Written Assignment 2

Due: Friday 10/17/2025 @ 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 solutions (please use some form of typesetting: do **NOT** turn in handwritten solutions), please note who you worked with at the very beginning of the pdf.

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)

Note: this is a **proof** question, meaning you must follow formal proof structure (see the examples on piazza for guidance).

Question 2: Oracle-Advised Two-Player Zero-Sum Adversarial-Search (25 points)

In deterministic adversarial search, we have to expand a game tree where nodes are either MAX (our turn to go) or MIN (our opponent's turn to go). This leads us to the Minimax algorithm, where we have to account for our opponents agency, and single-player solutions are not sophisticated enough to account for this agency. Imagine we have access to an oracle $O(s)$. This oracle takes a state as input and returns our opponent's optimal move (for them) if they were allowed to go in that state. This is called an oracle because it is never wrong, whenever we use the oracle it returns our opponent's optimal action 100% of the time.

Assume the game is deterministic. Design an algorithm to turn this two-player game into a single player game that calculates the optimal move we should make and prove its correctness.

Note: this is a **proof** question, meaning you must follow formal proof structure (see the examples on piazza for guidance).

Question 3: Optimizing the AC3-algorithm (25 points)

The AC3 algorithm, whenever *any* value is deleted from the domain of variable X_i, puts *every* arc (X_k, X_i), even if each value of X_k is consistent with several remaining values of X_i. What if we stored, for every arc (X_k, X_i), the number of remaining values of X_i that are consistent with each value of X_k. Derive a way to update these numbers efficiently, and arrive at the conclusion that with this modification, arc consistency can be enforced with total runtime $O(n^2d^2)$.

Note: this is a **proof** question, meaning you must follow formal proof structure (see the examples on piazza for guidance).

Question 4: 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.

Note: this is a **proof** question, meaning you must follow formal proof structure (see the examples on piazza for guidance).

Extra Credit: Creating Crossword Puzzles (25 points)

Consider the problem of building a crossword puzzle (**creating** the crossword puzzle, **not** solving it). A crossword puzzle is defined as fitting words into a rectangular grid. The grid itself is given as part of the problem and specifies which squares are blank (and may be filled), and which are shaded (and therefore unavailable). You have at your disposal a list of words (i.e. a dictionary), and the task is to fill in the blank squares by using any subset of the list. Formulate this problem as:

a A general search problem. Choose an appropriate search algorithm and specify a heuristic function. Is it better to fill in blanks one letter at a time or one word at a time?

b A CSP. Should the variables be words or letters?

c Which formulation is better? Why?

Note: this is **not** a proof question, meaning you do **not** have to follow formal proof structure for this question. English justification is fine.