Luke Di Giuseppe, Audrey Webb
Dr. Sarah Erfani
COMP30024 - Artificial Intelligence
12 April 2019

Project Part A - Searching Chexers

In Project Part A, we play a single-player variant of the game Chexers in various situations involving either one player or multiple players of the same color. Before choosing an algorithm that would find the best path, or sequence of actions, for the player to win, we first had to formulate our game as a search problem. We represent a single-player Chexers game as the following:

- States: Game board configuration; specifically the location of the hexes that have a player piece occupying them or are occupied by the colourless blocks.
- Actions:
    - MOVE from one hex to the adjacent hex.
    - JUMP from one hex over an occupied hex to the hex directly next to the occupied one.
    - EXIT from the last hex on the board that is designated as an exit position for the player piece (Red, Green, or Blue players).
- Goal Test: All pieces have taken an 'EXIT' action, and therefore the board state has no player pieces remaining
- Path Cost: 1 per action.

After representing the single-player variant of Chexers as a search problem, our next step involved choosing the best search algorithm for our problem. Through our analysis we chose to use an informed search strategy called the A* Search algorithm. This algorithm is a special case of the Best-First Search algorithm which uses an evaluation function for each node to estimate it's desirability and thus expand the most desirable unexpanded node. What A* Search does is avoid expanding paths that are already expensive and uses the evaluation function, $f(n) = g(n) + h(n)$, where $g(n)$ is the path cost and $h(n)$ is the heuristic value cost. A* Search algorithm is complete (guaranteed to find a solution), it is optimal if we choose an admissible heuristic, and has feasible time and space complexity. Due to the compatibility between our search problem and A* Search's properties, we chose to use it to solve our single-player Chexers game problem. In addition, we take our nodes to be the varied board configurations.

For our A* Search algorithm, we defined our heuristic based on hexagonal Manhattan distance. Our function estimated the cost of getting a piece to an exit hex to be the halved Manhattan distance plus 1. This heuristic is admissible, as it predicts a piece can make jumps (i.e., move two spaces every one move) until it gets to its exit hex, and then makes the EXIT action. A piece cannot reach an exit hex faster than this, regardless of the board configuration, and so is admissable. As our heuristic is admissible, we know that our A* Search will find the optimal solution if given time to execute. Through experimentation we found that though the solutions are optimal, it often takes more than thirty seconds to execute, specifically in open boards with four pieces. A more advanced heuristic would allow the algorithm to more efficiently find the

We use code provided by Artificial Intelligence A Modern Approach (AIMA) throughout Project Part A

solution, but we found the problem of encouraging the search to include 'leapfrog' situations (a piece moves away from the goal in order to allow other pieces to quickly reach their goal) to be too complex to code, so we settled on our simpler solution.

To combat the issue of running overtime, we chose to use the A* Search algorithm with the constraint that if it takes more than twenty-five seconds to run, we would run a simpler version of the A* Search algorithm. This version considers pieces individually such that it ignores the other pieces on the board. It then chooses a valid ordering to exit the pieces, so there are no collisions. Single piece searches have a much smaller branching factor, resulting in significantly quicker run time but a suboptimal solution. Our strategy guarantees our program finding a suboptimal solution if an optimal solution is too complicated to find, keeping our program running under thirty seconds for all possible board configurations.

Based on our search problem and our algorithm input, we are able to understand A* Search's time and space requirements. Assuming an empty board, a piece on average can make 4.86 actions from a given hex. Thus, the worst case scenario for our A* Search algorithm is a time and space complexity of: $O(4.86n)^d$, where n is the number of pieces and d is the number of moves to have all pieces exit. This is very time consuming and scales quite poorly with regard to n and d, hence the poor performance of our algorithm with more than three pieces. Even though our algorithm will ignore many of these branches, and we know that most boards will have much fewer valid and non-repeating moves, the search is still eventually halted by the growing n and d.

In conclusion, we found in practical tests that our A* Search algorithm was able to handle many cases of four pieces and find optimal solutions. Only a minority of constructed situations, specifically those requiring a high number of moves, proved to be too complicated.

We use code provided by Artificial Intelligence A Modern Approach (AIMA) throughout Project Part A