COMS 4701: Artificial Intelligence

Homework 2 Sample Solutions and Feedback

Problem 1

1. (1-across, 1-down) $\in$ {(big, book), (big, buys), (bus, book), (bus, buys), (has, hold)}
2. 2-down for either heuristic.
3. 1-across is reduced to {big}. 3-across is reduced to {land}. 4-across's domain becomes empty, and we should not proceed with further assignments.
4. Full arc consistency will render every domain empty. You can also stop as soon as one domain becomes empty, which will vary depending on the order you process the constraints. We should not proceed with further assignments.
5. Every domain is left with two elements, and making any arbitrary assignment will reduce every other domain to one element, giving us a solution. Any assignment to a variable using the domain value that is not present in this first solution will result in a second unique solution.

(m, n, k)-Game

Responses

1. O will always win in this scenario. If O moves first, it gains an even bigger advantage over X (starting player can always at minimum tie the game). Choosing a different move when we have multiple equally good options will not change the outcome either, since both players are playing as best as they can (optimally).
2. X wins (3,4,3). O is playing optimally in that it knows that it cannot win, so trying to "prevent" an X victory is the same as making any arbitrary move. Since moves are processed from the top left, O picks those moves first. A similar situation is occurring in (3,4,4), but the advantage is in O's favor instead of X. X cannot win because it must get 4 in a row, which is always blockable by O. So the game ends in a "tie", or a win for O.
3. We have a Xset, which is the set of indices filled by X, and we have a Xbset, which is the set of indices either still blank or filled by X. First, let's look at the sequences() function. It takes in a parameter i which serves as the starting index, and a parameter k which is the number of consecutive cells needed to win. It returns 4 sets of indices, each set corresponds respectively to the "rows" of k cells going right from i, going down from i, going upper-right from i and going lower-right from i.

   So, now what is the "i" that we pass in? It is every blank and X-filled cell, because those might be the start of a winning line. So sequences() returns all potential winning lines, and

we want to check if they are actually feasible for X. We do this by checking if the indices for a line are a subset of the current X's and blanks. If yes, then X has a chance of filling in the blanks and winning. Otherwise, some of the indices are either already occupied by O or they go off the board, so these would be dead ends for X.

Now we calculate the "ratio". Since we need to fill all k cells with X to win, what's the proportion that is already filled? Xscore thus measures progress toward a potential win. We measure that by taking the largest "ratio" during the loop, which is the maximum current progress of X among all its winning scenarios, or the max number of X's in a potential winning line divided by k.

Finally, the difference. Xscore is X's proximity to winning, Yscore is Y's proximity to winning, which means that their difference is X's advantage over Y to the proximity of winning the game. Positive value indicates X is closer to winning, and vice versa.

4. X wins in the first scenario regardless of max depth. However, the sequence of game states and moves varies. As the max depth increases, the players are able to see the end outcome more clearly. For the second scenario, O wins with the default settings, but the advantage turns to X when we make its max depth higher than O's (with the exception of 5 to 4). This is because X is able to look further into the search tree, allowing it to make better moves than O.