Mini-Project 1: Sheep and Wolves

Ivan Lopez
ILopez41@gatech.edu

Abstract—The Sheep and Wolves agent was written to solve the classic game where a boat must move the two types of animals from the left side of the river to the right. The paper will discuss the agent's method for solving the test, performance, efficiency, and human-like reasoning.

1 AGENT'S METHOD

The agent uses a Breadth-First Search (BFS) algorithm to traverse the possible states of the Sheep and Wolves game. The base pseudo-code for the BFS algorithm was found in Artificial Intelligence: A Modern Approach (Russell et al., 2021). It starts by adding a node with the initial state space onto the frontier (see Figure 1). Second, it pops off the next node on the frontier in a FIFO queue to check if the node has any sub-nodes it can explore (see Generating the New States). Finally, if the sub-nodes are valid (see Testing the New States), then it will add the sub-node to the frontier. This process repeats the last two steps until the goal state is found; the goal state is when there are no animals on the left side of the river and the boat is on the right side. If the goal state is not found, then an empty list is returned. If it is found, the steps to get to the goal state are returned.

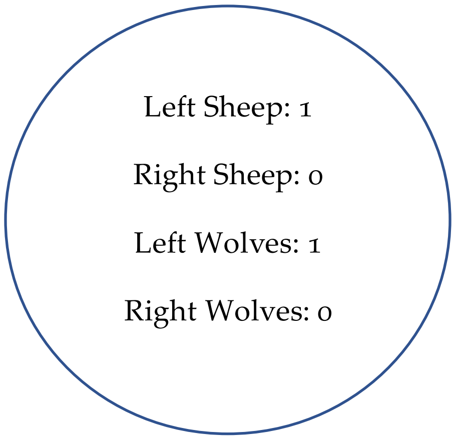

Figure 1—Initial Node with 1 Sheep and 1 Wolf

1.1 Generating the New States

To create new states or sub-nodes, simple conditional logic is used. The first conditional check is to verify the side of the river the current state (node) is on. If the node is on the left, then the next conditional test will be based on the Left Sheep and Left Wolves' values. If the boat is on the right, the next test will use the Right Sheep and Right Wolves values. Then, five independent conditional checks are made (seen in Figure 2). For each true result, a potential sub-node is created. Figure 3 shows an example of the checks for Figure 1: Initial Node. If the possible sub-node is validated successfully (see Testing the New States), it is added to the frontier as a true new state.

1. 2 or more Sheep
2. 2 or more Wolves
3. 1 or more Wolves and 1 or more Sheep
4. 1 or more Wolves
5. 1 or more Sheep

***Figure* 2**—5 Independent Conditional Checks

1. 2 or more Left Sheep: False, no possible sub-node
2. 2 or more Left Wolves: False, no possible sub-node
3. 1 or more Left Wolves and 1 or more Right: True, possible sub-node
4. 1 or more Left Wolves: True, possible sub-node
5. 1 or more Left Sheep: True, possible sub-node

***Figure* 3**—Example of 5 Independent Conditional Checks using Initial Node

1.2 Testing the New States

Two tests were created to validate the potential sub-nodes. The first is to verify that moving to the potential sub-node would not violate the Wolf Rule. The second is to ensure that the potential sub-node does not exist on the frontier or has already been checked. However, this was only a quasi-test, as it is part of building an efficient Breadth-First Search algorithm (Russell et al., 2021).

2 AGENT PERFORMANCE

Performance is being able to pass the test cases generated for the assignment in a reasonable amount of time for the number of animals being used. The agent performs very well against the cases used in the auto-grader. Not only did it pass all test cases, but it also found the optimal solution for each in less than 0.006 seconds.

2.1 Performance with Number of Animals Rising

The performance stays consistent even with a rising number of animals. As seen in Figure 4, the worst performance is around 86 Sheep and 86 Wolves, but the total compute time is still only 0.0056 seconds. It should be noted that performance testing only included up to 99 Sheep and 99 Wolves and that the tests included the same number of sheep and wolves in each test. This could be skewing the results. Memory performance, although not explicitly assessed, will decrease as the number of animals scales (see Agent Efficiency).

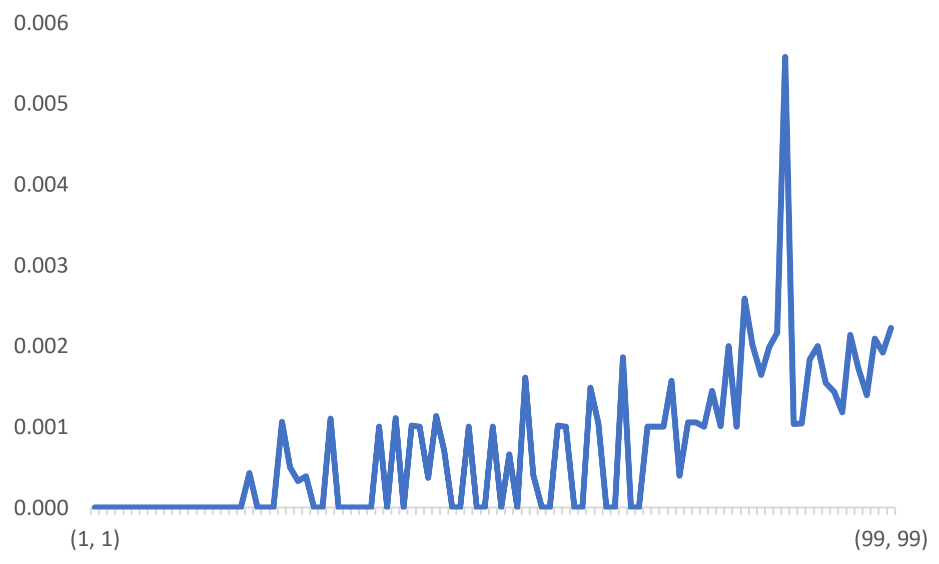

Figure 4— Time Performance

3 AGENT EFFICIENCY

The agent is semi-efficient because there is some room for improvement to ensure the code is fully optimized. For example, there is no need to track right animals and left animals. Simply subtracting left animals from the initial amount would have saved memory instead of just storing everything. This unnecessary memory usage is preventing the agent from being fully optimized. There are other similar areas of opportunity to clean up the code that I did not take due to the time constraint and being sick.

4 HUMAN-LIKE REASONING

The agent solves problems like that of a human. The pseudocode that I wrote before starting to program was based on the solving technique seen in the video lectures. The only profound difference is the agent solves the problems significantly faster. The agent solves the problem the same way that I solve it. That is what made debugging the agent easier; I could confirm each new state with my worked-out solution on paper.

4.1 Cleverness

The agent uses vanilla human-like reasoning to solve the answer. The only "clever" aspect of the agent is that it generates the sub-nodes while searching the frontier. It is a stretch to call that clever as that is the way to build the trees in most graph problems after lower-level undergraduate classes.

5 REFERENCES

1. Russell, S. J., Norvig, P., & Chang, M.-W. (2021). Artificial Intelligence: A Modern Approach. Pearson.