Adversarial Search

CS 486 /686
May 19, 2005
University of Waterloo

Introduction

- So far we have studied environments where there is only a single-agent
- Today we look at what happens if we are in a setting where there are multiple agents planning against each other
  - Game theory: zero sum games

Outline

- Games
- Minimax search
- Evaluation functions
- Alpha-beta pruning
- Coping with chance
- Game programs

Games

- Games are one of the oldest, most well-studied domains in AI
- Why?
  - They are fun
  - Games are usually easy to represent and the rules are clear
  - State spaces can be very large (so more challenging than "toy problems")
    - In chess the search tree has $\sim 10^{154}$ nodes
  - Like the "real world" in that decisions **have** to be made and time is vitally important
  - Easy to determine when a program is doing well
    - i.e. it wins

Types of games

- Perfect vs imperfect information
  - Perfect info means that you can see the entire state of the game
  - Chess, checkers, othello, go,…
  - Imperfect info games include scrabble, poker, most card games
- Deterministic vs stochastic
  - Chess is deterministic
  - Backgammon is stochastic

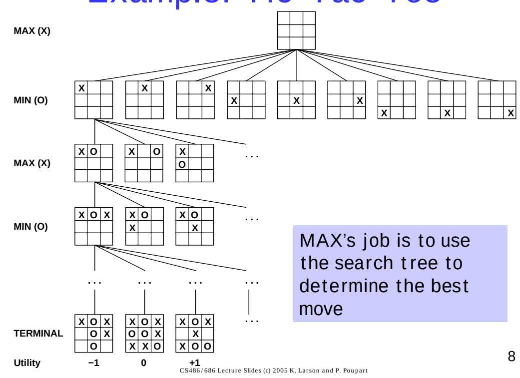
Games as search problems

- Consider a 2-player perfect information game
  - **State:** board configuration plus the player who's turn it is to move
  - **Successor function:** given a state returns a list of (move,state) pairs, indicating a legal move and the resulting board
  - **Terminal state:** states where there is a win/loss/draw
  - **Utility function:** assigns a numerical value to terminal states (e.g. In chess +1 for a win, -1 for a loss, 0 for a draw)
  - **Solution**: a strategy (way of picking moves) that wins the game

Game search challenge

- What makes game search challenging?
  - There is an opponent!
  - The opponent is malicious – it wants to win (i.e. it is trying to make you lose)
  - We need to take this into account when choosing moves
    - Simulate the opponent's behaviour in our search
- Notation: One player is called **MAX** (who wants to maximize its utility) and one player is called **MIN** (who wants to minimize its utility)

Example: Tic-Tac-Toe



MAX's job is to use the search tree to determine the best move

Optimal strategies

- In standard search the optimal solution is a sequence of moves leading to a winning terminal state
- But MIN has something to say about this
- **Strategy** (from MAX's perspective):
  - Specify a move for the initial state, specify a move for all possible states arising from MIN's response, then all possible responses to all of MIN's responses to MAX's previous move.....

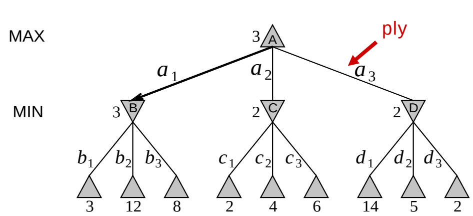
Optimal strategies

- Want to find the optimal strategy
  - One that leads to outcomes at least as good as any other strategy, given that MIN is playing optimally
  - Equilibrium (game theory)
  - Zero-sum games of perfect information are "easy games" from a game theoretic perspective

Minimax Value

MINIMAX-VALUE(n) =

$$\begin{cases} \text{Utility}(n) & \text{if } n \text{ is a terminal state} \\ \text{Max}_{s \in \text{Succ}(n)} \text{ MINIMAX-VALUE}(s) & \text{if } n \text{ is a MAX node} \\ \text{Min}_{s \in \text{Succ}(n)} \text{ MINIMAX-VALUE}(s) & \text{is } n \text{ is a MIN node} \end{cases}$$



Minimax algorithm

```
function MINIMAX-DECISION(state) returns an action
   v ← MAX-VALUE(state)
   return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← −∞
   for a, 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 a, s in SUCCESSORS(state) do
      v ← MIN(v, MAX-VALUE(s))
   return v
```

Returns action corresponding to best possible move

Properties of Minimax

- Complete if tree is finite
- Time complexity: $O(b^m)$
- Space complexity: $O(bm)$ (it is DFS)
- Optimal against an optimal opponent

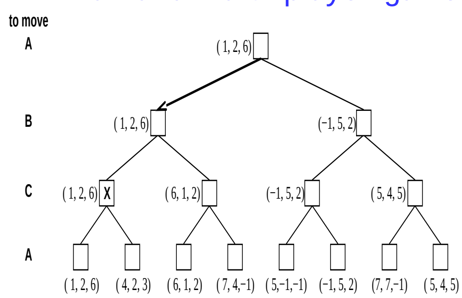
m is depth of the tree

Properties of Minimax

- Complete if tree is finite
- Time complexity: $O(b^m)$
- Space complexity: $O(bm)$ (it is DFS)
- Optimal against an optimal opponent
  - If MIN does not play optimally then we might be able to do better following a different strategy

m is depth of the tree

Minimax and multi-player games



Can not handle alliances, sidepayments....

- Can we now write a program that will play chess reasonably well?

- Can we now write a program that will play chess reasonably well?
  - For chess b~35 and m~100
  - Do we really need to look at all those nodes?

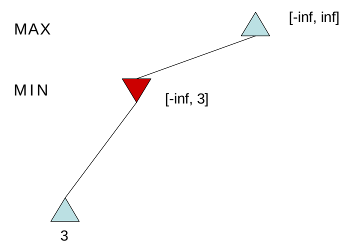
Alpha-Beta Pruning

- No!
  - If we are smart (and lucky) we can do **pruning**
    - Eliminate large parts of the tree from consideration
- Alpha-Beta pruning applied to a minimax tree
  - Returns the same decision as minimax
  - Prunes branches that cannot influence final decision

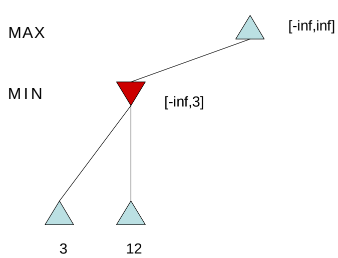
Alpha-Beta Pruning

- Alpha:
  - Value of best (highest value) choice we have found so far on the path for MAX
- Beta:
  - Value of best (lowest value) choice we have found so far on path for MIN
- Update alpha and beta as search continues
- Prune as soon as the value of the current node is known to be worse than current alpha or beta values for MAX or MIN

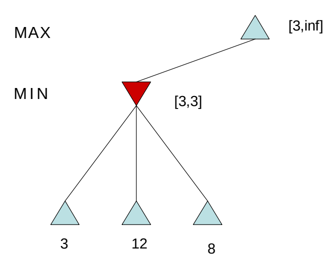
Alpha-Beta example



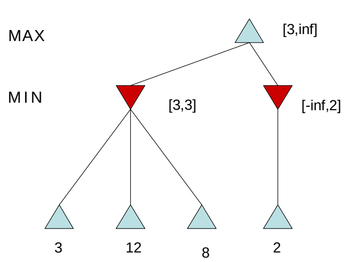
Alpha-Beta example



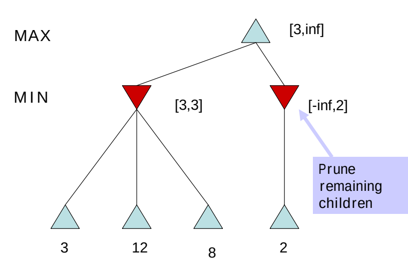
Alpha-Beta example



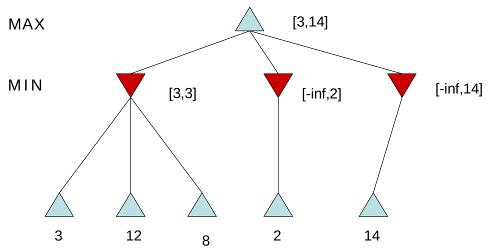
Alpha-Beta example



Alpha-Beta example

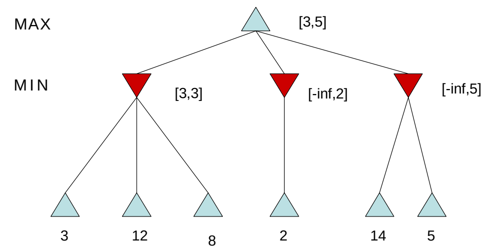


Alpha-Beta example



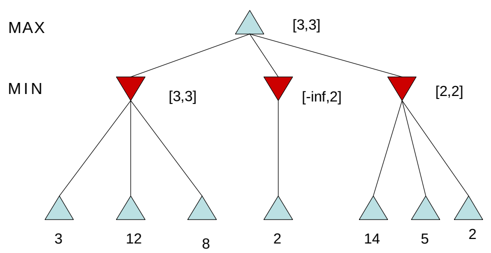
25

Alpha-Beta example



26

Alpha-Beta example



27

Properties of Alpha-Beta

- Pruning does not affect the final result
  - You prune parts of the tree that you would never reach in actual play
- The order in which moves are evaluated are important
  - With bad move ordering will prune nothing
  - With perfect node ordering can reduce time complexity to $O(b^{m/2})$

28

Real-time decisions

- Alpha-beta can be a huge improvement over minimax
  - Still not good enough as we need to search all the way to terminal states for at least part of search space
  - Need to make a decision about a move quickly
- Heuristic evaluation function + cutoff test

29

Evaluation functions

- Apply an evaluation function to a state
  - If terminal state, function returns actual utility
  - If non-terminal, function returns estimate of the expected utility (i.e. the chance of winning from that state)
  - Function must be fast to compute

30

Evaluation functions

- Evaluation functions can be given by the designer of the program (using expert knowledge) or learned from experience
- If features can be judged independently, a weighted linear function is good
  - $w_1f_1(s)+w_2f_2(s)+\ldots+w_nf_n(s)$ with s as board state

31

Cutting off search

- Instead of searching until we find a terminal state, we can cut search sooner and apply the evaluation function
- When?
  - Arbitrarily (but deeper is better)
  - Quiescent states
    - States that are "stable" – not going to change value (by a lot) in the near future
  - Singular extensions
    - Searching deeper when you have a move that is "clearly better" (i.e. moving the king out of check)
    - Can be used to avoid the **horizon effect**

32

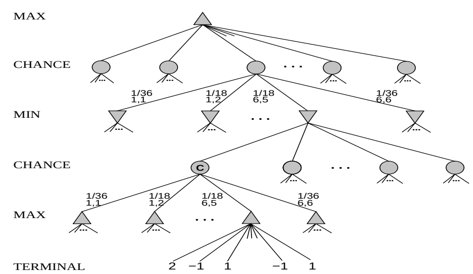
Cutting off search

- How deep do we need to search?
  - Novice chess human player
    - 5-ply (minimax)
  - Master chess human player
    - 10-ply (alpha-beta)
  - Grandmaster chess human player
    - 14-ply + a fantastic evaluation function, opening and endgame databases,..., special purpose hardware would be nice but is no longer really needed (Fritz)

33

Stochastic games

- In games like Backgammon chance plays a roll



34

Stochastic games

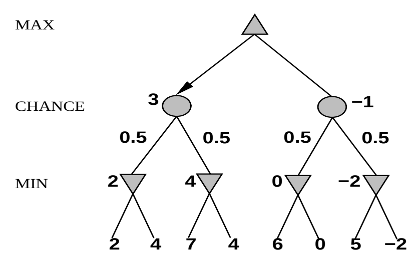
- Need to consider best/worst cases + probability they will occur
- Recall: Expected value of a random variable x

$$E[x]=\sum_{x\in X} P(x)x$$

- Expectiminimax is like minimax but at chance nodes compute the expected value

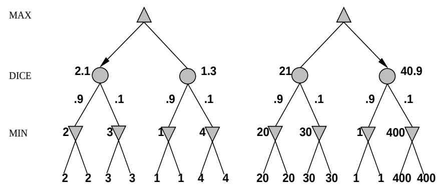
35

Expectiminimax



36

Expectiminimax



WARNING: exact values do matter! Order-preserving transformations of the evaluation function can change the choice of moves. Must have **positive linear transformations** only

Some Game Programs

Checkers: Tinsley vs. Chinook

Mr. Tinsley suffered his 4th and 5th losses **ever** against Chinook

Checkers

- Chinook: http://www.cs.ualberta.ca/~chinook
  - World Man-Machine Checkers Champion
  - Alpha-beta search
  - Opening database
  - Its secret weapon: **Endgame database**
    - Precomputed database of all 444 billion positions with 8 or fewer pieces, each with perfect win/loss/draw info
    - Perfect knowledge into the search
  - Checkers is now dominated by computers

Chess: Kasparov vs. Deep Blue

Kasparov		Deep Blue
5'10"	**Height**	6' 5"
176 lbs	**Weight**	2,400 lbs
34 years	**Age**	4 years
50 billion neurons	**Computers**	32 RISC processors + 256 VLSI chess engines
2 pos/sec	**Speed**	200,000,000 pos/sec
Extensive	**Knowledge**	Primitive
Electrical/chemical	**Power Source**	Electrical
Enormous	**Ego**	None

1997: Deep Blue wins by 3 wins, 1 loss, and 2 draws

Jonathan Schaeffer

Chess

- Its secret:
  - Specialized chess processor + special-purpose memory optimization
  - Very sophisticated evaluation function
    - Expert features and hand-tuned weights
  - Opening and closing books
  - Alpha-beta + improvements (searching up to 40 ply deep!)
  - Search over 200 million positions per second (though lots of these possible moves are silly moves by human standards...)

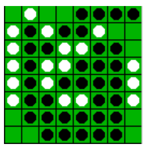
Chess

- There are now programs running on PCs that are on par with human champions
  - Deep Junior vs Kasparov in 2003: 3/3 tie
  - Deep Junior: 8 CPU, 8GB RAM, Windows 2000, 2000000 pos/second
- Is Chess still a human game or have computers conquered it?

Backgammon

- TD-Gammon (Gerry Tesauro at IBM)
- One of the top players in the world
- But only searches two moves ahead!
- Its secret: One amazing evaluation function
  - Neural network trained with reinforcement learning during ~1million games played against itself
  - Humans play backgammon differently now, based on what TD-Gammon learned about the game
  - Very cool AI ☺

Othello: Murakami vs. Logistello



Takeshi Murakami
World Othello Champion

1997: The Logistello software crushed Murakami by 6 games to 0

Jonathan Schaeffer

Othello/Reversi

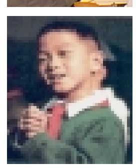
- Logistello (Michael Buro from U of Alberta)
- Human world champion crushed by the program
  - Humans no match for machine
- Its secret: Evaluation function
  - Automatically discovered and tuned knowledge
    - Samples patterns to see if its presence in a position can be correlated with success
    - Tuned 1.5 million parameters using self-play games with feedback

Bridge

- GIB (Matt Ginsberg – U of Oregon)
  - World's first expert level bridge playing program (Finished 12th in human world championship in 1998)
  - Humans are still doing better, but the gap is narrowing quickly
- Its secrets:
  - Does simulations for each decision
    - Deals cards to opponents consistent with available information
    - Chooses action that maximizes expected return
    - Plus other tricks...

Go: Goemate vs. ??

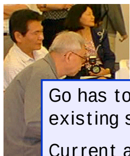
Name: Chen Zhixing
Profession: Retired
Computer skills:
self-taught programmer
Author of Goemate (one of the best Go program available today)

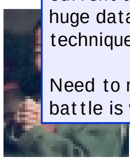


Gave Goemate a 9 stone handicap and still easily beat the program, thereby winning $15,000

Jonathan Schaeffer

Go: Goemate vs. ??





Name: Chen Zhixing
Profession: Retired
Computer skills:

Go has too high a branching factor for existing search techniques (b~100)

Current and future software must rely on huge databases and pattern-recognition techniques

Need to make strategic decisions – Which battle is worth fighting?

Jonathan Schaeffer

Summary

- Games pose lots of fascinating challenges for AI researchers
- Minimax search allows us to play optimally against an optimal opponent
- Alpha-beta pruning allows use to reduce the search space
- A good evaluation function is key to doing well
- Games are fun

Next class

- We will begin reasoning under uncertainty
  - Chapter 13