Evolving a Hex-Playing Agent

Michael McCarver and Rob LeGrand, Ph.D.

Abstract

Hex is a two-player adversarial board game in which there is always exactly one winner. Although it is known that a winning strategy exists for the first player, such strategies are difficult to find due to the large branching factor of Hex's game trees. A subset of Artificial Intelligence research is devoted to optimizing search algorithms, such as minimax, pursuant to searching these game trees and solving Hex boards for any game position. Our research is not concerned with perfect playing strategies. Instead of minimax approaches, we use Artificial Neural Networks and Genetic Algorithms to test the bounds of how quickly and how effectively Artificial Neural Networks are able to learn to evaluate board-states of a game. We experiment with network topology and evolution strategies and compare different approaches using metrics we developed.

*****

Introduction and related work

Hex was invented in 1942 by Piet Hein and independently reinvented in 1948 by John Nash. The board is in the shape of an *n*×*n* parallelogram and is made up of *n*×*n* hexagon shaped tiles. Each player has a store of tiles, colored differently than their opponents'. Players alternate placing tiles on the hexagons of the board, capturing one hexagon per move. The goal of the game is for a player to connect opposing sides of the board. The sides of the board each player aims to connect is determined before the game starts. Figure 1 shows an example of a completed game. John Nash offered an existence proof in 1949 of a first-player advantage, but no general winning strategy for the game is known to exist (Gardner).

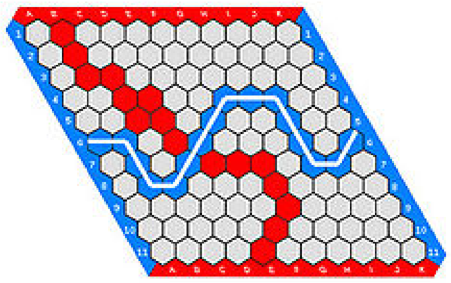

Figure 1. A completed game of Hex on an 11x11 board (from Wikipedia)

Research into Hex is often concerned with solving the game. Here, solving generally means finding perfect moves that give the player a guaranteed win. Levels of "solved" vary from knowing only the best first move (weakly solved) to knowing the best move at all points in the game (strongly solved). As of 2011, humans have only been able to solve by hand some center moves weakly for boards sized 8×8 and 9×9 (Arneson, Hayward and Henderson 2-3). In 2014, Pawlewicz and Hayward developed a Scalable Parallel Depth-First Proof Number Search and used it to solve all previously intractable 9×9 openings and one 10×10 opening. The hardest 9×9 opening took 111 days to solve (1).

Other avenues of research have been explored in the context of board game play. David Fogel trained a neural network with an evolutionary algorithm to teach it to play checkers. The neural network was accompanied with a minimax search of variable depth, where the depth was determined by the amount of time available to make a move. The final agent was able to perform better than 99.61% of players on a checkers-playing forum (Fogel 283). Young, Vasan and Hayward used deep Q-learning to train a CNN (convolutional neural network) to play Hex on a 13×13 board. They began their experiment with supervised training over a database of generated games to accelerate the speed with which the CNN learned early and mid-game strategy. Then, they used self-play to update their heuristic values with a deep Q-learning algorithm. They did not use minimax during gameplay, and they were able to win 20.4% of games as first player and 2.1% of games as second player against a version of the ICGA Olympiad Hex champion (Young, Vasan and Hayward 1).

Experimental Setup

Our research is inspired by the work of David Fogel's Blondie24, and, similarly, we set out to evolve the weights of an ANN using a genetic algorithm. We create a population of 100 Hex players, arranged on a torus so that players at the top and bottom of the map are neighbors, as well as players on the left and right. Each position that a player occupies is a hexagon, so each player is neighbored by six other players. One iteration of evolution consists of 600 matches; each player plays one match as first player and one match as second player against each of its six neighbors. Our algorithm keeps track of the number of games won by each player; we use this statistic in a fitness function during the breeding phase. In breeding, we move through each weight of each player and determine whether to keep the original weight or to replace it, and if we replace it, we use a weighted probability, determined by our fitness function, to choose between our neighbors and our currently looked-at player. After a weight is chosen, we shake the weight by applying a normal distribution with a mean set as the original weight. Finally, with some probability, we randomly swap pairs of weights within the ANN.

One key difference between our research and Fogel’s research is that we do not use a minimax search to guide our evolutions. When a minimax search is used, the player is aware of future board evaluations following a line of hypothetical moves and aware of winning moves in future states of game play. Minimax allows the player to evaluate the consequences of their moves, both from their perspective and their opponent’s perspective. Also, once a winning path of moves is discovered, the player will stop using its heuristic function and rely entirely on the minimax search to determine its next moves. By eliminating the minimax search, we hope to test the bounds of what our ANN is capable of learning concerning the strength of board positions at any point of gameplay. The only mechanic of gameplay that our players are aware of is the validity of potential moves. A sufficiently complicated ANN is capable of learning move validity also, but that is outside the scope of our current research.

We decided on an evolution configuration, which we called “vanilla”, to act as a control for our experiments. The specifications of the vanilla configuration are:

- Two-layered feed-forward ANN with five nodes leading into one final output node.

- Board state is read as a one-dimensional vector so that every index refers to a hex tile on the board. Tiles owned by our player are given a value of one, tiles owned by opponents have a value of negative one, and unoccupied tiles have a value of zero.
- Players are initialized with random values determined by a normal distribution with standard deviation of one and a mean of zero.
- During the breeding phase, the probability that players keep their original weight without evaluating the fitness function (what we call inertia) is set to zero.
- The fitness function is a simple comparison of the number of games won by each player. The probability of a given player being chosen for breeding is the number of games that player won divided by the total number of games won among the currently looked-at set of neighbors.
- The normal distribution used for shaking each weight has a standard deviation of one.
- After the breeding phase of evolution, players will always swap one pair of weights within their ANN exactly once.

Experimental Results

The variables we changed during experimentation were:

- The size and shape of the ANN.
- The probability/number of swaps per player.
- The level of inertia (probability of keeping a player's original weights).
- The function used to apply number of wins as a fitness function.

We generated six experiments with the Vanilla configuration to ensure a diverse baseline, and we generated two experiments of each of the variable strategies (time constraints were a limiting factor to the number of experiments we could generate). We created a few tools to compare the evolutionary configurations with each other. One tool allows us to pick specific players from given experiments and play them against each other, one on one. This is useful for gaining an approximate idea of the types of strategies players from an experimental population are learning, but is limited in that what we see is only truly representative of a single player. To gain a more holistic perspective, we can play every player of a given experimental population against every player of a different population (and against all players of their own respective populations as well), and rank them according to number of wins. This approach allows us to see the relative strengths of evolutionary configurations in respect to one another, but lacks objective measurement. For a more objective look, we play each player of a given population against 2000 random players so that each player plays 1000 games as first-player and 1000 games as second-player. We rank each player by their number of wins. Figure 2 shows these results for the six Vanilla configurations. This evolutionary configuration appears to be stable, in that each experiment produces results that are similar to each other. In contrast, Figure 3 shows the results for the configuration which differed from Vanilla only in that it never swapped weights within networks. This approach produces homogeneous players within a population and has a wide variance of effective playability.

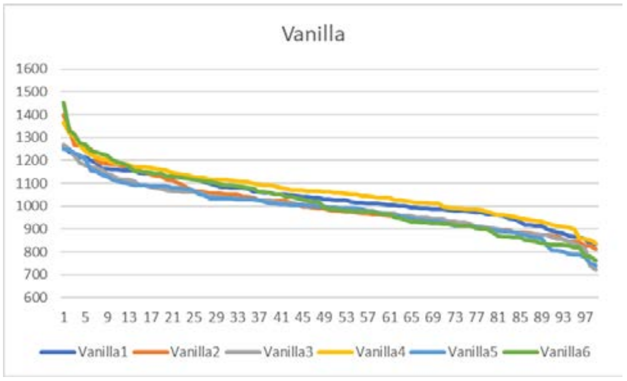

Figure 2. Comparison of all six Vanilla configurations

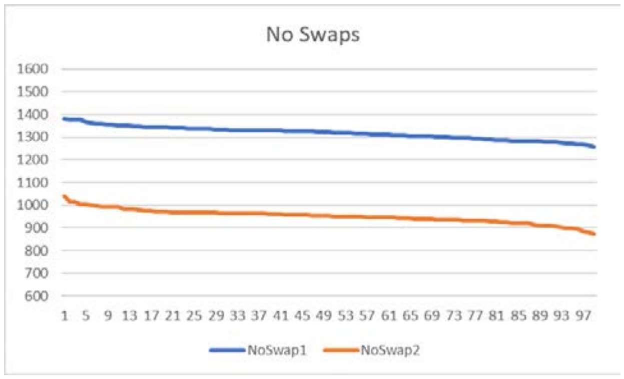

Figure 3. Comparison of configurations without swap mutation

Another experimental configuration yielded interesting results. Figure 4 shows the results of an experiment in which we changed the fitness function used during breeding. Instead of weighing the number of games won by a player in proportion to the total number of games won among a set of neighbors, we favored games won exponentially. We raised 2 to the power of each player's number-of-games-won statistic, and used this exponentiation to weigh our probability proportions. This configuration seems to consistently grow stronger players than the Vanilla configuration.

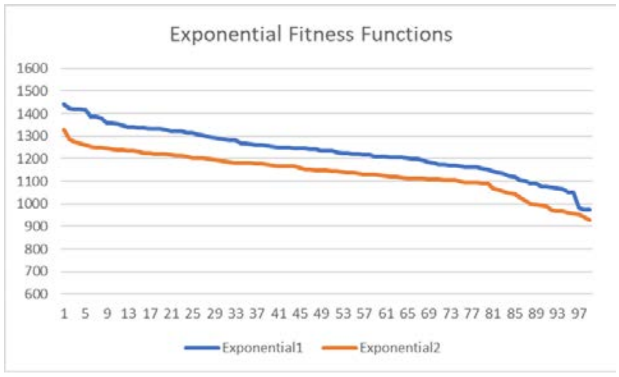

Figure 4. Comparison of variable fitness function

Other variable strategies we tried produced data similar to the six Vanilla data sets. These strategies include:

- Every fitness evaluation including a 50% chance of keeping the original weight (inertia).
- Continuing to make swaps within a network according to a negative binomial distribution, where as long as a certain probability passes, we continue making swaps. We tried this strategy with a 25% and a 50% passing probability.
- A 3-layer network topology of 25 nodes to 5 nodes to 1 output node.

Conclusions

The exponential fitness function configuration seems to be a strong evolutionary strategy, having produced two strong populations. The population that did not use swap mutation created populations of players that lacked diversity, and seemed inconsistent in the types of players that it produced. There are not enough data to determine whether the 25% and 50% probability negative binomial distributions outperformed the Vanilla configuration; more experiments need to be run. This is true also for the 50% inertia configuration. The three-layer ANN didn't perform noticeably better than the vanilla experiments; however, this isn't indicative of a poor strategy. It is possible, and probable, that a more complicated network topology requires more generations of evolution to successfully learn playing strategy. None of our strategies produced players that would beat human players familiar with the game Hex. This is unsurprising, as the state-space for Hex is vast. ANNs are capable of learning any function, even Hex board states, but it is likely that more complicated networks are required, as well as longer experiment iterations, before competitive players are produced.

Future Work

There are many variations to our experiment that would yield interesting research. Growing an ANN without any prior knowledge to game mechanics, including move validity, captures the spirit of our research as well as that of David Fogel's. Other possibilities include:

- Changing the shape of the population space: Currently, 100 players are arranged on a hexagonal torus with local neighborhoods of 6 players. Changing the size of neighborhoods may change the rate and effectiveness of a population in distributing evolutionary changes.
- Re-arranging the positions of players within the population based on their performance during the previous iterations: There may be an ideal distribution of good and bad players throughout a population that assists stable and quick evolution.
- Copying a weight within an ANN instead of swapping weights during the breeding phase: Swapping weights injects a lot of chaos into the breeding phase because newly created weights have no relation to fitness function performance. With a weight swap, two weights have new values, but copying weights only changes the value of one gene. Copying weights may reduce the chaos induced from mutation while maintaining the change desired.
- Changing the method of reading in board-states: The way our code reads in the board-state determines what information agents are aware of in the decision making process. One idea is, instead of storing board-state values as negative one, zero, or one in a single vector, use two vectors to separate inputs from each player. This would allow a network to find different values for opponent-owned and self-owned tiles. Another idea is to include information about the layout of the pieces on the board, such as whether pieces are connected to sides of the board.
- Changing the standard deviation by which to shake genes during breeding: The standard deviation is currently one, so large-valued weights (typically generated after many iterations) will not be drastically effected. A new method that scales standard deviation based on the value of each weight may make each shake more effective.
- Adding more information to be considered in the fitness function: More information about each agent may better direct the fitness function in choosing good players. Incentivizing players to win quickly by keeping track of the total number of turns played by that player may help the genetic algorithm better learn different aspects of the game, such as end-game or possibly early-game.

- Developing a sort of Mohs-hardness scale with which to compare different strategies: Currently, we can compare strategies by their performance against a set of completely randomized hex-players. This method gives us good information about players that don't play well, but as experiments continue and better evolution strategies are discovered, the method may become less useful. A set of tools that measure performance at different levels of game play would assist data interpretation, and could also possibly be included in the fitness function.
- Changing the activation function used: Sigmoid is traditional, but other activation functions, such as the threshold, hyperbolic tangent, softsign, rectifier, leaky rectifier and softplus functions, also allow learning arbitrary concepts and may perform better for a given problem.
- Applying this evolving-neural-networks framework to other games.

References

Arneson, Broderick, Ryan B. Hayward and Philip Henderson. Solving Hex: beyond humans. *7th International Conference on Computers and Games*, pp. 1-10, 2010.

Fogel, David B. *Blondie24: Playing at the Edge of AI*. Morgan Kaufmann, San Francisco, California, 2002.

Gardner, Martin. The game of Hex. *Hexaflexagons and Other Mathematical Diversions: The First Scientific American Book of Puzzles and Games*, pp. 73-83, 1959.

Pawlewicz, Jakub, and Ryan B. Hayward. Scalable parallel DFPN search. *8th International Conference on Computers and Games*, pp. 138-150, 2013.

Young, Kenny, Gautham Vasan and Ryan Hayward. NeuroHex: A deep Q-learning Hex agent. *5th Workshop on Computer Games*, pp. 3-18, 2016.