Artificial Intelligence Chapter 5: Adversarial Search



Pop Quiz



• White to mate...

Pop Quiz



• White to mate...in 292 moves!

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
- In AI, the most common games are deterministic, turntaking, two-player, zero-sum games of perfect information (i.e. fully observable environments).

Types of Games

	Deterministic	<u>Chance</u>
Perfect Information	Chess, checkers, go, othello	Backgammon, monopoly
Imperfect Information		<i>Bridge, poker, scabble, nuclear war</i>

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

Formal Elements of Games

- S_o: Initial state, specifies how game is set up to start.
- PLAYER(s): Defines which player has the move in a state.
- ACTION(s): Returns the set of legal moves in a state.
- RESULT(s,a): The transition mode, which defines the result of a move.
- TERMINAL-TEST(s):A terminal test, which is true when the game is over and false otherwise.

Formal Elements of Games

- UTILITY(s,p): A **utility function** (also called a payoff function) that defines the final numeric value for a game that ends in terminal state *s* for player *p*.
- Some games have a winder variety of possible outcomes; the payoffs in backgammon range from 0 to +192.
- A zero-sum game is defined as one where the total payoff to all players is the same for every instance of the game.
- Chess is zero-sum because every game has payoff 0+1, 1+0, or ¹/₂ + ¹/₂.

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, $state \leftarrow the \ a, \ s \ in \ Successors(s_{tate})$ 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) (m is the maximum tree depth, b is the number of legal moves at each point)
- Space
 - O(bm) depth first exploration of the state space

Resource Limits

- Suppose there are 100 seconds, explore 10⁴ nodes / second
- 10⁶ nodes per move
- Standard approach
 - Cutoff test depth limit
 - quiesence search values that do not seem to change
 - Change the evaluation function

- In 1950, Shannon proposed that programs should cut off the search earlier to apply a heuristic **evaluation function** to states in the search.
- This technique effectively turns non-terminal nodes into terminal leaves.
- The idea is to replace the utility function by a heuristic evaluation function (EVAL), which estimates the position's utility, and replace the terminal test by a **cutoff test** that decides when to apply EVAL.
- An evaluation function returns an estimate of the expected utility of the game from a give position, just as heuristic functions return an estimate of the distance to the goal.

- The performance of a game-playing program depends strongly on the quality of its evaluation function – but how to design?
- The evaluation function should order the terminal states in the same way as the true utility function: states that are wins must evaluate better than draws, etc.
- (2) The computation must not take too long!
- (3) For nonterminal states, the evaluation function should be strongly correlated with the actual "chances of winning."

- Note that if the search must be cut off at nonterminal states, then the algorithm will necessarily be *uncertain* about the final outcomes of those states.
- Usually, an evaluation function calculates various features of a state (e.g. number of pawns, etc.).
- The features define various categories or equivalence classes of states: the states in each category have the same values for all features.
- For example: suppose experience suggests that 72% of states encountered in a two-pawn vs. one-pawn category lead to a win (utility: +1); 20% to a loss (0) and 8% to a draw (1/2).
- Expected value: $(0.72 \text{ x } 1) + (0.20 \text{ x } 0) + (0.08 \text{ x } \frac{1}{2}) = 0.76$.

- Most evaluation functions compute separate numerical contributions from each feature and then combine them to find the total value.
 - Typical evaluation function is a linear sum of features
 - $\text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$
 - $w_1 = 9$
 - f₁(s) = number of white queens) number of black queens
 - etc.

Game tree

From M. T. Jones, Artificial Intelligence: A Systems Approach



Game tree

From M. T. Jones, Artificial Intelligence: A Systems Approach



From M. T. Jones, Artificial Intelligence: A Systems Approach



Evaluation function f(n) measures "goodness" of board configuration n. Assumed to be better estimate as search is deepened (i.e., at lower levels of game tree).

Evaluation function here: "Number of possible wins (rows, columns, diagonals) not blocked by opponent, minus number of possible wins for opponent not blocked by current player."



Minimax search: Expand the game tree by *m* ply (levels in game tree) in a limited depth-first search. Then apply evaluation function at lowest level, and propagate results back up the tree.



...



. . .



. . .



. . .

 \leq



. . .

 \leq

Minimax algorithm: Example

From M. T. Jones, Artificial Intelligence: A Systems Approach







What is value at the root?

- The problem with minimax search is that the number of games states it has to examine is exponential in the depth of the tree.
- We can't eliminate the exponent however, we can effectively cut it in half!
- The key idea is that it is possible to compute the correct minimax decision without looking at every node in the game true.
- We "prune" away branches in the game tree that cannot possibly influence the final decision.

- 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 = 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











Algorithm: Minimax with Alpha-Beta Pruning

```
function alphabeta(node, depth, \alpha, \beta, Player)
•
            if depth = 0 or node is a terminal node
                        return the heuristic value of node
            if Player = MaxPlayer
                        for each child of node
                          \alpha := \max(\alpha, alphabeta(child, depth-1, \alpha, \beta, not(Player)))
                          if \beta \leq \alpha
                            break ; Prune
                        return α
            else
                        for each child of node
                          \beta := \min(\beta, alphabeta(child, depth-1, \alpha, \beta, not(Player)))
                          if \beta \leq \alpha break ; Prune
                        return β
    end
```

- ; Initial call
- alphabeta(origin, depth, -infinity, +infinity, MaxPlayer)

From M. T. Jones, Artificial Intelligence: A Systems Approach



From M. T. Jones, Artificial Intelligence: A Systems Approach

alpha = value of the best possible move you can make, that you have computed so far

beta = value of the best possible move your opponent can make, that you have computed so far

If at any time, **alpha >= beta**, then your opponent's best move can force a worse position than your best move so far, and so there is no need to further evaluate this move



Alpha-beta pruning exercise



(a) What is value at the root, using minimax alone?

(b) What nodes could have been pruned from the search using alpha-beta pruning? Show values of alpha and beta

Alpha-beta pruning exercise



(a) What is value at the root, using minimax alone?

(b) What nodes could have been pruned from the search using alpha-beta pruning?

- Alpha-beta pruning effectiveness is highly dependent on move ordering.
- Under optimal ordering conditions, alpha-beta pruning needs to examine O(b^{m/2}) nodes to pick the best move, instead of O(b^m) for minimax.
- This means that the effective branching factor becomes root(b) instead of b – for chess about 6 instead of 35.

- Put another way, alpha-beta can solve a tree roughly twice as deep as minimax in the same amount of time.
- If successors are examined in random order rather than best-first, the total number of nodes examined will be roughly O(b^{3m/4}) for moderate b.

- Adding dynamic move-ordering schemes, such as trying the first moves that were found to be best in the past, brings the search effectiveness closer to the theoretical limit.
- In this way, the "past" could be the previous move, or it could come from previous exploration of the current move.
- One way to gain such valuable information is with iterative deepening search. (First search 1 ply and record path of best moves, then 1 ply deeper, etc.)

- As we saw in Chapter 3, iterative deepening on an exponential game tree adds only a constant fraction to the total search time, which can be more than made up from better move ordering.
- The best moves are often called **killer moves** and to try them first is called killer move heuristic.
- To avoid repeating states we can use a hash table of previously seen positions (called a transposition table like the explored list in GRAPH-SEARCH).

A Few More Notes on Alpha-Beta

- We assume (typically) that opponent is rational and both players have perfect information at each evaluation.
- 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)

Cutting Off Search

- A more sophisticated type of cut off search applied evaluation functions only to positions that are **quiescent** (meaning: unlikely to exhibit wild swings in value in the near future).
- Non-quiescent positions can be expanded further until quiescent positions are reached. This extra search is called a **quiescent search**; sometimes it is restricted to consider only certain types of moves (e.g. capture moves).
- To mitigate the **horizon effect**, one can use **singular extension** (apply move that is clearly better than all others at a given position).

Stochastic Games

• In nondeterministic games, chance is introduced by dice, card shuffling

• How do we make correct decisions for stochastic games? We calculate expected value.

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)

- Arthur Samuel's checkers program, written in the 1950's.
- In 1962, running on an IBM 7094, the machine defeated R. W. Nealy, a future Connecticut state checkers champion.
- One of the first machine learning programs, introducing a number of different learning techniques.



Rote Learning

- When a minimax value is computed for a position, that position is stored along with its value.
- If the same position is encountered again, the value can simply be returned.
- Due to memory constraints, all the generated board positions cannot be stored, and Samuel used a set of criteria for determining which positions to actually store.

Learning the evaluation function

• Comparing the static evaluation of a node with the backed-up minimax value from a lookahead search.

• If the heuristic evaluation function were perfect, the static value of a node would be equal to the backed-up value based on a lookahead search applying the same evaluation on the frontier nodes.

• If there's a difference between the values, the evaluation the heuristic function should be modified.

Selecting terms

• Samuel's program could select which terms to actually use, from a library of possible terms.

• In addition to material, these terms attempted to measure following board features :

center control

advancement of the pieces

mobility

• The program computes the correlation between the values of these different features and the overall evaluation score. If the correlation of a particular feature dropped below a certain level, the feature was replaced by another.

Deep Blue

http://www.dave-reed.com/csc550.S04/Presentations/chess/chess.ppt

- First Created in 1997
- Algorithm:
 - iterative-deepening alpha-beta search, transposition table, databases including openings of grandmaster games (700,000), and endgames (all with 5 pieces or more pieces remaining)

Hardware:

- 30 IBM RS/6000 processors
 - They do: high level decision making
- 480 custom chess processors
 - all running in parallel
 - They do :
 - deep searches into the trees
 - move generation and ordering,
 - position evaluation (over 8000 evaluation points)
- Average performance:
 - 126 million nodes/sec., 30 billion position/move generated, search depth: 14

From "Deep Blue (chess computer)"

http://en.wikipedia.org/wiki/Deep_Blue_(chess_computer)

On May 11, 1997, the machine won a six-game match by two wins to one with three draws against world champion Garry Kasparov.[1] Kasparov accused IBM of cheating and demanded a rematch, but IBM refused and dismantled Deep Blue.[2] Kasparov had beaten a previous version of Deep Blue in 1996.

After the loss, Kasparov said that he sometimes saw deep intelligence and creativity in the machine's moves, suggesting that during the second game, human chess players had intervened on behalf of the machine, which would be a violation of the rules. IBM denied that it cheated, saying the only human intervention occurred between games.

A decade later: Topalov vs. Kramnik

http://en.wikipedia.org/wiki/Veselin_Topalov

On 14 December 2006, Topalov directly accused Kramnik of using computer assistance [from the *Fritz* chess computer] in their World Championship match.

On 14 February 2007, Topalov's manager released pictures, purporting to show cables in the ceiling of a toilet used by Kramnik during the World Championship match in Elista.

Summary

- A game can be defined by the **initial state**, the legal **actions** in each state, the **result** of each action, a **terminal test**, and a **utility function** that applies to terminal states.
- In two-player, zero-sum games with **perfect information**, the **minimax** algorithm can select optimal moves by a depth-first enumeration of the game tree.
- The **alpha-beta** search algorithm computes the same optimal move as minimax, but achieves much greater efficiency by eliminating subtrees that are probably irrelevant.

Summary

- Usually, it is not feasible to consider the whole game tree (even with alpha-beta), so we need to cut the search off at some point and apply a heuristic evaluation function that estimates the utility of a state.
- Games of chance can be handled by extension to the minimax algorithm that evaluates a **chance node** by taking the average utility of all its children, weighted by the probability of each child.