Cooperative Halma Agent

Neil Edelman – 110121860

2014-04-06

Abstract

An artificial agent was created for the game Halma with two opposing players co-operating. The goal was for the two teams of two, oppositely placed, to get all of their pieces in the other team player's squares first.

1 Introduction

The game of Halma has a clearly defined start, middle, and end.[1] The version we play is co-operative, so you may have a waiting state where all your pieces are done and you're waiting for the other player to finish.

The board is a graph. The finite pieces are put into the graph in a start state. We require that only one piece occupies a node. Each piece has one of four distinguishable colours. The nodes are arranged in a square; the number of nodes across the top and sides is x_0 (which is 16, in our case.) The pieces are initially in teams at the four corners, and opposite corners co-operate; the goal is to have both you and the co-operating player switch spots before the other team does.

Define a grid, starting with zero, $\binom{x}{y}$. Define the parity as odd whenever $x \oplus y$. We assume, without loss of generality, that the player is on the upper-left.

The maximum advancement is one square per turn. This naturally leads us to define define the move cone $C_{t_0}(t)$ as anywhere any pieces could have moved at time t, where time is discrete and measured in turns. It is a vector Each t_0, the time selected, has a possibly different C_{t_0}, but they will have smooth boundaries between C_{t_n} and $C_{t_{n+1}}$. We define our beginning as when the C_t of different players do not intersect. Similarly the end game is the reverse.

From the start state C_0(t), one assumes the horizon is 5 moves, then we we be in the middle game. In general, the formula is $\lfloor \frac{x}{2} - x_i \rfloor = \lfloor \frac{16}{2} - 4 \rfloor = 4$. For the first four moves, we can hard code them. Ideally, we would learn them from self-play.

We need symmetry breaking to move. We keep in mind that double lineups are a wall.

In the middle game, we need to deal with blocks by the other players. You do this by changing directions in a line, as seen on Figure 1.

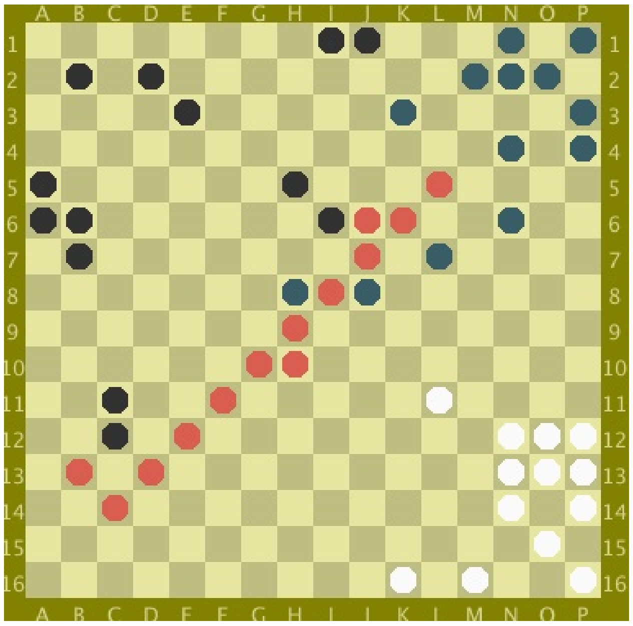

Figure 1: The leftmost piece should go left instead of down where it will be blocked.

Very much choose the furthest distance since the opposing players can move, then we choose the next-furthest piece, since the players can move again.

The co-operative nature of play causes a good strategy to get the closest piece on the co-operating player and go towards it.

In the end game is when the move cones diverge so you don't need to worry about the other players. Just apply the farthest piece from the end zone heuristic.

There is noting we can do once all the goal pieces are in the final position. We just watch and hope.

The collection of pieces can move horizontally or diagonally. We want to move diagonally, so the pieces should be in diagonal squares. Parity of the board can be used as the heuristic. The parity of the number of starting pieces across the one of the sides determines what starting move is optimal.

Number of players on an even square: 7. Number of players on an odd square: 6. So the even squares get more weight. Although, one even square is in the corner.

2 Motivation

This agent takes into account two main observations.

As the size of the groups, s, that move across the board increases, the number of moves drastically drops. The average width of the board from start to finish in Manhattan distance can be denoted d; similarly, the total number of pieces, n. With independent moves, $s = 1$, the number of moves is of the order of nd. If you take two pieces across at a time, the rough estimate is $\frac{nd}{2}$, and in general, $\frac{nd}{s}$. If you take all the pieces along at once, d. This assumes that the pieces are already in a line and that no one blocks it. The agent should play to get a single line.

The second observation is that, due to the co-operative nature of the game, we only need bridging the gap halfway. Hopefully our teammate will have a, possibly different, strategy. Since the game is commutative, we can take advantage of their strategy without knowing it. In fact, it doesn't help us if we did know it.

Notice that in several configurations it is impossible to win (Figure 2, and in some rare cases it will be impossible to move.

3 Theory

We used gradient-ascent with multiple co-operating agents.[2] $\mathcal{O}(n)$, where n is the number of moves to consider.

In Figures 3 and 4 we learned an good strategy for playing the game. This led naturally to a hill that has features that tweak the AI to behaving how you would like.

First we enumerate the moves. The moves are considered for each piece by a breath-first search. We did not use $\alpha\beta$-pruning because the numbers were not monotonic.[3] They are compared on-line and the top one is chosen.

The metric taken from the single moves is Equation 1 and is cache-friendly with no cases. The general space has a complicated metric (you can jump over pieces,) so this is as good as we're going to get.

$$\max\left(|y_1 - y_2|, |x_1 - x_2|\right) \tag{1}$$

The reward for going into a goal state.

A reward for going in a line.

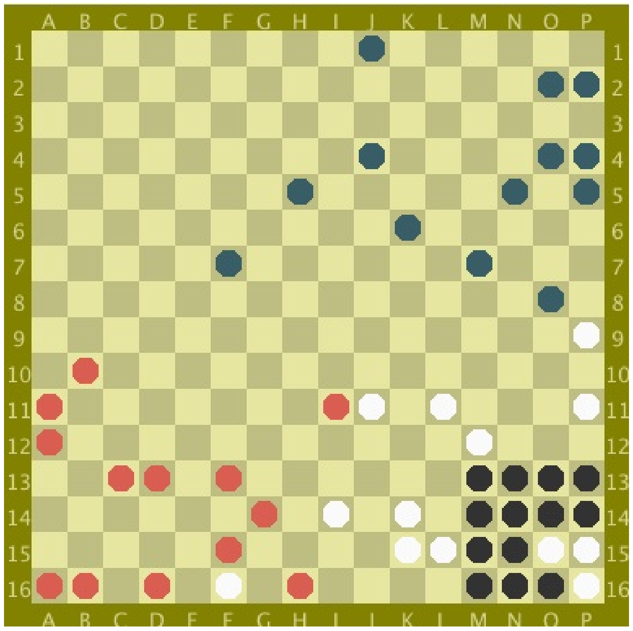

Figure 2: Deadlock.

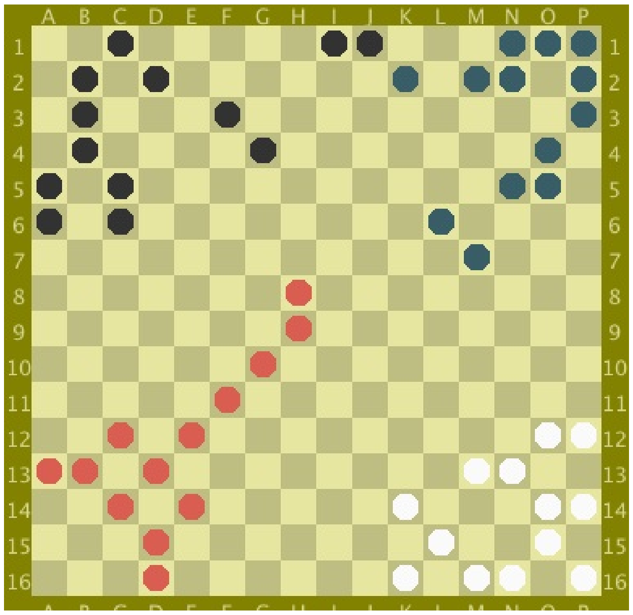

Figure 3: Play as a human.

Reward for going in front of a player. This encourages co-operation. We implemented it as a constant function added to the diagonal.

Although you are allowed to pass in halma, our game

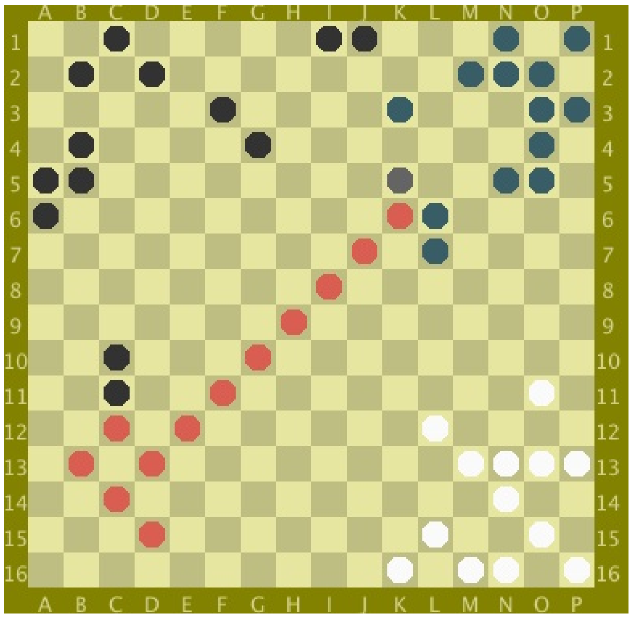

Figure 4: Play as a human.

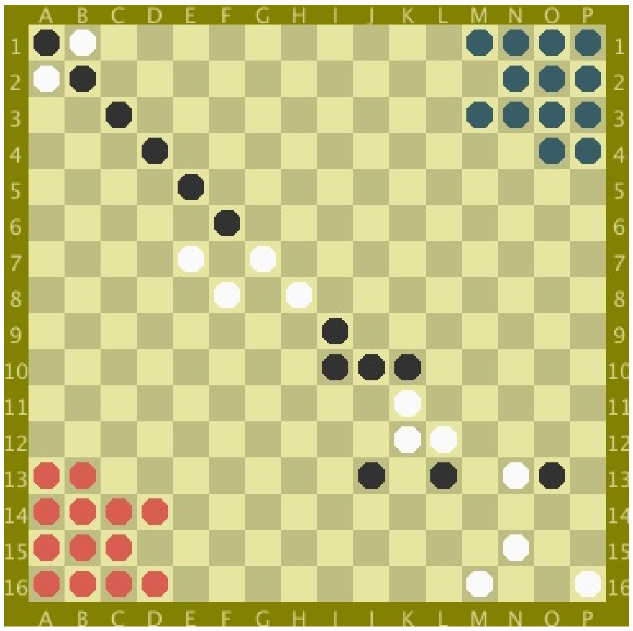

Figure 6: The groove parameter set to 5.

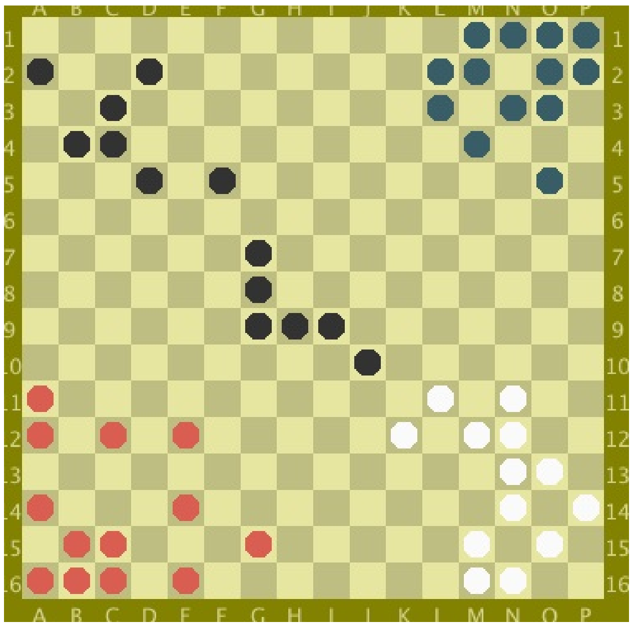

Figure 5: The groove parameter set to 0.

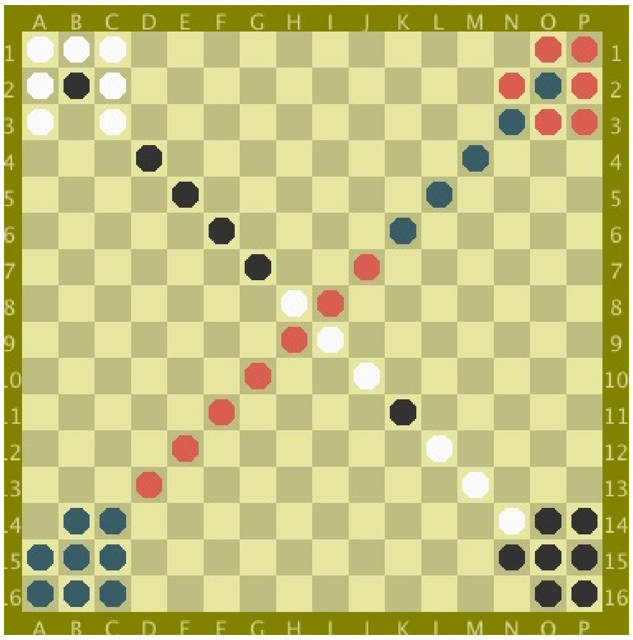

Figure 7: The groove parameter set to 10.

doesn't let you; therefore we allow negative moves

We don't know the player ids when we are connecting. This happens once the player ids are set (chooseMove is called,) and we initialise.

The jumps one square and multiple squares are calculated separately, then they are combined. We have two choices; be careful – remember, it's not quite like real halma where you can pass your turn. It is quite possible to be stuck in a situation where we should get the smallest negative value. It is conceivable that getLegalMoves returns empty, then there would be nothing we could do.

4 Advantages and Disadvantages

The main disadvantage is that it all but ignores the opposing team. I would like to hinder them at the same time causing as little hindrance to me.

Has no memory.

Doesn't adapt strategy.

Is very conducive to GPU programming where it could be accelerated greatly.

The hop function doesn't use $\alpha\beta$-pruning because the value is not monotonic. Namely, we want the agent to consider going back if it can go forward more. We use a search tree, but every node we visit we take off the graph; this results in a fairly sparse tree. We can do this because the game is conservative; that is, memoryless.

5 Approaches

We started off using an A* algorithm, with the heuristic being the negative slope. A* is good when you have to have a clearly defined metric, which we don't. We might as well use hill-climbing.[4]

We considered a three-step solution that would use three different strategies for the beginning, middle, and end. The hill-climbing encodes several of the key features of this approach in the same algorithm. The constant parameters of the hill-climbing were carefully planned to achieve a good beginning. The end was lifted up so that pieces move into their final positions.

6 Improvements

Figure 8 shows manually varying the groove parameter and how that affected winning against a random player. This could have been learnt instead of hard-coding.

As initially envisioned, the path would change based on the actual path. This allows adaptation to the other player's moves, by using the closest path that encompasses other player's pieces. Now the hill-climbing algorithm sets the hill and keeps it static.

Co-operation is a key element that would have been improved. If the co-operating player is assumed to be intelligent (ie, not a random player) we have half the work done for us. As well, any piece can be jumped, not just co-operating pieces.

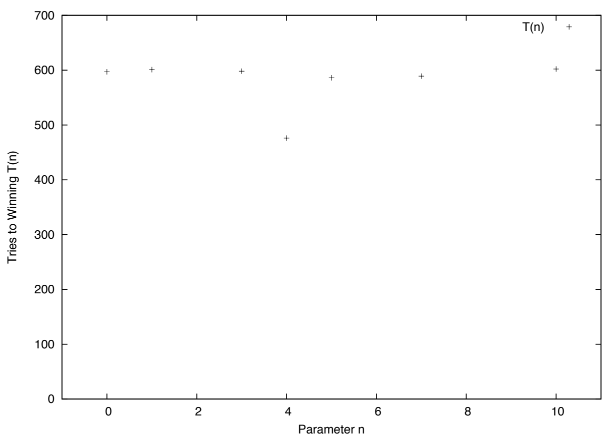

Figure 8: Manually varying hill-climbing groove parameter.

This is a greedy algorithm; it does not account for going back to get an improved position. It is susceptible to being blocked.

It doesn't co-operate to get to the end; this is not so important as the co-operating player is moving out (hopefully.)

References

[1] B. Whitehill, "Halma and chinese checkers: Origins and variations," Board Games in Academia, pp. 37–47, 2002.

[2] D. Chakraborty and S. Sen, "Mb-aim-fsi: A model based framework for exploiting gradient ascent multiagent learners in strategic interactions," in Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1, pp. 371–378, International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[3] D. E. Knuth and R. W. Moore, "An analysis of alpha-beta pruning," Artificial intelligence, vol. 6, no. 4, pp. 293–326, 1976.

[4] D. Gelperin, "On the optimality of a¡ sup¿¡/sup¿," Artificial Intelligence, vol. 8, no. 1, pp. 69–76, 1977.