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)

Proof by Induction:
Base Case: If s is a terminal state, then $\texttt{Minimax}(s) = \texttt{Alpha-Beta-Pruning}(s, \alpha, \beta) = v$, which is the value of the terminal state. Therefore, the base case holds.
Inductive Step: Assume that the statements are true for all children of s. We need to prove that they also hold for s. Consider the case when s is a MAX node. The $\texttt{Minimax}(s)$ algorithm computes the maximum value of its children. When running $\texttt{Alpha-Beta-Pruning}(s, \alpha, \beta)$, the algorithm prunes branches when the current node value is greater than or equal to β, hence the children's values do not affect the outcome beyond this point. This ensures that:

- *If $\alpha \leq v \leq \beta$ for a child of s:* Since v is within the alpha-beta bounds, there is no pruning and the child's value is considered for updating s's value. If all children's values are within these bounds, the maximum of these values will be v, hence $v' = v$ for the node s.
- *If $v \leq \alpha$ for a child of s:* This implies that the maximizing player has a better option elsewhere (at least as good as α). Thus, any value less than or equal to α will not affect the final decision for the maximizing player since s is a MAX node. Therefore, the pruning will occur, and v' for node s could be at most equal to α.
- *If $v \geq \beta$ for a child of s:* When a child node produces a value greater than or equal to β, we can prune the remaining siblings, as the minimizing opponent will never allow this scenario. Hence, we infer that v' will be at least β because the pruning occurred due to the guarantee that no better option for the minimizing player exists beyond this point.

The case when s is a MIN node follows similarly but with roles of α and β inverted for pruning. Here, β is updated as we iterate through children, and pruning occurs when a child's value is less than or equal to α.
Conclusion: By induction, if the true Minimax value is between α and β, Alpha-Beta pruning returns the correct value. If the true Minimax value is outside of this range, Alpha-Beta pruning may return a different value, but this value is appropriately bounded. Therefore, the properties of Alpha-Beta pruning as stated are correct.

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.

Proof: Consider a CSP with variables $X_1, X_2, \ldots, X_n$ having respective domains $D_1, D_2, \ldots, D_n$ and an n-ary constraint $C(X_1, X_2, \ldots, X_n)$.

- Assume there is a new synthetic variable Y, whose domain D_Y is the Cartesian product of the domains of the original variables involved in the constraint C, that is $D_Y = D_1 \times D_2 \times \ldots \times D_n$.
- For each original variable X_i, we construct a binary constraint $B_i(X_i, Y)$ which enforces that the value of X_i must correspond to the appropriate component of the tuple selected for Y. Specifically, if the value of Y is a tuple $(y_1, y_2, \ldots, y_n)$, then B_i constrains $X_i = y_i$.
- By doing so, the original n-ary constraint $C(X_1, X_2, \ldots, X_n)$ is equivalent to the conjunction of the binary constraints $B_1, B_2, \ldots, B_n$ and the constraint that Y must take values in its domain D_Y. The original n-ary constraint is satisfied if and only if there exists a value in D_Y that satisfies all binary constraints simultaneously.

This reduction shows that any CSP, regardless of the arity of its constraints, can be converted into a binary CSP. In conclusion, through the introduction of synthetic variables and the transformation of n-ary constraints into binary ones, all CSPs can effectively be treated as binary CSPs.

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.

Markov Blanket covers a node's parents, children, and the children's parents.

$$Pr[x_i'|mb(X_i)] = \alpha Pr[x_i'|Parents(X_i), Child_1, Child_2, Child_3, \dots]$$

We reconstruct this equation using chain rule, which means that the joint probability of a set of variables can be decomposed into a product of conditional probabilities.

- The probability that x_i' is true given its parent can be written as $Pr[x_i'|Parents(X_i)]$.
- The probability that child y_j is true given its parent can be written as $Pr[y_j|Parents(Y_j)]$. We'll do that for all children of X_i, and their product captures all the influence of children and children's parents on X_i.
- α is a normalization factor to make sure that the probability sums up to 1.

Thus, we derive this equation:

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