UNITED STATES MILITARY ACADEMY

HW 2 - 3

CS486: ARTIFICIAL INTELLIGENCE

SECTION F1

LTC ALEXANDER MENTIS

By

CADET SANG KEUN OH ’19, CO C4

WEST POINT, NEW YORK

28 SEPTEMBER 2018

\_\_\_\_ MY DOCUMENT IDENTIFIES ALL SOURCES USED AND ASSISTANCE

RECEIVED IN COMPLETING THIS ASSIGNMENT.

\_\_\_\_ I DID NOT USE ANY SOURCES OR ASSISTANCE REQUIRING DOCUMENTATION IN COMPLETING THIS ASSIGNMENT.

SIGNATURE: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

HW 2-3

1a. A state has to contain the depth/level number (how many recursions the player has gone down), the location of the player (A, B, C, -, +), and the path/actions taken in previous states.

1b. The space size for this search graph is infinite because of the recursive nature of this maze. For example, consider the top left corner. You could travel to an infinite depth inside “A” by continually traversing the leftmost node on the top row.

1c. The estimated branching factor for the search graph is 9. Locations A, B, and C have 11, 8, and 8 possible paths they could take, respectively. The average of 11, 8, and 8 is 9. Therefore, the best estimate for a branching factor is 9.

1d. BFS, UCS, and iterative deepening would all be appropriate, but I would use BFS. DFS is not appropriate as the recursive nature of the maze could easily lead the search to an infinite length path. Greedy is not appropriate because it is not optimal and we are searching for the shortest path. A\* is also not appropriate because very few, if any, heuristics are available for this problem. There are also few ways to assign utilities to different paths except for the recursive level at each location. UCS, for similar reasons, is not any more effective than BFS because there is really only one way to calculate utilities for paths. The costs would then all be initialized to 1, and that is the same as BFS. Iterative deepening depth first functions as a sort of DFS, which also means that it is not optimal. BFS is optimal for shortest paths in this case and is simpler to implement than UCS.

2a. Missionaries and Cannibals problem is a CSP because there are 3 main constraints; the boat needs at least one person to move back and forth, missionaries cannot be outnumbered on either bank, and the boat can hold at most 2 people at a time. The constraint satisfaction problem could be visualized as a schedule of boat trips back and forth between the banks that eventually moves everyone to the other side while fulfilling all constraints.

2bi. The set of variables can be described as , where Zxy is the position x, y on the board (10 x 10). The domains for each variable are {battleship, cruisers, destroyers, submarines, no ship}. A possible solution set of variables and domains follows.

Zxy for x = 1 and y = 1 through 4 is the battleship

Zxy for x = 3 and y = 1 through 3 and 5 through 7 are the 2 cruisers

Zxy for x = 5 and y = 1 through 2, 4 through 5, 7 through 8 are the 3 destroyers

Zxy for x = 7 and y = 1, 3, 5, 7 are the 4 submarines

2bii. If Zxy = any *one* ship, then Zx’y’ = no ship for (x’, y’) = (x-1, y-1), (x+1, y-1), (x+1, y+1), (x-1 y+1), (x+1, y), (x-1, y), (x, y-1), (x, y+1). This defines the constraint that two individual ships cannot be located in adjacent positions, even diagonally.

2biii. I would choose iterative improvement with min-conflicts because getting to a valid solution from a near-end state might be faster than starting from scratch. All the ships take up a sum of 20 positions out of 100. Even considering that ships must not be adjacent, it is easy to see that there are very many valid placements of all ships within the constraints. Using an efficient “improvement” function by moving a conflicting piece to a part of the board with the least spots occupied favoring the edges could often yield a solution quicker than a classic backtracking search. The benefits of using iterative improvement grow as the problem scales with respect to the board size.

3. A “dynamic ordering” technique used for alpha-beta pruning for chess games is called the killer heuristic (Huberman 1968). The killer heuristic orders moves by how often they have caused other moves to be pruned at the same depth. If some move at a certain depth has pruned two moves, a separate data structure keeps track of that move, and if that move appears at the same depth again, the move is placed first if no other moves pruned more than 2 moves at the same depth. This goes off the principle that one good move in chess is likely to still be a good move because of the position the particular piece establishes is likely important at the point in time (depth) (Wikipedia contributors 2018 Apr 24).

References

Huberman BJ 1968. A program to play chess end games. [Internet]. Stanford University Department of Computer Science, Technical Report CS 106, Stanford Artificial Intelligence Project Memo AI-65; [cited 2018 27 Sep]. Available from: <http://www.dtic.mil/dtic/tr/fulltext/u2/673971.pdf>

Wikipedia contributors. Killer heuristic [Internet]. Wikipedia, The Free Encyclopedia; 2018 Apr 24, 06:24 UTC [cited 2018 Sep 28]. Available from: <https://en.wikipedia.org/w/index.php?title=Killer_heuristic&oldid=837984356>.

Wikipedia contributors. Battleship (puzzle) [Internet]. Wikipedia, The Free Encyclopedia; 2018 Feb 11, 00:09 UTC [cited 2018 Sep 28]. Available from: <https://en.wikipedia.org/w/index.php?title=Battleship_(puzzle)&oldid=825016344>.