Mini-Project 1: Sheep & Wolves: CS7637

Josh Adams
jadams334@gatech.edu

1 HOW DOES YOUR AGENT WORK?

My agent works by taking advantage of the Breadth First Search (BFS) algorithm. The pseudocode is provided below and further explanation after.

```
"""
   Pseudocode

   def Agent_BFS(self, root):
          Initialize STATE_QUEUE with the root
          Initialize VISITED_STATES
          Initialize flag SOLVED to False
          While STATE_QUEUE not empty and not SOLVED:
              GENERATE_STATES from STATE_QUEUE.POP
              Iterate over generated states:
                  If state is solution:
                      SOLVED is TRUE
                  Add state to VISITED_STATES
                  Add state to STATE_QUEUE

   def GENERATE_STATES(some_state):
         Initialize MOVES (1, 0), (2, 0), (1, 1), (0, 1), (0, 2)
         Initialize STATE_CONTAINER
         Iterate over MOVES:
            Create new state base on move
            Validate State
            if generated_state valid:
               Add generated_state to STATE_CONTAINER
         return STATE_CONTAINER
"""
```

The agent starts off by generating the root state. This is done when the initial sheep and initial wolves are passed in. Each state is a tuple containing ((left-side sheep, right-side sheep), (left-side wolves, right-side wolves), *Pointer to parent state, Move used to arrive at that state, The boat location). The agent then initializes a queue with the root state, to hold states which have yet been processed, and a container to keep these states which have been visited. It then pops the first element from the state queue, add that state to the visited container and begins processing.

To generate new possible states the agent takes the state being processed and iteratively generates states based on the five possible moves. The possible moves are (1, 0), (2, 0), (1, 1,), (0, 1), (0, 2), which correspond to the (sheep, wolves) moved from that side to the other. As there are only five possible moves, there can be at most 5 newly generated states. Those states are then validated in many ways. First the agent checks if the newly generated state has been visited previously, if the state has been visited, that newly generated state is thrown out. The next validation is to check the generated state if there are more wolves than sheep on any side, when at least one sheep is present on that side. The second to last validation is of the limits for the animals. The newly generated state cannot have less than 0 or greater than the initial count for that animal on either side. The final check is that the new state cannot have a single animal on the side with the boat. If any of these validations fail the generated state is thrown out.

These newly generated states are then checked to see if they are the solution to the problem, if they are the problem ends and the result is returned. Otherwise, the states are added to the visited state container and the process starts again.

2 HOW WELL DOES YOUR AGENT PERFORM?

My agent performs well on every problem I have thrown at it. Intuitively speaking the most time-consuming problem will be the problems without a valid solution. The reason is that the agent must process all the states and does not have the ability to end early, like it would if it had found a solution. I have not come across problems that my agent struggles to solve but I would expect the agent to be able to solve or return some result in a finite amount of time.

3 HOW EFFICIENT IS YOUR AGENT?

My agent became very efficient once a final bug was resolved. This final bug was caused by adding states to the visited states container once they were processed. The solution was to add the states to the visited container once generated. My agent went from taking almost one second to solve a seven sheep and three wolf problem to 0.001 seconds.

As the number of animals increase, so does the runtime for my agent. Figure 1 is heatmap of runtimes for my agent and various number of sheep,

wolve combinations. Most of the black area above the 'heat' is due to those combinations being initialized with more wolves than sheep, which would be an invalid state.

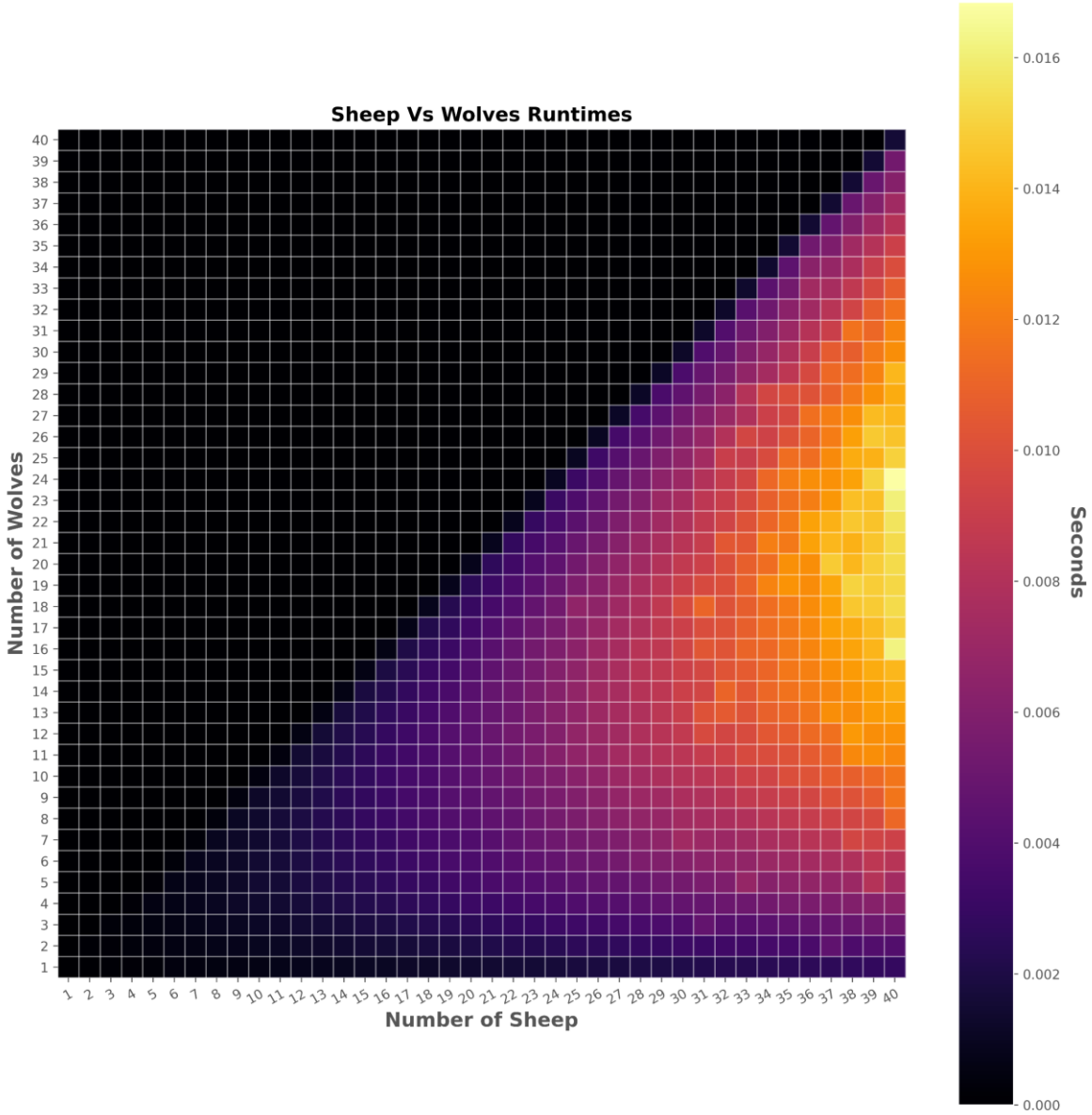

Figure 1 — Shows the average runtimes for various combinations of sheep and wolves in the problem.

Figure 1 clearly shows the relationship between the number of animals used in the problem and the run time. One very interesting thing in Figure 1 is that the most time-consuming combinations are not those with the largest total animals.

For example, look at the problem with forty sheep and thirty-nine wolves. The runtime was very similar to having forty sheep and two or three wolves. The most time-consuming combinations were having more than thirty-five sheep and between ten and thirty wolves in the problem. The table below shows some of the run times for larger problems such as 0.0930 seconds for a problem with 401 sheep and 400 wolves. The table is just to provide more evidence of efficiency.

Sheep	Wolves	Runtime (seconds)
11	10	0.0015
51	50	0.0070
101	100	0.0140
201	200	0.0270
401	400	0.0930

4 DOES YOUR AGENT DO ANYTHING PARTICULARLY CLEVER TO TRY TO ARRIVE AT AN ANSWER MORE EFFICIENTLY?

My agent does not do anything particularly clever to be more efficient when solving the problem. It uses the standard BFS algorithm.

5 HOW DOES YOUR AGENT COMPARE TO A HUMAN?

I believe that my agent does solve the problem is a similar fashion as a human would. One major difference is speed, I would imagine a human would not be able to solve problems more difficult than three sheep and three wolves, in less than a second. For the computer, this is trivial and can solve much more complicated in less time, e.g., 16 sheep and 15 wolves in 0.0025 seconds. The agent does solve the problems in the same ways as I would.

6 REFERENCES

1. https://en.wikipedia.org/wiki/Breadth-first_search
2. http://lucylabs.gatech.edu/kbai/spring-2021/mini-project-1/