cmput 355 2024 homework 5, with hints

Explain each answer, show all work.

1. For $n \times n$ hex for each $n \leq 5$, show all winning first black moves. Repeat for white.

2. Using the solver `hex/hex_simple.py` on your favorite machine, for `x` to play and then for `o` to play, solve 3×3, 3×4 (3 rows, 4 columns), 4×4 hex: you will need to change ROWS and/or COLS. **If this takes more than 1 minute, just kill the process and ignore that part of the question.** In each case, give

   a) who wins
   b) a winning move if first-player wins
   c) the number of calls made in solving
   d) the runtime.

3. In `hex_simple.py`, uncomment the lines that assign a better move order to CELLS and repeat the previous question.

4. For the 4×3 hex board, give a CELLS move order that reduces the number of calls made by `hex_simple.py` both for solving `x` and for solving `o`.

5. a) Which of these features does `hex_simple.py` use? For each unused feature: would it significantly improve runtime?

   prune isomorphic positions   transposition table   alphabeta search

   b) In `hex_simple.py`, what algorithm does `has_win()` use to determine whether a player has won the game?

6. Define *search space* as the number of reachable positions in a game. Estimate each of the following. Sort from lowest to highest.

   3×3 go search space, 3×3 hex search space, tic-tac-toe search space, $3^9 = 19\,683$

7. Here we estimate $n \times n$ hex search space. Assume black plays first. a) Explain why each reachable position with b black stones and w white stones has $b = w$ or $b = w + 1$.

   b) Define $f(n)$ as the number of $n \times n$ Hex positions that satisfy the condition in a). Recall from CMPUT 272 that $c(n, k)$ is the number of k-element subsets of an n-set. Explain why $f(3)$ equals this:

   $c(9, 0) + c(9, 1) + c(9, 1) * c(8, 1) + c(9, 2) * c(7, 1) + c(9, 2) * c(7, 2) + c(9, 3) * c(6, 2) + c(9, 3) * c(6, 3) + c(9, 4) * c(5, 3) + c(9, 4) * c(5, 4) + c(9, 5) * c(4, 4)$.

   c) Give similar expressions to those in b) for $f(1)$ and $f(2)$.

   d) Is $f(n)$ an upper bound on the search space, a lower bound on the search space, or neither?

   e) Program `hex/hex_count.py` computes values $f(n)$. Roughly, what is the ratio of $f(n)/(3^n)$?

8. `https://webdocs.cs.ualberta.ca/~hayward/355/mcts24.pdf` shows output from the start of an execution of `mcts/main.py` (`mcts x`) for the empty 2×2 hex board.

   a) After the search, what criterion is used to pick the move to play? In this case, what move was played: a1, a2, b1, or b2?

   b) Is the move from a) provably winning, probably winning, provably losing, probably losing, or none of these?

   c) Explain each line of ouput from the slides

   d) Find a simulation in which the leaf selection was made by picking a most-likely-winning child at each step.

   e) Find a simulation in which the leaf selection was made by not always picking a most-likely-winning child, or explain why there is none.

   f) Show the mcts tree after the first 9 simulations. Label each node with its cell name, wins, and visits (simulations).

9. a) Run `python3 main.py` in `mcts` on the 4×4 board. What move does it make? Is this a winning move?

   b) In `mcts/mcts1.py`, replace this line

   ```
   uct = mean_res+(self.c*sqrt(log(self.root_node.sims)/child.sims))
   ```

   with line `uct = mean_res` and repeat a). How do your results compare with a)?

10. Computing logarithms and square roots can be slow on some systems. In `mcts/mcts1.py`, replace square root factor from the previous question with factor `self.root_node.sims/(child.sims + self.root_node.sims)`. What is the ratio of number of simulations performed with this change compared to previously, say on 4×4 hex from the empty board?

11. Here is one way that the mcts algorithm in `mcts/` could be improved. At a node, whenever a true win (i.e. minimax win, so in hex once a player has joined their two sides) is detected,

prune their siblings.

a) For any two player game, what else could be done at this point?

b) For Hex, what else could be done at this point?

12. For each of the winning 3×3 hex strategies below:

    a) is this a first-player or second-player strategy? and for black or for white or for both?

    b) explain the strategy in words

    c) If you represented the strategy using a dag or tree, about how many nodes would be in the dag or tree?

    $b2 \wedge (b1 \vee c1) \wedge (a3 \vee b3)$

    $a3 \wedge (a2 \wedge (a1 \vee b1) \ \vee \ c1 \wedge (b2 \vee c2 \wedge (b3 \vee c3)))$

    $a2 \wedge (a1 \vee b1) \wedge (a3 \ \vee \ c2 \wedge (b2 \vee c1) \wedge (b3 \vee c3))$

13. See slide 32 in

    `https://webdocs.cs.ualberta.ca/~hayward/talks/hex.someques.pdf`

    a) Explain why the black stone is virtually connected to the top. Write this virtual connection in logical form using the notation from the previous question.

    b) Explain why the black stone is virtually connected to the bottom. Write this virtual connection in logical form using the notation from the previous question.

    c) Explain why the top is virtually connected to the bottom. Write this virtual connection in logical form using the notation from the previous question.

14. See slide 45 in

    `https://webdocs.cs.ualberta.ca/~hayward/talks/hex.someques.pdf`

    a) This position is white-to-play. If white's move is not in one of the 9 dark cells, explain how black can win.

    b) If white's move is not at the 1 dark cell in slide 48, explain how black can win.

    c) For a given hex position and given player-to-move, explain what a *mustplay region* is.

hints

1. slide 20ff `https://webdocs.cs.ualberta.ca/~hayward/talks/hex.someques.pdf`

2. `black-to-play, black plays top-to-bottom, mac 2021 desktop`

   ```
   3x3     b     c1     5677     0.0
   3x4     b     a1   161899     0.5
   4x4     b      ?        ?   > 60.0
   ```

3.
   ```
   3x3     b     b2      678     0.0
   3x4     b     b2    21668     0.1
   4x4     b     c2  7034997    26.3
   ```

4.
   ```
   4x3     w     --   456373     1.3  default order
   4x3     w     --    40294     0.2  (4,7,5,6,3,2,8,9,1,10,0,11)
   ```

   We want `o` to find a winning strategy quickly. Put 4,7 at the front of the order: then `o` always gets at least one of these and `x` is in trouble. Let me know if you can do better `:)` .

5. a) none. there are generally fewer symmetries in hex than ttt: the most common is rotation by 180 degrees, you are unlikely to reduce the search space by more than .5. There are lots of transpositions in hex, the dag of all continuations (DOAC) would probably have less than .5 the nodes of the TOAC. There are only two outcome values in hex (win/loss), compared to three in ttt (win/loss/draw), so alphabeta won't save as much as it does with ttt, especially because `hex_simple` is faster than pure minimax, as it returns as soon as a winning child is found.

6. tic-tac-toe: search space is numuber of entries in tt after solving (why?). I modified `tt24.py` to give this info. 4520 nodes.

   3x3 hex: `search_space()` in `hex_simple.py`. 5514 nodes.

   3x3 go: see `https://tromp.github.io/` and `https://tromp.github.io/go/legal.html` : 3x3 legal go positions 12675.

   caveat since legal moves from a postion depends not just on the position but only the complete move history, a better measure of go search space is the number of paths (with no repeated node) from the root in a graph with 12675 nodes. this will be much bigger than 12675 (interesting research question: estimate this number).

7. a) number of moves made by black will always be equal, or one more than, number of moves made by white
   b) after black has placed j stones and white has placed k stones, black had $c(n*n, j)$ ways to place her stones, and white had $c(n*n-j, k)$ ways to place her stones.
   c) $f(1) = c(1,0) + c(1,1) = 2$. $f(2) = c(4,0) + c(4,1) + c(4,1) * c(3,1) + c(4,2) * c(2,1) +$

$c(4, 2 * c(2, 2) = 35$.

d) upper bound: some of the postions counted by $f(n)$ are not reached because the game ended earlier, e.g. a 2×2 hex position with `x` at a1,a2 and `o` at b1,b2 is never reached, because black wins after her 2nd move.

e) it's getting close to $1/n$

8. covered in detail in the lecture Mar 20.

9. a) the program is non-deterministic, so the winning move varies. I ran the program 20 times with 1s runtime, it found a winning move 19 times. Each run performed about 11000 simulations.

   b) the number of simulations increased from around 11000 to around 16000, which is to be expected since the floating point function calls to square root and logarithm take significant time. The search changes from an exploitation-exploration algorithm to an exploitation-only algorithm, so we might expect it to make a bad move here: I ran it 20 times on the 4×4 board, it found a winning move 15 times.

10. The number of simulations improved from around 11000 to around 13000, so about 11/13. I ran it 20 times on the 4×4 board, it found a winning move 17 times.

11. a) Assume that a node p in the game tree has a child v and that v has a child c and that c is a win. Then v is a loss, so we can use this info in the rest of our search. For example, we might check whether v has any siblings, and if so, just prune v.

    b) In Hex (also tic-tac-toe) once we know that c wins we can prune all children of p except moving to cell c, because if we don't then the opponent can just move to c after we move to not-c.

12. We answer for the second connection: we leave the other two to you.

    a) this strategy works for black (top-to-bottom). it supplies an initial move, so it is for the first player.

    b) black plays at $a3$, leaving a black top-to-bottom virtual connection. this vc is formed by two semi-connections (first-player) S_1 and S_2.
    S_1 is this: black plays at a2 and then plays one of $\{a1, b1\}$ (white cannot take both).
    S_2 is this: black plays at c1 and then plays either the one-cell sc $S_3 = \{b2\}$ or the three-cell sc S_4.
    S_4 is this: black plays at c2 and then plays one of $\{b3, c3\}$.

    c) Here is the answer for the tree: the dag has fewer nodes, as there are some transpositions (e.g. a3 a1 c1 a2 and a3 a2 c1 a1). The tree root corresponds to the starting position (empty board). This is a strategy tree, so each node with player(black)-to-move has exactly one child (the move from the strategy), each node with opponent(white)-to-move has all possible moves (one for each empty cell).

Below is part of the tree. Each node `*` is a subtree isomorphic to the subtree at the same level with root `c1`: these three subtrees each have 21 nodes.

Each node `+` is a subtree isomorphic to the subtree at the same level with root `c3`. these five subtrees each have 13 nodes. The total number of tree nodes is $1{+}1{+}8{+}8{+}3{*}21{+}5{*}13 = 146$.

```
                         root
                          |
                         a3
          /    /    /    / \   \     \    \
        a1   a2   b1   b2   b3   c1   c2   c3
        |    |    |    |    |    |    |    |
        c1 c1*  c1*   a2+   a2+ a2+  a2+   a2
  /  / / \  \  \                      /  / /\  \  \
a2 b1 b2 b3 c2 c3                   a1 b1 b2 b3 c1 c2
 |  |  |  |  |  |                    |  |  |  |  |  |
b2 b2 c2 b2 b2 b2                   b1 a1 a1 a1 a1 a1
     //\\
   / /  \ \
 a2 b1 b3 c3
  |  |  |  |
 b3 b3 c3 b3
```

13. covered in the lectures
    a) the cell set is $\{c1, d1\}$: if white plays at one of these, play can play at the other to join the black stone at $c2$ to the top.
    b) the cell set is $\{d2, b3, c3, d3, a4, b4, c4, d4\}$. this is the 4-3-2 side template: if white plays in $\{b3, a4, b4\}$ black plays at $d3$; if white plays in $\{d2, c3, d3, c4, d4\}$ or outside the cell set black plays at $b3$.
    c) combine the two virtual connections. logic form is left for you.

14. covered in the lectures
    a) the diagram shows the cells of a black win-threat: after black plays at the dot (e4), play has a top-to-bottom vc using the remaining 8 grey cells.
    b) slides 45, 46, 47 each show a black win-threat. the intersection of the cell sets of these 3 win-threats is the one cell shown on slide 48 (d5). If white plays at any other cell cx, then cx does not intersect all 3 win-threats, so it misses at least one win-threat, so after white plays at cx, black can play at any win-threat that cx did not iterfere with, turning the win-threat into a winning vc. the intere