Written Assignment 3

Due: Friday 03/22/2024 @ 11:59pm EST

Disclaimer

I encourage you to work together, I am a firm believer that we are at our best (and learn better) when we communicate with our peers. Perspective is incredibly important when it comes to solving problems, and sometimes it takes talking to other humans (or rubber ducks in the case of programmers) to gain a perspective we normally would not be able to achieve on our own. The only thing I ask is that you report who you work with: this is **not** to punish anyone, but instead will help me figure out what topics I need to spend extra time on/who to help. When you turn in your solution (please use some form of typesetting: do **NOT** turn in handwritten solutions), please note who you worked with.

Remember that if you have a partner, you and your partner should submit only **one** submission on gradescope.

Question 1: Correctness of Alpha-Beta Pruning (25 points)

Let s be the state of the game, and assume that the game tree has a finite number of vertices. Let v be the value produced by the minimax algorithm:

$$v = \texttt{Minimax}(s)$$

Let v' be the result of running Alpha-Beta Pruning on s with some initial values of α and β (where $-\infty \leq \alpha \leq \beta \leq +\infty$):

$$v' = \texttt{Alpha-Beta-Pruning}(s, \alpha, \beta)$$

Prove that the following statements are true:

- If $\alpha \leq v \leq \beta$ then $v' = v$
- If $v \leq \alpha$ then $v' \leq \alpha$
- If $v \geq \beta$ then $v' \geq \beta$

This means that if the true minimax value is between α and β, then Alpha-Beta pruning returns the correct value. However, if the tru minimax value if outside of this range, then Alpha-Beta pruning may return a different value. However, the incorrect value that Alpha-Beta pruning returns is bounded in the same manner that the true minimax value is (i.e. if the true minimax value is $\leq \alpha$ then the value produced by Alpha-Beta pruning is also $\leq \alpha$ and vice versa). Note that this implies that Alpha-Beta pruning will be correct with initial values of $(-\infty, +\infty)$ for (α, β).

Hint: use induction. If s is not a terminal state, then you can correctly assume that the claim above holds for all children of s. Use this assumption to prove that it also holds for s (the base case is trivial: minimax and Alpha-Beta pruning produce the same value for terminal states)

Alpha-Beta

As said in the comment, both Alpha-Beta and Minimax will return the same values for terminal states. We can use this as our base case. Now consider a generic node s with children's values t_i which satisfy the 3 statements given in the question.

- If the ideal move (found using the brute-force minimax method) has a value v that lies between α and β, the value returned by Alpha-Beta, v', is the same:

  If the node s is in the maximizing player's turn, we want to find the child state with the greatest value. Since we are sure the eventual v is never greater than β, we never return v prematurally before we've explored each child. Since v is greater than α, we'll never set α to higher than the correct v, so in the recursive calls to Alpha-Beta we'll continue having an α less than or equal to v, and we'll never run into this issue. Because an optimal v is known to exist, and we're checking each possible location that optimal v could be (Note that this relies on all children having the correct t_i), we're sure to find it.

  The same holds true but in reverse for the minimizing player's turn. We just need to switch our language from "v is never greater than β" to "v is never less than α", and so on.

- If the ideal move's value v is less than α:

  If the node s is in the maximizing player's turn, we want to find the child state with the greatest value. Knowing that this true value is less than or equal to α, we want to show that the value we output from Alpha-Beta is also less than or equal to α. By induction, there cannot exist a valid Alpha-Beta call to a child that will return a higher value than α. Thus, the maximum value v' can take on is α because we update v' using the max of the current v' and the call to the child.

  If the node s is in the minimizing player's turn, we know there must exist a value of a child that is less than or equal to α (The true v). Since we only stop exploring the child nodes when we find a v' less than α, we will either explore until we find the true v or explore until we find some other value less than α. In either case, $v' \leq \alpha$.

- If the ideal move's value is greater than β:

  If the node s is in the maximizing player's turn, we know there must exist a value of a child that is greater than or equal to β (The true v). Since we only stop exploring the child nodes when we find a v' greater than β, we will either explore until we find the true v or explore until we find some other value less than α. In either case, $v' \leq \alpha$.

  If the node s is in the minimizing player's turn, we want to find the child state with the smallest value. Knowing that this true value is greater than or equal to β, we want to show that the value we output from Alpha-Beta is also greater than or equal to β. By induction, there cannot exist a valid Alpha-Beta call to a child that will return a smaller value than β. Thus, the minimum value v' can take on is β because we update v' using the min of the current v' and the call to the child.

So it turns out that by induction, this will hold true for every node, as every vertex is either a terminal node or upstream of a terminal node.

Question 2: CSP Reduction (25 points)

Prove that any n-ary constraint can be converted into a set of binary constraints. Therefore, show that all CSPs can be converted into binary CSPs (and therefore we only need to worry about designing algorithms to process binary CSPs).

Hint: When reducing a n-ary constraint: consider adding synthetic variable(s) (i.e. inventing new variables). Each synthetic variable should have a domain that comes from the cartesian product of the domains of the original variables involved in that constraint. We can then replace the original constraint with a set of binary constraints, where the original variables must match elements of the synthetic domain's value.

If there is an n-ary constraint, we can create a synthetic variable that has the domain of *each possible* permutation that the n variables involved in the constraint can take on. As the hint says, this is the Cartesian product of the domains of each variable. Now the constraint is met for values of our domain that satisfy the n-ary constraint from before (using the past variables). That is, if our non-synthetic variables take on the values of the vector entries in our synthetic variable's value, and the n constraints are all met, the binary constraint takes on a value of true.

Extra Credit: Markov Blankets (50 points)

A **Markov blanket** of Random Variable X is the set of Random Variables that are the parents of X, the children of X, and the parents of X's children. Prove that a Random Variable is independent of all other variables in a Bayesian network, given its Markov blanket and derive the following equation:

$$Pr[x_i'|mb(X_i)] = \alpha Pr[x_i'|Parents(X_i)] \prod_{Y_j \in Children(X_i)} Pr[y_j|Parents(Y_j)]$$

where $mb(X)$ denotes the Markov blanket of Random Variable X.

Not enough time. Was helping my friend on their 210 `\salute`