CS 331: Artificial Intelligence
Adversarial Search II

Outline

1. Evaluation Functions
2. State-of-the-art game playing programs
3. 2 player zero-sum finite stochastic games of perfect information

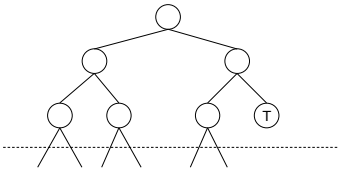
Evaluation Functions

Evaluation Functions

- Minimax and Alpha-Beta require us to search all the way to the terminal states
- What if we can't do this in a reasonable amount of time?
- Cut off search earlier and apply a heuristic evaluation function to states in the search
- Effectively turns non-terminal nodes into terminal leaves

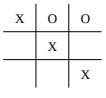
Evaluation Functions

- If at terminal state after cutting off search, return actual utility
- If at non-terminal state after cutting off search, return an estimate of the expected utility of the game from that state

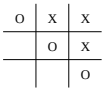


Example: Evaluation Function for Tic-Tac-Toe

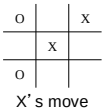
X is the maximizing player



Eval=+100 (for win)

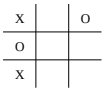


Eval=-100 (for loss)



Eval=1

X's move



Eval=2

X's move

Properties of Good Evaluation Functions

1. Orders the terminal states in the same way as the utility function
2. Computation can't take too long
3. Evaluation function should be strongly correlated with the actual chances of winning

Exact values don't matter. It's the ordering of terminal states that matters. In fact, behavior is preserved under any monotonic transformation of the evaluation function

Properties of Good Evaluation Functions

1. Orders the terminal states in the same way as the utility function
2. Computation can't take too long
3. Evaluation function should be strongly correlated with the actual chances of winning

Even in a deterministic game like chess, the evaluation function introduces uncertainty because of the lack of computational resources (can't see all the way to the terminal state so you have to make a guess as to how good your state is).

Coming up with Evaluation Functions

- Extract features from the game
- For example, what features from a game of chess indicate that a state will likely lead to a win?

Coming up with Evaluation Functions

Weighted linear function:

$$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$$

w_i's are weights

f_i's are features of the game state (e.g. # of pawns in chess)

The weights and features are ways of encoding human knowledge of game strategies into the adversarial search algorithm

Coming up with Evaluation Functions?

- Suppose we use the weighted linear evaluation function for chess. What are two problems with it?
  1. Assumes features are independent
  2. Need to know if you're at the beginning, middle, or end of the game

Alpha-Beta with Eval Functions

Replace:
if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

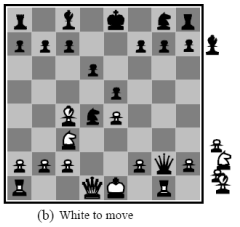
With
if CUTOFF-TEST(*state*,*depth*) **then return** EVAL(*state*)

Also, need to pass *depth* parameter along and need to increment *depth* parameter with each recursive call.

The depth parameter

- CUTOFF-TEST(*state*,*depth*) returns:
  - True for all terminal states
  - True for all *depth* greater than some fixed depth limit *d*
- How to pick *d*?
  - Pick *d* so that agent can decide on move within some time limit

Quiescence Search



(b) White to move

- Suppose the board at the left is at the depth limit
- Black ahead by 2 pawns and a knight
- Heuristic function says Black is doing well
- But it can't see one more move ahead when White takes Black's queen

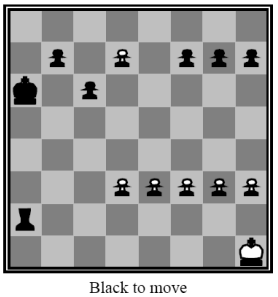
Quiescence Search

- Evaluation function should only be applied to quiescent positions
- i.e. positions that don't exhibit wild swings in value in the near future
- Quiescence search: nonquiescent positions can be expanded further until quiescent positions are reached

Horizon Effect

- Stalling moves push an unavoidable and damaging move by the opponent "over the search horizon" to a place where it cannot be detected
- Agent believes it has avoided the damaging, inevitable move with these stalling moves

Horizon Effect Example



Black to move

Singular Extensions

- Can be used to avoid horizon effect
- Expand only 1 move that is clearly better than all other moves
- Goes beyond normal depth limit because branching factor is 1
- In chess example, if Black's checking moves and White's king moves are clearly better than the alternatives, then singular extension will expand search until it picks up the queening

Another Optimization: Forward Pruning

- Prune moves at a given node immediately
- Dangerous! Might prune away the best move
- Best used in special situations e.g. symmetric or equivalent moves

19

Chess

- Branching factor: 35 on average
- Minimax lookahead about 5 ply
- Humans lookahead about 6-8 plies
- Alpha-Beta lookahead about 10 plies (roughly expert level of play)

If you do all the optimizations discussed so far

20

State-of-the-art Game Playing Programs

21

State of the Art Game Programs

- Checkers (Samuel, Chinook)
- Othello (Logistello)
- Backgammon (Tesauro's TD-gammon)
- Go (AlphaGo – guest lecture Wednesday!)
- Bridge (Bridge Baron, GIB)
- Chess

22

Chess

- Deep Blue – Campbell, Hsu, Hoane
- 1997 – Deep Blue defeats Garry Kasparov in a 6 game exhibition match

23

Chess

- Deep Blue Hardware:
  - Parallel computer with 30 IBM RS/6000 processors running the software search
  - 480 custom VLSI chess processors that performed:
    - Move generation (and move ordering)
    - Hardware search for the last few levels of the tree
    - Evaluation of leaf nodes

24

Chess

- Algorithm:
  - Iterative-deepening alpha-beta search with a transposition table
  - Key to success: generating extensions beyond the depth limit for sufficiently interesting lines of forcing/forced moves
  - Reaches depth 14 routinely, depth 40 in some cases
  - Evaluation function:
    - Had over 8000 features
    - Used an opening of about 4000 positions
    - Database of 700,000 grandmaster games
    - Large endgame database of solved positions (all positions with 5 pieces, many with 6 pieces remaining)

Chess

- So was it the hardware or software that made the difference?
  - Campbell et al. say search extensions and evaluation function were critical
  - But recent algorithmic improvements allow programs running on standard PCs to beat opponents running on massively parallel machines

2 player zero-sum finite stochastic games of perfect information

But First…A Mini-Tutorial on Expected Values

What is probability?

- The relative frequency with which an outcome would be obtained if the process were repeated a large number of times under similar conditions

Example: Probability of rolling a 1 on a fair dice is about 1/6

Expected Values

- Suppose you have an event that can take a finite number of outcomes
  - E.g. Rolling a dice, you can get either 1, 2, 3, 4, 5, 6
- Expected value: What is the average value you should get if you roll a fair dice?

Expected Values

What if your dice isn't fair? Suppose your probabilities are:

Value	Prob
1	0
2	0
3	0
4	0
5	0
6	1

OR

Value	Prob
1	0.5
2	0
3	0
4	0
5	0
6	0.5

OR

Value	Prob
1	0.1
2	0.1
3	0.2
4	0.2
5	0.3
6	0.1

Expected Values

The expected value is a weighted average of the probability of an outcome times the value of that outcome

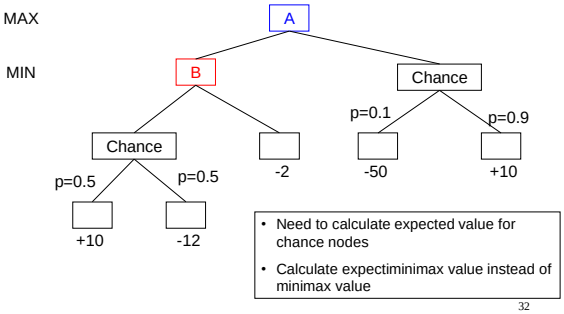
$$\sum \text{Prob(outcome)} * \text{value(outcome)}$$

Value	Prob
1	0.1
2	0.1
3	0.2
4	0.2
5	0.3
6	0.1

Expected Value
= (0.1)(1)+(0.1)(2)+(0.2)(3)+(0.2)(4)+(0.3)(5)+(0.1)(6)
= 0.1 + 0.2 + 0.6 + 0.8 + 1.5 + 0.6
= 3.8

31

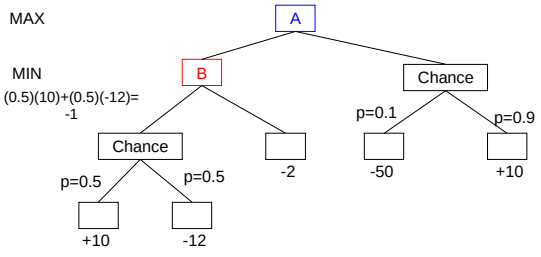
2 player zero-sum finite stochastic games of perfect information



- Need to calculate expected value for chance nodes
- Calculate expectiminimax value instead of minimax value

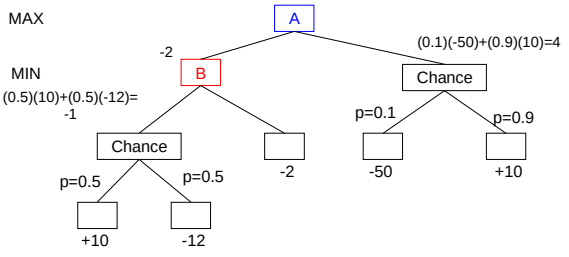
32

2 player zero-sum finite stochastic games of perfect information



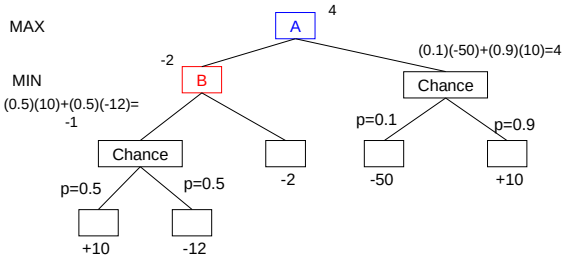
33

2 player zero-sum finite stochastic games of perfect information



34

2 player zero-sum finite stochastic games of perfect information



35

Expectiminimax

$$\text{EXPECTIMINIMAX}(n) = \begin{cases} \text{UTILITY}(n) & \text{If n is a terminal state} \\ \max_{s \in \text{Successors}(n)} \text{EXPECTIMINIMAX}(s) & \text{If n is a MAX node} \\ \min_{s \in \text{Successors}(n)} \text{EXPECTIMINIMAX}(s) & \text{If n is a MIN node} \\ \sum_{s \in \text{Successors(n)}} P(s) \cdot \text{EXPECTIMINIMAX}(s) & \text{If n is a chance node} \end{cases}$$

36

Complexity of Expectiminimax

- Minimax – $O(b^m)$
- Expectiminimax – $O(b^m n^m)$

n = # of possibilities at a chance node (assuming all chance nodes have the same number of possibilities)

Expectiminimax is computationally expensive so you can't look ahead too far! The uncertainty due to randomness accounts for the expense.

37

What you should know

- What evaluation functions are
- Problems with them like quiescence, horizon effect
- How to calculate the expectiminimax value of a node

38