Mini Project 1: Sheep and Wolves

Tess Greene
tgreene34@gatech.edu

1 INTRO

The Sheep and Wolves problem is a river-crossing problem, identical to the Guards & Prisoners problem, also known as the Missionaries & Cannibals problem and Jealous Husbands problem. There are two groups that need to cross a river via a boat that can only carry two people, and one group can never be the minority on either side of the river. In most cases there are three people of each group, but for this project I set out to find how to solve it for an arbitrary number of each, excluding situations where wolves outnumber sheep to begin with.

1.1 Human Approach

To teach an agent how to approach the problem, I wanted to see how I did so first. My goal was to find ways to reduce the problem as much as possible, or at least outline to myself the human cognitive process. Note: test cases will be denoted by (number of sheep, number of wolves), while transport procedures will be denoted [(sheep moved, wolves moved)].

I started with the basic cases: (1,1), where there was one sheep and one wolf; (2,2), where there were two of each; building up to (7,3), 7 sheep and 3 wolves. I always started with either sending two sheep across the river [(2,0)] or one of each [(1,1]).

1.2 Patterns

Patterns began to appear, though it took some trial and error. As I worked through the cases we were told would definitely be tested, I thought the total number of steps in an optimal solution would be (2 * sheep # + wolf #). This held for (4,3), (5,3), (6,3) and (7,3), with 11 moves, 13 moves, 15 moves, and 17 moves, respectively. However, when I began cases with higher values, this broke down. I attempted (8,5) by hand and found it took 23 steps. I will explain why in the next paragraph.

The other pattern I was finding was how the animals were optimally transported. I will start with cases where there were two or more sheep than wolves. I would

start with [(2,0)]. This meant I had to send one sheep back [(2,0), (1,0)]. Then I would take one of each across [(2,0), (1,0), (1,1)] and then send a sheep back again, [(2,0), (1,0), (1,1), (1,0)]. This pattern kept repeating until I ran out of wolves, where I would then begin ferrying sheep across by repeating [(2,0), (1,0)]. To understand how I was getting the optimal number of steps, I began counting how many of each river crossing type occurred. This is where I cracked why (8,5) was 23 steps. There were always (sheep # -1) steps of [(2,0)], (wolf #) steps of [(1,1)], and (sheep # + wolf # -2) steps of [(1,0)]. For (8,5) this meant (8-1)+(5)+(8+5-2) = 23, and it held for the previously attempted test cases where (sheep # >= wolf # +2). I now knew the optimal pattern and how many steps it should take.

For cases where there was one more sheep than wolves, there was a similar but distinct pattern. I would start attempts with [(1,1)], then have to send a wolf back [(1,1), (0,1)] to ensure the sheep were never outnumbered. Then I would bring 1 of each across again, but the following trip back I could send a sheep [(1,1), (0,1), (1,1), (1,0)]. This pattern of four steps repeated again and again in different cases. I also found that the total optimal steps equation still held.

For cases where the number of sheep and wolves were equal, I found the answer early. I wanted to know more about the problem and see what patterns had already been discovered. I found a paper that explained, "With four or more couples, however, it's impossible to accomplish the crossings under the required conditions" (Peterson, 1996). Therefore, I did not have to worry about (5,5) or (4,4) or any other equal pairs greater than (3,3). Now the problem was reduced to checking only the numbers of sheep and wolves and generating the corresponding answer.

2 CREATING THE AGENT

Before I had begun doing the problems by hand, I was concerned by what might be necessary to build an agent capable of solving this problem. Would it need to build a decision tree? Would it need an evaluation function? How many nodes might it use? Would it be recursive? I thought it would have to be fairly knowledge-intensive. However, once I had figured out the patterns, I knew my agent could succeed with a weak method. I called it Shepherd.

2.1 How Shepherd works

First, I gave Shepherd the base cases I had already solved. It checks if the initial sheep and initial wolves are equal to certain pairs: (1,1), (3,3), (5,3), etc. I hard-coded the answers to those. It also checks if initial sheep is equal to initial wolves, and if those are equal to or greater than 4. If so, it returns an empty list.

If none of those conditions are true, Shepherd calculates the total steps necessary for an optimal solution based on the number of sheep and wolves. It creates a list that long, where its elements are just a list of ascending numbers equal to its length.

If the difference between the numbers of sheep and wolves is greater than 1, Shepherd fills the list by placing tuples in certain patterns. It starts with placing [(2,0)] as the first element, and in every fourth element from there. It places [(1,0)] in the 2nd element, and in every 2nd element from there. Shepherd then puts [(1,1)] in the third element, and in every fourth element from there, *until* it runs out of wolves. When Shepherd has run out of wolves to move, it starts placing [(2,0)] from that point in the list, and in every other element from there. For example, in the case of (6,3), Shepherd created the following pattern:

[(2, 0), (1, 0), (1, 1), (1, 0), (2, 0), (1, 0), (1, 1), (1, 0),

(2, 0), (1, 0), **(1, 1)**, (1, 0), (2, 0), (1, 0), (2, 0)]

Figure 1 — A solution to (6,3). It takes 15 moves. It finishes transporting wolves at step 11 and begins ferrying only sheep.

If there is only a difference of 1 between sheep and wolves, Shepherd starts the pattern with [(1,1)] and repeats that every other step. It then moves a wolf back in the 2nd element, [(0,1)] and repeats that every fourth step. In the fourth element, Shepherd moves a sheep back [(1,0)] and repeats that every fourth step. Since it knows that there is only 1 less wolf than sheep, it does not have to check when wolves run out to start moving only sheep.

Shepherd does not check if it's correct; it is only a generator without a tester. I only checked Shepherd's solutions to my handwritten ones and on Gradescope. It successfully completed every problem.

2.2 Performance

Shepherd is not clever, but it gets the job done. I could not find any cases where Shepherd "struggled" because it was simply repeating patterns a certain number of times. Calculation times were graphed below.

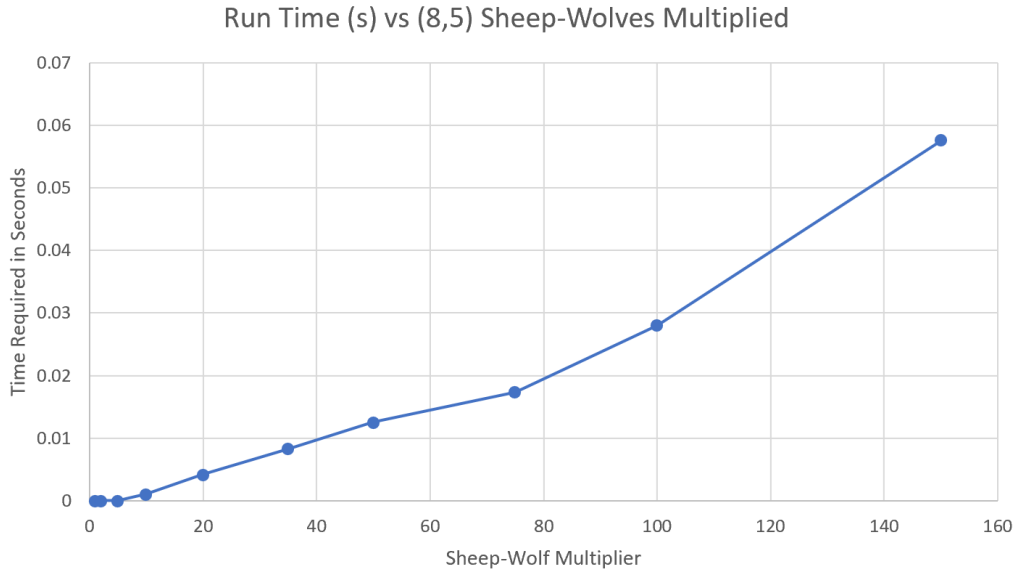

Figure 2 — Calculation times for different values of Sheep and Wolves as multiples of (8,5), for 1x, 2x, 5x,10x, 20x, 35x, 50x, 75x, 100x, and 150x. To clarify, case (800, 500) took <.03 seconds to complete. It is important to note that the runs above (8,5) are not verified to be correct. For reference, it took the human writer >1 minute to complete (8,5).

2.3 Comparison

Shepherd does think in a human way, provided knowledge of patterns a human discovered. I do not think in game trees and nodes. I find patterns at the small scale, check them at slightly larger scales, and hope they work at much higher scales. Shepherd repeats patterns as many times as I ask it to. It is an accurate (to the extent of tests verified) generator. Shepherd cannot learn or reason, but it can absorb and deliver human knowledge at much higher speed than any human.

3 REFERENCES

1. Peterson, I. (2003, December 12). Tricky crossings. ScienceNews. Retrieved January 12, 2022, from https://www.sciencenews.org/article/tricky-crossings