Prof. Dr. Goran Glavaš,
M.Sc. Fabian David Schmidt
M.Sc. Benedikt Ebing
Lecture Chair XII for Natural Language Processing, Universität Würzburg

8. Exercise for “Algorithmen, KI & Data Science 1”

1 Adversarial State Space Search

1. Define the tic-tac-toe game as adversarial state space search problem giving the (a) initial state, (b) successor function, (c) terminal function, and (d) payoff function.

    (a) Initial state s_0: matrix $A^{3\times 3}$, where each element is 0

    (b) Successor function $succ(s)$: if s is a max node: change any element of A that is equal to 0 to 1, else (s is a min node) change any element of A that is equal to 0 to -1

    (c) Terminal function $terminal(s)$: if sum over rows, columns or diagonals equals to 3 or -3 OR no more 0 elements in A return true, else false

    (d) Utility function: $utility(s)$: if sum over rows, columns or diagonals equals to 3 return 1, elif sum over rows, columns or diagonals equals to -3 return -1, else return 0

2. How might the actual utility of player MAX change compared to its expected utility ((a) increase, (b) decrease, or (c) stay the same), if MAX follows the optimal strategy (MINMAX-method), but MIN does not (e.g., MIN chooses randomly). Give an example game tree or argue why an option is not possible.

    Optimal strategy: Maximizing the worst-case utility

    (a) Increase: Possible because MIN might make a move that is not optimal. In the example in lecture 16, slide 10, MIN might randomly choose the left most transition resulting in utility 3 for MAX.

(b) Decrease: Not possible because MAX is already maximizing the worst-case utility.

(c) Stay the same: Possible because MIN might randomly make the optimal moves (e.g., lecture 16, slide 10)

3. Does the number of pruned nodes in alpha-beta pruning depend on the ordering of explored successor states? Explain.

   Yes, it does. Let's assume we prune below a MIN node and the root is a MAX node. We traverse the children of a node from left to right and only the rightmost node is smaller than alpha. In that case, we would need to traverse all nodes. If the depth of the tree is 2 nothing is pruned. In contrast, if the leftmost node is smaller than alpha, we can immediately prune the whole subtree (see lecture 16, slide 20).

4. How could you improve the ordering/selection of successor nodes such that alpha-beta pruning becomes more efficient (i.e., prunes more nodes)?

   a) Keep track of already explored states and their best successor (is infeasible for large state spaces)

   b) First explore successful moves in other branches of same depth (e.g., in the first branch you detected that marking the center of tic-tac-toe is successful. Therefore, you try it first in other branches of the tree)

   c) Use domain knowledge to order the moves (e.g., experts tell you to always mark the center in tic-tac-toe, if possible)

5. Implement the $ticTacToe$ class in the given $.ipynb$. The class represents a tic-tac-toe game. Given a state s it should return the best possible next move. The state s is represented as a 3×3 matrix, where an empty field is represented by 0, an "X" is represented by 1, and "O" is represented by -1. The two players are "MAX" (using "Xs" or $1s$) and "MIN" (using "O" or $-1s$). Implement the following methods:

   a) Player function $player(s)$: Given a state s return whose turn is next (either "MAX" or "MIN"). Assume that from the initial state MAX always makes the first move.

   b) Successor function $succ(s)$: Given a state s return the set of possible successor states.

c) Winner function $winner(s)$: Given a state s return $False$ if no one has won yet or the name of the player (either "MAX" or "MIN") otherwise.

d) Terminal function $terminal(s)$: Given a state s return $True$ if the state is terminal and $False$ otherwise.

e) Utility function $utility(s)$: Given a state s return 1 if "MAX" wins, -1 if "MIN" wins or 0 otherwise

f) Max value function $max\_val(s)$: Given a state s return the maximum of all minimum values (pseudocode in lecture 16, slide 12)

g) Max value function $min\_val(s)$: Given a state s return the minimum of all maximum values (pseudocode in lecture 16, slide 12)

h) Minmax function $minmax(s)$: Given a state s return $False$ if the game ended without winner, return the winner if there is one or return the best possible next state.