

Section 3: Games and CSPs

CS 182 - Artificial Intelligence

Recall the formalization of a **Deterministic Game** from lecture:

- States: S (with start state S_0)
- Actions: A (may depend on player/state)
- Transition Function: $R : S \times A \rightarrow S$
- Players: $P = \{1 \dots N\}$ (usually take turns)
- Terminal Test: $T : S \rightarrow \{True, False\}$
- Terminal Utilities: $U : S \times P \rightarrow R$

And that a solution for a player is a policy $\Pi : S \rightarrow A$.

Note that the difference between a game and a search problem is that introduction of multiple players and their policies and a goal test / terminal utility instead of a cost of actions and a goal state set.

Minimax: Recall that in games, every state has a value V . For terminal states, $V(s)$ is known. In single-player games, you can calculate the value for non-terminal states as

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

Now imagine a two-player, zero-sum, turn-based game like Tic Tac Toe. We still know $V(s)$ for terminal states, but now we have to anticipate the opponents' turn. In the minimax algorithm, we assume that the opponent plays rationally and tries to minimize your reward (thus maximizing theirs). Therefore, for your opponents' turns, we calculate the value of a state as

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

Therefore, with two rational agents the optimal strategy is to maximize your moves and assume your opponent will do the same (thus minimizing your utility on their turn), thus minimax. Of course if our opponent isn't optimal, then we can still use this type of procedure if we consider the opponent's move in expectation, thus expectimax!

Alpha-Beta Pruning is an extension of Minimax which increases efficiency by reducing the number of nodes searched. While we traverse the tree, we keep track of two values, α , and β . α is MAX's best solution on the path to root, β is MIN's best option on the path to root. We then follow the procedure below allowing us to eliminate the need to search over sections of the tree to which we know will not be a part of the optimal solution.

```
def max-value(state,  $\alpha$ ,  $\beta$ ):  
    initialize  $v = -\infty$   
    for each successor of state:  
         $v = \max(v, \text{value}(\text{successor}, \alpha, \beta))$   
        if  $v \geq \beta$  return  $v$   
         $\alpha = \max(\alpha, v)$   
    return  $v$ 
```

```
def min-value(state,  $\alpha$ ,  $\beta$ ):  
    initialize  $v = +\infty$   
    for each successor of state:  
         $v = \min(v, \text{value}(\text{successor}, \alpha, \beta))$   
        if  $v \leq \alpha$  return  $v$   
         $\beta = \min(\beta, v)$   
    return  $v$ 
```

Practice Problems

1. Describe Tic Tac Toe as a deterministic game. How many states exist?

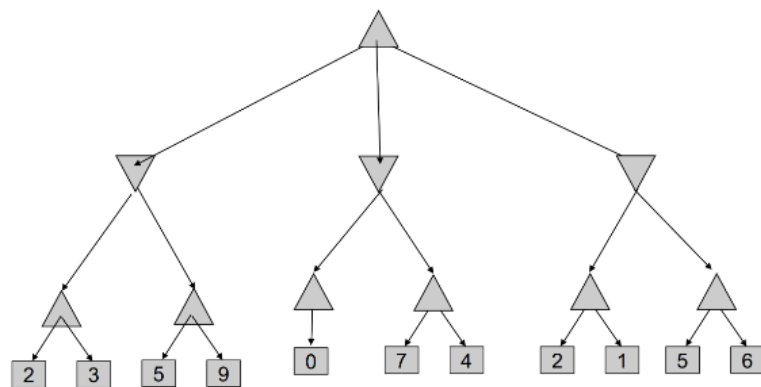
2. Draw the search tree including the terminal utilities for the board below. When drawing the tree, you can assume that X plays perfectly and that you know X 's response to any play in advance. Therefore, you only need to expand O 's options in the tree.

		X
		X
O	O	X

3. What move would O play? How would you change the utility function for O to play something different?
4. Under what assumptions about player 2 should player 1 use minimax rather than expectimax search to select a move? Under what assumptions about player 2 should player 1 use expectimax rather than minimax search?

5. Imagine that player 1 wishes to act optimally (rationally), and player 1 knows that player 2 also intends to act optimally. However, player 1 also knows that player 2 (mistakenly) believes that player 1 is moving uniformly at random rather than optimally. How should player 1 use this knowledge to select a move?

6. Conduct alpha-beta pruning for the following minimax tree.



CSPs

CSPs are formalized as a triple $\langle X, D, C \rangle$.

1. $X = \{X_1, \dots, X_n\}$: A set of variables in the problem
2. $D = \{D_1, \dots, D_n\}$: The domains for these variables
3. $C = \{C_1, \dots, C_m\}$: Constraints

Constraints encode the limits on the values/domains for each variable contingent on the values/domains of other variables.

One way to solve a CSP is through **backtracking search** which is simply DFS with two changes:

1. Fix the variable ordering ($X_1 = D_1 \rightarrow X_2 = D_2 \equiv X_2 = D_2 \rightarrow X_1 = D_1$).
2. Check constraints as you go and backtrack if you violate any.

Tasks

Assume there are 5 CS classes that meet on MWF:

1. CS50 - Intro to CS: Meets from 8:00-9:00am
2. CS124 - Algorithms: Meets from 9:00-10:00am
3. CS181 - Machine Learning: Meets from 10:30-11:30am
4. CS182 - Artificial Intelligence: Meets from 8:30-9:30am
5. CS187 - Natural Language Processing: Meets from 9:00-10:00am

and 3 professors who will be teaching these classes:

1. Professor A, who can teach CS50, CS181, and CS182.
2. Professor B, who can teach CS124, CS187, and CS181.
3. Professor C, who can teach CS50, CS124, and CS187.

You are in charge of scheduling and are constrained by the fact that each professor can only teach one class at a time (Problem Adapted from Berkeley material).

1. Formulate this problem as a CSP problem in which there is one variable per class, stating the domains, and constraints. Constraints should be specified formally and precisely, but may be implicit rather than explicit.

2. Draw the constraint graph associated with your CSP

3. Describe the standard backtracking search strategy to solve this problem. Write down a possible solution that you found using the strategy.

4. Explain why it is a good heuristic to choose the variable that is most constrained but the value that is least constraining in a CSP search