# Adversarial Search

05 November 2020 15:22

## Objectives:

- Adversarial Search
- Minimax
- Alpha-Beta Pruning
- Imperfect Decisions
- Games with Chances

A competitive multi-agent environment where goals are in conflict – gives rise to games

Other agents – opponents – which introduce uncertainty

- An adversarial search agent must deal with contingency
- High complexity and time sensitive – typically have to make a best guess based on experience and time available
- Chess has a branching factor of 35 and a game is 100 moves – thus we have 35^100 nodes – we have to make the best move given the situation

## Uncertainty

- From the opponent trying to make the best move for themselves
- Randomness – e.g. throwing a dice
- Insufficient time to determine consequences

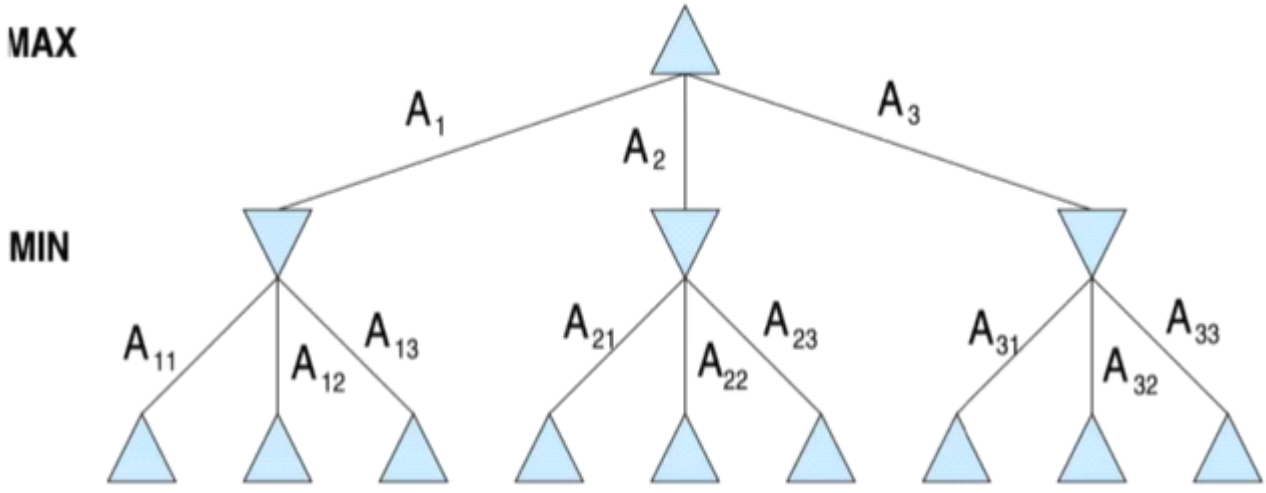
## Formal View of a Game

- Initial state – the board position and player to move
- Set of operators defining legal moves and resulting states
- A terminal test to determine a terminal state
- A utility function or payoff function – gives a numeric value for terminal states e.g. +1, -1, 0 for chess win lose draw

We can build a game tree based on initial state and operators for each player

## Ply

A single move in a 2 player game is takes 2 half-moves, or is "2 ply"



# Minimax

In the case where two agents are competing so that a positive reward for one is a negative reward for the other, we have a two-agent zero-sum game. The value of such a game can be characterized by a single number that one agent is trying to maximise and the other agent is trying to minimize. Having a single value fora  two-agent zero sum game leads to a minimax strategy. Each node is either a max node, if it is controlled by the agent trying to maximise, or is a min node if it is controlled by the agent trying to minimize.

- Gives and optimal strategy for max
- Choose move with the highest minimax value

The minimax value of a state is the utility (for Max) of being in that state assuming both players play optimally from that state until the end of game

Obtains best achievable payoff against best play

## Algorithm

- Generate a complete game tree
- Use utility function to rate terminal states
- Use utility of terminal states to give utility of nodes one level up
- Continue backing up tree until reaching the root
- Max should choose move that leads to highest utility

This is the minimax decision:

**Maximises utility under the assumption that the opponent will play to minimise it**

- Complete if tree is finite
- Optimal against an optimal opponent
- Space - O(bd) because DFS
- Time – O(b^d) - a killer for real games
- Minimax requires a complete search tree which is not practical
- Forms basis for realistic algorithms

## Multi-player Minimax

Can extend minimax to multiple players by using vectors of utilities

# Alpha-Beta Pruning

Complete search tree is impractical – alternative is to prune branches that will not influence decision

- Consider a node n that might be chosen
- If a better choice exists, then n will never be reached so prune it
- As soon as we discover a better choice than n, prune it

Minimax is a DFS, and ABP gets its name from the parameters backed up the path

- Alpha = value of best choice along the path for Max (highest utility)
- Beta = value of best choice for min (lowest utility for Max)

ABP updates alpha and beta as it searches, pruning as soon as value of current node is known to be worse than the current alpha or beta for max or min respectively
Pruning is done by terminating the recursive call

## Order of Examining Successors

- The effectiveness of ABP is dependent on order of examining successors
- So, we should try to examine the best successors first
- If we could do this, the ABP looks at O(b^(d/2)) instead of O(b^d) for minimax – roughly twice the lookahead
- For random order successors, APB looks at O(b^(3d/4) nodes
-

In practice, a simple ordering function can give significant advantage
e.g. for chess we might look at capture moves first then threats, then forward moves, then backward moves

## Imperfect Decisions

APB prunes much of the search tree, while minimax needs the complete tree
But APB still needs to search to the terminal states for some of the tree which is still impractical

Alternative: cut off the tree earlier, using:

- A heuristic evaluation function to get a value for states
- A cut-off test to determine when to stop going down the tree

Evaluation function gives an estimate for expected utility for a given position

Cutting off tree turns nonterminal nodes into terminal leaves.

Eval function should:

- Order terminal states as per utility function
- Approximate actual utility state

Uncertainty is unavoidable since we are not considering a complete tree

- Most evaluation functions calculate features of a state e.g. number of each piece
- These give equivalence classes for states which will lead to a win, draw or less with some probability
- Combine features with a weighted linear function
- Assumes features are independent

### Cutting Off Search

Simplest approach is to set a fixed depth – cut-off test succeeds at depth d
More robust approach is to use an iterative deepening, continue until out of time, then return the best move found so far
Both of these approaches are unreliable

Solution: only apply eval function to quiescent positions – those whose value is unlikely to change significantly in the near future

- Non-quiescent positions expanded until quiescent positions reached
- This extra search is called quiescent search
- Quiescent search restricted to certain types of more to quickly resolve uncertainties in position
- Horizon problem: faced with unavoidable damaging move from opponent, a fixed depth search is fooled into viewing stalling moves as avoidance

Singular extension search as a means of avoiding horizon problem
Singular extension is a move that is clearly better than all others
In chess, can search to see whether opponent can advance pawn to 8$^{th}$ row, turning it into a queen

Forward pruning: immediately prune some moves from a node with no further considerations

- Only safe in special cases like when the two moves are symmetric or equivalent and only

consider on of them, or nodes are very deep in search tree

## Games with Chance

- Many games contain chance
- Legal moves are dependent on roll of dice, so cannot construct a complete game tree
- Have to include chance nodes to solve this
- These nodes can be labelled with the possibility of the expected value taken over the chance nodes