National University of Singapore
School of Computing
CS3243 Introduction to AI

Tutorial 6: Adversarial Search

Issued: March 7, 2022 Discussion in: Week 9

Important Instructions:

- **Assignment 6** consists of **Question 1** from this tutorial.
- Your solutions for this tutorial must be TYPE-WRITTEN.
- You are to submit your solutions on LumiNUS by **Week 9, Sunday, 2359 hours**.
- Refer to LumiNUS for submission guidelines

Note: you may discuss the content of the questions with your classmates, but you must work out and write up your solution individually. Solutions that are plagiarised will be heavily penaltised.

---

1. Consider the minimax tree given in Figure 1. In the figure, black nodes are controlled by the MAX player, and white nodes are controlled by the MIN player. Payments (terminal nodes) are squares; the number within denotes the amount that the MIN player pays to the MAX player (an amount of 0 means that MIN pays nothing to MAX). Naturally, MAX wants to maximize the amount they receive, and MIN wants to minimize the amount they pay.

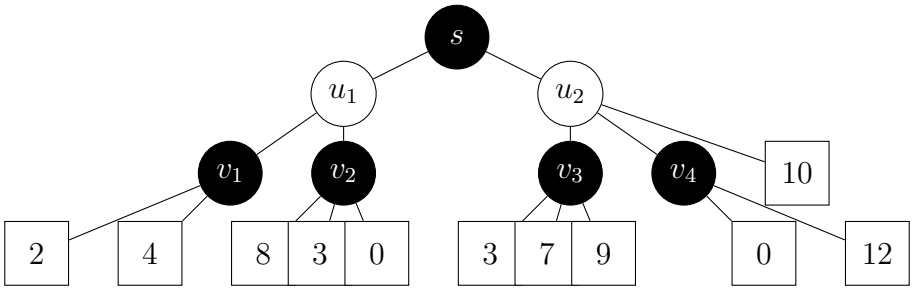

Figure 1: A minimax search tree.

Suppose that we use the α-β `PRUNING` algorithm, given in Figure 6.7 of AIMA 4th edition (reproduced in Figure 2).

(a) **Assume that we iterate over nodes from right to left**; mark with an 'X' all edges that are pruned by α-β pruning, if any.

(b) **Assume that we iterate over nodes from left to right**; mark with an 'X' all edges that are pruned by α-β pruning, if any.

(c) Determine if the effectiveness of pruning depends on iteration order (i.e., provide a **yes or no** answer. (There is no need to elaborate on this answer for your assignment submission However, you should consider why iteration order is important for discussion during the tutorial.)

```
function ALPHA-BETA-SEARCH(game, state) returns an action
  player ← game.TO-MOVE(state)
  value, move ← MAX-VALUE(game, state, −∞, +∞)
  return move

function MAX-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a), α, β)
    if v2 > v then
      v, move ← v2, a
      α ← MAX(α, v)
    if v ≥ β then return v, move
  return v, move

function MIN-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a), α, β)
    if v2 < v then
      v, move ← v2, a
      β ← MIN(β, v)
    if v ≤ α then return v, move
  return v, move
```

Figure 2: $\alpha-\beta$ `PRUNING` algorithm.

2. Consider the following game: we have an attacker looking at three targets: t_1, t_2 and t_3. A defender must choose which of the two targets it will guard; however, the attacker has an advantage: it can observe what the defender is doing before it chooses its move. If an attacker successfully attacks it receives a payoff of 1 and the defender gets a payoff of -1.

   Model this problem as a minimax search problem. Draw out the search tree. What is the defender's payoff in this game?

3. With the `MINIMAX` algorithm, we know that the value v, computed at the root (i.e, the utility for the MAX player), is a worst-case value. This means that if the opponent MIN does not act optimally, the actual outcome v' for MAX can only be better, and never worse than v. That said, the `MINIMAX` algorithm may not select the optimal move given sub-optimal play from the MIN player.

   Construct an example where, should the MIN player play sub-optimally, the `MINIMAX` algorithm makes a sub-optimal move.