Jordan Gilman, Oleksii Zadniprianyi

Programming Assignment #1 report

Methodology

We created PuzzleState class that contains the following attributes:

- lb_chickens - number of chickens on the left bank
- rb_chickens - number of chickens on the right bank
- lb_wolves - number of wolves on the left bank
- rb_wolves - number of wolves on the right bank
- boat_left - boolean, is True if boat is on left bank, else on right

We also created Node class that contains following attributes:

- state - the puzzle state corresponding to this node
- parent - the parent of this node
- action - the action taken to reach the state of this node
- path_cost - the cost of all actions taken to reach the state of this node

Each Node has its state and contains the representation of some particular action, which is usede for finding optimal solution.

Three important functions that are used throughout our algorithms are expend_node(), is_valid_action() and gen_ps().

- expand_node() – expands current node, if there are future states
- is_valid_action() – checks the puzzle state for evidence of an invalid action
- gen_ps() – creates a new state based on the action applied
- is_goal_state() – check if goal state is reached

We created is_in_state_history() function to check for duplicates of states already performed in state history. We found that it greatly optimizes our runtime because it removes redundant loops while searching for the goal state. Because of this check we did not need to specify the depth limit for DFS as it was able to find solution quickly. Whereas without it, we may have never found the solution.

With the help of these functions and classes we developed algorithms based on pseudo code from the book. In each of the algorithms we would create instance (initial node) of Node class, expand_node() to see its children, check is_valid_action() and perform it, gen_ps() based on taken action, and call is_goal_state() to see if we reached the goal.

For A* our cost function f(n) = g(n) + h(n) where g(n) is the path_cost of the current node and h(n) is the estimated cost from the node to the solution. We know that for any number of animals, m, still on one side of the bank, it will take *at least* 3m/2 more actions to get to the goal state. This is because it takes two actions to move one or two animals, but we will always prefer to take two animals to the goal side and one animal away from the goal side in order to minimize the number of moves. This heuristic is admissible (i.e. optimistic) because it always assumes we can take two animals whereas there may be cases where we can only take one based on problem-specific constraints.

Results

	BFS	**DFS**	**IDDFS**	**A***
Test 1	26/11	11/11	23/11	11/11
Test 2	368/31	45/31	322/31	45/31
Test 3	NA	1214/953	NA	1214/953

Where: nodes expanded/solution path, NA – were not able to find because of long run time

Discussion

We expected for the BFS and DFS to have similar performance because both algorithms have same time complexity. However, DFS was outperforming BFS by a lot. This is an interesting behavior. We think that is because the longer the path the higher chance of it becoming a solution. In other words, DFS finds solution, but it is not the optimal one. Moreover, our optimization using is_in_state_history() function, which limits redundant loops has greater effect on DFS, allowing it to explore in depth faster. Before it was implemented, BFS and DFS ran equally long for the third test case. IDDFS algorithm takes longer than DFS as it looks for optimal solution. For A* we took our DFS and added heuristic function to it, so we expected it to behave similarly to DFS.

Conclusion

In our case, the fastest algorithms are DFS and A*. This is because instead of using depth limit for DFS we optimized it to only take action which results in a unique state. This in itself can be a heuristic function. At the time, we did not realize that. For A* algorithms we could have potentially created a better algorithm with closer analysis of the action patterns present in solutions. By looking for specific groups of actions in the action history, many nodes could be eliminated from expansion.