Jonathan Herr
4/18/2014
CS580 Project Interim Report

Tablut Game AI with Temporal Difference Learning

Overview

Hnefatafl is an ancient Celtic and Germanic strategy board game played on a square grid of, variously, 9x9, 11x11 or 18x18 cells, depending on the specific rules employed. No comprehensive record of game rules exist, although Tablut, the version used here and described by Linnaeus in the 1700s, is the best understood version. Game play consists of one side, using white pieces, with a smaller number of guards and a king positioned at the center of the board attempting to escape the board via the corners and the other side, the black pieces, consisting of a larger number of soldiers arrayed around the edges of the board, attempting to capture the king or all of his soldiers. Movement is as the rook in chess, any number of moves in a single direction over open space, with no jumping allowed. Capture rules require intention, two pieces must surround an opponent on opposite sides(horizontally or vertically only) and captures can only take place on the capturing sides turn, so that moving between two opponents does not indicate capture. The center and the four corners can only be occupied by the king however other pieces may pass through them. The king is allowed to participate in captures in this version of the game.

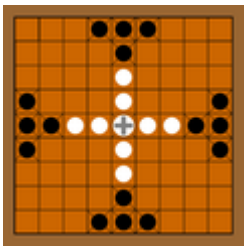


Figure 1 – Tablut Board Layout

Problem Definition & Algorithm

The goal of this work is to produce an algorithm which can play Tablut significantly better than a random player and then improve itself through play against itself with the ultimate goal of achieving human like performance. Unlike more popular games, Tablut does not have a database of played games from which we could draw better measures of success, or a scoring system such as chess has. We will measure progress by the rate and magnitude of a player's increase in performance against a random player initially, and later against earlier versions of itself. We will use minimax with alpha-beta pruning to build the game tree and train a neural net to learn the evaluation function to evaluate each state of the tree. Temporal Difference learning with backpropagation will be used to update the neural net. We also build several naïve algorithms to play against the neural net which also take advantage of the minimax tree but implement different evaluation functions.

We will consider the player successful when it has plateaued at a point where a game against a novice player can be finished in fewer than 25 moves consistently. For reference, in random play, the mean number of turns is 146. Expert players(of which there appear to be few) have found that the sides are approximately equal under the rules we've implemented so that is an additional goal. Note that, with random play, in this implementation, this is not the case; the attacker's side is slightly favored, winning 55% of the time, based on a run of 10,000 games with a max of 500 turns before a draw is called.

Implementation

We will implement the algorithm in Java, in order to take advantage of existing libraries for evolutionary computation. The Neuroph Neural Network toolkit will be used to learn evaluation functions.

Approach

We use minimax to develop a decision tree for each player and alpha/beta pruning to reduce the search space. We look ahead up to three plies and score the game at that point based on several simple heuristics as detailed in Table 1 and replace that evaluation function with one learned by a neural net for the learning player.

Two hand-tuned players (described elsewhere here as fixed minimax players) were developed for comparison to the neural net player and as an attempt to help improve its play. These use the rules in Table1 with the weights in Table 2. A simple player which is concerned only with piece count and king's proximity to the corner is also mixed into play for variety and training purposes. A random player is used to balance the rule based players. The random player simply chooses a random valid piece, direction and length with no consideration for move value.

The minimax player is extended with a learning step in order to learn its evaluation function. This player is labeled the TD player because it uses a temporal difference scheme to update its weights after each turn as detailed below. The TD player forms the basis for our analysis of minimax with TD learning as a means to train a Tablut player.

Table 1 - Hand Picked Features

Rule	Starting Weight	Description	Purpose
Seek Corners	1.0	Add value to a state based on the sum of the distances of a player's pieces from their nearest corner. Closer is higher.	Both sides need the corners, white to guard the king(and exit) and black to block the king from exiting
Seek King	1.0	Add value to a state based on the nearness of a player's pieces to the king.	White defends the king, black attacks him, both need to be near
Seek Opponent	1.0	Add value to a state based on nearness of pieces to an opponent piece.	A secondary way to win is via capture of all opponent pieces, and in general capture may help to win, so reward pieces near an opponent.
Capture	1.0	Add value when a state includes a capture	Captures improve chances of win by removing obstacles
Number of pieces	1.0	Add value according to	The side with the most

		the number of pieces relative to starting number	pieces left is likely winning
Freedom	1.0	Update score relative to the number of spaces available to a player	More freedom means the ability to move about the board
Escape	1.0	Used for the king piece only, adds value when a movement places the king in the direct line of sight to an escape node	Encourage king to move to be in line with escape nodes

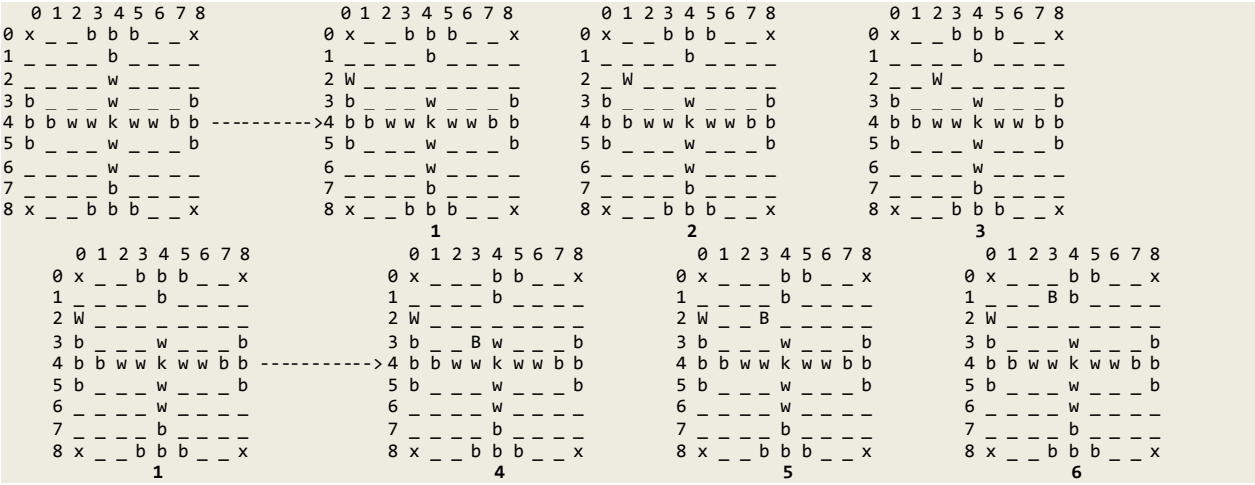
Table 2 - Hand Tuned Feature Weights

	Seek King	Seek Opponent	Seek Corner	Number of Pieces	Capture	Escape	Freedom
Black	5.0	2.0	1.0	1.0	3.0	1.0	1.0
White	5.0	1.0	8.0	3.0	1.0	5.0	2.0

Game Tree

Figure 1 shows the first few states of the first few levels of a generated minimax tree below. The first node is the selected node/current state, other nodes are available states. Indented nodes are expansions of the previous level, here showing only the first node expanded as if moving down the leftmost side of the graph. A minimum of three plies of look-ahead is generated.

To conserve memory and improve efficiency the search tree for each move is generated simultaneously with alpha-beta pruning and scoring of each node. This helps to decrease the space and time complexity of typical minimax, which are O(bd) and O(b^d) where b is the branching factor and d the height of the tree. The average branching factor in the beginning of a game (when there are long open paths) drops from approximately 42 down to 28 under alpha-beta pruning. Later branching factors drop off in a mostly linear fashion, depending on how the game plays out, as there are fewer pieces and shorter paths available to move along once pieces begin to obstruct each other.



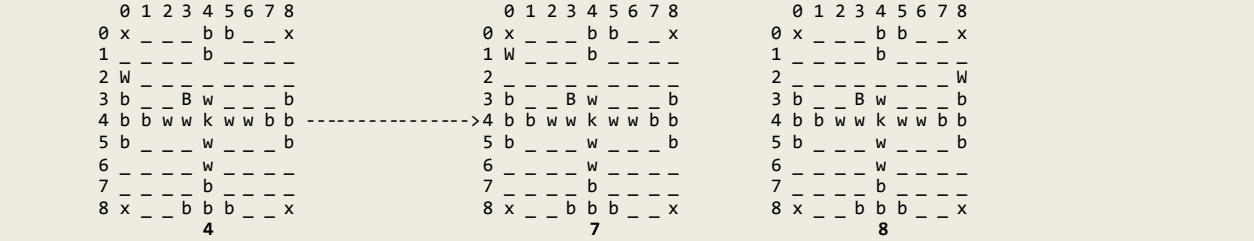


Figure 1 – Small Sample of a 9x9 game tree. Capitalized letters indicate a moved piece. Ww=white, Bb=Black, x=exit node,k=king

Learning

Borrowing heavily from Tesauro's TD-Gammon and Samuel's checkers, we learned an evaluation function using a multi-layer perceptron and a fully incremental version of Temporal Difference with backpropagation. On each turn we update the neural net weights based on the difference between its prediction and the actual value of the subsequent state (following the opponent's turn). Rewards are set to zero unless the last move was the final move of the game, at which point the expected value of the game is one if the player won or zero if not.

Network Structure

We use a three layer feed-forward network with one input per board cell, 49 in a 7x7 game, one input to indicate white's turn and one to indicate black's turn. A final input indicates the number of remaining pieces for the current player for a total of 52 input nodes. There are 50 nodes in the hidden layer and a single output layer indicating the network's prediction of success when using that state. The transfer function is standard sigmoid.

The input data describes the current state of the board by assigning a 0 to empty cells, 1 to a black-occupied cell, -1 to a white-occupied cell and -2 to a king occupied cell.

Results

In order to increase performance we reduced the search space by testing and training on a 7x7 board as opposed to the standard 9x9 board previously discussed. No rules were changed to accommodate this change and we believe the learned feature vectors would work on other size boards though this has not yet been tested. This change resulted in a drop in branching factor from 61 to 42 and ultimately 28 with alpha-beta pruning.

Below we present results from running trials with different combinations of players. The first column is the algorithm being tested. Results are presented from the perspective of the Algorithm, but results for the Opponent can be simply derived. Several themes are explored here – Quality of random players, quality of the fixed minimax algorithms, and success of the TD learner against those players and itself and the ability of the TD learner to generalize.

Note that unlike chess, checkers or backgammon the players here have different pieces, which while they move the same, have different goals. This leads to an additional wrinkle when learning an evaluation function – can we find one which satisfies strategies for both players?

Algorithm	Opponent	Games	Max Turns	Mean game time(s)	Mean Moves	White Win %	Black Win %	Draw %	Search Depth	Notes
Random Player – White	Random Player – Black	1000	150	.01	58	51	49	0	n/a	Random player scores are symmetrical
Fixed Minimax Player - White	Fixed Minimax Player – Black	1000	150	6.08	29	33	47	20	3	See table 1 for feature weights. It appears the weights for black are better than those for white
Fixed Minimax Player – White	Random Player – Black	1000	150	.13	93	52	9	39	1	Single ply is quick but doesn't win more often than random for white
Fixed Minimax Player – White	Random Player – Black	1000	150	2.08	44	62	38	0	3	Deeper searches avoid draws and improve scores at the cost of time
TD White Player	Random – Black	1000	150	.05	41	92	7.4	1	1	Trained on 1000 games against random
TD White Player	Fixed Minimax Black	1000	150	2.66	42	98	1.1	.8	3	Despite never seeing this player, the TD player soundly defeated it.
TD Player Trained as White – Black	Random – White	1000	150	.06	71	54.6	36	9.4	1	Here we test a network trained to play white as black with predictable results given the nature of Tablut.

Table 1 - Selected Algorithm Metrics. All games run on a modified 7x7 Tablut board.

Discussion

The fixed minimax player with hand selected static evaluation function was very difficult to tune. Perturbing weights based on in-game performance and updating weights based on end of game feedback alone proved very inefficient and was ultimately unsuccessful simply due to time constraints. Direct learning of the parameterized weights of these players through an evolutionary algorithm sounds most promising but was dropped in favor of the well-known solution used by TD-Gammon and others to learn the weights of the evaluation function with a feed-forward neural net and temporal difference learning. Despite this, a look-ahead of three moves (the most we ever completed) and a well selected set of feature weights (applied to the features defined above) did generate some success.

The TD Learning approach was much more successful; although it too had drawbacks we'd like to remedy in future work. It appears that the choice of network structure, which emphasizes piece location over exits or other quick winning strategies, leads to a player which emphasizes captures over winning. Boards very quickly are reduced to one or two pieces on a side, leading to many draws (i.e. 150 turns exceeded) or to the white player winning by finally bumping into an exit. A different network structure, which emphasized group movement (for defense of the king) or the locations of the exit nodes, may produce different kinds of play. These have not been explored here but make for possible future work.

Despite that, the combination of minimax, alpha-beta pruning, TD learning and a feedforward neural net did produce a successful player, even if we did not meet our original goals of winning within 25 moves against random players. This is clearly a function of the number of trials we ran and the diversity of the players we trained against as well as the neural net structure.

Code

All code can be found on github at https://github.com/jonathanherr/TablutCS580.

References

Ashton, John C. "The Heroic Age." *Ashton—Linnaeus's Game of Tablut and Its Relationship to the Ancient Viking Game Hnefatafl*. The Heroic Age, 1 Aug. 2010. Web. 18 Apr. 2014.

Barto, Andrew G. "TD-Gammon." *Reinforcement Learning*. By Richard S. Sutton. N.p.: Bradford, n.d. N. *Book*. Bradford. Web. 04 May 2014.

Russell, Stuart J., and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Upper Saddle River: Prentice-Hall, 2010. Print.

"Tafl Games." *Wikipedia*. Wikimedia Foundation, 18 Apr. 2014. Web. 18 Apr. 2014.

Zoran Sevarac. *Neuroph Java Neural Network. Sourceforge, 2008. Web. May 5, 2014.*

Tsinteris, Kimon, and David Wilson. *TD-Learning, neural networks and backgammon. Web. May 5, 2014.*