COMP30024 Part A Report

Jack Cantwell (1473189) & Thomas Eley (1473363)

With reference to the lectures, which search strategy did you use? Discuss implementation details, including choice of relevant data structures, and analyse the time/space complexity of your solution

The search strategy implemented in this task is A* search, a variant of best-first search that attempts to avoid expanding paths that are already expensive. A* does this by maintaining a list of evaluated path costs and expands the lowest cost move first. The A* cost function is defined as f(n) = g(n) + h(n) where g(n) is the amount of steps required to reach the current state from the start state, and h(n) is the heuristic that estimates the cost of reaching the goal state from the current state. A* search was chosen because it is both optimal and complete, meaning that it always finds the solution, if there is one, with the lowest total cost, assuming the heuristic is admissible and consistent.

A priority queue was used to store the list of states that have been expanded in the search, ordered in increasing order of f(n). The underlying data structure for the priority queue was a min-heap data structure, where the root node contains the state with the lowest f(n) cost value, and these values increase as they move down the tree. Whenever a state is generated (pushed into the queue) or expanded (popped out of the queue), the heap reorders the priority queue based on f(n). If two states have the same f(n) value, the tie breaker is which one was generated first. On expansion of a state, it is evaluated to see if it is a solution, if not, the next possible moves are evaluated, ensuring that previously expanded states are not revisited.

The time complexity of A* search in the worst case is generally $O(b^d)$, with b as the branching factor which in this case is 5, corresponding to the five possible moves, and d as the depth. The space complexity is also $O(b^d)$ in the worst case, as A* keeps all generated states in memory. However, a good heuristic function can prune away many of the b^d states that an uninformed search would expand.

If you used a heuristic as part of your approach, clearly state how it is computed and show that it speeds up the search (in general) without compromising optimality. If you did not use a heuristic based approach, justify this choice

The heuristic function is a crucial part of the efficiency of the A* search algorithm, it estimates the cost from any given state to the goal state. In order to guarantee optimality, the heuristic must be admissible, meaning that it never overestimates the true cost. Without this, the heuristic could compromise the optimality of the search by selecting suboptimal paths.

A simple heuristic, such as Manhattan distance from the red frog to the nearest winning cell would not be admissible, because it ignores the hopping mechanic that allows the red frog to jump over blue frogs, moving two squares in one move. To account for this, we could assume that the red frog can consistently hop towards the bottom row, essentially halving the Manhattan distance. However, given the “hopping” feature, which allows the frog to jump over multiple blue frogs in succession, with a move cost of 1, there is a possibility that a frog could jump from the current state to the goal state. Using this logic, any frog on an even row can at best

hop all the way to the 6th row, requiring at least one more move to reach the goal state, whereas any frog on an odd row could conceivably reach the goal state in one "hop". This forms the basis of our heuristic, whereby frogs on even rows return an h value of 2, and frogs on odd rows return an h value of 1. This is admissible, as there is no "cheaper" possibility of reaching the end of the board. This admissibility guarantees A* search will find an optimal path.

The heuristic improves the search efficiency in general while guaranteeing optimality. Without the heuristic function, the A* search essentially runs as a uniform-cost search (UCS), which expands all states in increasing order of g(n), and can lead to expanding states in directions that don't contribute to reaching the goal. The heuristic focuses on paths that could minimise total cost, ensuring that paths where the red frog reaches an odd state are prioritized, but not to the extent that it becomes a greedy search, as the heuristic never overestimates the true cost to the goal. Paths where there is a clear 'leapfrog' path to the bottom row will be prioritised, resulting in a more efficient search compared to non-heuristic based approaches. This heuristic is a very relaxed version of the problem, and is far from a perfect heuristic, however, it was implemented as it is guaranteed to be optimal while improving efficiency in some scenarios.

Imagine that all six red frogs had to be moved to the other side of the board to secure a win (not just one). Discuss how this impacts the nature of the search problem, and how your team's solution would need to be modified in order to accommodate it.

Expanding the problem to moving all six red frogs to the other side of the board would still be solvable by the taken approach, that being applying A*. However, the overall heuristic would need to be updated to be the sum of individual heuristics. This would still be admissible as only one frog can move per turn.

The main impact that this change has on the nature of the search problem is the drastic increase in the branching factor. The branching factor now increases to 30, as each of the 6 pieces can move in 5 different directions, and only one piece can move at a time. It should be noted that the time and space complexities of this problem will remain as $O(b^d)$, with the understanding that b is increasing 6-fold.

In order to accommodate for this alteration, it must first be considered that each of the 6 red frogs' positions must be stored, rather than just tracking one, this extends to the way that the "visited" set is used. Moreover, when expanding neighbouring states the algorithm must now check all possible moves for each of the different frog's positions. Functionally, the fact that red frogs can hop over OTHER red frogs, as well as blue frogs, must now also be considered, resulting in hopping chains with both red and blue frogs along the path. The only other major change that would need to be made would be updating the goal test function to check if all 6 red frogs are in the 7th row.

Given that there is no "grow" function at this stage in the project, it is a fair assumption that the likelihood of the existence of a path that takes us to the goal state is relatively small. However, the approach, with the adaptations outlined above, would still be successful in finding the optimal solution, should it exist.