COMP30024 Assignment 1 Report

Jack Macumber, Tiannan Sha

1. Form the search problem

We define states, actions, goal test and path costs as follows:

State: board configuration. That is, the entirety of the board, including the coordinates of pieces and blocks. The initial state is the initial board configuration, in which pieces are at their initial positions. Note that two states would only differ in the coordinates of pieces.

Actions: all the possible movements of all the pieces on a given board. That is, choose any piece to jump or move to any valid board position as defined in the spec, or exit.

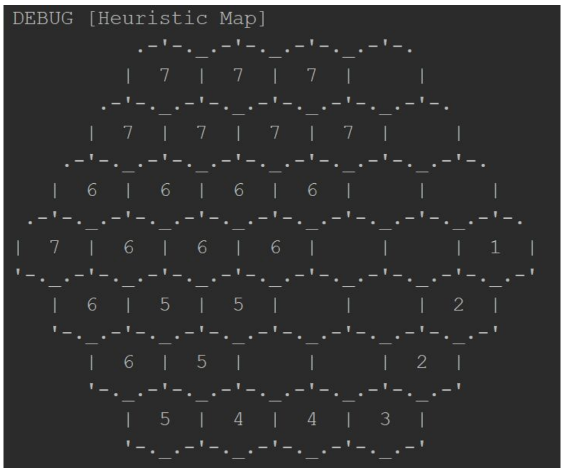
Goal tests: the board configuration where all pieces have exited the board.

Path costs: number of actions (moves, jumps and exits) performed from the initial board configuration to the final board configuration where all pieces have exited the board.

2. Algorithm choices

<u>Heuristic</u>

To speed up our search, we use heuristic to inform our search algorithm. The heuristic is relaxed over what the true cost by assuming the pieces can always jump (even when there's no piece to help), taking into account blocks. Before searching, we assign a heuristic value to every coordinate of the board to label the estimated number of actions left to exit from that position. Then when performing the search, for all the coordinates occupied by a piece, we sum up the heuristic values for these positions to obtain the total estimated number of actions left for all pieces to exit.



```
DEBUG [Heuristic Map]
          .-'-._.-'-._.-'-._.-'-.
         |  7  |  7  |  7  |     |
       .-'-._.-'-._.-'-._.-'-._.-'-.
      |  7  |  7  |  7  |  7  |     |
    .-'-._.-'-._.-'-._.-'-._.-'-._.-'-.
   |  6  |  6  |  6  |  6  |     |     |
 .-'-._.-'-._.-'-._.-'-._.-'-._.-'-._.-'-.
|  7  |  6  |  6  |  6  |     |     |  1  |
'-._.-'-._.-'-._.-'-._.-'-._.-'-._.-'-._.-'
   |  6  |  5  |  5  |     |     |  2  |
   '-._.-'-._.-'-._.-'-._.-'-._.-'-._.-'
      |  6  |  5  |     |     |  2  |
      '-._.-'-._.-'-._.-'-._.-'-._.-'
         |  5  |  4  |  4  |  3  |
         '-._.-'-._.-'-._.-'-._.-'
```

To assign a heuristic value to each coordinate, we employ a BFS like approach. We start from the positions where pieces can exit, set their heuristic value to be 1, as in these positions, pieces are 1 action away from exit. Then in each iteration, we update the unassigned positions that are one action away (either jump or move) to be current heuristic value +1, until all the positions that are not occupied by blocks have been assigned a heuristic value.

It is easy to see our heuristic is admissible. All the pieces are assumed to be able to jump without helper piece, so the estimated cost is guaranteed to be no greater than the actual cost.

<u>***A****</u>

The algorithm we employ is ***A****, with the heuristic as described above. As our heuristic is admissible, our ***A**** is optimal. As we only have finite number of nodes with $f \leq f(G)$, ***A**** is complete. The time complexity of ***A**** is exponential in [relative error in h * length of soln.]. (The time and space complexity details are discussed in section 3.) The difference between our heuristic value and the actual cost left to reach the goal is only that our heuristic assumes the potential of a few more jumps. For each extra jump, we only underestimate one action. As a result, our [relative error in h] is actually very small. Therefore, the time complexity of our ***A**** is small as the exponent term is small. When the number of pieces is ≤ 4, our program can find the solutions within a second (on our computational environment). The only drawback of our ***A**** is the space complexity, which is the number of nodes generated, as ***A**** requires all nodes be kept in memory, unlike IDS or DFS. But we still choose ***A**** because ***A**** has better performance with respect to time and memory space was not a limiting factor in the sample tests we had created.

3. Factors that impact complexity

Let *b = max branching factor*, *d = depth of the least-cost solution*, *m = max depth*, *p = number of pieces* and *k = number of blocks*, *h = cost to goal evaluated by the heuristic*, *h* = actual cost to goal*

The time complexity of ***A**** is *exponential in [relative error in h * length of soln.]*. That is, $O(b^{d\,(h^*-h)/h^*})$, where $(h^* - h)/h^*$ is the *relative error in h* (AIMA , page 98).
The space complexity of ***A**** is number of nodes generated, as ***A**** keeps all nodes in memory.
Time and space complexity are related via the number of nodes generated. When b or $(h^* - h)/h^*$ or d grows, time complexity would grow, which indicates more nodes are generated, as time complexity measures number of nodes generated. Moreover, all nodes are kept in memory, therefore when b or $(h^* - h)/h^*$ or d grows, space complexity also grows.

Max branching factor $b = 6p$ as in the worst case, each of the p pieces can choose to either move or jump (if a piece can jump, then it has a helper piece and it can't move, and vice versa), to any of the six directions. As p increases, b would increase 6 times faster. Because b is the base of the exponential term, b has a significant impact on both time and space complexity. Therefore, as p grows, more nodes are visited, so both time and space complexity would grow significantly, and vice versa.

Depth of the least cost solution d is decided by the initial positions of the pieces. When the pieces' initial positions are closer (need fewer actions to get) to the exit positions, the length (the number of actions) of the shortest path is shorter, and hence *d* is smaller. *d* decides the exponent of the exponential term, therefore as *d* decreases, fewer nodes are visited, so both time and space complexity would decrease significantly, and vice versa.

$\textit{Relative error of the heuristic is } (h^* - h)/h^*$ (AIMA , page 98).
$h^* - h \;=\; (\textit{actual number of actions left} \;-\; \textit{number of actions left when you can always jump})$. As the gap between h^* and h grows, the relative error grows, and hence the exponent would grow, more nodes are visited, so both the time and space complexity would grow.

As *number of blocks* k grows larger, in general there will be fewer possible actions for pieces. As a result, the branching factor might decrease. Also, more blocks might also mean more chances to jump, so d would decrease. Therefore, depending on the positions of the blocks, as k increases, both time and space complexity can decrease. However, if there are many blocks cramming together and block the road (e.g. in **Figure 1**), pieces would be blocked and wouldn't have more chances to jump, and d would increase. Therefore, in this case, when k grows, both time and space complexity would grow.