Using Genetic Programming to Evolve Board Evaluation Functions

Gabriel J. Ferrer and W. N. Martin
Department of Computer Science
Thornton Hall
University of Virginia
Charlottesville, Virginia 22903
{ferrer, martin}@virginia.edu

ABSTRACT

In this paper, we employ the genetic programming paradigm to enable a computer to learn to play strategies for the ancient Egyptian boardgame Senet by evolving board evaluation functions. Formulating the problem in terms of board evaluation functions made it feasible to evaluate the fitness of game playing strategies by using tournament-style fitness evaluation. The game has elements of both strategy and chance. Our approach learns strategies which enable the computer to play consistently at a reasonably skillful level.

1. Introduction

Developing intelligent computer players of strategy games is a problem which AI research has been addressing since the field began. As excellence in the play of strategy games has often been considered to be a sign of intellectual excellence, some felt that developing an intelligent game player could well be a big step on the road to developing a more generally intelligent machine. This paper examines using the genetic programming paradigm [6] in the context of Senet, a strategy game popular in ancient Egypt. As early as 3100 B.C., Senet boards and pieces were among the standard furnishings found in tombs. Murals depicting Senet games in progress were common decorations on the walls of the tombs as well. Its position in ancient Egyptian culture was akin to the position of chess in our culture, the timeless game of strategy. [5] Unlike chess, checkers, and similar games with perfect knowledge, but like backgammon, the play of Senet involves both strategic planning and thought as well as an element of probability that controls what moves are possible on a turn. This probabilistic element makes a straightforward lookahead search process difficult to implement and evaluate. We use the genetic programming paradigm to evolve board evaluation functions (e.g. Samuel's checkers player [7]) from an initial population of randomly generated functions. Evolution proceeds through the mechanism of the-survival-of-the-fittest and in our formulation the fitness of individuals is estimated by having a Senet tournament among the members of the population. This process resulted in the evolution of players who appear to have gained a relatively high level of skill. We also experimented with seeding the base population of randomly generated functions with a small number of handcrafted individuals who already play with some degree of skill.

A brief overview of genetic programming is given in section 2. The details of the application of genetic programming to evolving board evaluation functions for Senet are discussed in section 3. Our experiments, testing methods and results are discussed in section 4. The rules of the game of Senet are described in Appendix A.

2. Genetic Programming

The basic idea of genetic programming is to breed computer programs to solve a particular problem. John R. Koza formalized these principles of genetic programming [6] into a unified paradigm for designing algorithms. The basic outline of the paradigm is roughly as follows. The first generation of computer programs is randomly generated and generally rather low in fitness. Fitness in this context refers to how suitable a solution an individual represents to the problem at hand. This is formalized for each particular problem by a fitness function. For example, we could attempt to evolve a computer program to fit the data elements of a data set, the symbolic regression problem. It would take as inputs a set of values for the independent variables in the function, and produce as outputs values for the dependent variables. The fitness of each individual would be based on the error computed between the values predicted by the individual function and the values in the data set. A small error would correspond to higher fitness. Some individuals in the initial random population will, in spite of being random, nevertheless be more fit than others, and some of these will survive to see future generations and also help fill the spaces (created by the removal of the less fit) with offspring containing various combinations of their genetic material, as well as a small number of random mutations. The pairs of individuals are selected to generate offspring with a preference for the most fit, and their spawn are composed of subexpressions from their parents. Each member of this new population of programs is then evaluated for fitness, and the process continues. Over a large number of generations, the average individual fitness should improve. The single best program in the population at the time of termination is designated to be the result, which may be a solution (or approximate solution [6]) to the problem posed.

3. Evolving Evaluation Functions

In using the genetic programming paradigm a major consideration is the formulation of the problem as evaluable programs. In the game of Senet each turn begins with a randomized construction (by the use of throw sticks) of a set of moves. The player then gets to select the sequence in which the moves in the constructed set are executed. (A summary of the rules can be found in Appendix A, while the detailed rules can be found in [5].) We have chosen to represent each individual player as a board evaluation function [7] [8]. The system can then simulate the play of an individual by constructing the set of moves, applying the represented evaluation function to each move in the set, and executing the moves in the order of preference indicated by the resulting evaluation numbers. Thus, the individuals of our population are functions that take a board position and return an evaluation number. We happen to stipulate that larger numbers indicate moves that the player finds most preferable. In the event that the evaluation function returns the same largest number for multiple moves, a default behavior randomly selects one of the moves with equal probability. Note that with this formulation any function that returns a number whenever executed thus becomes a feasible player. Our system will maintain this property for individuals and thereby avoids the potential complications of introducing individuals into the population who do not play the game legally.

In Koza's paradigm [6] the programs are expressed as LISP functions constructed from a predetermined (and application specific) set of function templates and constants (referred to as terminals). This brings us to the issue of precisely which terminals and functions should be available as components of the player. Some kind of conditional is necessary in order for an individual to do any decision making, so the *if* function was included. Incorporating the logical operators *and*, *or*, and *not* is also a natural requirement for decision making. A series of functions to query the current state of the game was included to insure that an individual would have the potential to use knowledge about the current state of the game to make informed decisions. These state functions include functions to return the number of legal moves currently available to the individual, functions to determine the contents of a particular board location, functions to give the individual the starting location and distance of the current move, and functions to determine the location of the individual's pieces. The integers from 1 to 30 were included as terminals for two reasons. One reason is to indicate a board location to one of the board querying functions. Another reason is to give the individual a mechanism for expressing a preference numerically. The arithmetic operators $+$, $-$, and $*$ (multiplication) were included to enable preferences outside of the 1 to 30 range to be expressed, as well as to facilitate the manipulation of data obtained from the information gathering functions.

A population of individuals is evolved in the following manner. A competition scheme modeled on a sports tournament is used to determine fitness, similar to that used for the game of tic-tac-toe by Angeline and Pollack [1]. The individuals in the population are paired off arbitrarily and then play three games against each other. The individual who wins two games is declared the winner and progresses to the next round of the tournament. The losers at each level all have the same fitness. We chose to evaluate fitness in this fashion because a tournament provides a straightforward method for determining which players in the population are best suited for winning games. The tournament approach is well suited for use with a board evaluation function representation for the players. Selection for both reproduction and crossover is done in a rank-proportionate manner [2]. One-eighth of the new population is reproduced from the original population at each new generation, with the remainder of the new population being generated via crossover. Since it is theoretically possible for a Senet game to last indefinitely, a limit of 500 turns was imposed on the duration of a game. Our experiments have shown that Senet games rarely take even half that many turns to complete, so we believe that 500 turns is a very reasonable limitation. In a sample of 38,100 games, the average number of turns played per game was approximately 158.

In the event that a game lasts as long as 500 turns, the victor is determined randomly. This evolutionary process ends whenever the last requested generation has been completed.

The reproduction operator copies selected individuals from the old population into the new population. For each individual reproduced, there is a one in eight chance that that individual will mutate. A mutation consists of replacing a randomly selected subtree of the individual with a function generated randomly in the same manner that the individuals in the initial population are generated. The crossover operator works by randomly picking one parent to be the base function for the child. The root of that parent's parse tree will have at least two children. One of those children nodes will be randomly selected. If that child node is an terminal, then it is the crossover point and a random subtree of the second parent is spliced into the tree. Otherwise, there is a one in three chance that the subtree rooted at that child is replaced by a random subtree of the second parent. If this does not happen, the child node is treated as the root node as before and the process repeats.

The initial population of board evaluation functions is generated as follows. For each individual, a root function is selected which is a function requiring at least 2 arguments. Another function or terminal is randomly selected for each argument. If it is a function, more functions or terminals are randomly determined to be its arguments, and so forth. This growing process terminates when the terminals, which are either numbers or move queries with no arguments, are encountered. We evolved four different populations in order to determine whether this collection of techniques would produce interesting Senet players. All of the populations were of size 512 (the power of two slightly simplifies the tournament process).

The creation and crossover mechanisms described above are slightly different than Koza's [6]. The primary implication of this difference is that the size of the individuals is not limited. In a future study, we plan to investigate the importance, if any, of this difference.

4. Experiments and Results

The element of probability in Senet resulting in the lack of perfect knowledge about the game makes it difficult at best to know what an optimal board evaluation function for Senet should be. This requires us to create an empirical evaluation scheme. The fitness of the players for evolutionary purposes is determined by how good they are at defeating each other. But how do we know that the overall winner, that is, the tournament winner in the last generation, has any objective value whatsoever? Unlike previous efforts to implement computer game players (such as [1] [7] [8]), Senet is not a well-understood game. There is no literature describing good Senet strategy, not even from the ancient Egyptians. We have nevertheless attempted to develop and use a series of baseline players. Each baseline player is implemented from the same set of terminals and functions available to the individuals being evolved through genetic programming, and are subject to the same default behavior. In order to quantify the performance of the evolved individuals against the baseline players, we used a statistical confidence interval method [3] to enable us to state with 95% certainty how frequently an evolved individual will win against a particular baseline strategy, with a margin of error of plus or minus 5%.

One group of baseline players used for testing purposes is a set of eight random individuals (referred to as RND1 through RND8 below) generated by the code that creates initial populations. The other group consists of five handcrafted players (referred to as HC1 through HC5 below) inspired by strategies we have used in playing the game. The players in this group use a variety of combinations of some of the following strategies:

- Minimize the number of my pieces on the board.
- Maximize the number of my opponent's pieces on the board.
- Avoid the Waterhouse pitfall.
- Place my opponent in the Waterhouse pitfall.
- Maximize the distance between my opponent's pieces and the goal.
- Capture an opponent's piece.
- Set up a barricade.

One additional player we included is an individual that always returns a constant for each board position, which in conjunction with the system's default behavior has the effect of selecting a random move at each turn, with an equal probability of selecting each move in the set constructed for the turn. (This player will be referred to as Const below). For the results presented below, we compared pairs of players by having them play a set of 200 games against each other. Note that this testing phase is different than the tournament structure used for fitness calculation and is done after the fact as an external quality measure. Figure 1 shows how the best evolved players did against eight randomly generated opponents, as well as against Const described above. Figure 2 shows how these same players did against the five handcrafted players described above.

Two of our test populations were evolved from initial populations consisting entirely of randomly generated individuals. The player labelled First Unseeded in figures 1 and 2 was the top evaluation function in the first population after 33 generations of evolution. Second Unseeded was the top evaluation function in the second population after 30 generations of evolution. (The total number of generations is different for the runs because, unfortunately, the termination point of each run was when the LISP system ran out of memory. This was a consequence of our experimental decision to allow the trees representing the players to grow arbitrarily in size.) Figure 1 shows that First Unseeded and Second Unseeded won no fewer than 145 victories out of 200 against each of these opponents, evidence that the results of this application of genetic programming were substantially better than simply generating random individuals, and also superior to making uniformly random moves.

We also tested First Unseeded and Second Unseeded against the baseline players. Figure 2 shows the results. First Unseeded and Second Unseeded continue to demonstrate a reasonable level of skill at playing Senet, although Second Unseeded did have some trouble playing against HC2. Overall, though, they consistently match or exceed the performance of the handcrafted baseline players.

We wanted to see how population seeding would work in the context of genetic programming, since it has been used successfully before with more standard genetic algorithms [4]. Our expectation was that incorporating some of our knowledge about the game would likely improve the results of the search process, as the baseline individuals provide promising directions for where the search might proceed. At the same time, we seeded the population with only a small number of individuals in order to assure diversity in the population.

Two additional runs were evolved from initial populations which contained 504 randomly generated individuals and 8 individuals who were copies of some of the baseline strategies mentioned above, in order to observe the effect of incorporating handcrafted features into the population. For each run, the randomly created part of the initial population was different than any of the others.

First Seeded was the top player after 23 generations starting with a partly seeded initial population. Second Seeded was the top player of its population after 22 generations starting with a partly seeded initial population. The 8 individuals used to seed the run for First Seeded were also used in the run for Second Seeded. As with the unseeded players, figures 1 and 2 demonstrate the good performance of First Seeded and Second Seeded against the random players and baseline heuristics. First and Second Seeded also tended to win more often against the same opponents compared with First and Second Unseeded, indicating that the search was improved by incorporating knowledge into the initial population.

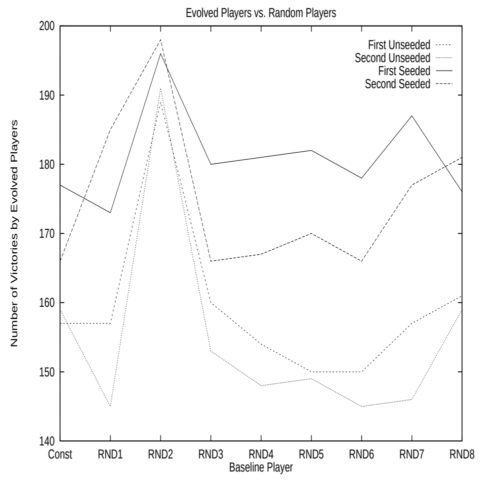

Fig. 1: Evolved Players vs. Random Players

These experiments have shown that using genetic programming with the representation described above results in the evolution of board evaluation functions which can play the game of Senet with a reasonable level of skill.

References

[1] Peter J. Angeline and Jordan B. Pollack. Competitive environments evolve better solutions for complex tasks. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 264–270, 1993.

[2] J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, 1985.

[3] John E. Freund and Ronald E. Walpole. *Mathematical Statistics.* Prentice Hall, 1980.

[4] J. Grefenstette. Incorporating problem specific knowledge into genetic algorithms. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, 1987.

[5] Timothy Kendall. *Passing Through the Netherworld.* Kirk Game Company, 1978.

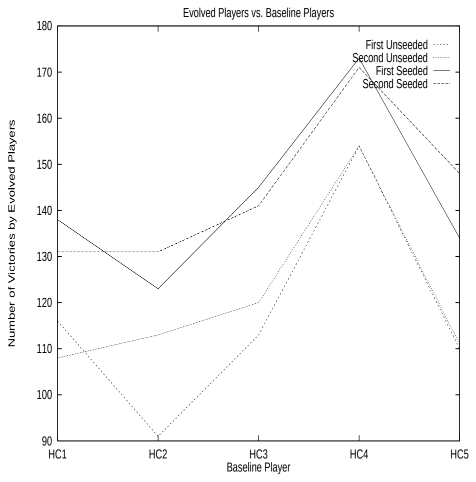

Fig. 2: Evolved Players vs. Handcrafted Heuristics

[6] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[7] Arthur Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development 3*, pages 210–229, 1959.

[8] Gerald Tesauro. Practical issues in temporal difference learning. In *Machine Learning 8*, pages 257–277, 1992.

A. The Rules of Senet

The original rules of Senet are unknown to us, but a good approximation of what they were like is not impossible to find. Versions of the game have been found with anywhere from 5 to 10 pieces per side, although 7 pieces per side appears most common, and is the number used for each side in this implementation. For this implementation of Senet, the rules were based upon the rules developed by Timothy Kendall for a version of Senet he published as a boardgame. Here is a summary of Kendall's rules, paraphrased directly from his rule booklet:

Set-up The 7 spools and the 7 cones are set up alternating with each other in the first 14 squares of the board. The pieces advance in an S-shaped course along the board, going left-to-right in the first row, right-to-left in the second row, and left-to-right again in the third row. A picture of the Senet board at the start of the game can be seen in figure 3.

Objective To move all of one's pieces off the board before one's opponent succeeds in doing so.

Determining the Moves There are 4 throw-sticks used in the game, each on e of them bearing a marked and an unmarked side. There are 5 possible resulting configurations, which have the following results: (1 = marked side up, 0 = unm arked side up)

1110: Move one house and get an extra throw.
1100: Move two houses. End of turn.
1000: Move three houses. End of turn.
0000: Move four houses and get an extra throw.
1111: Move five houses and get an extra throw.

A player should not move until all throws are completed. When the throwing is done, the throws may be used in any order or combination desired, one at a time. Once a throw is used, it is gone.

The opening move The players alternate throwing the sticks, handing them over to the other player upon a throw of 2 or 3. The first player to throw a 1 has the cones as his pieces and moves first. His opponent has the spools as his pieces and begins playing on the next turn.

Moving, Hitting, and Defending Since no two pieces may simultaneously occupy the same house, a player may only move a piece on his turn to a vacant house or a house occupied by an undefended enemy piece. A defended piece is a piece which has at least one of its friends in a house either immediately preceding or following it. When an undefended piece is hit, it trades places with the opposing piece which attacked it.

Forced Moves If a player cannot use a throw to move a piece forward, then he must move the piece backward the appropriate number of houses. If this causes the piece to land on an opposing piece, defended or not, the opposing piece gets to move forward at the original piece's expense. If there are no possible backward moves, the remaining throws are forfeit.

Endplays To bear pieces off the board, the player must deal with special rules governing movement through houses 26 to 30. To move a piece beyond house 26 (the "Beautiful House"), it must first land there by exact count. It may be subsequently carried forward as follows: On a throw of:

5: Bear off at once.
4: Move to house 30. Bear off on any future throw.
3: Move to house 29. Bear off on a future throw of 2.
2: Move to house 28. Bear off on a future throw of 3.

None of a player's pieces can be borne off the board until the player has completely vacated

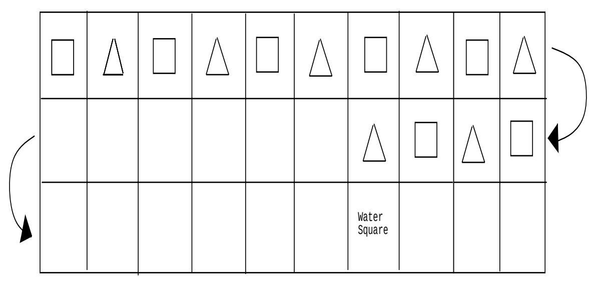

Fig. 3: The Senet Board

the first ten squares. Once a piece makes it into one of the above houses, it is no longer required to travel backwards as described earlier.

The Water House This house is a pitfall that should be avoided if possible. A piece can land here either by moving 1 from house 26 (usually avoided, since a throw of 1 always enables another throw) or by getting bumped there by an opponent going from house 26 to one of the end houses. When a player has a piece in this house, he loses a turn. If he throws a 4 on his next turn, he may bear the drowned piece off the board. Otherwise, the drowned piece is transported to house 15, the "House of Repeating Life", or the nearest vacant house preceding it. In addition, while a player's piece is drowned, none of his other pieces are considered defended.

As previously mentioned, the above rules are based upon Kendall's published version of Senet. A few rules were very slightly changed to simplify some of the programming. The above set of rules is what was implemented.