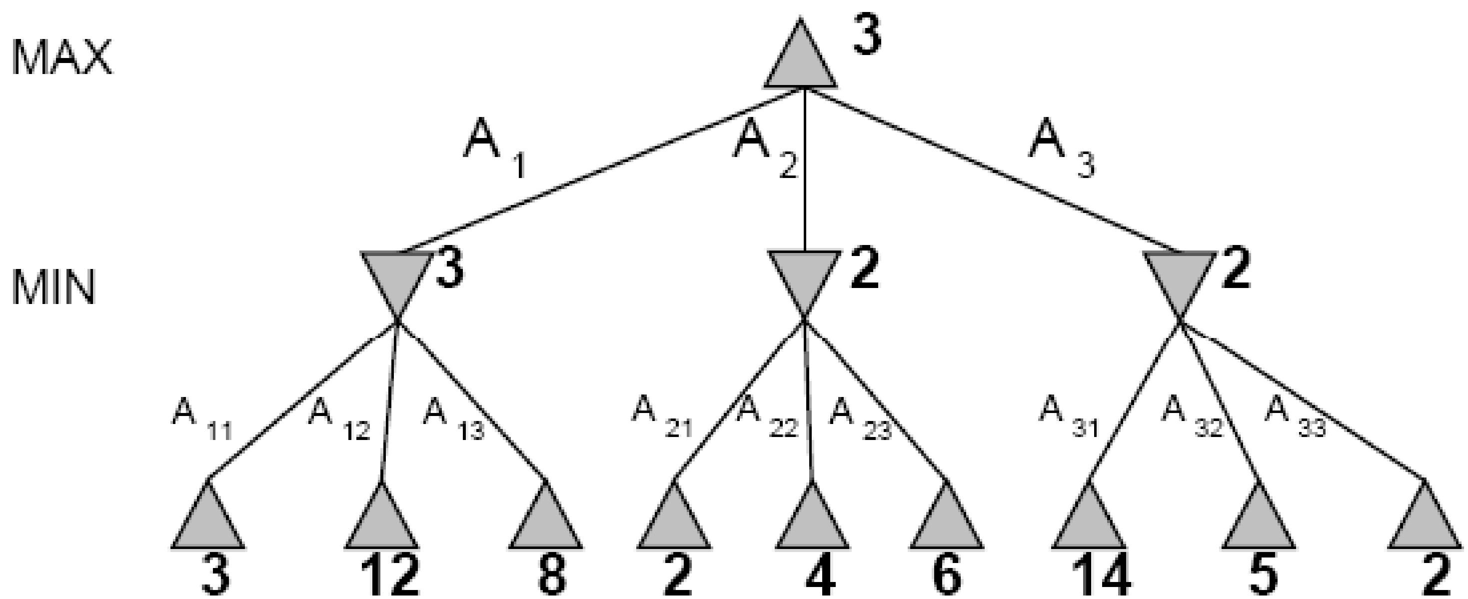
Minimax Strategy

- Basic Idea:
  - Choose the move with the highest minimax value
    - best achievable payoff against best play
  - Choose moves that will lead to a win, even though min is trying to block

- Max's goal: get to 1
- Min's goal: get to -1

- Minimax value of a node (backed up value):
  - If N is terminal, use the utility value
  - If N is a Max move, take max of successors
  - If N is a Min move, take min of successors

Minimax Strategy



Minimax Algorithm

```
function MINIMAX-DECISION(state, game) returns an action
   action, state ← the a, s in SUCCESSORS(state)
            such that MINIMAX-VALUE(s, game) is maximized
   return action
--------------------------------------------------------------
function MINIMAX-VALUE(state, game) returns a utility value
   if TERMINAL-TEST(state) then
      return UTILITY(state)
   else if MAX is to move in state then
      return the highest MINIMAX-VALUE of SUCCESSORS(state)
   else
      return the lowest MINIMAX-VALUE of SUCCESSORS(state)
```

Properties of Minimax

- Complete
  - Yes if the tree is finite (e.g. chess has specific rules for this)
- Optimal
  - Yes, against an optimal opponent, otherwise???
- Time
  - $O(b^m)$
- Space
  - O(bm) depth first exploration of the state space

Resource Limits

- Suppose there are 100 seconds, explore 10^4 nodes / second

- 10^6 nodes per move

- Standard approach
  - Cutoff test – depth limit
    - quiesence search – values that do not seem to change
  - Change the evaluation function

Evaluation Functions

- Example Chess:
  - Typical evaluation function is a linear sum of features
  - $Eval(s) = w_1f_1(s) + w_2f_2(s) + \ldots + w_nf_n(s)$
    - $w_1 = 9$
    - $f_1(s)$ = number of white queens) – number of black queens
    - etc.

Alpha-Beta Pruning

- The problem with minimax search is that the number of game states is has to examine is exponential in the number of moves

- Use pruning to eliminate large parts of the tree from consideration

- Alpha-Beta Pruning

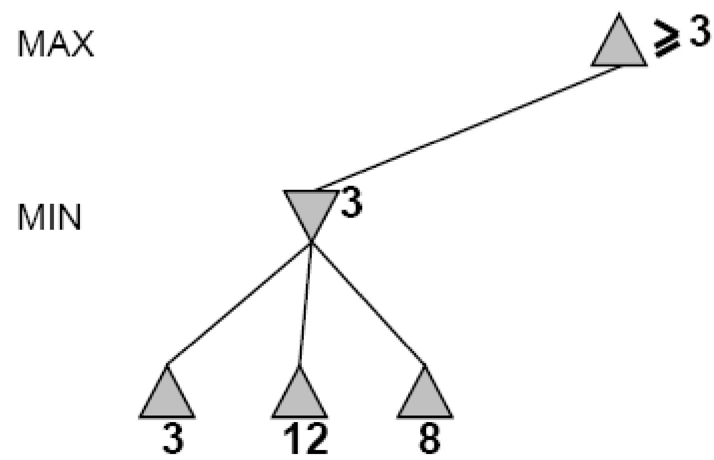
Alpha-Beta Pruning

- Recognize when a position can never be chosen in minimax *no matter what its children are*
  - Max (3, Min(2,x,y) …) is always ≥ 3
  - Min (2, Max(3,x,y) …) is always ≤ 2
  - We know this without knowing x and y!

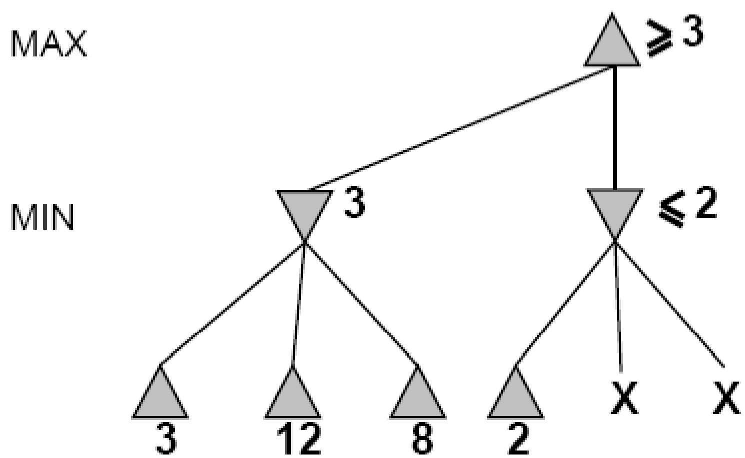
Alpha-Beta Pruning

- Alpha = the value of the best choice we’ve found so far for MAX (highest)
- Beta = the value of the best choice we’ve found so far for MIN (lowest)
- When maximizing, cut off values lower than Alpha
- When minimizing, cut off values greater than Beta

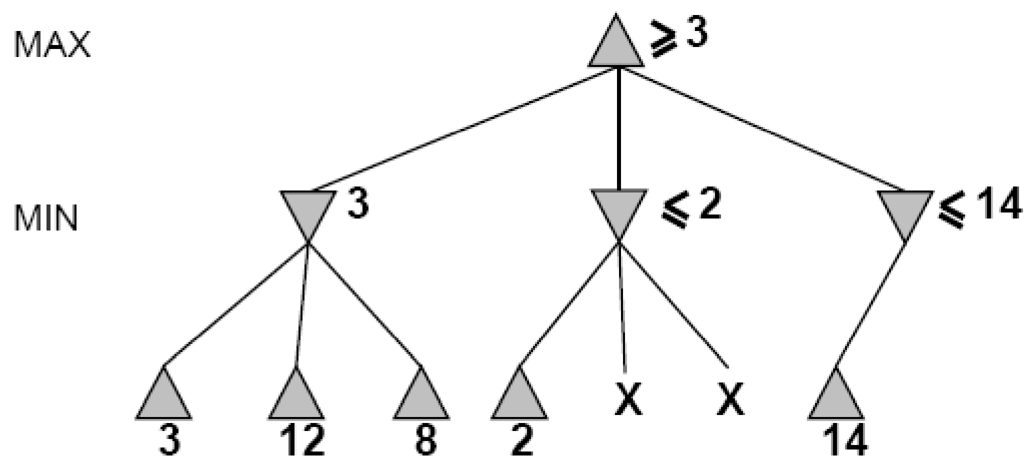
Alpha-Beta Pruning Example



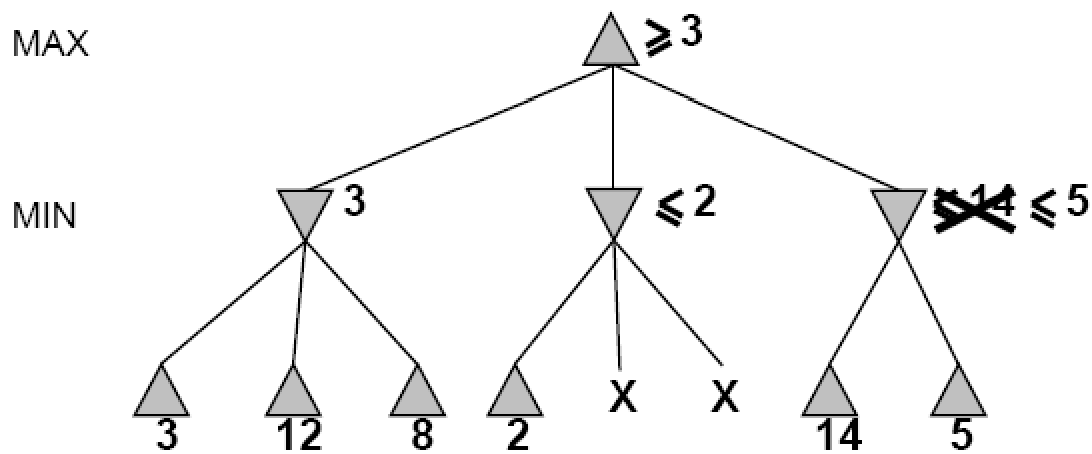
Alpha-Beta Pruning Example



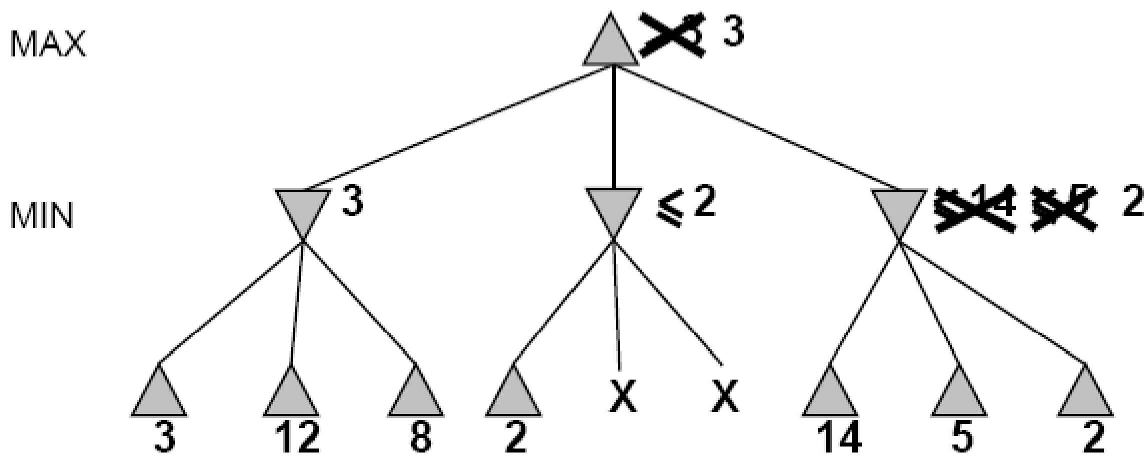
Alpha-Beta Pruning Example



Alpha-Beta Pruning Example



Alpha-Beta Pruning Example



A Few Notes on Alpha-Beta

- Effectiveness depends on order of successors (middle vs. last node of 2-ply example)

- If we can evaluate best successor first, search is $O(b^{d/2})$ instead of $O(b^d)$

- This means that in the same amount of time, alpha-beta search can search twice as deep!

A Few More Notes on Alpha-Beta

- Pruning ***does not*** affect the final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering", time complexity $O(b^{m/2})$
  - doubles the depth of search
  - can easily reach depth of 8 and play good chess (branching factor of 6 instead of 35)

Optimizing Minimax Search

- Use alpha-beta cutoffs
  - Evaluate most promising moves first
- Remember prior positions, reuse their backed-up values
  - Transposition table (like closed list in A*)
- Avoid generating equivalent states (e.g. 4 different first corner moves in tic tac toe)

- But, we still can’t search a game like chess to the end!

Cutting Off Search

- Replace terminal test (end of game) by cutoff test (don’t search deeper)
- Replace utility function (win/lose/draw) by heuristic evaluation function that estimates results on the best path below this board
  - Like A* search, good evaluation functions mean good results (and vice versa)
- Replace move generator by plausible move generator (don’t consider “dumb” moves)

Alpha-Beta Algorithm

```
function ALPHA-BETA-SEARCH(state, game) returns an action
   action, state ← the a, s in SUCCESSORS[game](state)
         such that MIN-VALUE(s, game, −∞, +∞) is maximized
   return action
```

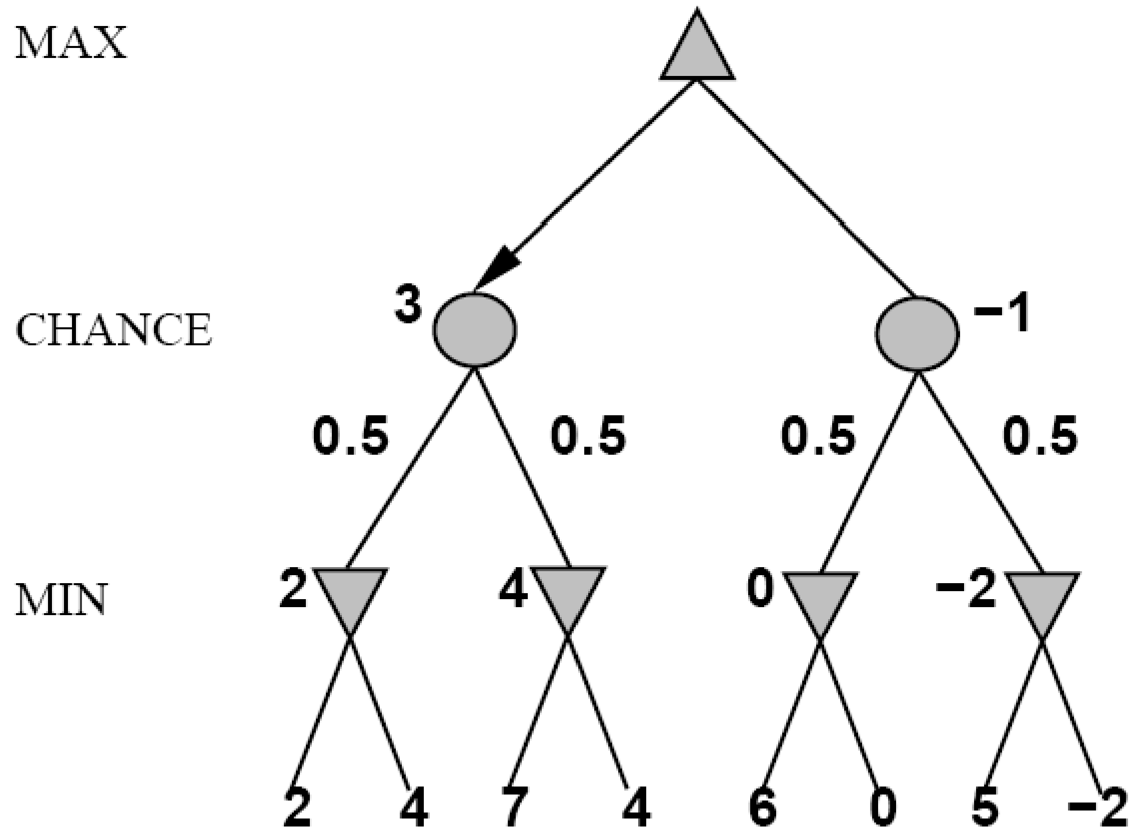
```
function MAX-VALUE(state, game, α, β) returns the minimax value of state
   if CUTOFF-TEST(state) then return EVAL(state)
   for each s in SUCCESSORS(state) do
      α ← max(α, MIN-VALUE(s, game, α, β))
      if α ≥ β then return β
   return α
```

```
function MIN-VALUE(state, game, α, β) returns the minimax value of state
   if CUTOFF-TEST(state) then return EVAL(state)
   for each s in SUCCESSORS(state) do
      β ← min( β, MAX-VALUE(s, game, α, β))
      if β ≤ α then return α
   return β
```

Nondeterministic Games

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

Nondeterministic Games



Algorithm for Nondeterministic Games

- Expectiminimax give perfect play
  - Just like Minimax except it has to handle chance nodes

- if state is a MAX node then
  - return highest Expectiminimax – Value of Successors(state)
- if state is a MIN node then
  - return lowest Expectiminimax – Value of Successors(state)
- if state is a CHANCE node then
  - return average Expectiminimax – Value of Successors(state)

Summary

- 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
- Games are to AI as the Grand Prix is to automobile design