COMP30024 Part A Analysis

# *Search Problem*

For Project Part A, we viewed the game as a search to find the optimal move for each piece that brought that piece closest to the end state. All necessary information was provided and the program therefore required offline problem solving.

**Percept:** The percept comes from a separate file, detailing the necessary board information

**State:** The initial state for our project is the game board with the initial information from the percept. The state is updated after every action. The game board is made up of a collection of nodes

**Nodes:** Each node is referenced by its coordinates and stores the path cost and possible moves from that position

**Pieces:** Each piece is located at a specific node and its initial location is determined by the percept

**Action:** Each action requires moving a specific piece located at a node to another neighbouring node on the board

**Goal:** The explicit goal of the program is to move all pieces off the board, by reaching an exit node. The exit nodes are determined by the percept.

**Path Cost:** The cost of a path is determined by finding the minimum number of actions required to exit the board from the location of a node.

# *Search Algorithm*

Our program uses a local, hill climbing search algorithm to solve the problem. Our algorithm takes the current board state and an individual piece and moves the piece to a node with a lower path cost than itself. We use this search algorithm independently for each piece, starting with the piece closest to the exit until the goal state is reached. We can split our piece configurations into two cases, either there is one piece on the board, or there are multiple pieces.

## One Piece

**Completeness:** Our algorithm is complete and will always find an exit. This is because our percept always has a solution, and as a result there will always be a node that is closest to the exit relative to the current piece. Our path cost is continuously decreasing, until the goal state is reached and so there is no local minimum that the algorithm can return.

**Time:** The algorithm is O(b\*d), where b is the number of neighbours and d is the distance (depth) of the piece away from the exit. We only ever evaluate the current piece and search through its neighbours for the one closest to the exit. Therefore, the amount of time needed to run the program is the number of times this neighbourhood search needs to be conducted, in other words the distance of the piece away from the exit. This is an efficient algorithm, and is constant in time, because it is bounded above by 6\*36 = 216, which is the maximum number of neighbours and maximum pieces on the board.

**Space:** The algorithm is O(b), where b is the number of neighbours away from the exit. For the algorithm to work, we only need to evaluate the current neighbours of a piece. The space complexity is therefore constant, as we only need to store the maximum of 6 neighbours of any piece at a time

**Optimal:** The algorithm is optimal. The board state ensures that for every node, the minimum path cost away from the exit is known. Consequently, by moving the piece to a neighbouring node with the lowest path cost, we ensure that we move along a path that is minimal in length

## Multiple Pieces

**Completeness:** Our algorithm is complete and will always find an exit for all pieces on the board. Again we are ensured that there is a solution from the percept. By moving the piece closest to the exit, we find that there is never a situation where a path for a further piece is hindered by moving a piece closer to the exit. By moving each piece independently, it is certain that every piece will exit the board.

**Time:** The algorithm is O(b\*d), where b is the number of neighbours and d is the distance (depth) of the piece away from the exit and p is the number of pieces. The algorithm is unchanged by the number of pieces on the board, because it only acts on one piece at a time. The total running time increases when multiple pieces are considered and we have a maximum bound of 4\*6\*36 = 864, which is still constant.

**Space:** The algorithm is O(b), where b is the number of neighbours away from the exit. Again, the algorithm only ever considers one element and its neighbours at a time, and so the space complexity is the same

**Optimal:** The algorithm is far from optimal. By only considering one piece at a time, we neglect to consider the impact on the entire board after making a move. Moreover, by moving the nearest piece to the exit, we lose the ability of jumping over other pieces to decrease the time needed to exit.

# *Problem*