INFO8006 - Introduction to Artificial Intelligence

Exercise session 2

Game formulation

A search problem is defined by

- A representation for states.
- The initial state of the agent.
- A player function $p = player(s)$ which defines who moves in s.
- A set of actions allowed in every state s.
- A transition model $s' = result(s, a)$ that returns the resulting state s' for using action a in state s.
- A terminal test which determines if the game is over.
- A utility function $utility(s, p)$ that assigns a final numerical value to player p in terminal state s.

Adversarial search

In a two-player game, agent share the same utility. The first want to minimize it (MIN agent) whereas the other wants the opposite (MAX agent). Assuming that MAX moves first and is the agent that we want to win, the problem can be framed as a search problem where

- A goal state is a terminal state where MAX wins.
- MAX agent need a model of its opponent.

The minimax value is the largest utility accessible for MAX from a state s, assuming MIN acts optimally.

H-minimax

Searching the exact minimax solution is most of the time not feasible. One way to bypass this issue is by cutting the search. To do so, we define

- An evaluation function $eval(s)$ that estimates the utility that would be reached from a state s.
- A cutoff test $cutoff(s, d)$ that replaces the terminal test. The later determines if the search must be stopped at depth d in state s.

expectiminimax

When the game is stochastic, we can use the minimax algorithm on the expected value of a state. We have to insert intermediate chance nodes between moves to account for th distribution over actions. The algorithm becomes

$$\text{Expectiminimax}(s) = \begin{cases} \text{Utility}(s) & \text{if Terminal}(s) \\ max_a \text{Expectiminimax}(\text{Result}(s,a)) & \text{if Player}(s) = \text{MAX} \\ min_a \text{Expectiminimax}(\text{Result}(s,a)) & \text{if Player}(s) = \text{MIN} \\ \sum_r P(r) \text{Expectiminimax}(\text{Result}(s,r)) & \text{if Player}(s) = \text{CHANCE.} \end{cases}$$

In session exercises: Ex. 1, Ex. 2

Exercise 1 Tic-Tac-Toe (AIMA, Ex 5.9)

Tic-Tac-Toe is a game for two players, X and O, who take turns marking the cells of a 3×3 grid. The player who succeeds in placing three of their marks in a straight line (horizontal, vertical or diagonal) wins the game. If neither of the players win before the grid is full, its a draw.

We consider X as the max player and O as the min player. We define X_n as the number of rows, columns or diagonals with exactly n X's and no O's. Similarly, O_n is the number of rows, columns, or diagonals with just n O's. A position s is terminal if $X_3(s) \geq 1$, $O_3(s) \geq 1$ or if the grid is full. The utility function assigns $+1$, -1 or 0 to such position, respectively. For non-terminal positions, we use an evaluation function defined as $eval(s) = 3X_2(s) + X_1(s) - 3O_2(s) - O_1(s)$.

1. Define the search problem associated with the Tic-Tac-Toe game.

   Let a state of the game be a matrix $s = (s_{ij}) \in \{-1, 0, 1\}^{3\times 3}$, where the values -1, 0 and 1 respectively denote O, empty and X cells.

   Initial state

   $$s_0 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

   Player function The next player should be X if there are as many X's as O's on the grid and O otherwise. Choosing $+1$ for X and -1 for O,

   $$player(s) = \begin{cases} +1 & \text{if } \sum_{i,j} s_{ij} = 0 \\ -1 & \text{otherwise} \end{cases}$$

   Actions An action is represented by the position (i, j) of an empty cell in the grid. Then, the set of possible actions is $actions(s) = \{(i, j) : s_{ij} = 0\}$.

   Transition model For an action $(i, j) \in actions(s)$, $result(s, (i, j)) = s'$ is the same as s except that $s'_{ij} = player(s)$.

   Terminal test $terminal(s)$ is true if $X_3(s) + O_3(s) \geq 1$ or $\prod_{i,j} s_{ij} \neq 0$, false otherwise.

   Utility function $utility(s) = 1$ if $X_3(s) \geq 1$, -1 if $O_3(s) \geq 1$ and 0 otherwise.

2. Approximately how many possible game states of Tic-Tac-Toe are there?

   If we disregard unreachable states, we have $3^{3\times 3} = 19\,683$ possible states and $9! = 362\,880$ possible games.

3. Show the whole game tree starting from an empty grid down to depth 2 (one X and one O on the board), taking symmetry into account.

4. Annotate your tree with the evaluations of all the positions at depth 2.

5. Using the H-Minimax algorithm, annotate your tree with the backed-up values for the positions at depths 1 and 0, and use those values to choose the optimal starting move.

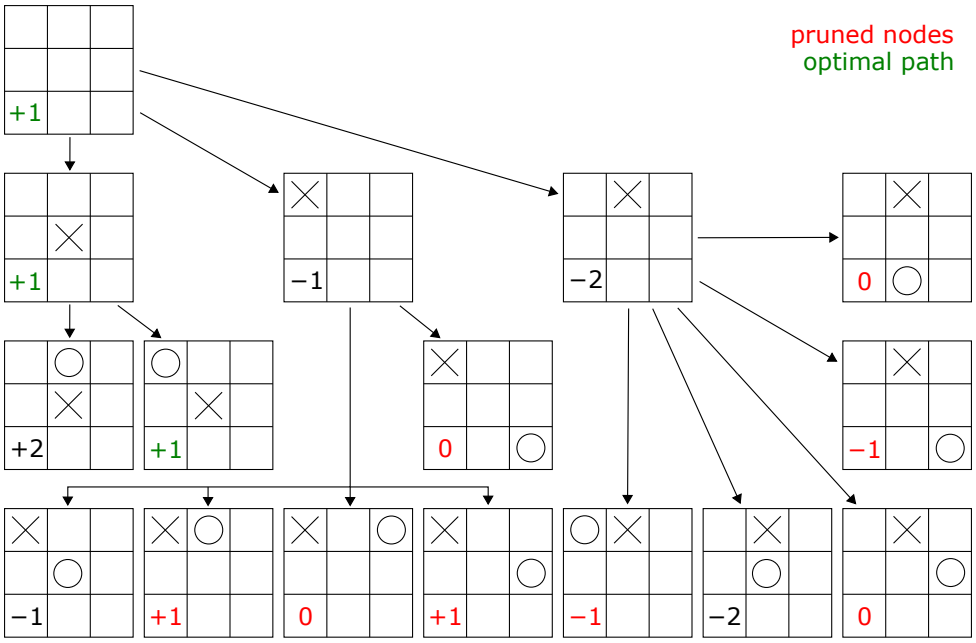
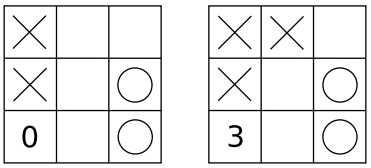


Figure 1. Nodes in red corresponds to nodes that would have been pruned, *i.e.* not evaluated, if $\alpha - \beta$ pruning was applied, assuming the nodes are generated in the optimal order for $\alpha - \beta$ pruning.

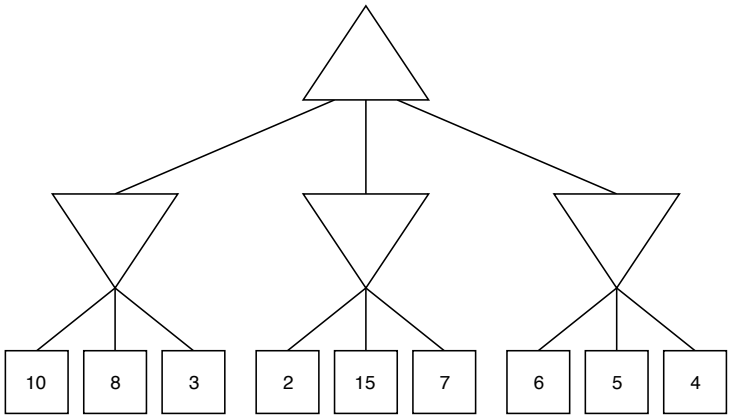
6. Is this evaluation function a good heuristic? If not, provide one or more states s for which $eval(s)$ is misleading.

   This is not a good heuristic, mainly because it does not take into account which player's turn it is. This results in states for which X is winning with lower evaluation than other states for which O is winning.

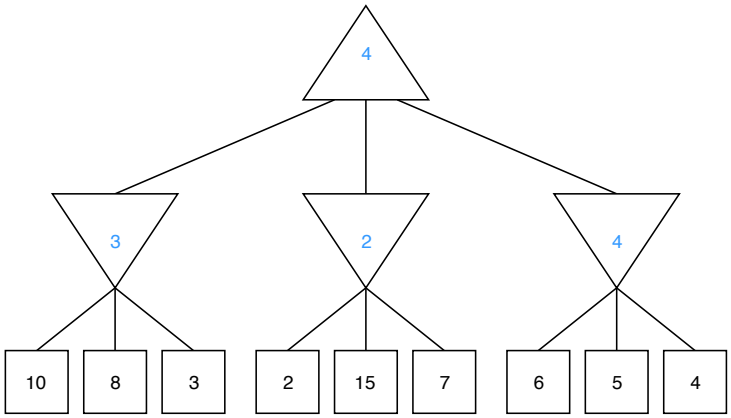


   This heuristic does not preserve the Minimax *ordering* of intermediate states.

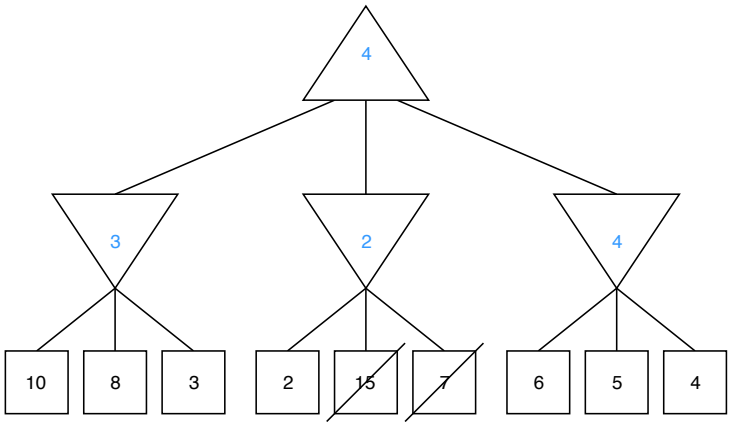
Exercise 2 Minimax (CS188, Fall 2019)



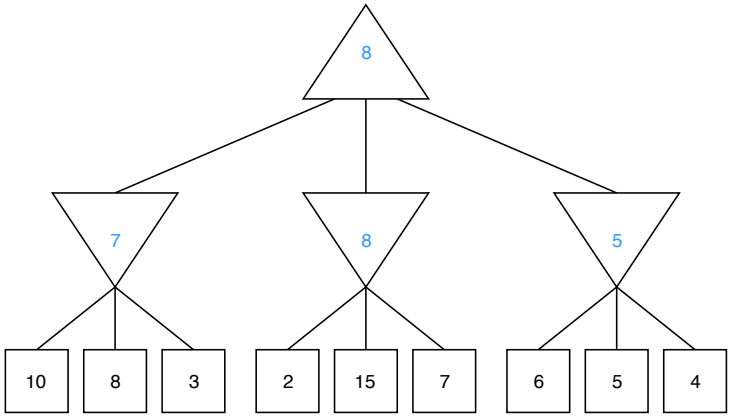
1. Consider the zero-sum game tree shown above. Triangles that point up, such as at the top node (root), represent choices for the maximizing player; triangles that point down represent choices for the minimizing player. Assuming both players act optimally, fill in the Minimax value of each node.



2. Which nodes can be pruned from the game tree above through alpha-beta pruning? If no nodes can be pruned, explain why not. Assume the search goes from left to right; when choosing which child to visit first, choose the left-most unvisited child.



3. Again, consider the same zero-sum game tree, except that now, instead of a minimizing player, we have a chance node that will select one of the three values uniformly at random. Fill in the Expectminimax value of each node. The game tree is redrawn below for your convenience.



Exercise 3 21 misery game (January 2019)

The game "21" is played with any number of players who take turns increasing a counter. The counter starts at 1 and each player in turn increases the counter by 1, 2, or 3, but may not exceed 21; the player who says "21" or larger loses.

1. Define the search problem associated with the 2-player version of the "21" game.

   Let a state of the game be a pair $s = (v, p) \in \mathbb{Z} \times \{0, 1\}$, where v is the current value of the counter and p the player to play next.

   Initial state $s_0 = (1, 0)$.

   Player function $player(s = (v, p)) = p$.

   Actions $actions(s) = \{1, 2, 3\}$.

   Transition model For an action $a \in actions(s)$,

   $$result(s = (v, p), a) = (v + a, p + 1 \bmod 2).$$

   Terminal test $terminal(s = (v, p))$ is true if $v \geq 21$, false otherwise.

   Utility function $utility(s) = 1$ if $p = 0$, 0 otherwise.

2. For the following, consider the game of "5" (still in its 2-player version), which has the same rules has "21" except that you should not say 5 or more. Show the whole game tree.

3. Using the Minimax algorithm, annotate your tree with the backed-up values, and use those values to choose the optimal starting move.

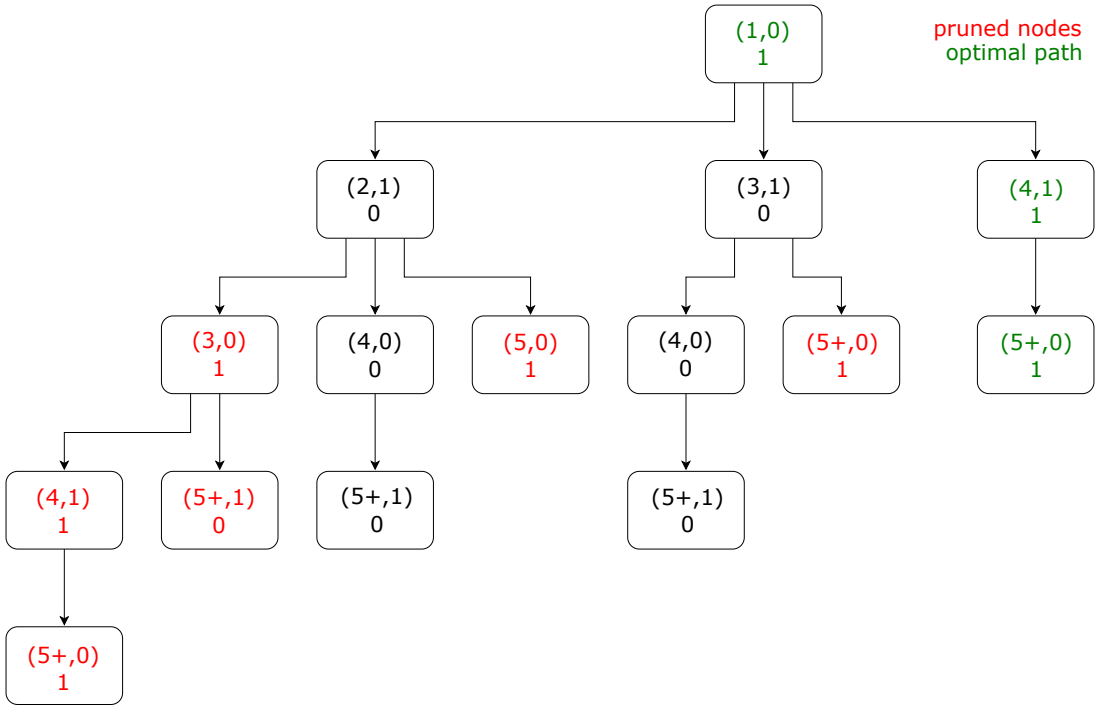
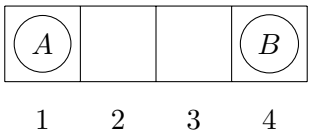


Figure 2. Nodes in red corresponds to nodes that would have been pruned, *i.e.* not evaluated, if $\alpha - \beta$ pruning was applied, assuming the nodes are generated in the optimal order for $\alpha - \beta$ pruning.

Exercise 4 Leapfrog (AIMA, Ex 5.8)



Consider the following two-player turn-taking game which initial configuration is shown in the figure above. Player A moves first. Each player must move their token to an adjacent free cell in either direction. If the opponent occupies an adjacent cell, then a player may jump over the opponent to the next free cell, if any. For example, if A is on 3 and B is on 2, then A may move back to 1. The game ends when a player reaches the opposite end of the board. If player A reaches cell 4 first, then the value of the game to A is $+1$; if player B reaches cell 1 first, then the value of the game to A is -1.

1. Define the search problem associated with this game.

   Let a state of the game be a tuple $s = (i, j, p) \in \{1, 2, 3, 4\}^2 \times \{0, 1\}$, where i and j are the respective positions of A and B and p is 0 when it is A's turn and 1 when it is B's.

   Initial state $s_0 = (1, 4, 0)$.

   Player function $player(s = (i, j, p)) = p$.

   Actions $actions(s) \subseteq \{\text{left}, \text{right}\}$. The "left" (resp. "right") action is only available if there is a free cell to the left (resp. right) of the current player.

   Transition model For an available action $a \in actions(s)$,

$$result(s = (i, j, p), a) = \begin{cases} (next(i, j), j, 1) & \text{if } p = 0 \text{ and } a = \text{right} \\ (prev(i, j), j, 1) & \text{if } p = 0 \text{ and } a = \text{left} \\ (i, next(j, i), 0) & \text{if } p = 1 \text{ and } a = \text{right} \\ (i, prev(j, i), 0) & \text{if } p = 1 \text{ and } a = \text{left} \end{cases}$$

   where

$$next(x, y) = \begin{cases} y + 1 & \text{if } x + 1 = y \\ x + 1 & \text{otherwise} \end{cases}$$

$$prev(x, y) = \begin{cases} y - 1 & \text{if } x - 1 = y \\ x - 1 & \text{otherwise} \end{cases}.$$

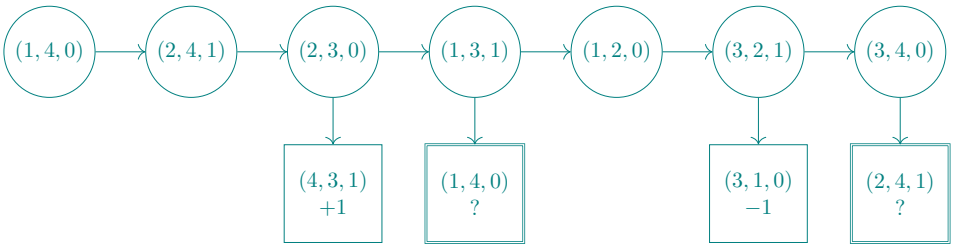
   Terminal test $terminal(s = (i, j, p))$ is true if $i = 4$ or $j = 1$, false otherwise.

   Utility function

$$utility(s = (i, j, p)) = \begin{cases} +1 & \text{if } i = 4 \\ -1 & \text{if } j = 1 \end{cases}.$$

2. Draw the complete game tree, using the following conventions:

   - Put each terminal state in a square box and annotate it with its game value.
   - Put loop states (states that already appear on the path to the root) in double square boxes. Since their value is unclear, annotate them with a "?" symbol.

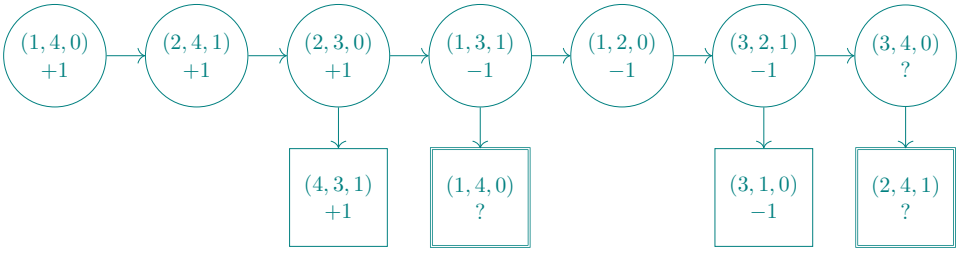


3. Explain why the standard minimax algorithm would fail on this game.

   The Minimax algorithm is designed to work on finite acyclic game trees. In our case, the tree is not acyclic, which leads the standard Minimax algorithm into infinite loops.

4. Annotate each node with its backed-up minimax value. Explain how you handled the "?" values and why.

   We handle the "?" values by assuming that a player only chooses to go in loop states if no winning action is available. That is, A (resp. B) prefers $+1$ (resp. -1) to "?", but "?" to -1 (resp. $+1$).



   Our strategy is equivalent to replacing "?" by any value strictly between -1 and $+1$, which only works for this game tree. For other games, it is not clear how to compare "?" values with intermediate outcomes like draws, wins of different degrees (as in score games) or even other "?" values. In such cases, algorithms more complex than Minimax must be used.

5. This 4-cell game can be generalized to n cells for any $n > 2$. Prove that A wins if n is even and loses if n is odd.

   We already know that A wins if $n = 4$ and it is quick to verify that B wins if $n = 3$. For $n \geq 5$, we notice that the three first states are $(1, n, 0)$, $(2, n, 1)$ and $(2, n-1, 0)$. For optimal players, the state $(2, n-1, 0)$ is equivalent (in the sense that the winner is the same) to the initial state of the game with $n-2$ cells. Likewise, the outcome for n cells is the same as for $n+2$ cells and, by induction, for $n+2k$ ($k \geq 0$) cells. Therefore, we have that A wins games with $4+2k$ cells (even) and loses those with $3+2k$ cells (odd).

Exercise 5 Chess and transposition table (AIMA, Ex 5.15)

Suppose you have a chess program that can evaluate 16 million nodes per second.

1. Decide on a compact representation of a game state for storage in a transposition table.

   There are 32 pieces and we need to specify their positions in a 8×8 board. If a piece is not on the board anymore, we can fix its position as the position of the King. This position can be stored in 6 bits ($2^6 = 64$), for a total of 24 B per state.

2. About how many entries can you fit in a 4 GB in-memory table?

   We can store roughly $4 \times 10^9 \times \frac{1}{24} \approx 160 \times 10^6$ states in the table.

3. Will that be enough for the three minutes of search allocated for one move?

   It is not enough to store all $16 \times 10^6 \times 3 \times 60 = 2.880 \times 10^9$ evaluated nodes.

4. How many table lookups can you do in the time it would take to do one evaluation? Suppose that you have a 3.2 GHz machine and that it takes 20 operations to do one lookup on the transposition table.

   We can perform

   $$\frac{3.2 \times 10^9}{16 \times 10^6} \times \frac{1}{20} = 10$$

   table lookups in the same amount of time as a single evaluation. This demonstrates the importance of transposition tables.

Quiz

In Monte Carlo Tree Search (MCTS), in the formula

$$\frac{Q(n', p)}{N(n')} + c\sqrt{\frac{2 \log N(n)}{N(n')}},$$

which of the following is true?

- ■ The first term encourages the exploitation of higher-reward nodes, while the second encourages the exploration of less-visited nodes.
- ☐ The first term encourages the exploration of less-visited nodes, while the second term encourages the exploitation of higher-reward nodes.
- ☐ The first term encourages the exploitation of highly-visited nodes, while the second term encourages the exploration of lesser-rewarding nodes.
- ☐ The first term encourages the exploration of lesser-rewarding nodes, while the second term encourages the exploitation of highly-visited nodes.

In adversarial search,

- ☐ The horizon effect arises when the search is stuck in a cycle.
- ☐ The horizon effect arises when the evaluation function is perfect.
- ☐ The deeper in the tree the evaluation function is buried, the more the quality of the evaluation matters.
- ■ If not looked deep enough, bad moves may appear as good moves, because their consequences are hidden beyond the search horizon.

A quiescent state is

- ☐ A state in which the game will loop indefinitely.
- ☐ A state in which an agent is stuck.
- ■ A state in which the outcome of a game is unlikely to vary a lot in the near future.
- ☐ A state in which the MAX agent is certain to win.

Minimax algorithm

- ■ Cannot fail more against a sub-optimal agent than an optimal one.
- ■ Can fail more often if the opponent is sub-optimal but predictable and that we underestimate the opponent strategy.
- ■ Does not ensure that the MAX agent always wins.
- ■ All of the above.