Steven Bui
Alex Young
CS 331

Programming Assignment 1

Methodology

For this Chicken Wolf Boat problem we tested 3 test cases using four different search algorithms to solve for the optimal path to reach the end of the problem, which is to reach the goal state from the initial state. Our four algorithms (bfs, dfs, iddfs and astar) are based on the GraphSearch function pseudocode. What this means is that we generate a graph of nodes for each algorithm and iteratively traverse the states while updating the graph. In order to do this, we use two hashmaps, one for the frontier states (states that have been generated but not checked for successors or checked for the goal state), and one for the set of explored states. Each node keeps track of its own state and its parent state, which allows for easier traversal once the goal state has been found. For the iddfs algorithm, we set the depth limit as 10000 to stop the algorithm from running through too many nodes.

In addition, the heuristic algorithm that we use to calculate f(n) = g(n) + h(n) is simple. We measure the total number of steps based on the total number of chickens and wolves that need to cross to the other bank. In the optimal solution 2 animals would move across to the goal bank and then 1 would move back to the initial bank. This means that for every 2 steps, 1 animal would cross to the other bank in total, however with the last step 2 animals would step across the bank. This would look like (total animals on right bank - 1) * 2 - 1 = optimal path cost. In the case where the boat was on the opposite bank the optimal path cost would just be (total animals on right bank * 2). We use this heuristic function because it is admissible (we know this because the algorithm will at the smallest decrease the number of animals on a side by 1 and will always increase the depth/path cost by 1, meaning there will never be a decreasing edge in a path).

Results

Test Case 1

	bfs	dfs	iddfs	astar
# of nodes on solution path	11	11	11	11
# of nodes expanded	14	11	84	11

Test Case 2

	bfs	dfs	iddfs	astar
# of nodes on solution path	43	47	43	43
# of nodes expanded	110	82	4061	43

Test Case 3

	bfs	dfs	iddfs	astar
# of nodes on solution path	377	377	377	377
# of nodes expanded	760	562	161473	377

Discussion

The results that our algorithms got were expected for the number of nodes on the solution path. All of the algorithms except dfs got 11, 43, and 377 for the three test cases. For test case 2, dfs got 47 instead of 43 like the other algorithms. This makes sense given that bfs, iddfs and astar are all optimal under the correct conditions. Because dfs is not optimal, it was unable to find the optimal solution path in test case 2.

When looking at the number of nodes expanded, we found that iddfs had the highest number of nodes expanded which makes sense because everytime the depth limit is increased, it will have to start back from the top and expand the nodes that were already expanded from a previous iteration. We also found that astar had the lowest number of nodes expanded which makes sense because as an informed search algorithm, it should perform a smarter search.

One interesting behavior here was that for test 3, which had a larger number of nodes to keep track of, dfs was able to perform better than bfs with an optimal solution and less nodes expanded too. While this is interesting because dfs is not always optimal, if it gets lucky, it does still make sense that this scenario played out the way it did because dfs is able to look through the deeper nodes far faster than bfs. The most interesting behavior was for the astar search which got the optimal solution in the exact same number of nodes expanded to as the number of optimal steps. This was really surprising to our group because for a long time we had a running version of code for astar which we thought was correct that was giving a lot more expanded nodes.

Conclusion

Out of all the algorithms, we found that astar performed the best with by far the lowest number of nodes expanded and an optimal solution. From this we can conclude that informed search algorithms that can find optimal solutions (such as A* search) are far more efficient than uninformed search algorithms. We can also conclude that out of these search algorithms that if you value finding the optimal solution you should not use depth-first search and if you value finding a solution quickly (with fewer nodes expanded to), then you should not use iterative deepening depth first search.

In the end it was expected that A* performed the best because it is an informed search that uses knowledge beyond what is in the problem to optimize the search. It is able to make use of a heuristic algorithm to estimate the shortest possible cost to the solution and prioritize successor nodes based on this information that the other algorithms did not have.