

## EECS 649: PROBLEM SET #5

---

### Reading:

- R&N 5.1-7

**Total Points: 100**

### Format:

- Use standard sheets of paper (8.5 by 11 inches) for scanning
  - Perform all work neatly. When asked to write a paragraph, it should be typed (e.g., using a computer and LaTeX or Word).
- 

*The answer to the following problem should be **typed**:*

### **Problem 5.1 [10 points]** *Describe all the elements of solving a game*

Describe the state descriptions, move generators, terminal tests, utility functions, and evaluation functions for **only one** of the following stochastic games: Monopoly, Scrabble, bridge play with a given contract, **OR** Texas hold'em poker.

*The following two problems require hand calculation and are much faster if you do **not** typeset them (unless you are using LaTeX ...):*

### **Problem 5.2 [20 points]** *Minimax vs. Alpha-Beta*

[Rich and Knight] Print out this [linked game tree](#) (twice).

- Fill in the minimax values on one copy. Beneath the tree, note what move **max** Player 1 should make at Node A and **their payoff (which is the minimum payoff the player is assured if they make that move)**.
- Perform alpha-beta search on the second copy. Then, below the tree, list the nodes that would not have been expanded if alpha-beta pruning had been used (assuming nodes are expanded in **LEFT TO RIGHT ORDER**).

*Example Solutions for (b): Here is a linked [handout](#) with two alpha-beta search examples on the first page (which model solutions for you); plus a detailed walkthrough of the algorithm on the second page (if needed to understand it; your solutions do not have to include that detail).*

### **Problem 5.3 [30 points]** *Basic concepts of game playing (using tic-tac-toe)*

[R&N] This problem explores some basic game-playing concepts, using tic-tac-toe (aka noughts and crosses) as an example. We define  $X_n$  as the number of rows, columns, or diagonals with exactly  $n$  X's and no O's. Similarly,  $O_n$  is the number of rows, columns, or diagonals with just  $n$  O's. The utility function assigns +1 to any position with  $X_3 = 1$  and -1 to any position with  $O_3 = 1$ .

All other terminal positions have utility 0. For nonterminal positions, we use a linear evaluation function defined as  $\text{Eval}(s) = 3X_2(s) + X_1(s) - (3O_2(s) + O_1(s))$ .

- Approximately how many possible games of tic-tac-toe are there?
- Show the whole game tree starting from an empty board down to depth 2 (i.e., one X and one O on the board), taking symmetry into account.
- Mark on your tree the evaluations of all the positions at depth 2.
- Using the minimax algorithm, mark on your tree the backed-up values for the positions at depths 1 and 0, and use those values to choose the best starting move.
- Circle the nodes at depth 2 that would not be evaluated if alpha-beta pruning were applied, assuming the nodes are generated in the optimal order for alpha-beta pruning.

Notes:

- The  $s$  in  $\text{Eval}(s)$  refers to the current state(or configuration). You only have to approximate/bound the number of games.*
- Please take symmetry into account, as noted. Thus, instead of four initial boards with an X in each corner, you only have to deal with one in the upper left corner, e.g., because of the rotational symmetry.*
- In part (e), "optimal order" would be the one that leads to the most pruning after looking at a single node, typically this would be expanding the least value first at a min node and the largest value first for a max node.*

*The following problem requires **programming**:*

#### **Problem 5.4 [40 points]** *Min-conflicts aka 8-queens Revisited*

Consider the **8-queens** problem from R&N again. (Hopefully, this will drastically amortize the time you've spent on it last week.)

Solve it using

- Min-conflicts, choosing the variable (column) to minimize over at random
- Min-conflicts, choosing the variable to minimize in cyclic order: 1, 2, ..., 8, 1, 2, ..., 8, ...

Note: this is really the same algorithm, with just a slight difference in the way the variable to "de-conflict" is picked.

Again, you will be **maximizing** "fitness," defined as the number of **non-attacking** pairs of queens, which is 28 minus "the number of pairs of queens that are attacking each other, either directly or indirectly". Thus, when you find a configuration/state with fitness equals 28, you have found a solution.

*Note: You can set a maximum number of iterations (report that number) and keep track of the number/percent of those that reach it bc they are stuck in a local min ("failures").*

*Also, if more than one value minimizes the conflicts, choose among them at random.*

- a. Implement **one program** encompassing both algorithms above (the difference between them is very slight; use a #define, global variable, command-line option, **or** user input to switch between them). In each case, the algorithm should exit as soon as a solution is found, print the solution (as a string of digits like those depicted in Figure 4.6 of R&N), and print the total number of **fitness evaluations** required from the start of the algorithm.
- b. Test your program enough to convince yourself that it works and that solutions are being found appropriately. **(Do not report on this.)**
- c. Gather statistics regarding the operation of the two different algorithms, recording their performance. Here, you are to run each algorithm 100 times (from different random starting positions/populations) and report **only the average** of the number of evaluations until a solution is found.

Freely use the code you wrote in Problem Set 4 or the C++ or Python I supplied for it in [the linked notebook from PS4](#).

**Hints:** Pseudo-code appears in Figure 6.9 (p. 198) of R&N. Note that Min-Conflicts is similar to hill-climbing, except you only test 8 successors, not 56, so you can look at my supplied code for trying 56 successors and remembering the best.

Turn in your **code** and a **table** summarizing your results. How did this **compare** with your results from Problem Set 4?