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: $R : S \times A \to S$
- Players: $P = \{1 \dots N\}$ (usually take turns)
- Terminal Test: $T : S \to \{True, False\}$
- Terminal Utilities: $U : S \times P \to R$

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

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 anticipates 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, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

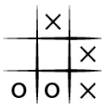
```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

Practice Problems

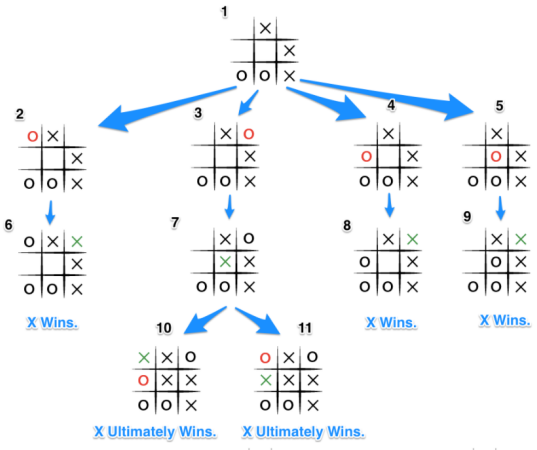
1. Describe Tic Tac Toe as a deterministic game. How many states exist?

   - S: A state in this problem describes the current board state. Every field can have any of three values: *empty*, X, and O. If we represent them using the numbers $0, 1, 2$, we can represent a board as an array of length 9, e.g. $[0, 0, 1, 0, 0, 2, 0, 0, 0]$. Using this representation, there are $3^9 = 19683$ possible states. Side note: This representation allows many illegal states (game is already over, too many X or O, etc). This is not a problem for search algorithms because they will never expand those illegal states. However, for an accurate calculation of possible legal states, you can look here: `http://brianshourd.com/posts/2012-11-06-tilt-number-of-tic-tac-toe-boards.html`
   - A: All legal actions change an empty field to either a X or O depending on the current player. A valid representation would be to encode this as a tuple (i, p), where i is the index of the empty field, and p is the current player.
   - Transition Function: R: The transition function takes the state and an action and returns the new state by switching the active player and adding an X or O into the appropriate empty space on the board.
   - P: Tic Tac Toe is a two-player game, so this is $\{0, 1\}$
   - Terminal Test: T: The test checks whether there are any three X or O in a diagonal, row, or column. As a simplification, the test only has to check the symbol for the player who most recently played their turn.
   - Terminal Utilities: U: The winning player gets $+1$, the losing player -1, a tie is worth 0.

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.

   the terminal utilities are all -1. No matter what O does, X will win.[1]

3. What move would O play? How would you change the utility function for O to play something different?

   The move that O plays would only depend on your implementation of this algorithm, since all the actions have the same value to O. A possible adjustment is to change the utility function to take the depth of the tree into account

[1] Pictures taken from `http://neverstopbuilding.com/minimax`

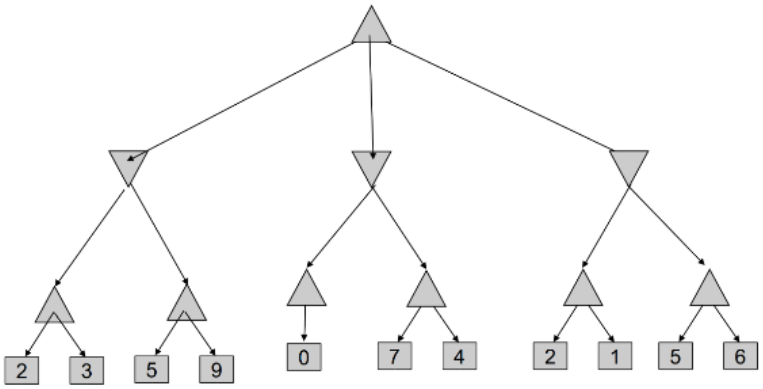
4. Under what assumptions about player 2 should player 1 use minimax rather than expectimax search to select a move? Under what assumptions about player 2 should player 1 use expectimax rather than minimax search?

   Player 1 should use minimax if they expect player 2 to move optimally. If player 1 expects player 2 to move randomly, they should use expectimax to maximize their expected utility.

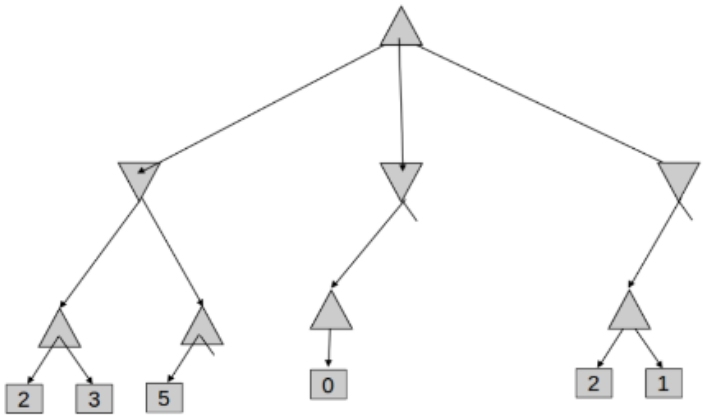
5. Imagine that player 1 wishes to act optimally (rationally), and player 1 knows that player 2 also intends to act optimally. However, player 1 also knows that player 2 (mistakenly) believes that player 1 is moving uniformly at random rather than optimally. How should player 1 use this knowledge to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

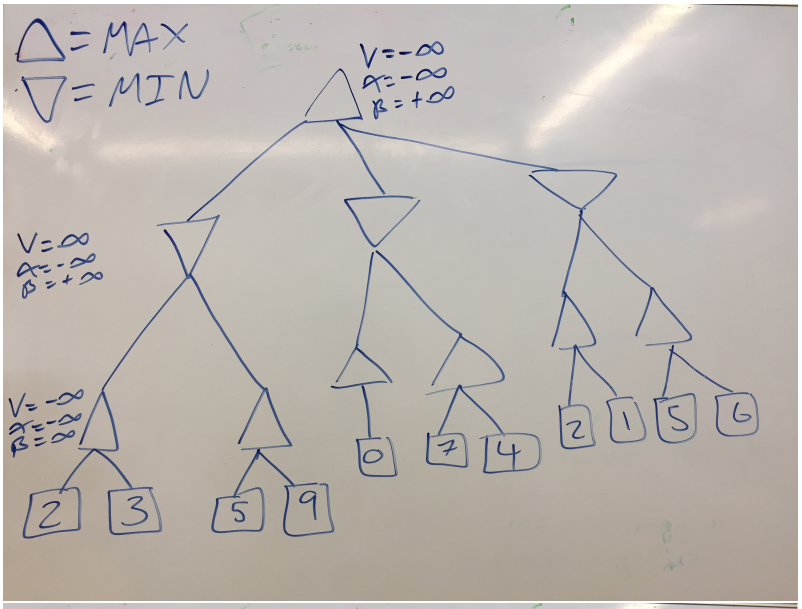
   This problem can be solved by adding a second game tree. The new game tree is used to anticipate player 2's action. Here, all nodes for player 1 are replaced by chance nodes and player 1 can use expectimax to find player 2's policy. In the original game tree, player 1 can prune all actions by player 2 but the one resulting from expectimax.

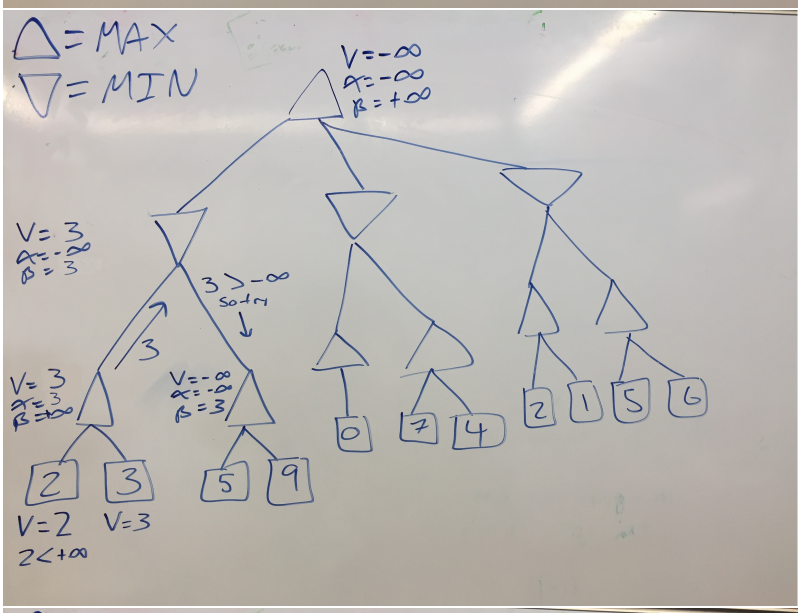
6. Conduct alpha-beta pruning for the following minimax tree.

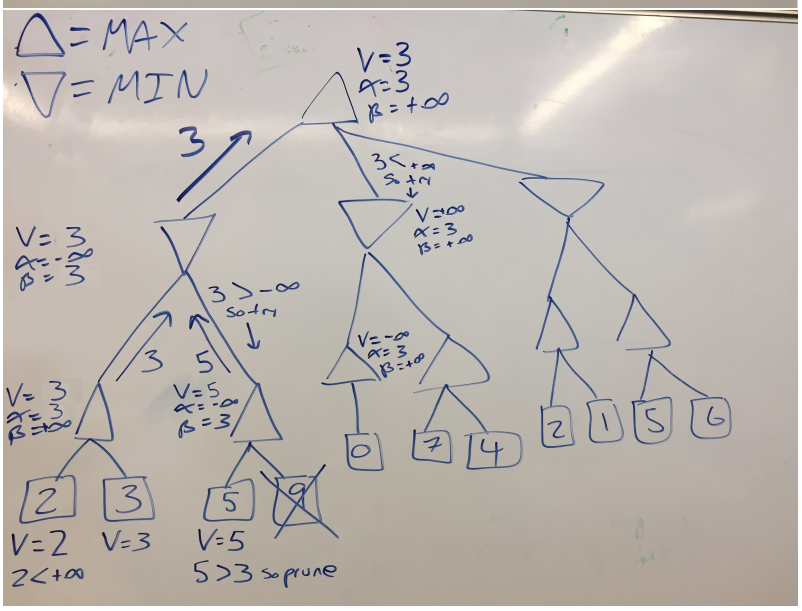


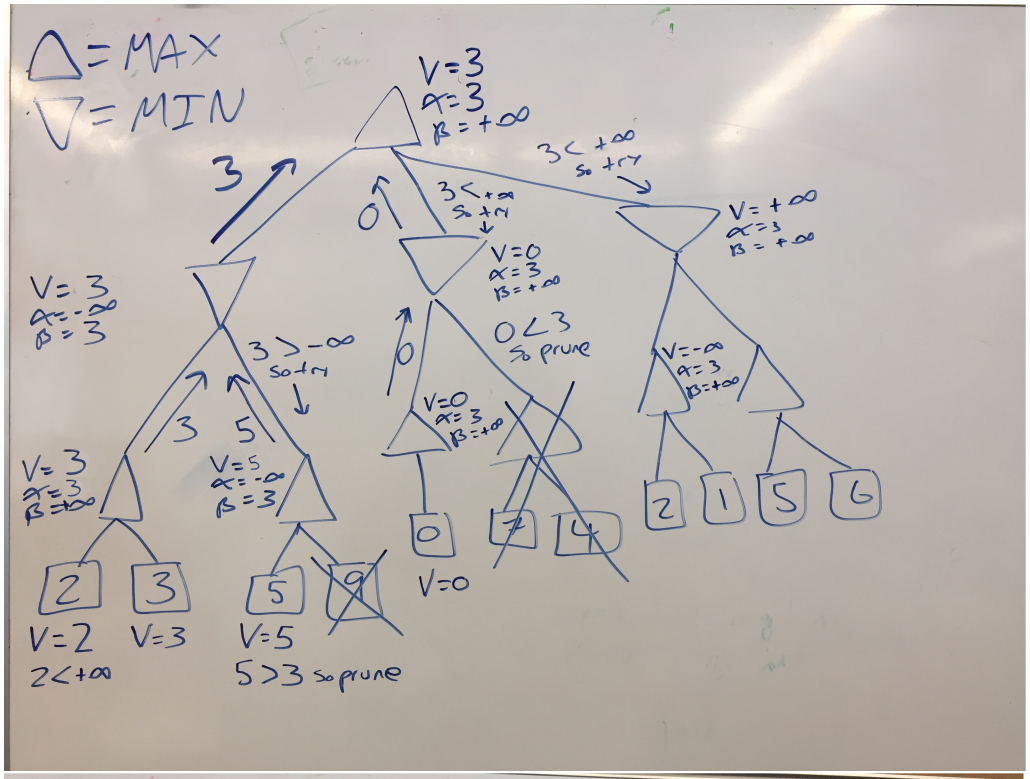
   The following pictures show the final solution and then the steps to computing the solution. If you want to practice your pruning skills, we recommend the website: `http://inst.eecs.berkeley.edu/~cs61b/fa14/ta-materials/apps/ab_tree_practice/`

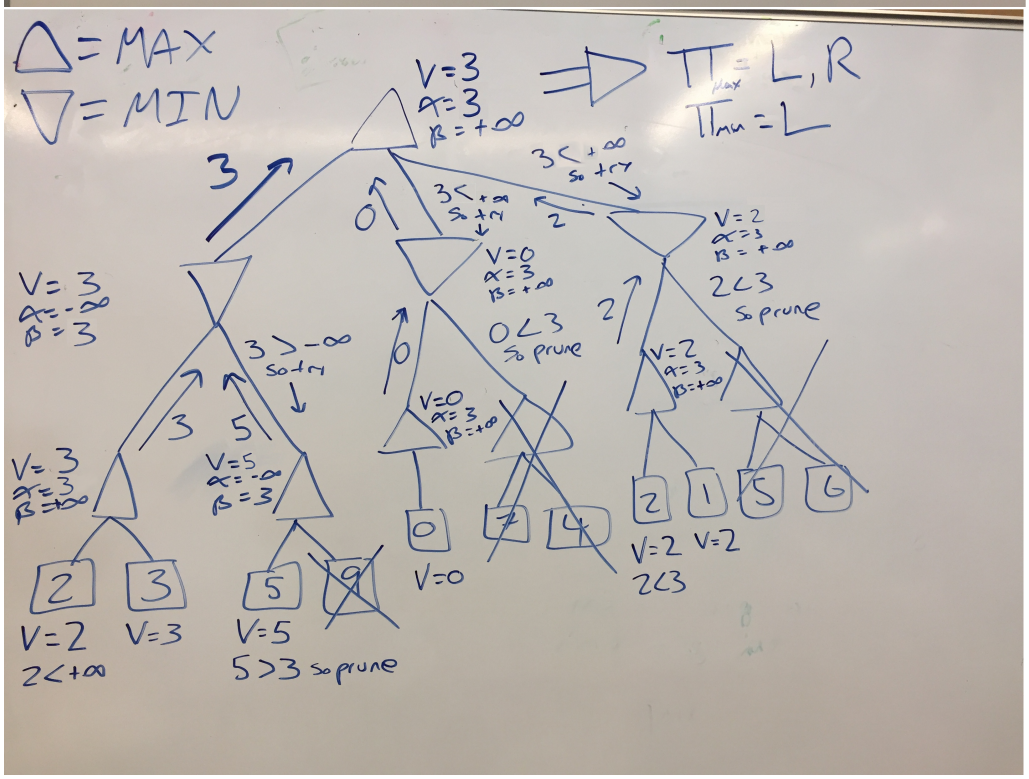



△ = MAX
▽ = MIN
V = -∞
α = -∞
β = +∞
V = ∞
α = -∞
β = +∞
V = -∞
α = -∞
β = ∞
2
3
5
9
0
7
4
2
1
5
6


△ = MAX
▽ = MIN
V = -∞
α = -∞
β = +∞
V = 3
α = -∞
β = 3
3 > -∞
so + ∞
3
V = 3
α = 3
β = +∞
V = -∞
α = -∞
β = 3
2
3
5
9
V = 2
2 < +∞
V = 3
0
7
4
2
1
5
6


△ = MAX
▽ = MIN
V = 3
α = 3
β = +∞
3
3 < +∞
so + ∞
V = ∞
α = 3
β = +∞
V = 3
α = -∞
β = 3
3 > -∞
so + ∞
3
5
V = 3
α = 3
β = +∞
V = 5
α = -∞
β = 3
V = -∞
α = 3
β = +∞
2
3
5
9
V = 2
2 < +∞
V = 3
V = 5
5 > 3 so prune
0
7
4
2
1
5
6


△ = MAX
▽ = MIN
V=3
α=3
β=+∞
3
3<+∞ so try
V=+∞
α=3
β=+∞
0
V=0
α=3
β=+∞
0<3 so prune
V=3
α=-∞
β=3
3>-∞ so try
3
5
V=-∞
α=3
β=+∞
V=3
α=3
β=+∞
V=5
α=-∞
β=3
2
3
5
9
0
7
4
2
1
5
6
V=2
2<+∞
V=3
V=5
5>3 so prune
V=0


△ = MAX
▽ = MIN
V=3
α=3
β=+∞
Π_max = L, R
Π_min = L
3
3<+∞ so try
2
V=2
α=3
β=+∞
2<3 so prune
0
V=0
α=3
β=+∞
0<3 so prune
2
V=3
α=-∞
β=3
3>-∞ so try
3
5
V=2
α=3
β=+∞
V=3
α=3
β=+∞
V=5
α=-∞
β=3
2
3
5
9
0
7
4
2
1
5
6
V=2
2<+∞
V=3
V=5
5>3 so prune
V=0
V=2
2<3

CSPs

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

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

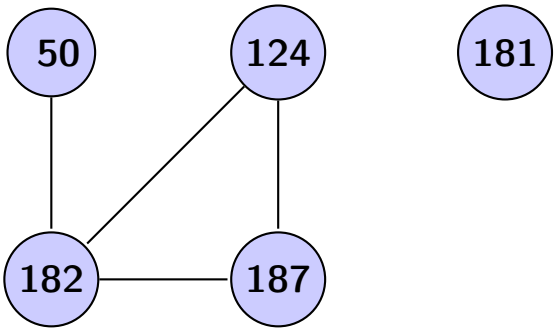
1. Formulate this problem as a CSP problem in which there is one variable per class, stating the domains, and constraints. Constraints should be specified formally and precisely, but may be implicit rather than explicit.

   The following list first shows the variable and then the domains:

   - $CS50$ - $\{A, C\}$
   - $CS124$ - $\{B, C\}$
   - $CS181$ - $\{A, B\}$
   - $CS182$ - $\{A\}$
   - $CS187$ - $\{B, C\}$

   Implicit constraints: $\{CS50 \neq CS182, CS124 \neq CS182, CS124 \neq CS187, CS182 \neq CS187\}$

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.

   We can use backtracking to solve this problem. The domains dictate the possible next assignments, and we consider one assignment per step. We then check whether constraints are violated after each assignment. Once an incomplete end is reached, we backtrack up the search tree. In the following solution, we use the standard backtracking approach and assign the variables numerical order and professors in alphabetical order (You can look at the lecture notes for heuristics for picking the next unassigned variable, or to filter possible solutions which will solve this problem faster!):

   (a) We first pick $CS50 : A$. There are no violated constraints.

   (b) We now pick $CS124 : B$. There are no violated constraints. The partial solution is {CS50: A, CS124: B}.

   (c) We now pick $CS181 : A$. There are no violated constraints. The partial solution is {CS50: A, CS124: B, CS181: A}.

   (d) We now pick $CS182 : A$. This violates the constraint $CS50 \neq CS182$. Therefore, we backtrack all the way to step (a) before we can address this.

   (e) We now first pick $CS50 : C$. There are no violated constraints.

   (f) We now pick $CS124 : B$. There are no violated constraints. The partial solution is {CS50: C, CS124: B}.

   (g) We now pick $CS181 : A$. There are no violated constraints. The partial solution is {CS50: C, CS124: B, CS181: A}.

   (h) We now pick $CS182 : A$. There are no violated constraints. The partial solution is {CS50: C, CS124: B, CS181: A, CS182: A}.

   (i) We pick $CS187 : B$. This does violate the constraint $CS124 \neq CS187$. We backtrack to step (h).

   (j) We pick $CS187 : C$. This does not violate a constraint. We are done. The final solution is {CS50: C, CS124: B, CS181: A, CS182: A, CS187: C}.

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

   The reason that this heuristic is effective is that the variable with the least options is the hardest to satisfy, while the least constraining value leaves open the most options for other variables. This gives the highest chance of success. If we had done this above we would have finished the problem faster – try it yourself!