Mini Project 1 Journal

Juho Youn
jyoun34@gatech.edu

1 HOW DOES YOUR AGENT WORK?

In this section, I will make a brief description of my SemanticNetsAgent class. Then, I will address how my intelligent agent generates new states of the Sheeps and Wolves problem as well as how it tests whether a new state is valid.

1.1 What's inside SemanticNetsAgent Class

- **__init__(self)** function : This function imports Queue Library and initializes some member variables such as a queue(**q**) to put new states into, a list to store the answer (**list_answer)**, a list to record previous states (**record_of_states**) and etc.
- **State_Node()** Class : It includes member variables such as the number of sheep on the initial side (**left_sheep**), the number of wolves on the initial side (**left_wolves**), the number of sheep on the other side (**right_sheep**), the number of wolves (**right_wolves)** on the other side and the location of the boat (**boat_located_right**), the move made from the previous state to transform to this state and a pointer to previous state.
- **solve(self, initial_sheep, initial_wolves)** function : This function initializes the goal state (**goal_state_tuple**) with its inputs. Then, it keeps calling **generator** function until it reaches the answer or it determines the problem is not solvable. This function returns the **list_answer** member variable.
- **check_valid_move(self, previous_node, move_to_do)** function : This function calls **get_new_state** function to get a new state. It returns **True** if a new state from the previous node is valid and returns **False** if it is not.
- **get_new_state(self, previous_node, move_to_do)** function : It returns a new state as a tuple **(left_sheep, left_wolves, right_sheep, right_wolves, boat_located_right)** which is calculated based on its inputs.
- **generator(self, previous_node, move_to_do, goal_state_tuple)** function : This function calls **check_valid_move** function to see whether a new state is valid. If a new state is valid, it creates a new **State_Node** object, puts it into **q** and appends the state tuple of this node to **record_of_states.** This function also checks whether the new state is equal to the goal state

(**goal_state_tuple**). If that is the goal, this function calls **get_path** function to modify **list_answer** and tells the intelligent agent to stop generating new states.

- **get_path_backward(self, goal_node)** function : This function returns a list of moves that will solve the problem in reversed order.
- **get_path(self, goal_node)** function : This function returns a list of moves that will solve the problem. It calls **get_path_backward** function to get a list of backward order of moves and reverse it to return the forward order of moves.

1.2 How does it generate new states?

In my code, a **State_Node** object corresponds to a **knowledge representation** in a **semantic network**. A **State_Node** object has all information that constructs a **knowledge representation**.

Then how does my intelligent agent generate new states? First, when **solve** function is called, the initial **State_Node** object is created and put into **q,** which is a queue data structure. Then, my intelligent agent dives into a **while loop**. In the loop, my agent pops one **State_Node** object from the **q** and generates possible and valid states. To generate the next layer of states, my agent tries five different moves including sending two sheep to the next side, sending two wolves to the next side, sending a sheep to the next side, sending a wolf to the next side and sending a sheep and wolf to the next side. Then, my agent calls **check_valid_move** function to check if a new state is valid. If a state is valid, my agent creates a new **State_Node** object and puts it into **q**. The **State_Node** objects in **q** will be used again to generate new states in the next layer.

1.3 How does it test them?

My intelligent agent tests if a new state is valid right before it generates a new **State_Node** object in **generator** function. This **generator** function is a **smart generator** that contains a tester function which is called **check_valid_move.**

Then how **check_valid_move** works? First, it checks if a new state follows the rule of the game, which is "There should not be more wolves on each side of river". It also checks if a new state has ridiculous values such as negative numbers of sheep or wolves on each side.

Second, **check_valid_move** checks if a new state has already occurred previously. This is possible because my agent records a state of **knowledge representation** whenever a **State_Node** object is created.

2 PERFORMANCE & EFFICIENCY

In this section, I will address how my intelligent agent performed on various test cases and how the efficiency changes as the number of sheep and wolves increase. In this section, "performance" refers to the accuracy of my intelligent agent and "efficiency" refers to the speed of my intelligent agent.

2.1 How well does your agent perform?

My intelligent agent passes all the test cases with full credits.

2.2 Does it struggle in any particular cases?

No. My intelligent agent works well in all cases.

2.3 Does your agent do anything particularly clever to try to arrive at an answer more efficiently?

There were two features that I considered when I designed my intelligent agent.

First, I used **Queue** as a data structure to store the created states. In this way, my intelligent agent can create each layer of states in the increasing order. For example, my intelligent agent does not create any states of the third layer before it completes to create all possible states of the second layer.

Second, my intelligent agent tests the validity of new states before they are created. Whenever a new state is created, it is recorded. This prevents my agent from creating and deleting unnecessary states.

2.4 How does its performance and efficiency change as the number of animals rises?

From the data, it seems that the complexity of my intelligent agent is on some level between **O(n)** and **O(n^2)**. In figure 2, when the number of sheep and wolves are small, the graph looks irregular. In figure 1, when the number of sheep and wolves rises from 1 to 1000, the graph looks like a sequence of linear

lines but the slope increases as x-value increases. It is anticipated that the slope increases much faster as x-value rises and the efficiency will drop much more.

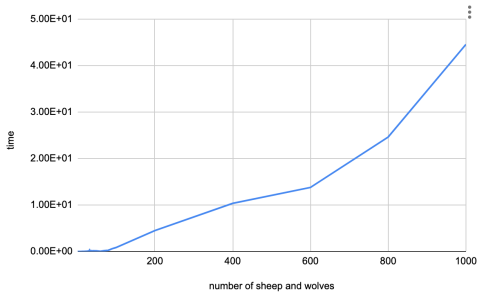

Figure 1— 1 ~ 1000

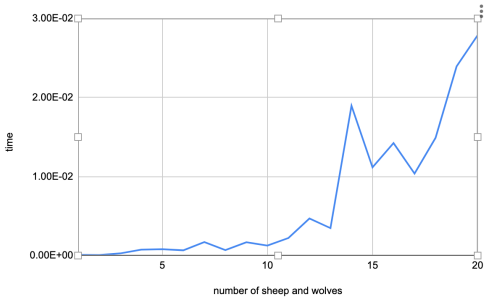

Figure 2— 1 ~ 20

3 HUMAN CONNECTION

3.1 How does your agent compare to a human? Does your agent solve the problem the same way you would?

I used my own problem solving approaches as a human when I designed my intelligent agent. My intelligent agent solves the problems almost in the same way I did. It starts with an initial state and generates a second layer of states from the initial state. Then my agent focuses on one of the states in the second layer and generates possible states from it. Intelligent agent keeps focusing on the other states in the second layer and generates all possible states of the third layer. It keeps generating next layers until it reaches the final state or it cannot generate states any more.

However, there is a difference between my strategy as a human and that of my agent. I utilized the **generate and test** method when I solved the problems on my own. Testing a new state before I create a **knowledge representation** was hard for me. I had to figure out if a new state follows the rules of the game or if a new state already occurred before without seeing any diagram. As a human I prefer drawing a **knowledge representation** first and testing if it's valid later. On the other hand, my agent uses the **test and generate** method. This term is not used in our lecture but I just named it as test and generate because my agent has a **smart generator** which tests the states before it generates a State_Node object that corresponds to a **knowledge representation**. The order of testing and generating is the only difference between my approach as a human and the design of my agent.