Applying Genetic Algorithm and Self-Learning on Computer Othello

Yung-Ling Lai\textsuperscript{1}, Shiao-Fan Chang, Ting-Chun Ko
National Chiayi University, Taiwan
yllai@mail.nctu.edu.tw, csf0427@hotmail.com, s0930317@mail.nctu.edu.tw

Abstract

Artificial intelligence algorithms have been applied on computer board games since 1953. Among all computer board games, because of its low branching property, Othello can easily defeat humans by designing with min-max search and alpha-beta pruning. Nowadays, the goal of compute Othello is no longer to challenge people but to compete against other computer programs. This paper improves the computer Othello’s opening strategy, mid-game strategy, and end-game strategy. The opening strategy is enhanced by improving self-learning efficiency using pattern recognition. The evaluation function is designed by combining the Line-pattern evaluation with other evaluation functions and by using an improved genetic algorithm to optimize the parameters. Then implement dynamic programming in min-max search and alpha-beta pruning to make the searching engine more efficient and to improve the depth of perfect-searching.

Keywords: Genetic Algorithm, Self-Learning, Computer Othello

1. Introduction

Computer board game using artificial intelligence was first proposed by Turing in 1953. In the past half century, the computer board games have played an important role in the research area of artificial intelligence. Table1 shows the developments of the chess programs.

<table>
<thead>
<tr>
<th>Board Games</th>
<th>Human vs Computer</th>
<th>summarize</th>
</tr>
</thead>
<tbody>
<tr>
<td>gobang</td>
<td>Solved</td>
<td></td>
</tr>
<tr>
<td>Othello</td>
<td>( &lt; )</td>
<td>Computer is better than human</td>
</tr>
<tr>
<td>Chess</td>
<td>( = )</td>
<td>Human was defeated by Deep Blue, 1998</td>
</tr>
<tr>
<td>Chinese chess</td>
<td>( &gt; )</td>
<td>Level 6, 7 for computer</td>
</tr>
<tr>
<td>Chinese checkers</td>
<td>( &lt; )</td>
<td>Computer is better than human</td>
</tr>
<tr>
<td>Shogi</td>
<td>( &lt; )</td>
<td>Computer is better than human</td>
</tr>
<tr>
<td>go in nine row</td>
<td>( &gt; &gt; &gt; )</td>
<td>Computer is far from human</td>
</tr>
<tr>
<td>go</td>
<td>( &gt; &gt; &gt; )</td>
<td>Computer is far from human</td>
</tr>
</tbody>
</table>

In the rule of the Othello, we can turn over the adversary’s pieces only after a legal move [8]. For each hand, there are around 1 (including pass) to 15 (seldom more than 15) legal moves which is average of eight moves. We see that the Othello’s possible moves for each hand is much less than other board games (compare to go game which has more than 200 possible moves for each hand or Chinese chess which has more than 50 possible moves for each hand). Even if the designing of the evaluation function are not precisely, with the help of high speed calculation ability of the recent computer, we may still get a fair result after several layers of min-max search and alpha-beta pruning when design the computer Othello program. This paper’s goal is to improve the Computer Othello by using the genetic algorithm and the line states evaluation to improve the evaluation function then use the self-learning strategy to enhance the data base of the opening of the game.

In an Othello game, the player moves 60 times (30 each)
in average. Assume that the legal moves in one hand is $n$, then the complexity of getting the optimal move of the first hand is $O(n^{60})$. As we described before, the average possible moves for each hand is 8, we need to search for $1.5 \times 10^{54}$ states to find the best move for the first hand. With this high complexity, the program cannot get the optimal solution by using searching strategy in a short time.

The board of the Othello is $8 \times 8$, and each lattice can be black, white or empty. There are $3^{64}$ (about $3.4 \times 10^{30}$) possible states, even if we consider the symmetric and erase part of the illegal states (which are not possible to appear), there are still a huge number of states which require a extremely high space complexity to store them. Therefore, before the winning strategy has been proved, the computer Othello still have room to be improved.

2. Development of the Othello

Because of the high complexity of the Othello, the common design for the computer Othello often uses min-max search tree and alpha-beta pruning, then use the evaluation function to get an optimal route of the search tree in a reasonable time. In order to improve the ability of computer Othello, the most direct way is to improve the accuracy of the evaluation function [1] and increase the depth of the search tree [6] or to make the opening state data base system [9]. To make a good opening state database, one has to collect the game history of famous plays between experts. There are many game history of competitions for the computer Othello programs and many world wide games in recent years, they are saved by wthor database format [3] and can be downloaded from internet [7]. In order to reach a higher level of computer Othello, we not only have to improve the searching efficiency, the accuracy of the evaluation function, but also have to enhance the self-learning ability.

Othello has several characteristics: because it changes the state a lot for each move, the state is hard for human but easy for computers to memorizes; there are no cycles for the states since each move will either be “pass” or add a piece to the state; and the final result is easy to tell. Because of those characteristics, it’s not hard to see that the computer Othello has higher capability than human being. There are four famous computer Othello programs, namely WZebra, Herakles, ntest-and-nboard, and edax. WZebra not only has high capability but also integrates the interface, analysis for state records, and the opening database. Herakles can consider the whole states adequately and have mobility in hand, it shows strongly its ability in the mid-game with a lose of spending too much time for the ending perfect search. Ntest-and-nboard is the one of the best computer Othello programs, whose design emphasizes on the capability of the program and does not much care about the interface so it is not easy to use. Edax has a precise and complete opening database and its capability is higher than WZebra in average.

3. System architectures and strategies analysis

Our system may be divided as three stages. In the first stage, we try to optimize the parameters using in genetic algorithm. By considering line state evaluation additionally the common state evaluations, we may improve genetic algorithm to modify the weight of the parameters in the evaluation function. The second stage proceeds the self-learning. When the depth of the search tree increase by one, the nodes in the tree will increase several times, the development of the self learning is obviously in the first week, but getting slow in the next week, and almost stuck after the third week. Even if we spend more time on self-learning, the searching depth might be increased but this does not have too much to do with the capability. After the self-learning is stable, we can use it with the evaluation function to estimate the state. We test the program and modify slightly in the third stage. When the state estimates are different between self-learning and the evaluation function, we use other computer Othello to play with our program and modify the parameters generated in the first stage and modify the weight.
of the self-learning in the evaluation function to get a more clinical result for the estimation.

To reduce the time complexity of the program is our goal when designing the computer Othello because it will make huge amount copies of the state information in the recursive steps and then it requires a lot of computations in the evaluation function. Since the legal movements only happen in the profile of the pieces on the state, we don’t have to scan all states in the search tree when searching the next optimal legal move. We define ROI as the “region of interested” so that each time we search only those states in ROI and we modify ROI after each movement to reduce the unnecessary searching.

In the stage of self-learning, we have to record a huge amount of states which requires a lot of memory. When using the information that has been learned by the program, we need an efficient comparison of the states. Hence we have to design the structure of the state in a way that it requires less memory but easy to compare. The structure of states in our system is shown as follows:

```cpp
class cAIData  //class of the self-learning
{
    public:
    class cAIData * Left, * Right, * Root;
    //point to the left, right subtree, and the father node
    unsigned char Map[2][4]; //array of the board
    char Mover;
    //which player can move
    unsigned long WhiteWin, BlackWin; //the winning times
    for each player of this state
};
```

There were many good methods for the states evaluation [10], here we use six methods to evaluate the states including line state evaluation, weighted node, mobility, stability, unbalanced wedge, and even balancing (even number of passing). This paper proposed a line state evaluation, which may enforce the evaluation methods that others proposed before, and optimize the parameters by genetic algorithm, avoid unbalanced wedge (or force the adversary to make unbalanced wedge), to make judgment of the evaluation function more precise. The line states indicate the states get from dividing the board into middle column, rows, and diagonal. Since each line state was made by no more than 8 lattices, there are no more than $3^8 = 6561$ line states. After expanding all line states recursively, we may use the estimation value for each line state to make a table so that the evaluation function may use it to make a more precise estimation in the competition. The key of winning the Othello often decides on the control of mobility. It often happens that after mid-game, the one who has weak mobility does not have legal movement and was forced to pass or to put his stone in a bad place. The mobility means the number of legal moves by the time of your turn. In order to control the mobility, we believe not to turn too many stones of adversary before mid-game, so there are many adversaries’ stones on the board which also means there are a lot of stones we can turn at this time. In this way, as long as we still have some stones on the board, we have a better mobility. We also consider to increase the stable stone (which are the stones that can not be turn over by adversary) and even balancing. The even balancing has two meanings, one is if we know one position is a good position, then the position adjacent to it is a bad position. Since the last hand can get the chance to turn over others stones and the adversary can not turn over again, it usually consider to be an advantage. In order to keep this advantage, the one hold white stone should make the number of “pass” to be even and the one who holds black stone should try to break this even. This is another meaning of even balancing.

These methods are related to each other. For example, the unbalanced wedge is a special case of the line state. In fact, this case has been calculated as a bad situation in the line state evaluation. In addition, the stable stones would be marked in the line state evaluation also. When we consider these evaluation methods individually, although it might not have high precision but there is some accuracy for each one. When combine them all together to do the evaluation, it's
possible that a move is marked as a “good move” in parameter A but is a bad one in parameter B. As a whole, line state is the grand evaluation parameter among these six parameters, others are just an accessory to help in the evaluate process. Figure 1 shows the relation between each dimension where the bold line indicate high relativity and the thin line indicate the low relativity while no line indicate they are either not related or has a very low relativity.

Figure 1: Relation between each parameter

After doing the optimization for each parameter, we set the weight for each parameter manually. Then give each parameter a weighted coefficient and optimize these coefficients by genetic algorithm again before manually reset the weight for each of them.

The genetic algorithm used in this paper references the backtracking in the neural network to improve the adapted function (see figure 2). Instead of supporting standard information in the beginning, our system provides three sets of initial value (1, -1, and 0) assign to each parameter. In the process of evolution, we set the goal for generation, e.g. 300 generation for the first turn, 400 generation for the second, and 500 generation for the third. When it reaches the goal of generation, the best generation is added to the adapted function to make a new adapted function. After many times of evolution, it has a big chance that the parameter assignment in the system can have an excellent ability. The advantage of this strategy is that it can precede the evolution without any standard information and the parameters got by evolution have a good ability. The time needed for evolution is the disadvantage of this strategy. Fortunately the evolution will be done in the system designing phase which will not affect to the competition.

Figure 2: The improved genetic algorithm
The searching in this paper first construct a min-max search tree then use alpha-beta pruning to cut off the unnecessary branches. Then use dynamic programming to record the states that have been evaluated and return the values of estimation. When the searching reach the same state, we then use the known value of estimation to avoid re-expand the node in the search tree. This will allow us to reach one depth more in the search tree using the same searching time.

Although some of the computer board game has added self-learning in the program design, there is no deterministic self-learning algorithm proposed. The space complexity of the chess games has limited the effect. The self-learning in this paper is used to enhance the opening strategy. We record the states of self-competitions and using the statistic results to assist the evaluation functions to make good judgment of the whole game. In order to increase the breadth of the self-learning, we use the breadth-first strategy in self-competition. We use pattern recognition to speed up the effect of self-learning. After analyze a large mount of the game records, we divide the game board into the high-affected area and low-affected area. The game board after subtract the low-affected area is recognized as a pattern which is learned during self-competition. The learning result from the self-learning can be the sample value for the evaluation functions. We use the pattern recognition as a weight, a little lower than the state evaluation, in the evaluation functions at the bottom search level, which can improve the learning effect.

As a whole, the opening strategy is enhanced by improving self-learning efficiency using pattern recognition. The mid-game strategy uses Line-pattern evaluation with other evaluation functions to enhance the precision of state judgment. The end-game strategy (about last 20 moves), we use improved search algorithm to make perfect search. Using these three strategies, the computer Othello can easily beat the human being players.

4. Results and discussions

Each evaluation function has different setting on parameters, and there is one more set of parameters after combining the evaluation functions. After the optimization process, the weight of parameters in the evaluation functions is as follows:

- Line state evaluation
- Weighted node
- Stability
- Mobility
- Unbalanced wedge
- Even balancing

The result shows that since the line state evaluation has the highest accuracy evaluation, in the optimization it will get the highest weight among all other parameters. Other parameter also showed their importance by the weight. Notice that the resulting weight of the mobility seems defeated the saying that “The one who holds the mobility holds the crux of the winning”. Since the mobility is very hard to estimate, one may find that the mobility can change a lot after only one move, there is no computer Othello program can evaluate the mobility precisely up to now [5]. The way to estimate the mobility in our system is not very precise either, hence the result after optimization for mobility is not as expected.

Our system was competed with Deep Green when testing the effect of the self-learning. The search depth is set to six, and the level of the evaluation function is set to highest and search time is allowed no more than 2 seconds. When compete with other named Othello programs, we use the same hardware in competitions, and have 80 runs (half of first hand and half of last hand), each run time was limited to five minutes. The results is shown in table 2

<table>
<thead>
<tr>
<th>Purposed program</th>
<th>Deep Green</th>
<th>3D Reversi</th>
<th>WZebra</th>
<th>Herakles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Holds black (first hand)</td>
<td>15W 5L</td>
<td>14W 6L</td>
<td>5W 15L</td>
<td>4W 16L</td>
</tr>
<tr>
<td>Holds white (last hand)</td>
<td>14W 6L</td>
<td>14W 6L</td>
<td>4W 16L</td>
<td>3W 17L</td>
</tr>
<tr>
<td>Total</td>
<td>29W 11L</td>
<td>28W 12L</td>
<td>9W 31L</td>
<td>7W 33L</td>
</tr>
</tbody>
</table>
Table 2 shows that our system is a little better than most of other Othello programs on the market, but not good enough to compete with WZebra or Herakles. This shows that the searching ability in our system is not complete and the time for each move is not properly admeasured.

5. Conclusions

Self-learning has the properties of the machine learning although it is not belonged to the genetic algorithm or the neural network. Although self-learning has been applied in many computer board games, there is no critical algorithm for it. This paper used the pattern recognition to speed up the efficiency of the self-learning, which was used to enhance the opening strategy, and purposed the line state evaluation, optimize with the other evaluation methods to improved the genetic algorithm for the high-precisely evaluation function, which improved the mid-game strategy. Finally we use dynamic programming to speed up min-max search with alpha-beta pruning to improve the efficiency of perfect search in the end-game.

After practical test, it has proved that the way of self-learning proposed in this paper is possible to enhance the efficiency of learning greatly. The optimization of the genetic algorithm can coordinate several evaluation functions in order to evaluate the state precisely and improve the computer Othello.

References