Mini Project 1: Sheep & Wolves

Anandita Chopra
achopra49@gatech.edu

***Abstract*—This document contains the journal for the first mini project – Sheep & Wolves. The project serves to build and solidify our understanding of a general AI problem-solving method called Generate and Test by implementing an optimal solution for the problem at hand based on the same.**

1 INTRODUCTION

In this project, the requirement was to build an agent to solve the Sheep and Wolves problem for any number of initial wolves and sheep (*Mini-Project 1 | OMS CS7637*, 2022). Here, we have a certain number of sheep and wolves on the left bank of a river and we intend to move them to the right bank with a constraint that at most two animals and at least one animal can board the boat while moving to either bank and wolves can never outnumber the sheep on either side of the bank. This problem can be solved via **Generate and Test**, a powerful problem-solving method in AI that comprises generating a set of solutions to a given problem and testing their efficacy to address the problem (*Lesson: Generate & Test*, 2022). The details of the agent workflow with special consideration on getting the solution efficiently are covered in Section 2. To evaluate the function and non-functional aspects of the implementation, we perform a comparative analysis using different values of sheep and wolves (Section 3). Lastly, we try to draw a parallel with how this agent is similar or different to the human approach for solving the problem (Section 4).

2 AGENT WORKFLOW

My agent implements a Bread-first-search-like approach to explore all possible valid (via smart generator) and productive (via smart tester) states for the given problem. Here, the **state** is defined as the number of sheep and wolves on either side of the bank, the direction where the shepherd/boat is present, the move in consideration and the parent state. The agent starts exploring from the initial state (with sheep and wolves or left side of bank) until either it reaches the goal

state (where all animals are on the right side of the bank, green box) or exhausts all the possible states in case of no solution (red box).

High Level Workflow of the agent is described in the flowchart below:

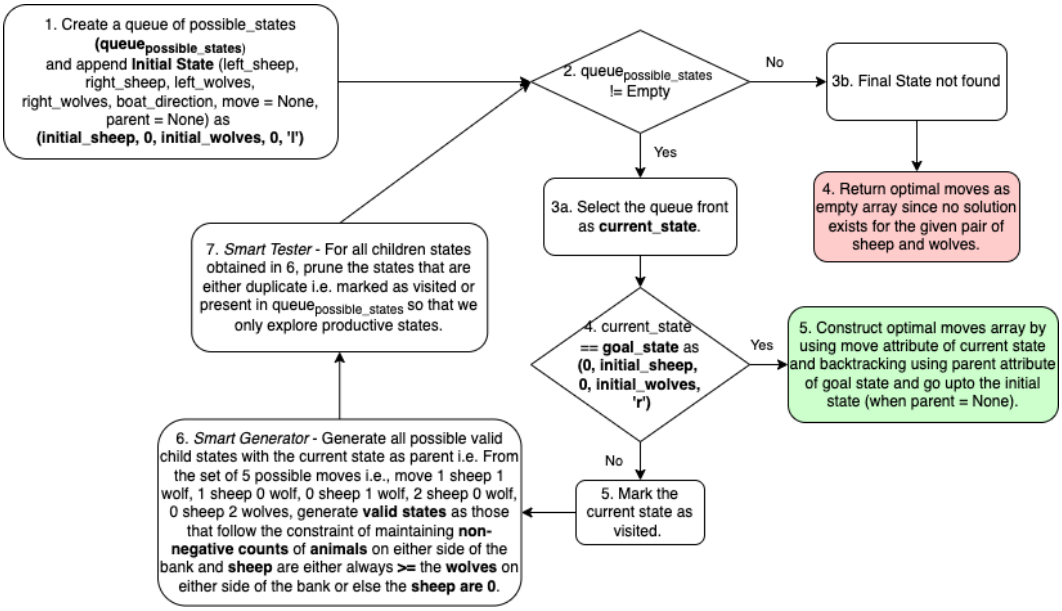

Figure 1 — High level Workflow

3 CORRECTNESS AND PERFORMANCE EVALUATION

3.1 Correctness Evaluation

Since my implementation followed BFS as the underlying principle, it is expected that the agent should be able to explore the entire search space and obtain a correct solution. Further, the logic in testers to avoid exploring the duplicate and the already explored states should ensure that the agent solves the problem optimally for every pair of wolves and sheep. To test this hypothesis and check whether the agent struggles with any test cases, I tested all possible combinations of count for each type of animal up to 25, with sheep always greater than or equal to the number of wolves, and found the optimal set of moves in cases where solution existed and empty array in cases of no possible solution. Optimal moves count for a subset of the valid combinations for animal count i.e., count as sheep and wolf up to 5 is depicted in the figure below:

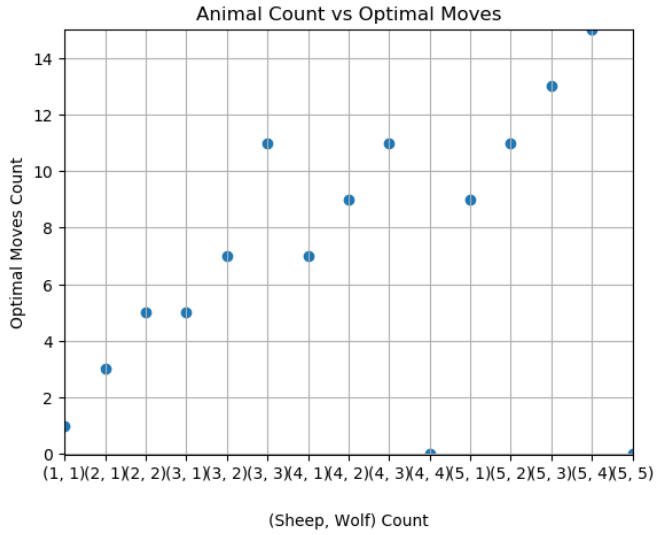

Figure 2 — Correctness Evaluation Experiment: Animal Count vs Optimal Moves

We can clearly see that we got optimal moves in all cases where the solution existed like 1 move for case of 1 sheep and 1 wolf, 11 moves for 3 sheep and 3 wolves, 13 moves for 5 sheep and 3 wolves and so on. Also, for cases, where there was no solution like 4 sheep and 4 wolves or 5 sheep and 5 wolves, we can see optimal moves count was 0 i.e. we got empty array for optimal moves.

3.2 Performance Evaluation

As already described above, I had implemented optimizations in both the generator and tester logic to optimally generate only valid states and avoid testing any duplicate/already explored states. Further, since the state at any point was an Object, I had overridden the __eq__ and __hash__ implementations for faster comparison between the objects which would come in picture in comparison for validating or checking duplication among different states. Given these optimisations, the agent should be fairly efficient. That said, as the number of animals increase, there would be increase in the time taken to solve since the search space would increase exponentially in the worst case. The worst-case time and space complexity would be exponential.

To test the performance change with the rise in animals, in the above experiment to run the agent for all possible sheep and wolves count up to 25, I captured the

time taken for each combination as well. Time taken for a subset of the valid combinations for animal count is depicted in the figure below:

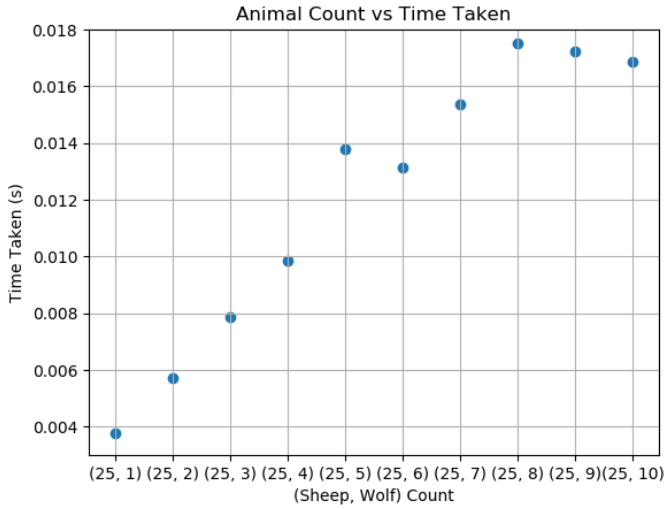

Figure 3 — Performance Evaluation Experiment – Animal Count vs Time Taken (in seconds)

4 COMPARISON OF AGENT WITH HUMAN

As mentioned above, my agent followed a breath-first-search-like approach with additional logic to smartly generate and test different states. It created an in-memory graph with the possible states space to find a solution. As a human, I can say that, even I tried to solve this problem by visually creating a graph with the possible states space like the agent. But one different thing is I ended up solving this problem with depth-first-search. Had I implemented my agent to use DFS, likely that would be faster than the current approach. Also, as humans, we may explore some states preferentially based on our perception of certain states as more productive than others.

5 REFERENCES

1. *Lesson: Generate & Test*. (2022). Ed — Digital Learning Platform. https://edstem.org/us/courses/16992/lessons/27588/slides/156766
2. *Mini-Project 1 | OMS CS7637*. (2022). Lucylabs. https://lucylabs.gatech.edu/kbai/spring-2022/mini-project-1/