Project report on

Artificial Intelligence Playing the Ricochet Robots Board Game

for 02180 Introduction to Artificial Intelligence

Contributors:	Jannis Haberhausen	(s186398)
	Jack Reinhardt	(s186182)
	Kilian Speiser	(s181993)
	Jacob Miller	(s186093)

Contents

1 Game Rules

2 Game Representation

3 State Space and Complexity

The state space and branching factor of Ricochet Robots is large, making the task of reaching the target non-trivial for both humans and AI search algorithms. The board consists of a sixteen-by-sixteen grid with walls placed in predetermined positions. With four robots that can be moved anywhere on the board (other than on top of another robot), this leaves a total of $(16*16)(16*16-1)(16*16-2)(16*16-3) = 4,195,023,360$ board configurations for a single wall setup and target placement. Depending on the board setup, some states may not be reachable, but the state space still remains large. Each robot can move in any of the four directions on the board until it reaches a wall or another robot in its path. Assuming that each robot has an obstruction in one of the four directions decreases the possible moves for each robot to three, giving the search algorithm a branching factor of $4*3 = 12$. This assumption will be true in most cases since the robot must be stopped by an obstruction before moving in a different direction.

The large state space and branching factor makes the problem of reaching the goal state difficult for traditional AI search algorithms with limited memory and time. In order to simplify the game and drastically reduce the size of the state space and the branching factor, we implemented Ricochet Robots in a way that allows the user to choose the size of the board (16x16 or 6x6) as well as the number of robots. In general, the size of the state space can be approximated by $(w*h)^n$ and the branching factor can be approximated by $n*3$, where w is the board width, h is the board height and n is the number of robots. This reduced implementation of Ricochet Robots allowed us to play and test our AI algorithms without surpassing our memory and time limitations.

4 Search Algorithms and Results

4.1 Recursive Depth-Limited Search

Given the large branching factor of Ricochet Robots, a depth-limited search gives the AI player a higher potential to find solutions to the more difficult board configurations. Some experimentation is necessary to find the optimal limit to the depth, but when the AI player is attempting to find a better solution than its human competitor, the limit could simply be set to one less than the human move count.

Specific design choices were made in order to optimize the depth-limited search for Ricochet Robots. The depth-limited search was implemented recursively and as a tree search rather than a graph search. Not keeping track of past states gives the algorithm much less overhead and allows it to expand nodes more quickly, increasing the success rate. The algorithm will start in the initial state by looping through each possible move for each robot and continue traversing the tree recursively until it either reaches a solution or it reaches the specified depth limit. Once it reaches the depth limit, it will return to the previous level of recursion to search the next branch at that depth.

One negative aspect of the depth-limited AI player is that it usually does not find an optimal solution. Most solutions contain extraneous moves that do nothing towards reaching the goal state. In order to alleviate this issue, two aspects were added to the algorithm. First, when checking if a move is possible, the algorithm also checks if the move is the opposite of the previous move (i.e if the red robot just moved North, do not allow it to move South). This cuts off an entire repeated branch of the search tree, giving the algorithm more time to search new branches. Second, once the algorithm finds a solution, it will optimize it by removing non-essential moves. It does this by iteratively deleting moves from the end and testing if the goal state is still reached.

Figure 1 shows the testing results for the depth-limited search with different maximum depth limits. Each algorithm was tested on 30 iterations of Ricochet Robots, starting with a random board

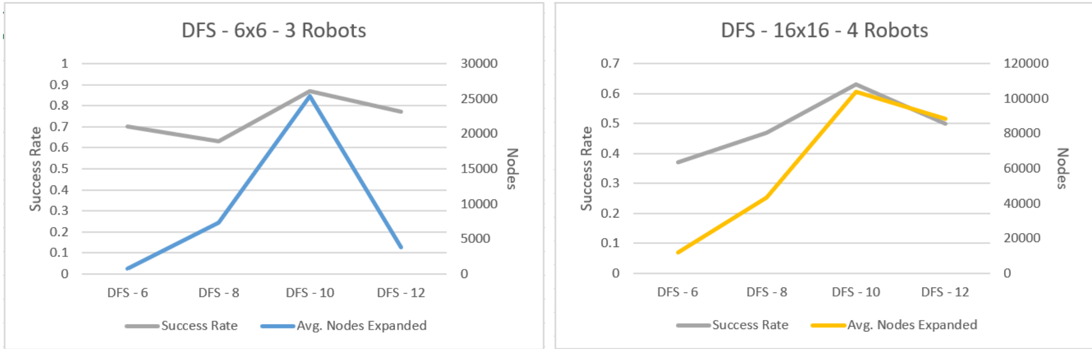

Figure 1: Depth-limited Search Testing Results

setup and continuing the next round where the previous round left each of the robots. Each iteration placed the target randomly, meaning that some configurations may be more difficult to find a solution or the optimal solution may be at a larger depth. The AI player was given sixty seconds to find a solution.

For both the simplified implementation (6x6 board and three robots) and the full implementation (16x16 board and four robots), a depth limit of ten was able to find a solution most frequently. Depth limits of six and eight performed considerably worse than that of ten for both tests, implying that many of the solutions were past these depth limits. The AI player with a depth limit of twelve performed poorly on the simplified implementation, but only marginally worse than that of ten in the full implementation. This is likely because the algorithm wastes time searching deep into branches of the search tree when the solutions to the simplified game are closer to the initial state. Overall, the depth-limited AI player would be an adequate match for a human player in both the full and simplified versions of Ricochet Robots.

4.2 Informed Search: Breadth and Depth ??

The team decided to develop a informed breadth first search to guarantee finding always the goal state by using as few moves as possible. However, this leads as mentioned in 3 this leads to a tree with up to 16^n knots.

- Difficulty of setting the limit to a certain level

4.3 A* Search

Two heuristic functions were tested with the A* graph search for their ability to quickly reach the target in Ricochet Robot.

1. Manhattan Distance - let h be equal to the one-norm of the target's location on the board and the location of the robot of corresponding color.
   $h = |robots.x - target.x| + |robots.y - target.y|$
2. Row/Column - for each robot accumulate h according to the following:
   - if the robot color matches the target color, then h += 0 if the robot is in the same row or column as the target and h += 1 otherwise
   - else, h += 0 if the robot is in an adjacent row or column to the target and h += 1 otherwise

Neither heuristic function is consistent and therefore this algorithm does not guarantee an optimal solution. While a breadth-first search is optimal, it is increasingly slow at each depth level due to the large branching factor. The A* search would prioritize certain nodes to expand based on the heuristics defined above, hopefully allowing it to reach the target more quickly.

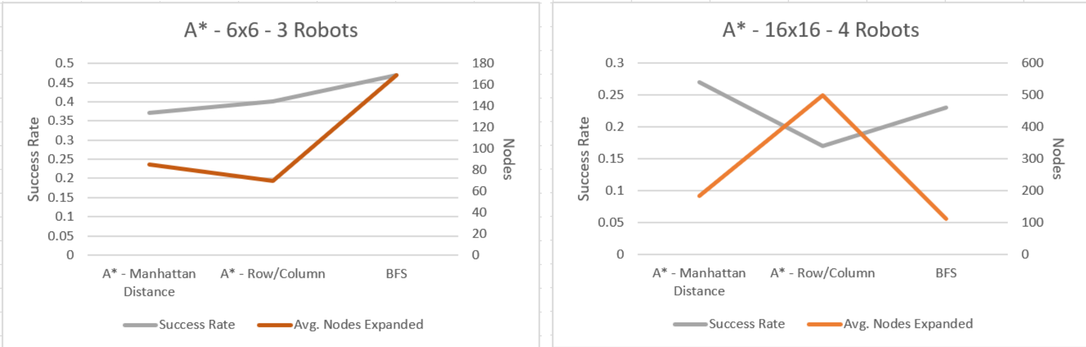

Figure 2: A* Testing Results

Figure 2 shows the testing results for the A* AI player given the two heuristic functions described above as well as a breadth-first search as a baseline (setting h = 0). Paralleling the tests done with the recursive depth-limited search algorithm, each AI player was tested on 30 random iterations of Ricochet Robots with a time limit of sixty seconds.

While BFS performed the best in the simplified implementation, it was beat out by the A* - Manhattan Distance in the full implementation. BFS will perform best when the goal state is at a shallow depth because it is optimal and therefore will never skip over a solution close to the initial state. This behavior explains BFS's success in the simplified game. Unfortunately, BFS will not be able to search deeper states in the search tree before it runs out of time. The Manhattan distance heuristic is not optimal, which may cause it to miss some shallower solutions that BFS would find, but in the full game the heuristic seems to have helped guide the AI player towards a solution quicker than BFS. The row/column heuristic performed worst in both games and does not seem to be a viable algorithm for Ricochet Robots. Due to the higher memory usage and slower run-time per iteration associated with the A* AI player, it cannot search nearly as many states as the recursive depth-limited AI player, and therefore had far lower success rates than that of the recursive depth-limited AI player.

4.4 Custom Search Algorithm

5 Conclusion