Jeremy Keys
3/28/17

Artificial Intelligence I – 2017 - Assignment 5

Let us consider the following bidirectional graph.

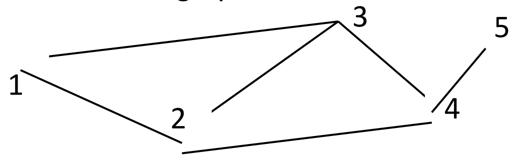


Figure 1: A Graph

Two players (A) and (B) play a game of 2-coloring as follows.

- (A) starts with selecting a vertice of the graph and colors it red
- (B) selects another vertice, that has not been colored before, and colors it with blue
- (A) continues with selecting a vertice that has not been colored by any one and colors it with red
- (B) continues …

In coloring the vertices, (A) and (B) should make sure that no two vertices of the same edge are colored with the same color. Whoever cannot select a vertice to color will lose. If all the vertices are colored, then the game is tied.

As an example, for the graph in Figure 1, if (A) picks 4 and colors with red, (B) picks 1 and colors with blue, (A) will lose. On the other hand, if (A) colors 2 first, (B) picks 3, then (A) can win by coloring 5.

Answer:

(1) The set of states of the game is the set of all possible positions, including all possible terminal states. One state is one position in the game, where a position consists of the graph with each node having an associated color (red, blue, or uncolored), as well as which player's turn it is. The first, and most obvious state, **S_0,** the starting state, consists of the "game board" (the graph) with each node being associated with the "uncolored" attribute. Each state might be given as a set of tuples, where the set has the arity of the number of nodes on the graph, the first element of each tuple is the vertex number, and the second element of each tuple is its color. So, with vertices (1) through (5), the state previously described could be given as:

S_0 = { (1,uncolored), (2,uncolored), (3,uncolored), (4,uncolored), (5,uncolored) }

Another possible state is a terminal state. For simplicity's sake, I'll use the terminal state given in the assignment description, which is a terminal state in which A has lost:

S_t = { (4,red), (1,blue), (2,uncolored), (3,uncolored), (5,uncolored) }

A mid-game state might look like this:

S_x = { (1,red), (2,uncolored), (3,uncolored), (4,red), (5,blue) }

(2) The Player function is defined such that **Player(S_0) = A**, and then the player alternates after every ply (every move by an individual player). In the first state I described in the answer above, which happens to be **S_0**, the function returns **A.** In the terminal state **S_t**, the function returns **A.** In the mid-game state **S_x**, the function returns **B**.

In concrete terms, the Player function might be defined so that a state with an equal number of colored states returns A, and a state with one more red position than blue position returns B (or, alternatively, an unequal number):

Player(s) = { A iff the number of states colored blue in *s* equals the number of states colored red, else B }

So in **S_0**, where # of red = 0 and # of blue = 0, the function returns A; in the next state, after A has colored a position red, the function returns B. In **S_x,** there are two red states and one blue state, so the function returns B.

(3) The Result(**s**,**a**) function is defined to return the state which consists of the position of state **s** with the action **a** applied to it, which in this game, would be a player choosing a vertex to color.

In the example with **S_0** and **Player(S_0) = A**, the player A can choose from the set of any of five actions: { (1,red), (2,red), (3,red), (4,red), (5,red) }. Applying any of these actions to **S_0** results in a state with four uncolored vertices, and one vertex colored red. Of course, by the concrete definition of the Player function I give, since there will be one more red-colored position than blue, the Player function will return B the next time it is called. In any terminal state, there are no valid actions, so the Result function should return **s** (the unchanged terminal state).

For the example **S_x** and **Player(S_x) = B**, the player B has the choice of the following two actions: { (2,blue), (3,blue) } . Game's almost over! If the player chooses (2,blue), the Result function returns:

S_{b2} = { (1,red), (2,blue), (3,uncolored), (4,red), (5,blue) }

Else if the player chooses (3,blue), the function returns:

S_{b3} = { (1,red), (2,uncolored), (3,blue), (4,red), (5,blue) }

(4) The termination test is, of course, a function which returns true when the state it is provided is a terminal state, and false otherwise. In our game, we know that a terminal state is any state **S_t** in which **P = Player(S_t)** has an empty set of valid actions; in other

words, there is no valid move left for the current player. This function can be implemented by iterating over the set of uncolored vertices, and if every uncolored vertex has an edge to a vertex already colored P's color, then the state is a terminal state. Let the function Color(*state, vertex*) take a state and vertex of that state (given as an integer) and return its associated color. In pseudocode in the style of the book (pg. 166):

```
function terminal(state) returns true iff the state is a terminal state, else false
    if all vertices are colored do
        return true

    let mycolor = red if Player(state) = A, else mycolor = blue
    for each uncolored vertex u in state do
        let valid = true
        for each vertex v in state which has an edge to u do
            if Color(state,v) equals mycolor do
                let valid = false

        if valid do
            return false

    return true
```

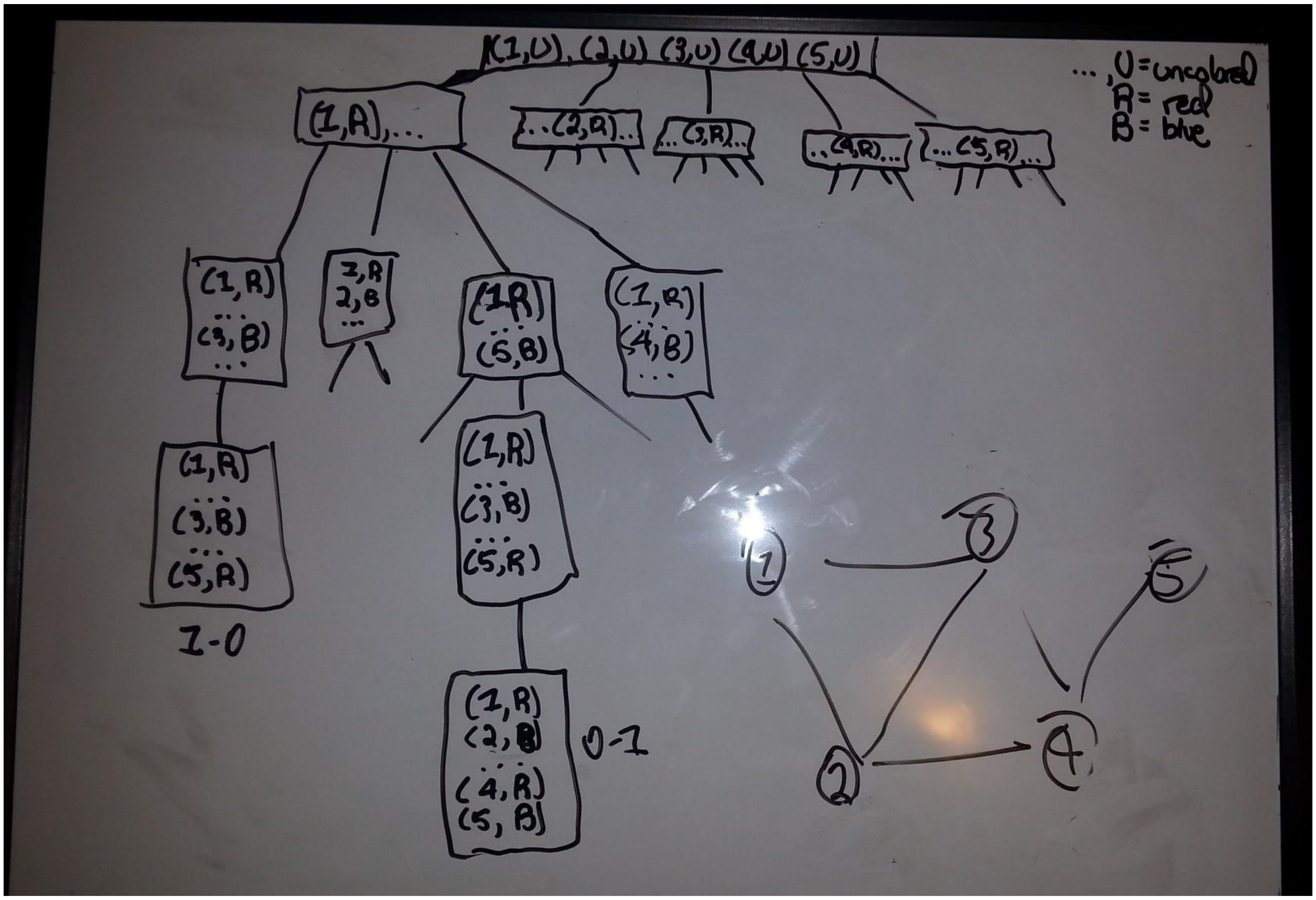
(5) The utility function Utility(**S_t**,**p**) is a function which takes a terminal state **S_t** and the current player **p**. (Note: This function does not need to take a player argument, since the function is given a state, and thus the function should be able to determine the player using the Player function. I'm not sure if this was an oversight by the authors, or if they have a reason for making the function take a player parameter.)

In our game, we can choose the utility outcomes using by popular zero-sum games like Chess, and arbitrarily choose positive for A, and negative for B. Thus the possible payoffs are: 1-0 (A wins), ½-½ (a draw), and 0-1 (B wins). To make this a simple integer payoff, we can modify that to be: 1 (A wins), 0 (draw), -1 (B wins). We can define the function so that it will return 0 if all states are colored, else it will return 1 if Player(**S_t**) = A, else it will return -1.

```
function utility(state,player) returns 0 if draw, -1 if B wins, 1 if A wins
    if all vertices are colored do
        return 0
    else if player equals A do
        return -1
    else do
        return 1
```

This simple function takes advantage of the fact that the loser is the person who must move, but cannot (has no valid moves). In other words, if not all vertices are colored in a terminal state, then we know that the person who needs to move has lost.

(6)


(1,U), (2,U) (3,U) (4,U) (5,U)
..., U=uncolored
R= red
B= blue
(1,R),...
...(2,R)...
...(3,R)...
...(4,R)...
...(5,R)...
(1,R) ... (3,B) ...
3,R 2,B ...
(1,R) ... (5,B)
(1,R) ... (4,B) ...
(1,R) ... (3,B) ... (5,R)
1-0
(1,R) ... (3,B) ... (5,R)
(1,R) (2,B) ... (4,R) (5,B)
0-1
1
2
3
4
5