Adversarial Search by Evolutionary Computation

Tzung-Pei Hong tphong@nuk.edu.tw
Department of Electrical Engineering, National University of Kaohsiung, Kaohsiung, Taiwan 826, ROC

Ke-Yuan Huang
Institute of Electrical Engineering, Chung-Hua University Hsinchu, Taiwan 300, ROC

Wen-Yang Lin wylin@isu.edu.tw
Department of Information Management, I-Shou University, Kaohsiung, Taiwan 840, ROC

Abstract

In this paper we consider the problem of finding good next moves in two-player games. Traditional search algorithms, such as minimax and α-β pruning suffer great temporal and spatial expansion when exploring deeply into search trees to find better next moves. The evolution of genetic algorithms with 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 thus 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 algorithm.

Keywords

Two-player game, game search tree, minimax search, genetic algorithm, reservation tree.

1 Introduction

Computer game-playing was one of the first and most interesting inquiries into Artificial Intelligence, and study has continued (Lee and Mahajan, 1990; Pitrat, 1977; Rosenbloom 1982; Slate and Atkin, 1977) from the first chess-playing machine proposed by Shannon in 1950, to the Deep-Blue chess system developed by IBM that recently even beat human world chess champion. This latest success shows the great potential of applying artificial intelligence to computer game playing, 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 (Abramson, 1990; Nievergelt, 1977), 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 (Abramson, 1989), such as *minimax* (Ballard, 1983; Campbell and Marsland, 1983; Kaindl, 1988) and α-β *pruning* (Knuth

and Moore, 1975) 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. Genetic algorithms (GAs) (Davis, 1991; Goldberg, 1989) have become increasingly important research tools for solving difficult problems since they can provide feasible solutions in limited amounts of time. They were first proposed by Holland in 1975 and have been successfully applied to the fields of optimization (Goldberg, 1989), machine learning (Goldberg, 1989; Wang et al., 1998), neural networks (Miller et al., 1989), and fuzzy logic controllers (Thrift, 1991), among others (De Jong, 1980; Kuncheva, 1993).

The evolving abilities of genetic algorithms to find global or nearly global optima in limited times seems promising, but they were rarely used for game-tree search, due primarily to the fitness functions in GAs, which usually find only maximum or only minimum values, but not iterative minimax values. However, minimax evaluation is necessary in two-player games. We thus 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 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 Adversarial game trees and search methods

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 an *adversarial 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-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 game 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 (Brassard and Bratley, 1996; Campbell and Marsland, 1983; Kaindl, 1988; Winston, 1992) has traditionally been used. The name was inspired by the game tree def-

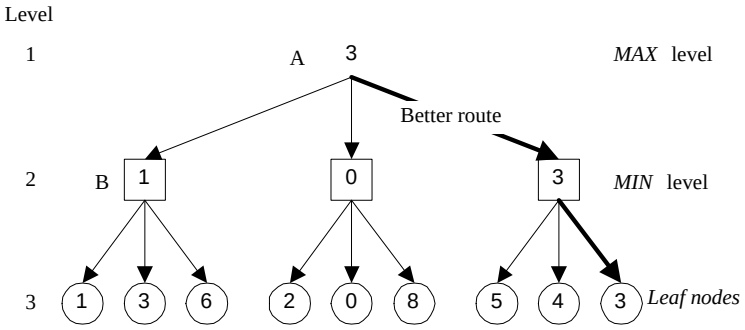

Figure 1: An example adversarial game-search tree with 3 levels.

initions; 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. For example, in Figure 1, the minimum of 1, 3, 6 is chosen for node **B** since it is on a MIN level, while node **A** is on a MAX level, so it gets the maximum of 1, 0, 3. Finally, the root receives a value of 3, which means the solution route is the right-most path downward to the leaf node for a board evaluation value of 3.

Several techniques have been proposed for improving the efficiency of the minimax strategy (Ballard, 1983; Kaindl, 1988; Knuth and Moore, 1975; Pitrat, 1977; Rivest, 1988). The most well known is the α-β *pruning* (Knuth and Moore, 1975). Rather than exploring the whole game tree to obtain the most promising next move, the α-β pruning approach can abandon the exploration of some branches whose evaluation has no effect on evaluating the higher-level nodes. For example, assume we start the search from the third branch of node **A** in Figure 1, which returns 3. Next assume we perform depth-first exploration of the second branch. When we reach the first node on level 3 and obtain value 2, we can stop exploring the other two branches on level 2 since the minimum of these branches is at most 2, which is less than 3. The last evaluation of the root cannot thus be changed from the second subtree. It can been shown that for a game tree with branching factor of b and depth of d, the number of nodes explored by alpha-beta pruning is approximately $2b^{d/2}$ for the best cases (knuth and Moore, 1975; Winston, 1992) and $(b/\log b)^d$ for the average cases. Still, the work required becomes impossibly large with increasing depth. Other improvements such as *iterative deepening*, *quiescence search*, *move ordering*, and *forward pruning* can be found in (Barr and Feigenbaum, 1989; Bolc and Cytowski, 1992; Winston, 1992).

In the field of algorithms, it is sometimes helpful to make decisions randomly instead of spending lots of time on deciding which alternative is the best choice, especially when the time required to obtain the optimal choice is prohibitively high. This concept builds the foundation of a class of algorithms called the *probabilistic algorithms* (Brassard and Bratley, 1996). There are various forms of probabilistic algorithms, each having its fitting situations and limitations. For two-player games, a class called the *Las Vegas algorithms* (Brassard and Bratley, 1996) seems to be promising. The basic

paradigm of these algorithms is repeatedly making random decisions to obtain candidate solutions, evaluating their correctness, and returning the solutions if predefined requirements are satisfied or if an allowable time limit is reached. A Las Vegas search approach, however, has some problems. First, it does not guarantee to find the answer. And, there is no bound on the time for obtaining a solution. The probability of success grows as the time available to the algorithm increases.

2.2 Genetic algorithms

The Genetic Algorithm (GA), first introduced by John Holland (1975), 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 propagate. 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. As there is no single chromosome representation to fit all searching problems, there is also no unified chromosome representation to fit all different games. Each game should choose an appropriate encoding way dependent on the game characteristics to represent the solution states. Below, a chromosome representation is proposed to solve the two-player game-tree search problems in which all the possible next moves for nodes on the same levels are the same. Each possible move in the search tree is then 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.

If all the possible moves for nodes on the same levels are not the same, appropriate repairs to offspring states from the crossover and mutation operators are desired. For example, in the Go-Moku game, each board position can be given a number, and the possible branches descending from a node can be encoded by the positions on which a piece may be played next. Note that not all the positions on the board are considered

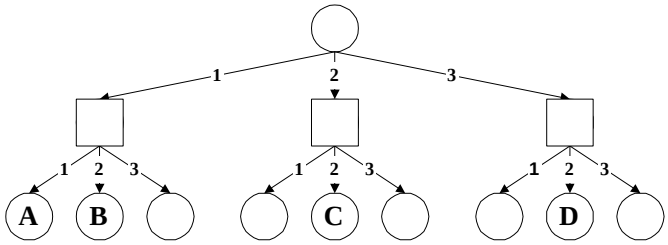

Figure 2: The coding scheme for a game-search tree.

for a next move, but only a limited range of positions determined from heuristics is considered.

The chromosome representation is similar to Holland's royal road function, in which a chromosome is regarded as a set of consecutive blocks, representing the parameters to be adapted. Each move in a possible move sequence then corresponds to a parameter in the royal road function. But the evaluation approaches are totally different.

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, as shown in Figure 3, 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.

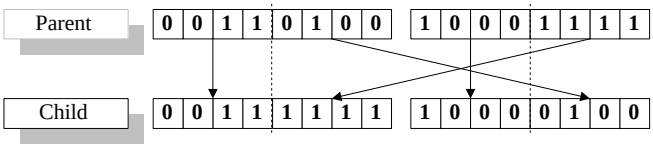

Figure 3: One-point crossover.

An example of applying the crossover operation to a two-player game is shown in Figure 4. Each offspring move sequence inherits some move patterns from its parents. 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 5 is used in the proposed algorithm.

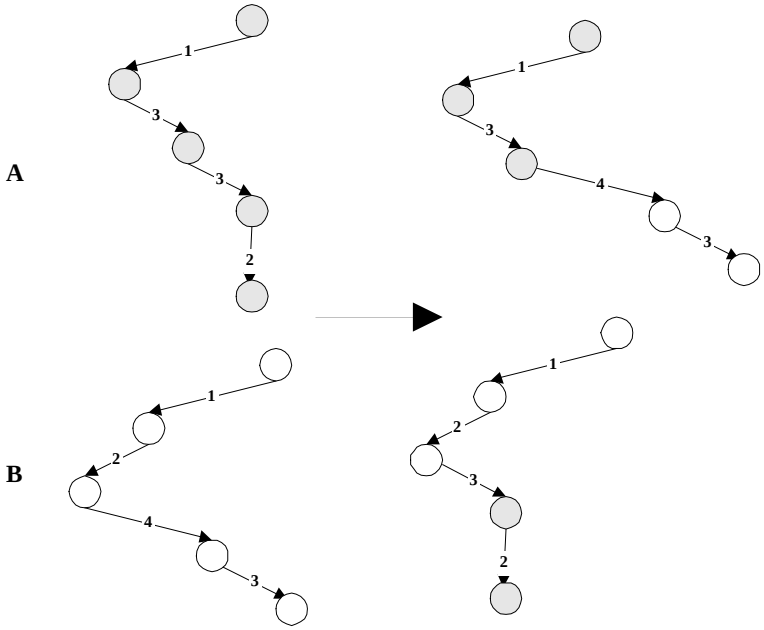

Figure 4: One-point crossover for two move sequences.

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 6. 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 6, 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**.

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 7. 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 **U** is on a MIN level and the original board evaluation value of **U** was 5, smaller than the value propagated from **V**, it is left unchanged.

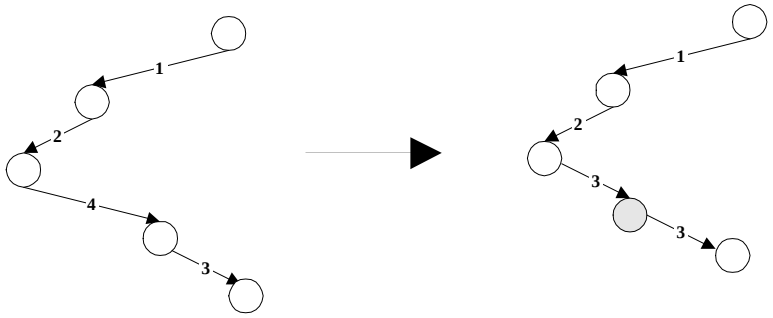

Figure 5: Mutation for a move sequence.

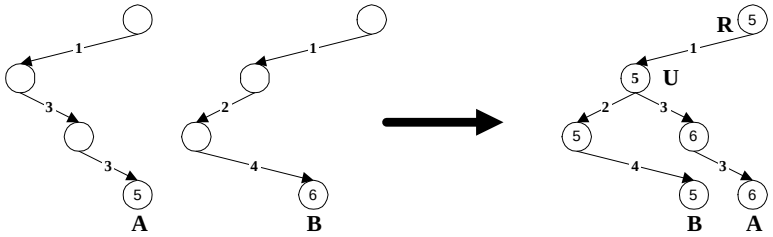

Figure 6: The reservation tree that results from combining two individuals, **A** and **B**.

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. Chromosomes with larger fitness values are thus chosen to produce the next generation. For example, in Figure 7, the highest levels from the bottom that the leaf values of **A**, **B**

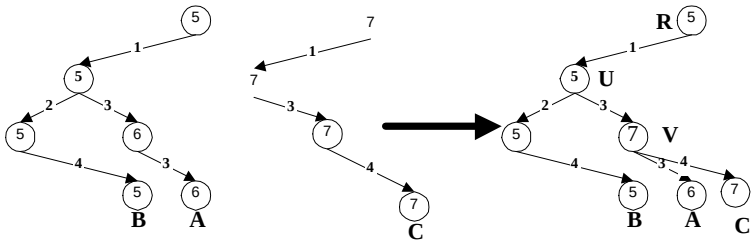

Figure 7: The reservation tree resulting from considering individual **C**.

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.

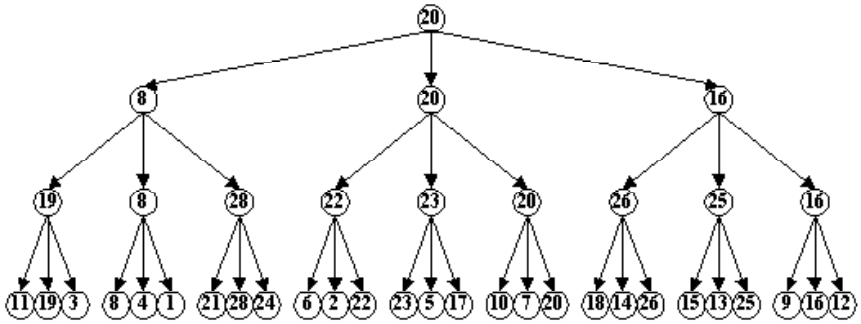

Figure 8: An example of a game-search tree with four levels.

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.

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. Below, an example is given to illustrate the proposed algorithm.

3.5 An example

Consider the game tree in Figure 8. Each node has three branches, representing three possible moves. The leaf values were obtained using a predefined board evaluation function for game play.

Suppose that the population size N is set at 3 and the initial population is $\{111, 221, 312\}$. A reservation tree containing these three chromosomes is shown in Figure 9.

Assume in Step 9, two pairs, 111, 221 and 221, 312, are chosen for crossover at position 2. The offspring are thus 121, 211 and 212, 321. Furthermore, assume in Step 10, chromosome 111 is chosen for mutation at position 3 to generate the new offspring, 113. The new reservation tree resulting from insertion of the five offspring, 121, 211, 212, 321, and 113, is shown in Figure 10; bold-faced circles denote nodes whose values have been updated.

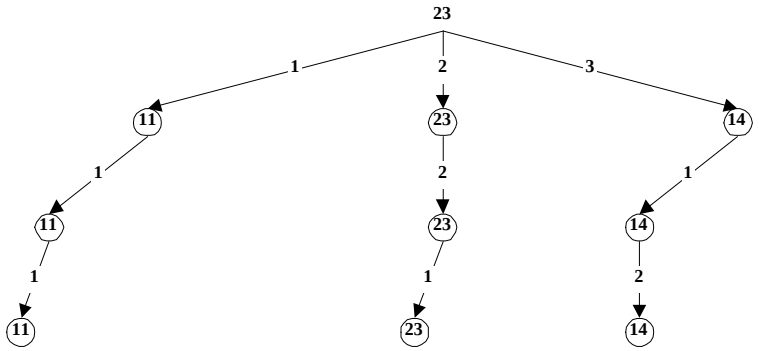

Figure 9: A reservation tree consisting of chromosomes 111, 221, and 312.

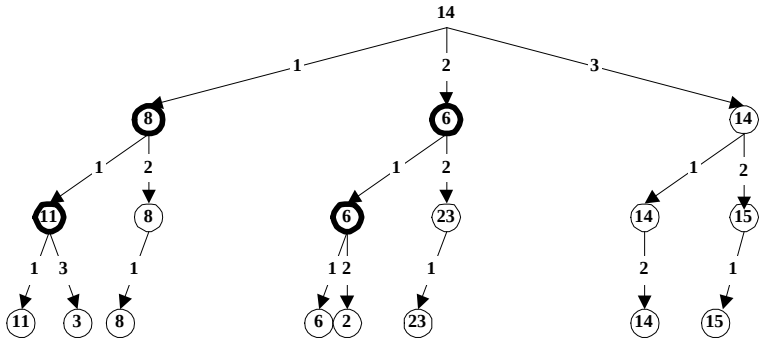

Figure 10: The reservation tree after Step 13.

We now have eight chromosomes, including the old population and the offspring, as shown in Table 1.

We proceed now to Step 14. The fitness values of the chromosomes in the reservation tree are shown in the last column of Table 1. Take the chromosome 121 as an example. Since 121 has a leaf value of 8, which is propagated to the second level of the reservation tree, its genetic value is $4 - 2 + 1(= 3)$. Others values can be calculated in similar fashion. Since 312, 121, and 211 are the best three chromosomes—they have the largest fitness values—they are therefore chosen as the new population in Step 15. If the termination criteria have not yet been satisfied, Steps 9 to 16 are executed again using the new population; otherwise, the best chromosome in this example, route 312, is output as the final solution, so the next move to play is the third branch.

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. Accuracy was considered first. The experiments were carried out as

Table 1: The chromosomes, their leaf values and fitness values.

No.	Source	Chromosome	Leaf value	Fitness
1	parent	111	11	2
2	parent	221	23	2
3	parent	312	14	4
4	crossover	121	8	3
5	crossover	211	6	3
6	crossover	212	2	1
7	crossover	321	15	2
8	mutation	113	3	1

follows. A computer game-search tree with a predefined number of levels was first generated. The minimax method, which is basically an exhaustive search, was used to find the best solution for a specific level in the search tree, and ranked it 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 7-level search tree with 7 branches was used in the experiments. The population size was set at 40. The crossover rate was set at 0.5 and the mutation rate was set at 0.1. The roulette wheel parent selection method was adopted for crossover and mutation. The superior individuals according to their fitness values were chosen to form the next generation. The experimental results for the 7-level game tree using 40 generations are shown in Table 2, and using 200 generations in Table 3; *number of levels* means the level of which the minimax and the genetic minimax methods searched.

Table 2: Accuracy of the search methods for the 7-level game tree after 40 generations.

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

Table 3: Accuracy of the search methods for the 7-level game tree after 200 generations.

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

From Tables 2 and 3, it is easily seen that the route obtained by our method is very close to the best route found by the original minimax method, and also, that increasing the number of generations, increased the accuracy of the resulting route.

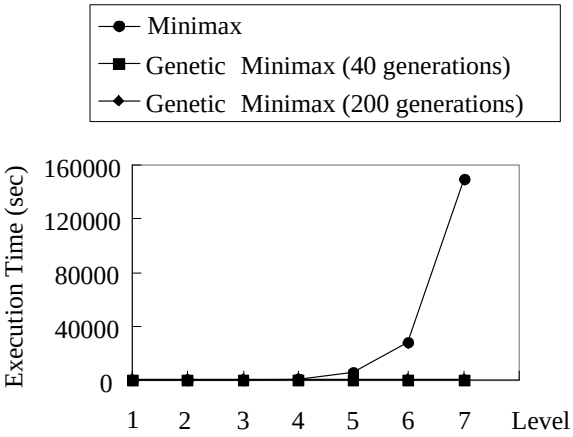

Figure 11: The execution times for the three methods.

We next compared the execution times required by the two search methods. The result is shown in Figure 11. 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.

Finally, we compared the accuracy of our algorithm with that of the minimax method using the same time limit. The experimental results for the 7-level search tree are shown in Figure 12. Clearly, our proposed method found a more accurate solution than the minimax method in the same time, since our algorithm used genetic heuristics to find good patterns rapidly, and was able to search deeper into the search tree in the same amount of time.

Note that in real game trees, there will always be some correlations between the leaf nodes based on how much of their paths they share. The crossover operator adopted can thus utilize this characteristic to generate good results.

5 Conclusion and Future Work

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. 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 (Korf, 1991; Mutchler, 1993).

As a conclusion, we like to say more about the comparison of our genetic minimax with the probabilistic style search. For the adversarial game search problem, a simple Las Vegas method could be performed as follows. A move sequence is generated according to the predefined searching depth, such that each move in the sequence is randomly chosen from the players' possible actions. The tip node is then evaluated to obtain a score and another path is generated. Using the reservation tree technique and the minimax principle, we can combine these two paths and perform roll-up evalua-

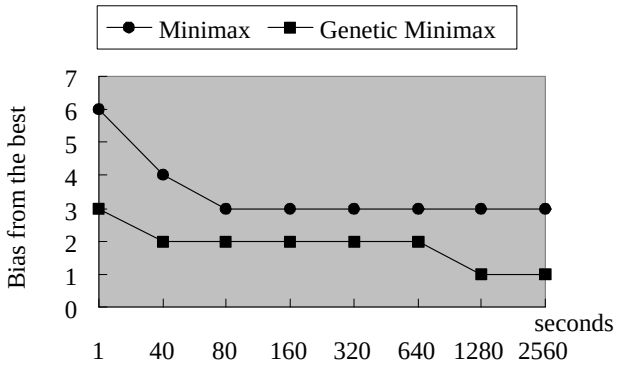

Figure 12: Accuracy of the proposed method and the minimax methods within the same time limit.

tion accordingly. This process continues until the preset time limit is reached. At last, the best move is chosen according to the final evaluation. At a first sight, this method is analogous to the initialization step of our genetic search method. The Las Vegas search generates the solution entirely by making random decisions, hoping to find a good next move by chance. No crossover or mutation operators are performed for local search. For a search tree of branching factor b and depth d, it can be derived that the probability to find the best move sequence by the Las Vegas search is p/b^d, where p is the number of paths generated. If the available time quantum is measured by the number of paths, say p, then the depth of the search tree to be explored should be less than $\log_b(2p)$ to yield a probability higher than $1/2$. The incorporation of probabilistic algorithms with state-of-the-art search techniques may be promising. This remains unexplored and needs further investigation.

In this paper, we assume that the two-player game-tree search problems have the same possible next moves for nodes on the same levels. If all the possible moves for nodes on the same levels are nearly the same, the above scheme may also be suitable but appropriate repairs for offspring states from the crossover and mutation operators are desired. When there are a variable number of possible moves at each node on the same level of a game tree, the maximum number of possible moves at each level can be used, buy many illegal offspring states may be derived. Although appropriate repair mechanisms can still be designed to resolve the conflicts, much additional computational time may be spent if many or most offspring chromosomes are illegal. For this situation, adopting different encoding schemes depending on the game characteristics is preferred. As an alternative, board states, instead of possible moves, may be encoded into chromosomes for GAs to be applied. We may also attempt to apply appropriate heuristics such as iterative deepening and SSS* according to the characteristics of games to the proposed algorithm for further improving the performance of the genetic adversarial search. This paper only provides a useful first step toward a thorough exploration of GAs to game tree searches. There is still much work to be done in this field.

Acknowledgment

The authors would like to thank the anonymous referees for their very constructive comments.

References

Abramson, B. (1989). Control strategies for two-player games. *ACM Computing Survey*, 21:137–161.

Abramson, B. (1990). On learning and testing evaluation functions. *Journal of Experimental and Theoretical Artificial Intelligence*, 2:241–251.

Ballard, B. W. (1983). The *-minimax search procedure for trees containing chance nodes. *Artificial Intelligence*, 21:327–350.

Barr, A. and Feigenbaum, E. A. (1989). *The Handbook of Artificial Intelligence Volume I*, Addision-Wesley.

Bolc, L. and Cytowski, J. (1992). *Search Methods for Artificial Intelligence*, Academic Press.

Brassard, G. and Bratley, P. (1996). *Fundamentals of Algorithmics*, Prentice Hall.

Campbell, M. S. and Marsland, T. A. (1983). A comparison of minimax tree search algorithms, *Artificial Intelligence*, 20:346–367.

Davis, L. (1991). *Handbook of Genetic Algorithms*, Van Nostrand Reinhold.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization Machine Learning*, Addison Wesley.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*, University of Michigan Press.

De Jong, K. A. (1980). Adaptive system design: a genetic approach. *IEEE Transactions on Systems, Man and Cybernetics*, 10:566–574.

Kaindl, H. (1988). Minimaxing—theory and practice. *AI Magazine*, 9:69–76.

Knuth, D. E. and Moore, R. W. (1975). An analysis of alpha-beta pruning. *Artificial Intelligence*, 6:293–326.

Korf, R. E. (1985). Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence*, 27:97–109.

Korf, R. E. (1991). Multi-player alpha-beta pruning. *Artificial Intelligence*, 48:99–111.

Kuncheva, L. (1993). Genetic algorithm for feature selection for parallel classifiers. *Information Processing Letters*, 46:163–168.

Lee, K. F. and Mahajan, S. (1990). The development of a world class othello program. *Artificial Intelligence*, 43:21–36.

Miller, G. F., Todd, P. M., and Hedge, S. U. (1989). Design neural networks using genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 379–384.

Mutchler, D. (1993). The multi-player version of minmax displays game-tree pathology. *Artificial Intelligence*, 64:323–336.

Nievergelt, J. (1977). Information content of chess positions. *SIGART Newsletter*, 62:13–14.

Pitrat, J. (1977). A chess combination program which uses plans. *Artificial Intelligence*, 8:275–321.

Rivest, R. L. (1988). Game tree searching by min/max approximation. *Artificial Intelligence*, 19:77–97.

Roizen, I. and Pearl, J. (1983). A minimax algorithm better than alpha-beta? yes and no. *Artificial Intelligence*, 21:199–220.

Rosenbloom, P. S. (1982). A world-championship-level othello program. *Artificial Intelligence*, 19:279–320.

Shannon, C. E. (1950). Programming a computer to play chess. *Philosophy Magazine*, 41:256–275.

Slate, D. J. and Atkin, L. R. (1977). *CHESS 4.5—The Northwestern University Chess Program*, Springer-Verlag, New York.

Thrift, P. (991). Fuzzy logic synthesis with genetic algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 509–513.

Wang, C. H., Hong, T. P., Tseng, S. S., and Liao, C. M. (1998). Automatically integrating multiple rule sets in a distributed knowledge environment. *IEEE Transactions on Systems, Man, and Cybernetics*, 28:471–476.

Winston, P. H. (1992). *Artificial Intelligence*, Third Edition, Addison Wesley, MA.