AI Project Part A Report

1 - Strategy overview

Taking into account the efficiency of our solutions, we decided to implement A* search algorithm and use Breadth-First Search (BFS) as a baseline to evaluate our algorithm's performance.

1a - Implementation

Data structure: A state stores a corresponding board and red frog's position. A node consists of a state, a parent node, an action, an indicator if the move is a jump, and realized and estimated costs. Because this problem allows the red frog to jump successively in one move, the parent of every node in a jump sequence is set to be the initial node. Consequently, the algorithm does not allow the player to break down multiple jumps into individual moves, however, because two separate moves would cost more than one successive jump, ignoring this case can reduce the space complexity.

Algorithm: Our A* search repeatedly removes the node with the lowest total estimated cost from the priority queue. If the goal is not reached, the node is expanded based on five possible actions, and its child nodes are inserted into the priority queue. No solution is returned if the queue is empty because no more possible states can be explored.

1b - Complexity

According to the lecture slides, theoretically, the complexities of A* does not count the complexities of priority queue operations and the heuristic function. If d is the solution depth, b is the branching factor, and n is the size of the board, in the worst case such as when there is no solution, A* has a time and space complexity of $O(b^d)$, similar to BFS. With a useful heuristic function, the time complexity of the algorithm becomes exponential in the relative error in the heuristic function and the solution length. As we can explore fewer nodes, the space complexity may also reduce.

However, in practice, the complexity of the heuristic can matter. In our implementation, the heuristic function calculates the distance to each element in a pre-computed list of coordinates corresponding to blue frogs and final-row lily pads. This list is cached to avoid redundant computation, but incurs a one-time preprocessing overhead of $O(n^2)$ where the board size is $n \times n$. The runtime of the heuristic per call is $O(n + m)$ where m is the number of blue frogs.

Because the error in h varies by problem, we designed three tests and compared A*'s actual runtime and space usage to BFS to analyze A*'s complexity:

- Test 1: The first test case of the assignment, where there is a goal and an optimal path.
- Test 2: There is no solution for this test, so we expected A* and BFS performed similarly.
- Test 7: With more lily pads than in Test 1, so b is around 3.
- Test 8: The same board as test 1, but with a lily pad on every empty grid, so b is maximized.

Test\Algorithm	BFS	A*
Test 1: time (space)	0.00263s (167 nodes)	0.00181s (97 nodes)
Test 2	0.00115s (45 nodes)	0.00193s (45 nodes)
Test 7	0.00279s (247 nodes)	0.00148s (68 nodes)
Test 8	0.00886s (705 nodes)	0.00484s (277 nodes)

As the branching factor increases, the time and space reduced by A* become more significant, as shown in Test 1, 7 and 8. In Test 1, A* runs almost twice as fast as BFS and uses about half the space. Even with a higher b, the heuristic can guide A* to expand node closer to the goal, hence, A* performs even better in Test 7. In Test 8, where the b is maximized, BFS has to expand about 5 children per node at each level, resulting in 705 nodes in memory. These numbers are over 4 times higher than in Test 1 although they have the same frogs' initial positions. In contrast, the A* only expands 277 nodes, only about 3 times higher than in Test 1. In Test 2, as expected, A* seems to perform slightly worse because of the computational time of heuristic function.

2 - Heuristic function

For a node n, we have the following evaluation function $f(n) = g(n) + h(n)$ where $g(n) =$ cost so far to reach n, $h(n) =$ estimated cost to goal from n , $f(n) =$ estimated total cost of path through n to goal. In this section, we will discuss the construction of $h(n)$ and explain how it speeds up the search without compromising optimality.

The primary source of complexity arises from the red frog's ability to jump over one or more blue frogs. This motivates the categorization of optimal paths from a given node n to the goal into 2 distinct groups: **(1) paths that do not involve any jumps,** and **(2) paths that involve at least one jump.**

Let m be the number of lily pads on the final row ($m \le 6$) and $l_1, l_2,..., l_m$ denote their positions. Let k be the number of blue frogs whose rows are at least the row of the red frog and $b_1, b_2,..., b_k$ denote their positions. Let t denote the position of the red frog.

2a - Construct a heuristic in case (1)

Consider 2 points $p_1 = (r_1, c_1), p_2 = (r_2, c_2)$ s.t. $r_2 \ge r_1$. Let $d(p_1, p_2) =$ estimated cost from p_1 to p_2 where **no** jumps are allowed. Let $A = \{(1, -1), (1, 1)\}$ denote the set of possible diagonal moves; $B = \{(0, 1), (0, -1)\}$ denote the set of possible horizontal moves; $C = \{(1, 0)\}$ denote the set of possible vertical moves. We can set $d(p_1, p_2) = x + y + z$ where $x, y, z \in N, a \in A, b \in B, c \in C\ s.t.$ $x * a + y * b + z * c = (r_2 - r_1, c_2 - c_1)$. For d to be admissible, we need to find x, y, z such that $x + y + z$ is minimized. Since a diagonal move is equivalent to a horizontal move followed by a vertical move, we need to maximize x. Hence, $x = min(|r_2 - r_1|, |c_2 - c_1|)$; $y = |c_2 - c_1| - x$; $z = |r_2 - r_1| - x$. Since our goal is reaching any of $l_1, l_2,..., l_m$, for h to be admissible, we set $h(n) = min\{d(t, l_1),..., d(t, l_m)\}$.

2b - Construct a heuristic in case (2)

Since there is at least 1 jump on the optimal path from a node n to the goal, the cost from n to the goal is at least the cost from n to the closest blue frog (where "closeness" is measured by d). Note that we can't "land" on a blue frog b, but $d(n, b)$ is still a good estimation since it never overestimates the total cost of moving from n to an adjacent cell to b **and** jumping over b. For h to be admissible, we set $h(n) = min\{d(t, b_1),..., d(t, b_k)\}$.

2c - Combine 2 cases

Since we don't know beforehand which case our optimal path will fall into, and we need h to be admissible, we set $h(n) = min\{d(t, l_1),..., d(t, l_m), d(t, b_1),..., d(t, b_k)\}$.

h is admissible by construction, so optimality is guaranteed. It remains to be shown that h speeds up the search compared to BFS. We can think of BFS as an A* search with a heuristic $h'(n) = 0$. Since $d(t, b) > 0\ and\ d(t, l) \ge 0\ \ \forall\ valid\ t$, h dominates h'. Furthermore, whenever t is not at a goal, h strictly dominates h'. Additionally, we can construct another admissible heuristic function h'' by replacing d with Euclidean distance. However, using triangular inequality, d dominates Euclidean distance, so h dominates h''.

3 - Extension

Consider the case where all 6 red frogs had to be moved to the other side of the board to secure a win. Since we can only move 1 red frog at a time, our action space is extended to $\{(i, a)\}$ where $i \in \{1, 2,..., 6\}$ denote the red frog to be moved and $a \in \{Down, Left, Right, DownRight, DownLeft\}$ denote the selected move (including the ability to jump over one or multiple blue or red frogs).

Let t_i denote the position of a red frog ($i \in \{1, 2,..., 6\}$).
Let $h_i(n) = min\{d(t_i, l_1),..., d(t_i, l_m), d(t_i, b_1),..., d(t_i, b_k), d(t_i, t_1),..., d(t_i, t_6)\}$.
We claim that $h^*(n) = max\{h_1(n),..., h_6(n)\}$ is an admissible heuristic for this extended problem.
Let t_j be the position of the red frog with the highest "individual heuristic" i.e. $h^*(n) = h_j(n)$.

Consider the problem of moving the jth red frog to the goal.
If moving other 5 red frogs doesn't affect the optimal path of the jth red frog, we can simply assume that they aren't moved and treat them as blue frogs. This reduces to the original problem with 1 red frog, where h_j is admissible, so h^* is admissible.
Otherwise, there must exists an rth red frog that can be moved to a position where the jth red frog can jump over and form a "more optimal" path. The number of moves it takes until jth red frog jumps over the rth red frog is at least $max\{h_r(n), h_j(n)\} = h_j(n)$. Hence, h_j never overestimates the total cost of moving all red frogs to the goal, so h^* is admissible.