Project Part A Sample Report

Matt Farrugia

Formulating the problem

How have you formulated the game as a search problem? (You could discuss how you view the problem in terms of states, actions, goal tests, and path costs, for example.)

A state in the problem consists of the location of the 1-4 pieces on the board at the beginning of a turn during the game (in particular, the *set* of piece coordinates is important, since the pieces are indistinguishable). The location of any blocks and the colour of the player are also of interest, however note that these will not change throughout a game.

Our actions correspond to the allowed actions on any turn in the game. The effect on a state will be based on the rules for Chexers (e.g. to change the location of one piece in the state's piece set, or, for an exit action, to remove one piece from the set). The goal test is then simple: Does a state have any pieces left? If so, the goal has not yet been reached. If not, then the goal has been reached.

Since we are interested in finding the shortest possible action sequence to win the game, it makes sense to consider each action to have uniform cost (say 1) and thus the path cost will simply count the number of actions in an action sequence.

Choice of algorithm and heuristic

What search algorithm does your program use to solve this problem, and why did you choose this algorithm? (You could comment on the algorithm's efficiency, completeness, and optimality. You could explain any heuristics you may have developed to inform your search, including commenting on their admissibility.)

It seems that many teams chose to use the A* search algorithm, so this solution does the same. It's a sensible choice: the algorithm is efficient, complete, and optimal (under some reasonable assumptions about the problem, all of which apply here).

To ensure the optimality of A* it's important that we use an admissible heuristic. Let's use the following admissible heuristic h:

$$h(S) = \sum_{p \in S} \left\lceil \frac{\text{exit-dist}(p)}{2} \right\rceil + 1$$

Here, S is a state (a set of pieces p), and 'exit-dist' is a function representing the hexagonal Manhattan distance to the nearest hex on the appropriate edge of the board for exiting for the given player[1]. This heuristic underestimates the number of actions to reach a goal state by assuming each piece can jump freely to the required edge of the board (hence divide the exit-dist by 2, and round up to account for a final move if the distance is odd) and take an exit action (hence add 1).

Better heuristics exist (e.g. we could modify exit-dist to take the distribution of blocks into account, as some teams did to great success), but this heuristic was simple to implement, fast to calculate, and allowed the algorithm to perform as fast as required during testing.

[1]See the redblobgames guide for information on adapting Manhattan distance to hexagonal grids. Note that in this case we can use a simpler calculation: Since the exit hexes form a line, the distance to the nearest exit hex from coordinate $p = (p_q, p_r)$ can be calculated directly as $\text{exit-dist}(p) = 3 - x$ where x is p_q for Red, p_r for Green, and $-p_q - p_r$ for Blue.

Factors affecting performance

What features of the problem and your program's input impact your program's time and space requirements? (You might discuss the branching factor and depth of your search tree, and explain any other features of the input which affect the time and space complexity of your algorithm.)

The maximum branching factor b of our search tree will depend on the number of pieces (since each piece allows for up to 6 actions) and on the distribution of blocks (e.g. when there are many blocks, the pieces movements may be significantly restricted).

The depth d of the optimal solution will also be influenced by the number of pieces (with more pieces we expect to require more actions to get all pieces to the correct edge and off the board with an exit action) and the distribution of blocks (many blocks may facilitate more jump actions and allow the correct edge of the board to be reached in fewer actions by each piece; or, if the blocks are connected so as not to allow jumps, they may instead increase the depth of the solution by requiring pieces to move around them on long detours). The depth of the optimal solution will also depend on the initial positioning of the pieces, which may be near to or far from to the correct edge.

A*'s time and space complexity both depend on the size of the search tree generated. In the worst case, the search tree's size is in $O(b^d)$. Thus we can expect the time and space requirements of this program to increase with the number of pieces in the initial configuration, and they may decrease as we consider boards filled with blocks.

The effectiveness of the heuristic is a major factor in determining the actual resource requirements of this program. In practice the search tree should come in significantly smaller than this worst case bound. The more our heuristic underestimates path costs, the less effectively A* be able to ignore suboptimal paths ('pruning' the search tree). Since the chosen heuristic assumes pieces mainly travel with jump actions the algorithm's performance will suffer when an input doesn't present opportunities for pieces to jump towards the exit edge.