Mini-Project 1

Derek Kwan
dkwan6@gatech.edu

1 QUESTION 1: AGENT OVERVIEW

My agent keeps track of state through an array of 2-tuples with the first tuple signifying number of sheep and wolves as (sheep, wolves) for the left coast and the second tuple for the right coast. I also have an array of possible moves of 2-tuples in the form (sheep, wolves) designating the number of animals on the boat that will lead from one state to another ([(1,0), (0,1), (1,1), (2,0), (0,2)]).

The state is initialized with the arguments passed into "solve" and assumes that the animals start on the left coast, following the guards and prisoners example from class. Thus, the initial state is [(initial sheep, initial wolves), (0,0)] and goal state is all animals on the right coast ([(0,0), (init. sheep, init. wolves)]). My agent performs a breadth-first search on the state space, following the implementation in the Norvig AI book (Russell and Norvig, 2010, pp. 81-83), and starts with exploring from the initial state with the initial boat direction "right". The neighbor states explored from a given state are found by using the given direction and enumerating over all possible moves. Thus, they are states achieved by adding the move's boat capacity (number of sheep and wolves) to the current state's coast specified direction and subtracting the number from the other coast. The new neighbor states have an associated direction opposite of their parent state.

My agent, in its utilization of the generate-and-test method, uses a smart tester instead of a smart generator and generates all states legal and not. As I generate each state, I test for its "validity" and if it is a goal state. Following the project directions, a state is valid if both numbers for both coasts are greater than zero and the wolf number on each coast is less than or equal to the sheep number on either coast or the sheep number is zero. Closely following the referenced BFS algorithm (Russell and Norvig, 2010, pp. 81-83), if the state is a goal state, terminate and return the moves (stored in a dictionary) it took to get there. If not, if it is a valid state (and it has not been explored and not already to be explored), add the state-direction pair to explored states and to the to-explore FIFO search queue. Also, add the move to an accumulating list of moves it took to get to the state to the dictionary. If there are no more states to explore and the goal has not been reached, I return an empty list, signifying that no solution can be found.

2 QUESTIONS 2 AND 3: PERFORMANCE AND EFFICIENCY

To get some sense of performance, I edited my agent to return both the moves, number of states explored, and number of states generated. I edited the test() function to call the solve() function for all numbers of sheep and wolves from 0 to 24 inclusive. I called test() indirectly using Python's timeit module to time how long the call took to be completed (averaging around 0.15 seconds).

In general as shown in Figure 1 (generated using matplotlib), given a fixed number of sheep, both the number of states explored and generated followed a parabolic shape: an increase exponentially with an increasing number of wolves that begins to taper off around half the number of sheep, then an increasing decay. Initial conditions where the wolf number is greater (or from my observations, equal to when the wolf number is greater or equal to four) than the sheep number led to no solutions although since my generator is not smart and I do not test the initial state, some states will be explored from these conditions anyways. Further editing my test() function and tracing this peak value across increasing numbers of sheep (and half the number of wolves, rounded downward when non-integer), the growth in both tracked numbers of states is exponential (as seen in Figure 2).

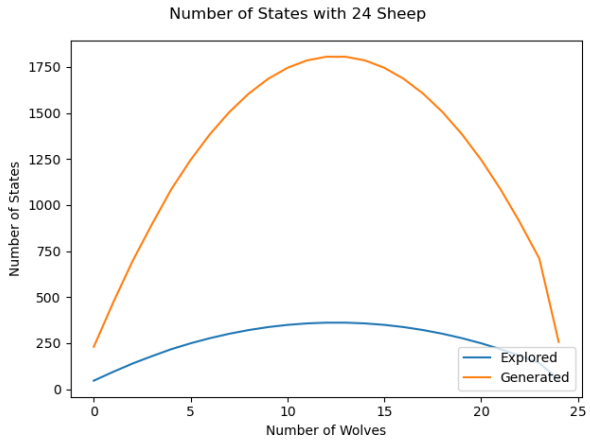

Figure 1—Number of states with 24 sheep.

Because the tester is smart and not the generator, my agent is not is as efficient as it could be. Greater efficiency could be reached with a smart generator and as stated earlier, impossible starting cases (since if the state is not illegal to start with, moves from it would quickly lead to illegal states) lead my agent

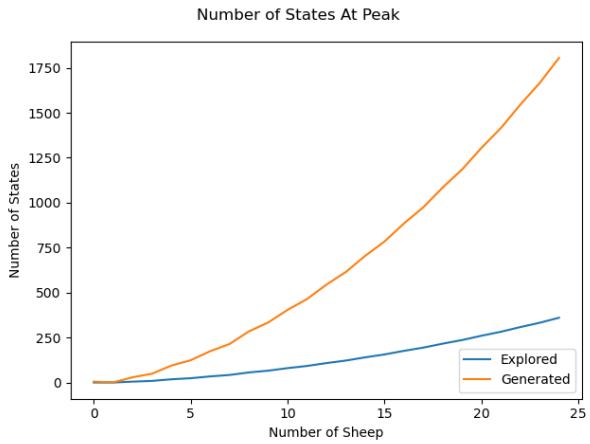

Figure 2—Number of states at peak.

to inefficiently generate and explore states it should not need to. As my agent manages to answer all questions correctly on Gradescope, I would say in terms of accuracy it performs well but in terms of "struggling" with cases I would say that this mentioned area would be a weak point.

In terms of efficiency, according to the Norvig AI book, BFS is $O(b^d)$ in (worst-case) time and space complexity where b is the branching factor (five in this case because of the number of possible moves) and d is the depth of the goal state and thus is exponential in the number of animals (given the starting state does not lead to an illegal state as the capacity of the boat is bounded) (Russell and Norvig, 2010, pp. 82-83), . From the same source, depth-first search has the same worst-case performance in time complexity but is O(bm) in space complexity where m is the maximum length path possible in the search space and thus fares much better than my BFS implementation as it is linear (Russell and Norvig, 2010, p. 87). Thus, I would not call my agent particularly efficient. However, I would surmise that my agent returns the optimal solution. In the Norvig AI book, BFS is stated as optimal "if the path cost is a nondecreasing function of the depth of the node" (Russell and Norvig, 2010, p. 82). Interpreting path costs as number of moves the boat has to take, each move to a new state yields a path cost of one and is thus nondecreasing.

3 QUESTION 4: OPTIMIZATIONS

My agent does not do anything particularly novel outside of the BFS implementation it is derived from. By its nature, I do not explore any state-direction pair

that has been explored already. Perhaps what can be considered an optimization is that I cache the results of the moves it took to get to a particular state-direction pair (instead of storing just a parent node) so that the results are saved and I do not have to backtrack to rebuild the move list upon algorithm termination, unlike the Norvig AI book's SOLUTION function (Russell and Norvig, 2010, p. 79), favoring speed optimzation over memory optimization. I do not explore neighbor states that fail the "validity" test but for further improvement, could augment the smart tester by applying this principle to the initial state as well as the observation that initial states where the wolf and sheep numbers are equal and both greater than four led to no viable solutions. My agent could do better in space complexity by switching to DFS as mentioned earlier (Russell and Norvig, 2010, p. 92). In fact, the Norvig AI book calls DFS "the basic workhorse of many areas of AI" (Russell and Norvig, 2010, p. 87). However, it also states that DFS does not necessarily return optimal solutions (Russell and Norvig, 2010, p. 86).

4 QUESTION 5: HUMAN COMPARISON

The agent solves the problem differently than I would as a human. It is human-like in that it does not bother exploring states that are "invalid" that thus could never reach a solution. However, I would not want to enumerate every possible neighbor state to a given state and furthermore, would not even bother generating states that had negative numbers of animals and then testing them (I however, found it much easier to code it out this way). If done in my head, maintaining that number of states requires a great deal of cognitive load. Even with pencil-and-paper, the branching factor is five (before testing) and I would not want to draw that out. Thus, I would do something akin to a best-first search, an "informed search strategy" that uses a heuristic to guide its move choices (Russell and Norvig, 2010, p. 92). With the (perhaps instinctual) background knowledge that moving more animals to the right coast than left (hopefully) gets me closer to my goal state of all animals on the right coast, I prefer to explore the moves satisfying that heuristic and upon failure of a test, backtrack until I find a better neighbor state-direction pair to explore. This somewhat resembles the Norvig book's recursive best-first search, especially so in that it is memory-bounded (Russell and Norvig, 2010, p. 99) and the limits of my memory (and paper space) are bounded. Using a memory-bounded algorithm agrees with the characteristics typical of AI agents as stated in lecture where agents "have only a limited computing power, processing speed, memory size" (Joyner and Goel, 2015).

5 REFERENCES

[1] Joyner, D. A. and Goel, A. (2015). *Characteristics of AI Agents [Video]*. Georgia Institute of Technology. Retrieved January 25, 2022 from https://www.youtube.com/watch?v=uwuZKEKpq8k.

[2] Russell, S. J. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. (Vol. 3, pp. 64-119). Pearson Education, Inc.