Miniproject 1:
Smart Agent to solve the wolves and sheep problem

Andres Cabezas
aquicano03@gatech.edu

Abstract—The following inform will give a summary of the behavior and design of the agent. A time performance report. A list of some clever decisions that it makes to generate the cases. And finally, a brief analysis of the comparison between the agent and the human behavior against this problem.

1 THE AGENT'S DESIGN

The design of the agent, followed the "Generate and Test" approach. The agent has 3 main modules: The **"Case Generator"**; The **"Tester"** and the data structure. Additionally, a **Find Steps** function that throws the final list of trips (see Figure 1).

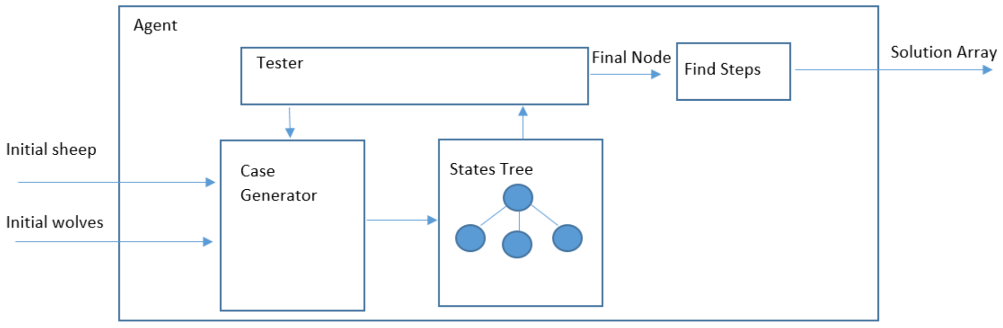

Figure 1—Design of agent for the wolves an sheep problem

1.1 Case Generator

The case generator is in charge to create all the states possible from a current state. For this purpose, the generator is going to check the following conditions:

1. If all the wolves and sheep crossed the river -> Returns empty
2. If a certain configuration of sheep and wolves is not valid -> Returns empty
3. If the direction of the trip is going. The possible states are reduced to 3 because the idea is to be able to transport all the animals in the least amount of steps possible. -> Returns state node with trips (1,1),(0,2) or (2,0)

4. If a certain side of the river has less animals than the boat capacity, discard the states that transport more than 1 animal -> Returns state node with trips (1,0), (1,1) or (0,1).

After have checked all these conditions, the generator returns a States Tree Node array. The parent node for each one of these nodes it's the one before apply the possible trips.

1.2 Tester

The main task of the tester is to store each of the nodes evaluated and verify that there is no repeated state node.

1.3 Data Structure: States Tree

The data structure selected to store the states was a non binary tree, where the root node it's the initial state of the problem.

1.4 Find Steps function

The function is in charge to give the answer array. And, the method used was inspired in a DFS approach. The Tester is going to have all the possible states stored in a dictionary called **"saved_states"**, which the algorithm uses to access the node desired. The algorithm is going to access the last node, push the trip and it is going to visit each of the parents of the node doing the same operation until arrive at the root node.

1. The algorithm will look up the final step which will be represented by the number of sheep = 0, number of wolves = 0, and the direction of the trip = -1. Where, the direction of trip = 1 will mean that the following trip would be cross the river, and direction of trip = -1 will be going back in the next trip. Hence final_step = (0,0,-1).
2. If final step is stored in states_saved:
   (A) Initialize current_node = states_saved[final_step]
   (B) While current_node has a parent:
      (a) Push current_node.trip into list_of_states
      (b) Current_node = current_node.parent
   (C) Returns list_of_steps.
3. If final steps is not stored in states_saved return an empty list_of_steps.

The flowchart of the algorithm can be seen in the Figure 2

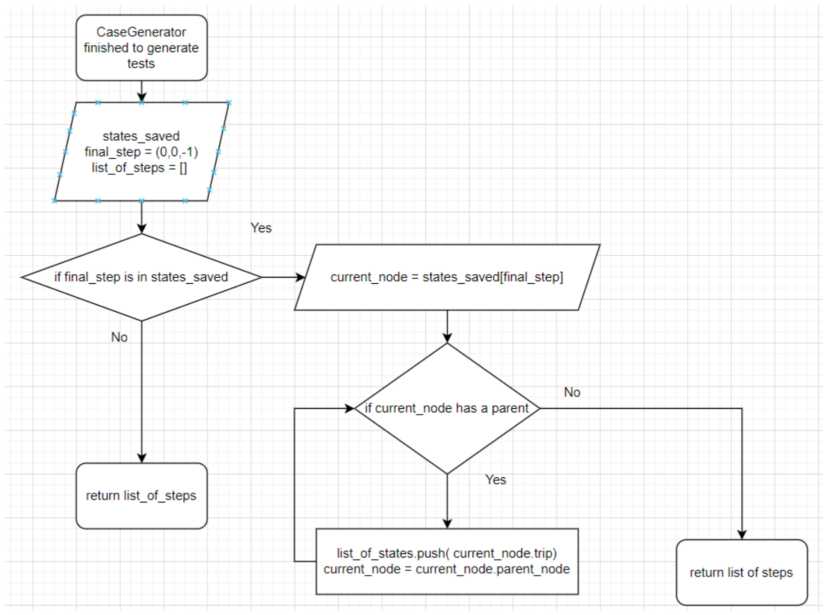
CaseGenerator finished to generate tests
states_saved
final_step = (0,0,-1)
list_of_steps = []
if final_step is in states_saved
Yes
current_node = states_saved[final_step]
No
if current_node has a parent
No
Yes
return list_of_steps
list_of_states.push(current_node.trip)
current_node = current_node.parent_node
return list of steps

Figure 2—Algorithm of the find steps function

2 PERFORMANCE OF THE AGENT

The agent's performance was verified with Gradescope. For 20 test cases, the agent solved each problem in the least amount of steps or the optimal path. The agent took more time to give the final solution for some cases. This behavior could be related to the possibilities space, in Figure 3 the case 13 sheep and 7 wolves take up to 0.012 seconds to solve the problem. Something similar could happen with the case of 20 sheep and 7 wolves.

3 EFFICIENCY OF THE AGENT

The agent can solve each one of the problems in less than 0.1 seconds (see Figure 3). The time complexity of the agent expected is given mainly by the **Case Generator** and the **Find Steps function** which is O(log(p)) and O(log(p)) respectively. Where p represents the total number of possibilities.

4 CLEVER DECISIONS MADE BY THE AGENT

(A) There are five combinations, two out of them, will not work to meet the goal of taking all the animals to the opposite riverside. These two discarded options are, number one; just one sheep on the boat, number two; just one wolf on the boat. The optimal path, will always consider two animals in the

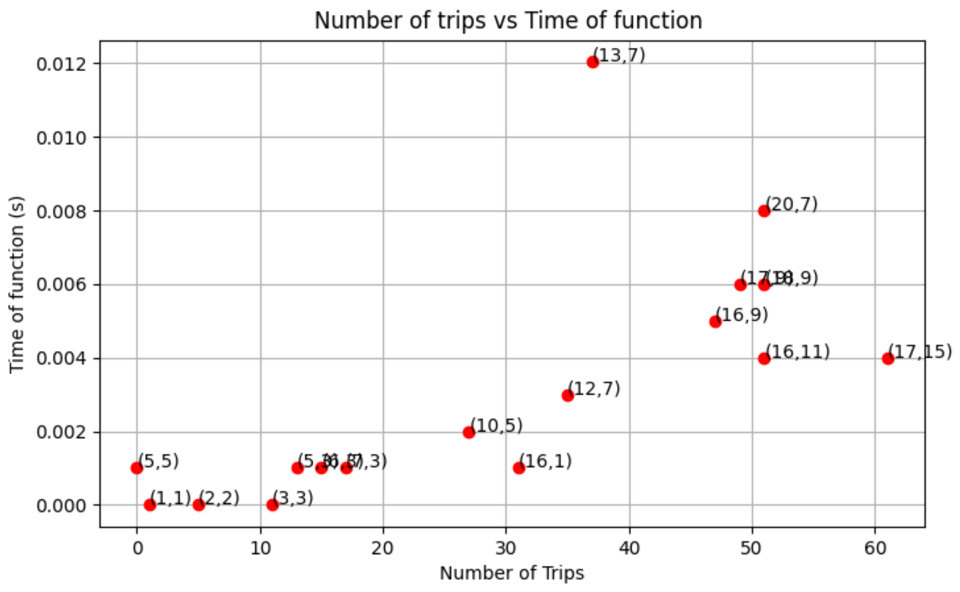

Figure 3—Number Of Steps vs Time of the function

ongoing trip.

(B) The agent stores all the state nodes in a dictionary. Therefore, to find the final state where there are no sheep and wolves in need to cross the river. The agent can find in a constant time complexity the node that holds the final step, which is the key to the efficiency of this agent.

5 AGENT REASONING VS HUMAN REASONING

The agent solves the problem in a different order than humans in this specific case myself. For instance, the solution given by the agent for the 2 sheep and 2 wolves case is the following: [(1, 1), (1, 0), (2, 0), (1, 0), (1, 1)]. In my logic, I will not choose to select as the first step to transport 1 wolf and 1 sheep. I would choose to select as a first step the (0,2). Because, I already have the bias of the lesson where was solved the case 3 sheep and 3 wolves. My solution for this problem is: [(0,2), (0,1), (2,0), (0,1), (0,2)], where I see that I tend to choose the option where the wolves cross the river in the first place. The main difference is that the agent chooses the solution without any previous learning, and its performance is not affected by external sources. Compared to humans that can struggle, even with finding an approach to solve the problem.