COMP3411/9414 Artificial Intelligence
Session 1, 2019

Week 5 Tutorial Solutions

This page was last updated: 04/30/2019 06:05:19

Activitity 5.1 Tic-Tac-Toe (Exercise 5.9 from R & N)

This problem exercises the basic concepts of game playing, using tic-tac-toe (naughts 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 +10 to any position with X_3>=1 and -10 to any position with O_3>=1. All other terminal positions have utility 0. For the nonterminal positions, we use a linear evaluation function defined as

$$\text{Eval}(s) = 3X_2(s) + X_1(s) - (3O_2(s) + O_1(s))$$

1. Approximately how many possible games of tic-tac-toe are there?

   There are 9 choices for the 1st move, 8 for the 2nd move, 7 for the 3rd move, etc., giving us an upper bound of 9! = 9*8*7*6*5*4*3*2*1 = 362880. But this is an overestimate, because some games end in 5, 6, 7 or 8 moves. The true figure is actually 255168.
   If we take symmetry into account, the number reduces substancially. For example, there are now only 3 choices for the first move and at most 5 choices for the second move. In fact, the total is reduced to 26830 distinct games, of which 172 end in 5 moves, 579 end in 6 moves, 5115 end in 7 moves, 7426 end in 8 moves, 8670 result in a win in 9 moves and 4868 result in a draw. There are a number of Web sites providing a full analysis. See for example `http://www.se16.info/hgb/tictactoe.htm`

2. 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.
3. Mark on your tree the evaluations of all the positions at depth 2.
4. 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.
5. 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.

Activity 5.2 Exploiting a Suboptimal Opponent

Continuing the tic-tac-toe example, if minimax (or alpha-beta search) were to search the entire game tree, it would evaluate all opening moves equally - because it assumes the opponent will play optimally, which leads to a draw in the end, no matter what the opening move. However, if we assume the opponent could make an error, we see that one particular opening move is much better than the others. Explain why.

Hint: for each leaf position from Activity 5.1, try to determine whether X or O can force a win from that position.

> The best opening move for X is in a corner (i.e. the second branch in the game tree above). Check each of the five children of this node and you will see that, for all but one of them, X can use a "fork" tactic to force a win. Even if O chooses the correct move (which is in the centre), X can play into the opposite corner giving him yet another chance to win (if O is silly enough to play in a remaining corner).

Activity 5.3 Applying Alpha-Beta Search

Apply the alpha-beta search algorithm to the following game tree, indicating clearly the values of alpha and beta at each node as the algorithm progresses, and circling the nodes that would not be evaluated.

Note:

1. It is helpful to first do a full minimax search, to tell us the "line of best play" (which is shown in bold).
2. The algorithm prunes off the last four nodes in one stroke, implicitly using logic which might be paraphrased as follows:

   > "By choosing the left option, MAX can guarantee a reward of at least 4. The right option would allow MIN to reduce the reward below 4, so MAX can reject this option without further evaluation."

Activity 5.4 Pruning in Games with Chance Nodes

This question considers pruning in games with chance nodes. This figure shows the complete game tree for a very simple such game. Assume that the leaf nodes are to be evaluated in left-to-right order, and that before a leaf node is evaluated, we know nothing about its value -- the range of possible values is $-\infty$ to ∞.

1. Copy the figure, mark the value of all internal nodes, and indicate the best move at the root with an arrow.

2. Given the values of the first six leaves, do we need to evaluate the seventh and eighth leaf? Given the values of the first seven leaves, do we need to evaluate the eighth leaf? Explain your answers.

The value of the left branch is 1.5 . After the first six leaves, we still need to keep evaluating, because if the 7th and 8th leaf were both greater than 3 then the right branch would be preferred. Once the 7th leaf is evaluated, we know that the value of the right branch is at most -0.5, so we do not need to evaluate the 8th leaf.

3. Suppose the leaf node values are known to lie between -2 and 2 inclusive. After the first two leaves are evaluated, what is the value range for the left-hand chance node?

   It is between 0 and 2 (inclusive).

4. Circle all the leaves that need to be evaluated under the assumption in Part 3.

   Only the first five leaves need to be evaluated. After that, we know that the right branch is between -2 and 1, so it is definitely inferior to the left branch.

Activitity 5.5 Further Discussion

---

1. Describe an optimal strategy for a simple version of the game Nim, which uses only a single heap of *n* stones. Players take turns to remove either 1, 2 or 3 stones from the heap. The player who takes the last stone (or, all the remaining stones) wins. For what values of n can the first player force a win? For what values can the second player force a win?

   If *n* is divisible by 4, the second player can force a win; otherwise, the first player can force a win. The winning strategy is to always remove a number of stones equal to *n* mod 4.

2. Discuss what you found out from the Tree Search for Simple Games activity on the Alpha-Beta Pruning page, concerning one or more of these games:

   - Hexapawn on a 3x3 board
   - Connect-4, or a simplifed version with four columns, where you only need to get three in a row in order to win
   - Sprouts, with two initial dots
   - another simple game of your choosing

3. The Chinook checkers program makes extensive use of endgame databases, which provide exact values for every position with eight or fewer pieces. How might such databases be efficiently generated, stored and accessed?

   An indexing function can be found which assigns a unique number to each position, and classifies the positions into "slices", based on the number of kings and checkers, and the rank of the most advanced checker, for each colour. These different "slices" of the database can be compressed and decompressed as needed during a game. The Chinook database contains 443,748,401,247 positions, but compresses into roughly 6 gigabytes.

The database is essentially a gigantic lookup table, whose values can be generated by a technique known as "Retrograde Analysis". We assume inductively that all positions with N-1 or fewer pieces have been evaluated, before extending to positions with N pieces. Positions with a forced capture are evaluated first (because they lead directly to a position with fewer pieces). The remaining positions are initially marked as "unlabeled", and then we repeatedly loop through them, each time labeling some positions as "win" or "loss". Positions can be labeled using either a "forward" or "backward" approach. The "forward" approach looks at an unlabeled position and generates all possible successor positions. The "backward" approach looks at a labeled position and generates all possible predecessor positions. If all the successors of a state have previously been labeled as "win" (for the opponent) then that state is labeled as "loss" (for the current player). If any successor state is labeled as "loss", then the current state is labeled as "win". If the successor states are all "unlabeled", or a mix of "win" and "unlabeled", then the current state remains unlabeled. We keep looping through the remaining positions until we get through a loop with no new positions being labeled. It follows that all remaining unlabeled positions must result in a draw.

Details can be found at `http://www.cs.ualberta.ca/~chinook/publications/`
