A Genetic Minimax Game-Playing Strategy

Tzung-Pei Hong Ke-Yuan Huang Wen-Yang Lin
Department of Information Management
I-Shou University
Kaohsiung, 84008, Taiwan, R.O.C.

Abstract

In this paper we consider the problem of finding good next moves in two-player games. Traditional search algorithms, such as minimax and $\alpha-\beta$ pruning suffer great temporal and spatial expansion when exploring deeply into search trees to find better next moves. The evolution of genetic algorithms, abilities to find global or near global optima in limited times seems promising, but they are inept at finding compound optima, such as the minimax in a game search tree. We propose a new genetic-algorithm-based approach that can find a good next move by reserving the board evaluation values of new offspring in a partial game-search tree. Experiments show that solution accuracy and search speed are greatly improved by our algorithms.

1. Introduction

Computer game-playing was one of the first and most interesting inquiries into Artificial Intelligence, and study has continued [11][14][16] from the first chess-playing machine proposed by Shannon in 1950 [17], to the Deep-Blue chess system developed by IBM that recently even beat human world chess champion. This latest success shows the great potential applying artificial intelligence to computer game-playing holds, and further inspires research into machine learning. Study of more complicated games, such as Go (Wei-Chi) is still in its infancy.

Three factors are usually involved in improving a computer chess system's proficiency - chess domain knowledge [2][13], hardware, and its search algorithm. Chess domain knowledge depends on the kind of chess played, and hardware speed depends on computer technology. The third key mechanism is the game search algorithm, which emulates human thinking and explores possible opponent moves in suggesting best next moves. Usually, the deeper the game search tree, the more accurate the prediction.

Traditional two-player game search algorithms [1], such as *minimax* [3][4][8] and $\alpha-\beta$ *pruning* [9] suffer great temporal and spatial expansion when exploring deeply into search trees to find better playing strategies. In this paper, we apply genetic algorithms to game-tree searches to improve performance. We propose a genetic minimax search strategy for determining good next moves that employs conventional genetic algorithms (GAs). It uses evaluation values calculated from a partial games-tree search to generate offspring. After generation, the best solution is output as the next game move. Experiments are presented to show that the effectiveness of our proposed algorithms and the results show that the proposed method finds more accurate solutions more quickly than the original minimax search strategy. The proposed algorithm is thus practical and can be applied to real computer games.

An outline of this paper is as follows. In Section 2 we provide some background information on game-search trees and genetic algorithms. Section 3 delineates our proposed method, including the chromosome representation, crossover and mutation operations, fitness functions, and the reservation-tree technique. Section 4 reports on an experiment in which we compared our method with the well-known minimax search. As the results show, our approach was accurate in finding optimal solutions and outperformed the minimax in space and time. Conclusions and some future work proposals are discussed in Section 5.

2. Background

2.1. Two-player game trees and minimax search

In a two-player game, such as chess, players move in turn, each trying to beat the other according to specific rules. The players' moves can be modeled using a structure known as a *game tree*. Each node in the game tree represents a game state; for example, in chess, it can represent the positions of all the pieces on the board. Each branch in the tree corresponds to a move. By convention, the two players are denoted as *MAX* and *MIN*. We assume that *MAX* moves first, and that nodes on odd-numbered tree levels correspond to instances in which *MAX* moves next; these are called *MAX* nodes. Nodes on even-

0-7803-4869-9/98 $10.00©1998 IEEE

numbered levels correspond to instances in which *MIN* moves next; these are called *MIN* nodes. Terminal nodes are those corresponding to winning situations for *MAX*, as determined by a specific function called an *evaluation function* that obtains values representing the degree which *MAX* is favored to win. Figure 1 is an example of a game tree with 3 levels.

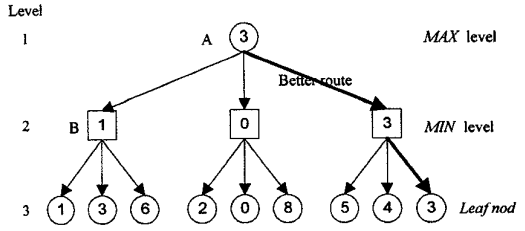

Figure 1. An example of a game-search tree with 3 levels.

The game-tree search objective is to find a winning strategy for *MAX*; that is, to find a good next move for *MAX* such that no matter what *MIN* does thereafter, *MAX* has the greatest chance to win the game. A well-known method called the *minimax* search has traditionally been used. The name was inspired by the game tree definitions; nodes on the *MAX* (*MIN*) level correspond to situations favorable to *MAX* (*MIN*). The minimax search first explores the game tree to a predefined depth using a depth-first search, evaluating leaf values, and propagating these values level-by-level upward to the root in accordance with the minimax principle: compute the maximum for *MAX* nodes and the minimum for *MIN* nodes. Several strategies have been proposed for improving the efficiency of the minimax strategy [3] [10][15].

2.2. Genetic algorithms

The Genetic Algorithm (GA), first introduced by John Holland in 1975 [7], is an approach that mimics biological processes for evolving optimal or near optimal solutions to problems. Beginning with a random population (group of chromosomes), it chooses parents and generates offspring using operations analogous to biological processes, usually crossover and mutation. Adopting the survival of the fittest principle, all chromosomes are evaluated using a fitness function to determine their fitness values, which are then used to decide whether the chromosomes are eliminated or retained to propogate. The more adaptive chromosomes are kept and the less adaptive ones are discarded in generating a new population. This new population replaces the old one and the whole process is repeated until specific termination criteria are satisfied. The chromosome with the highest fitness value in the last population gives the solution.

3. A genetic minimax search algorithm

As stated above, the GA is an innovative approach composed of many biological imitations, such as the chromosome representation, genetic operators, population selection, and fitness functions. Below, we state the primary issues involved in applying the GA to two-player games, and propose a new genetic minimax search algorithm for resolving them.

3.1. Chromosome representation

The encoding scheme is the first step in, and a key part of using GAs. To make available the crossover and mutation operations, and enhance the performance of the algorithm, a chromosome representation that stores current solution states is desirable. For two-player game-tree searches, each possible move in the search tree is encoded as shown in Figure 2: each branch number represents a move choice. Circles represent moves on the *Max* level and squares represent moves on the *Min* level. Any path from the root to a leaf node denotes a possible move sequence, which can be coded as a sequence of branch numbers. Thus, node *A* is coded as 11, *B* is 12, *C* is 22, and *D* is 32. Note that the chromosome length is determined by the depth of the search tree. Increasing the searching depth by one more level only requires increasing the length of each chromosome by only one additional number, which is small overhead for a GA.

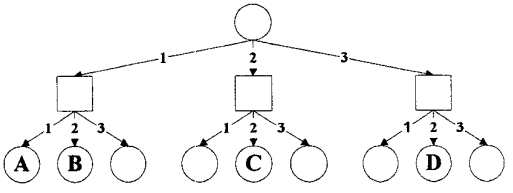

Figure 2. The coding scheme for a game-search tree.

3.2. Crossover and mutation

The crossover operation is used to generate offspring by exchanging bits in a pair of individuals (parents) chosen from the population. There are diverse forms of crossovers mentioned in the literature, but each has its limitations, which must be considered in light of the characteristics of the problem in question. Here, we adopt the simplest one-point crossover for two reasons. First, a substring represents a sequence of game moves and a better substring corresponds to a better choice of game strategy. By exchanging substrings, the one-point

crossover tends to keep the superior parts of parents. Second, from an implementation stand point, such an operation is easily realized using a pointer when chromosomes are stored as linked lists.

An example of applying the crossover operation to a two-player game is shown in Figure 3. Each offspring move sequence inherits some move patterns from its parents.

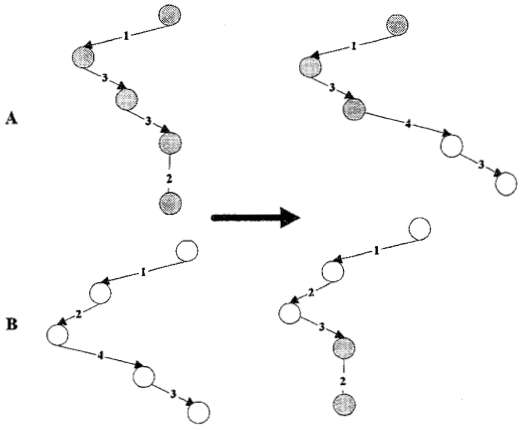

Figure 3. One-point crossover for two move sequences.

The mutation operator is used to randomly change some elements in selected individuals and leads to additional genetic diversity to help the search process escape from local optima traps. The mutation operator shown in Figure 4 is used in the proposed algorithm.

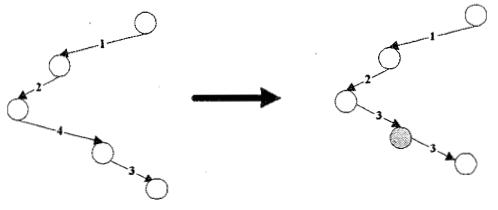

Figure 4. Mutation for a move sequence.

3.3. Fitness evaluation using a reservation tree

Designing the fitness function is important to create an effective GA, and it may be the most time-consuming and critical operation of all. The fitness function evaluates each chromosome and generates values representing their degrees of fitness. Chromosomes with smaller fitness values are usually discarded to increase the probability of retaining population superiority.

The minimax principle has to be used to justify the fitness of any new individual in a two-player game-search tree. Therefore, chromosomes must be compared with other chromosomes in the population to get accurate fitness values. This means a mechanism is needed by which the fitness of all chromosomes ever seen can be retained. Inspired by the minimax search, we propose a structure called the reservation tree to aid in fitness evaluation. The idea is explained below using an example.

Consider the search tree in Figure 5. Assume the two paths on the left represent, respectively, individuals **A** and **B**, which were randomly generated in the initial genetic minimax algorithm population. The resulting reservation tree, which is used to evaluate the fitness values of these two individuals, is formed by adding **A** and **B** to an initially null tree. The result is shown at the right of Figure 5, where **U** represents the least common ancestor node of **A** and **B**. Here node **U** is on a *Min* level. The evaluated value of **U** is 5 according to the minimax evaluation and so as the root node *R*.

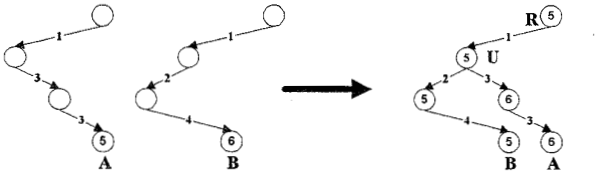

Figure 5. The reservation tree that results from combining two individuals, *A* and *B*.

Suppose a new individual **C**, coded as 134, is generated. **C** is also connected to the reservation tree at node **V**, as shown in Figure 6. Since **V** is on a *Max* level, its board evaluation value is updated from 6 to 7 and is propagated upward to node **U**. Since the original board evaluation value of **U** was 5 and is smaller than the value propagated from **V**, it is left unchanged.

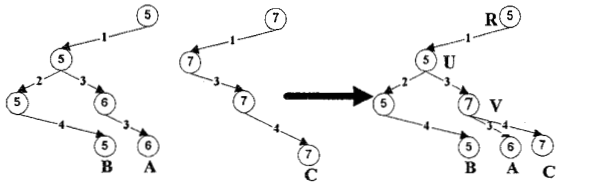

Figure 6. The reservation tree resulting from considering individual *C*.

In the reservation tree derivation above, we connected all inspected individuals to a tree structure, evaluated each individual's leaf node, and then propagated the board evaluation values upward along the current partial game tree. During propagation, the value of each node was evaluated using the minimax principle; that is, to nodes on the *MAX* level maximum evaluation was applied while to ones on the *MIN* level minimum evaluation was applied.

After all chromosomes in the current population have been attached to the reservation tree, their genetic fitness values are then calculated for use in genetic operations.

As is well known, genetic fitness values represent the importance of chromosomes in forming the next generation. It is thus necessary to define an appropriate fitness evaluation function to reflect the importance of good chromosomes. In a two-player game-search tree, the best move sequence (path) is the one whose leaf value is propagated along the path to the root node in the reservation tree. That is, each node on the path has the same value as the leaf node. A path whose leaf value can be propagated to a higher level can thus be thought of as more important. The fitness function in the proposed genetic minimax algorithm is thus defined as *the height from the bottom that the leaf value of a chromosome (path) can attain.* More precisely, if the height of the reservation tree is H, and the highest level that a chromosome's leaf value can reach is K, then the genetic fitness value of this chromosome is $H-K+1$. The larger the fitness, the closer the leaf value of a chromosome is to the root, meaning that the chromosome is more important to forming the next population generation. For example, in Figure 6, the highest levels from the bottom that the leaf values of ***A, B*** and ***C*** can attain are, respectively, 1, 4 and 2. Hence **B** has the greatest chance of being passed on to the next generation.

3.4. Description of the genetic minimax search algorithm

The genetic minimax search algorithm first uses the proposed chromosome representation scheme to describe the state of a two-player game. It next generates an initial population and performs three genetic operations (crossover, mutation, and reproduction) to generate the next generation. It uses the reservation tree to help in the evaluation process. The above procedure is then repeated until the termination criteria are satisfied. Finally, the chromosome with the greatest fitness value is output as the best solution. The genetic minimax search algorithm is described as follows.

The genetic minimax search algorithm:
Input: The current game situation.
Output: A best or nearly best next move.

Step 1: Define the time limit and the maximum number of generations.
Step 2: Define the depth of the game-search tree.
Step 3: Use the proposed coding scheme to code all possible situations in the game-search tree.
Step 4: Define the leaf-node board evaluation function (which is different from that used in genetic evaluation).
Step 5: Define the crossover and mutation rates.
Step 6: Initialize the reservation tree with only a root node value of $-\infty$.
Step 7: Randomly generate an initial population of N chromosomes and evaluate the leaf value of each chromosome using the board evaluation function defined in Step 4.
Step 8: Insert the initial population into the reservation tree, and update the node values in the tree using the minimax operation..
Step 9: Execute crossovers at the crossover rate to generate offspring.
Step 10: Execute mutations at the mutation rate to generate offspring.
Step 11: Delete any duplicate offspring.
Step 12: Compute the leaf values of all offspring chromosomes using the evaluation function defined in Step 4.
Step 13: Insert all offspring chromosome into the reservation tree and update the node values using the minimax operation.
Step 14: Compute the genetic fitness value of each individual in the reservation tree using the proposed evaluation function defined in Section 3.3.
Step 15: Select the superior N individuals according to their fitness values to form the next generation.
Step 16: If the termination criteria have been reached, then stop; otherwise jump to Step 9.

After Step 16, the chromosome with the best genetic fitness value is chosen as the best move sequence, and the first move in the sequence is the next move to play.

4. Experimental results

This section reports on experiments made to show the effectiveness of the proposed genetic minimax algorithm. They were implemented in Turbo-C on a Pentium-PC. Our proposed method was compared with the original minimax method in accuracy and execution time. A computer game-search tree with a predefined number of levels was first generated at random. The minimax method, which is basically an exhaustive search, was used to find the best solution for a specific level in the search tree, which route is ranked as first. Other solutions were also ranked in accordance with their board evaluations using the minimax method. The route obtained via the genetic minimax method was then ranked using the ranking order from the original minimax method. Then the bias of the route obtained using our method with the best solution from the original minimax method was used to justify the quality of the route. The smaller the bias, the better the quality of our method.

A randomly generated 7-level search tree with 7 branches was used in the experiments. The population size was set at 40. The experimental results for the 7-level game tree using 40 generations are shown in Table 1; *number of levels* means the level of which the minimax and the genetic minimax methods searched.

Table 1. Accuracy of the search methods for the 7-level game tree.

Number of levels	1	2	3	4	5	6	7
Rank of the minimax	1	1	1	1	1	1	1
Rank of genetic minimax	1	1	2	1	2	1	2
Bias from the first rank	0	0	1	0	1	0	1

From Table 1, it is easily seen that the route obtained by our method is very close to the best route found by the original minimax method.

We next compared the execution times required by the two search methods. The result is shown in Figure 7. Obviously, the minimax method performed well when the game tree is small, but execution time increased astonishingly as the number of levels is increased. By contrast, the execution time required by our method increased very slowly.

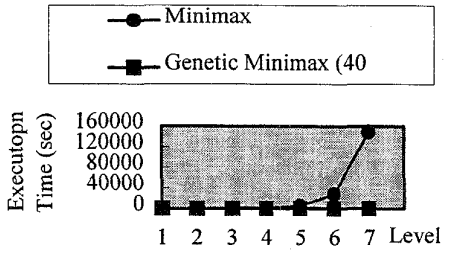

Figure 7. Search methods' execution times.

5. Conclusions

In this paper, we have proposed a new search algorithm for game-search trees. Our algorithm is a GA-based method, which is remarkable for its introduction of a reservation tree to help in genetic fitness evaluation. As our experimental results show, our method not only accelerated the search speed but also found a more accurate solution in a limited amount of time. The proposed method is thus practical as a search algorithm for computer game-playing. In the future, we will attempt to apply the genetic minimax algorithm to some real computer games, such as computer chess, and to further expand our method to multi-player games [10][12].

References

[1] B. Abramson, "Control strategies for two-player games," ACM Computing Survey, Vol. 21, No. 2, 1989, pp. 137-161.

[2] B. Abramson, "On learning and testing evaluation functions," *Journal of Experimental and Theoretical Artificial Intelligence*, Vol. 2, No. 3, 1990, pp. 241-251.

[3] B. W. Ballard, "The *-minimax search procedure for trees containing chance nodes," *Artificial Intelligence*, Vol. 21, 1983, pp. 327-350.

[4] M. S. Campbell and T. A. Marsland, "A comparison of minimax tree search algorithms," *Artificial Intelligence*, Vol. 20, 1983, pp. 346-367.

[5] L. Davis, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991.

[6] D. E. Goldberg, *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison Wesley, 1989.

[7] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[8] H. Kaindl, "Minimaxing - theory and practice, *AI Magazine*, Vol. 9, No. 2, 1988, pp. 69-76.

[9] D. E. Knuth and R. W. Moore, "An analysis of alpha-beta pruning," *Artificial Intelligence*, Vol. 6, 1975, pp. 293-326.

[10] R. E. Korf, "Depth-first iterative-deepening: an optimal admissible tree search," *Artificial Intelligence*, Vol. 27, 1985, pp. 97–109.

[11] K. F. Lee and S. Mahajan, "The development of a world class othello program," *Artificial Intelligence*, Vol. 43, 1990, pp. 21-36.

[12] D. Mutchler, "The multi-player version of minmax displays game-tree pathology," *Artificial Intelligence*, Vol. 64, 1993, pp. 323-336.

[13] J. Nievergelt, "Information content of chess positions", *SIGART Newsletter*, Vol. 62, 1977, pp. 13-14.

[14] J. Pitrat, "A chess combination program which uses plans," *Artificial Intelligence*, Vol. 8, 1977, pp. 275-321.

[15] R. L. Rivest, "Game tree searching by min/max approximation," *Artificial Intelligence*, Vol. 19, 1988, pp. 77-97.

[16] P. S. Rosenbloom, "A world-championship-level othello program," *Artificial Intelligence*, Vol. 19, 1982, pp. 279-320.

[17] C. E. Shannon, "Programming a computer to play chess," *Philosophy Magazine*, Vol. 41, 1950, pp. 256–275.