Adversarial Search Self-Check

605.645 – Artificial Intelligence

The purpose of this self-check is to make sure you understand key concepts for the algorithms presented during the module and to prepare you for the programming assignment. As you work through problems, you should always be thinking "how would I do this in code? What basic data structures would I need? What operations on those basic data structures?"

Although all the types of games we talked about in this Module can be unified at some level, it is convenient to separate out the two basic kinds of games we looked at. The first kind was a typical turn based game like Chess, Checkers or Backgammon. Although players often think of these games in a strategic sense (a strategy being a collection of moves), but they must alternative their moves and there are a great many moves on the part of an opponent that can foil a strategy. Payoffs are rarely modeled explicitly or known in advance. They tend to be win or lose. Even in Poker, the "pot" is not known ahead of time and you simply have a "winning" hand and get whatever pot there is.

The second kind of game is more strategic in the sense that there is only one "move" and that move is a strategy that might contain any number of actions. Payoffs are presumed to be known in advance. In fact, it can be quite problematic if you don't know your opponent's payoffs because it can distort what strategy you might pick. The interesting thing is that this "Game Theory" gives us a framework for thinking about decisions that we wouldn't ordinarily think of as games but at the same time gives us pause because we're hesitant to label them "games".

Although the programming assignment is concerned with the latter type of game. We are going to work through the former as well.

Problem 1. Tic-Tac-Toe

Tic-Tac-Toe is a solved game. This means that it is possible to completely enumerate the game tree and plot your success (or failure…actually, it would be to a "cat's game"). But that's okay because knowing that the tree can be enumerated isn't the point.

1. Think up a heuristic for Tic-Tac-Toe. Remember, a heuristic is a function that takes a state of the game and returns a value indicating how good that game state is. It is used in the following way. If I have 4 moves available to me, I can construct a game state for each of those moves "as if" I had made them and calculate the score. I should pick the move with the highest score (and by "I", I mean the Agent).

2. The language on moves, turns and ply is a bit confusing. This is because different games define "moves" and "turns" differently (see https://en.wikipedia.org/wiki/Ply_(game_theory) for more information.

   For us, we use the normalized term "ply" to indicate a single action by a single player. Player 1 placing a single "o" would be a ply.

   Using whatever symmetry shortcuts that are available, use your heuristic function to examine a game tree (extensive form game) from the starting board of Tic-Tac-Toe that expands 3 ply and find the best first move. The main challenge you will find when drawing these trees is that they spread out quickly since asymmetric moves break the subsequent ability to exploit symmetry. Do your best. You may need to turn the paper sideways, write small, think of alternative layouts. You may need to draw it out several times and then redraw a final draft.

Is this the move you learned as a child?

Problem 2. Normal Form Games ("Strategic" Games)

The programming assignment deals with Normal Form or what we are calling "Strategic" Games. Specifically, it uses Successive Elimination of Dominated Strategies (SEDS) algorithm. The basics of the SEDS algorithm are straightforward.

Consider any two of my available strategies. If the payoff from strategy A is greater than the payoff from strategy B no matter what my opponent does, I'm never going to be pick Strategy B. In such a case, Strategy A *strongly* dominates Strategy B and the dominated strategy can be eliminated from consideration. Essentially, the range of strategies I need to consider just got smaller. Yay.

It may be the case that some of the payoffs for Strategies A and B are equal for a given strategy of the opponent. If at least one is better than we can say Strategy A *weakly* dominates Strategy B and we can still eliminate Strategy B from consideration.

Consider the following two-person game. Use SEDS to find the pure strategy Nash Equilibrium. For this task, it's not just important to get the answer but to think about how you get the answer and how you would implement that algorithm. You can review the programming assignment to see what data structure you should use.

		Bar 2					
		$2		$4		$5	
	$2	10	10	14	12	14	15
Bar 1	$4	12	14	20	20	28	15
	$5	15	14	15	28	25	25

Make sure you show all your work as this will help you identify the process that you need to identify an algorithm for.

Would you believe me if I told you this was State Space Search? How would you formulate this problem as State Space Search? Under Weak SEDS, you can have multiple solutions. How would you reformulate State Space Search to return all solutions instead of just the first found?