Search Strategy Research and Analysis for Surakarta Chess Game

Chang Liu, Jiaxin Wang, Yunpeng Zhang

School of Computer & Communication Engineering, University of Science & Technology Beijing, Beijing 100083
E-mail: ustb_txliuchang@163.com

Abstract: As one of the most important part of Computer Game, the search algorithm is the key to every chess game. This paper formulates a new search strategy based on the basic algorithm. It briefly introduces the theory and technique of hybrid search strategy and the dynamic control of the depth of the search strategy. Efforts are also made to establish a kind of search depth which is not limited to integers. Experiments are conducted to have tested and evaluated the above strategies. They have been successfully used in Surakarta game program, which have helped us win the champion in the 2013 National Undergraduate Computer Game Competition.

Key Words: Surakarta Chess Game, Hybrid Search, Search Depth, Dynamically Controlled

1 INTRODUCTION

Computer Game, as an important branch of artificial intelligence, is a challenging and vibrant research field .The well known success of the Deep blue chess indicated that the dream of a computer overcoming the topmost human being chess master has become into reality [1].Further study of the research will undoubtedly bring new challenges to the game field. The program for Computer Game is basically in accordance with chess programming methods proposed by Shannon in 1950.The traditional game program is based on Shannon's model, usually consisting of the following four parts: chessboard show, move generator, situation evaluation ,and search strategy. Search strategy, as the most essential part of the game, is also the center of the program. Usually there are many kinds of search algorithms and Minimax search algorithm and Alpha-Beta pruning search algorithm are most common. On the one hand, Minimax search algorithm, is a basic search algorithm in traditional Computer Game[2].In game, we set a static evaluation function f (x).The larger f(x) gets, the more advantages one part called MAX will have. Through the backtracking method, the whole game tree is traversed and each node is visited once .In the end, the root node selects the maximum path. On the other hand, the Alpha-Beta algorithm is one of the effective methods for compressing the searching space[3].The algorithm is able to cut out the game tree in order to make the search more sufficient. With the simple search algorithm described above, however, some branches which have little effect on the situation are also searched, which wastes search time and reduces search efficiency. Therefore, this paper proposes two improved search strategies : hybrid search and the dynamic control of search depth .These two improvements are used in Surakarta chess game, which make the program have tremendous improvements. We prove that the two above strategies have certain advantages by tests.

2 IMPROVEMENTS IN SEARCH STRATEGY

2.1 Hybrid Search

In this part, the study does not focus on proposing a new search algorithm, or improving certain existing algorithms, but putting forward some new ideas about the search process of a game tree, and at the same time making the search more effective.

Hybrid search is a search strategy that combines multiple ideas of different algorithms. Key to the algorithm are differentiated search and the way of classification for move generator. Classification for move generator means program takes separated search for different move generators which have a big difference in branch factors so that the program can conduct a comprehensive search.

Differentiated search means that for the Si in the classified set of move generator {S1, S2... Sn} ,the program takes a particular search function. The program also has a move generator function matching with Si which distinguishes different move generators and then certain move generator will be searched emphatically. This is similar to Best-fist minimax extension in some degree[4][5]. What we propose is to combine Best-first search algorithm with Alpha-Beta search algorithm .Our program makes search depth different according to classification of move generator, not just mix two search algorithms.

2.2 The Dynamic Control of Search Depth

Nowadays, in order to make a search more balanced in a game, the program uses an iterative deep search[6] to limit search time. We hope to directly control search depth rather than limit time.

That is a supplementary to the fixed search depth algorithm like Alpha-Beta and Minimax. We may consider from the following aspects to achieve the dynamic control of search depth.

(1) The global control of search depth: Establishing different search depth according to different situations in a game is called the global control of search depth. The usual search limits the maximum search depth. Alpha-Beta search depth line, for example, is such a maximum limit in Alpha-Beta algorithm. The global control of search depth does not limit the maximum depth, and only needs a basic search depth (or calls minimum search depth).Similar to best-fist minimax expansion search algorithm, we set up a basic search depth d_0 to ensure that the search depth is not too low.d_0 represents the search depth in the most complicated starting situation.

(2)Dynamic search depth: For any kind of chess, search costs more time in the starting situation than in the later in terms of a large number of feasible move and huge branch factor in the starting situation. Therefore, we combine dynamic search depth with global control of search depth to make search time more balanced.

For a game whose starting branch factor is b_0 and basic search depth is d_0, the search quantity in the starting situation is about b_0^d_0.

For achieving our purpose to balance the search quantity, supposing later branching factor is b, the depth d of the search should satisfy the equation :

$$b^d=b_0^{d0}$$

Then the dynamic depth d can be obtained according to known information:

$$d=\log_b(b_0^{d0})$$

Since the above equation is inconvenient to be obtained directly in a general programming language , we can get the following form:

$$d=\ln(b_0^{d0})/\ln(b)$$

The above formula is used to determine search depth at certain time in the end.

(3)Make search depth real:an additional and operationally independent independent mechanism is presented apart from the two controls of search depth. Making search depth real is used with maximum search limitation. The depth of each search consuming is no longer to be set 1, but a real number, and the number will be represented by search depth consumption function. Our aim is to make branches of many nodes searched little, and branches of few nodes searched more. That only needs the function is monotonically decreasing.There is a search depth consumption function reference:

$$d_{cost}=\ln(Cb)$$

d_{cost} is search depth consumption, C is rate constant determined manually, and b is branching factor.

3 SURAKARTA CHESS SYSTEM WITH IMPROVED STRATEGIES

3.1 Features of Surakarta Chess

Surakarta chess is a kind of chessman-eating chess game, largely similar to International chess and Chinese chess ,but there are many different details. First, the usual chessman-eating games like Chinese chess, have the same normal rules as eating rules . For example, in the Chinese chess, horses usually move diagonally and elephants move squarewise .However, in Surakarta chess, normal move rules can only permit each chessman to move one cell to one direction, as long as there is no chessman in that direction. There are 8 optional directions, that are up-down, left, right, up-left, down-left, up-right, and down-right. But eating move rules can move a lot of cells along the grid lines while eating the chessman, which makes the normal move rules different from eating move rules when they are generated. So Eating move generator and normal move generator are naturally divided into two categories.

3.2 The Design of Surakarta Chess System with Improved Search Strategies

3.2.1 Chessboard Show in Surakarta Chess Game

We use a two-dimensional array of size of 6*6 where 1 to 12 represent the black chessman, 13 to 24 represent the white chessman, and 0 represents no chessman. The initial array is set as:

1	2	3	4	5	6
7	8	9	10	11	12
0	0	0	0	0	0
0	0	0	0	0	0
13	14	15	16	17	18
19	20	21	22	23	24

In every move generator, we use two positions S(s_i, s_j) and E(e_i, e_j) to represent.s_i and s_j are the abbreviation of $start_i$ and $start_j$, showing that the start point position is at the i line, j row. A correct move generator must place the chessman in the empty place and satisfies the following inequality :

$$(e_i-s_i)^2+(e_j-s_j)^2 \leqq 2$$

3.2.2 The Move Generator of Surakarta Chess

Enumeration method is used and all the validate moves are generated. Owing to Surakarta chess's own characteristics (spin eating), eating rules are generated separately. This also helps to distinguish search algorithms when searching. The normal move generator generates a algorithm in which the chessman can move to eight directions .In other words, we give the start point position and the target point position and then judge its legitimacy.

The eat move generator generates a algorithm in which the chessman can be eaten at any positions except for four corners. There are two directions to eat chessmen including horizon and vertical directions. Especially, the chessmen in line1 and line 6 cannot eat others in the horizon direction and the chessmen in row1 and row 6 cannot eat others in the vertical direction.

The following are steps of generating eat move generator:

Step 1: Choose the unselected chessman S sequentially. If there is no unselected chessman, the generator ends.

Step 2: Choose search direction "dir", following the sequence of"up, down, left, right". If all are searched, we return to Step1.

Step 3: Scan along the direction "dir" until we meet the chessman E.

If chessman E belongs to the opponent, the eat move generator(S,E) is generated and recorded. Otherwise, we repeated Step 2.

So the eat move generator is distinguished from the normal move generation when generated.

3.2.3 The Situation Evaluation of Surakarta Chess

General Surakarta chess will use comprehensive assessment methodology by means of basic position stones. The current valuation is:

$$val=v_p+v_s$$

v_p is the position score.v_s is the stones score. And v_p can be obtained through a table (PVT). For one side concerned:

$$v_p = \sum_{i=1}^{12} v_{pi}$$

If certain chessman is eaten, its position value is 0.On the contrary, if it is still on the chessboard, it can be evaluated according to PVT. Our PVT is a 6*6 array:

```
{{0.8,1 ,1 ,1 ,1 ,0.8}.
 {1,1 ,1.1,1.1,1 . 1},
 {1,1.1,1 ,1 ,1.1, 1},
 {1,1.1,1 ,1 ,1.1, 1},
 {1,1 ,1.1,1.1,1 . 1},
 {0.8,1 ,1 ,1 ,1 ,0.8}};
```

Here is a brief introduction of reasons for setting such a PVT. There are three values:0.8,1.0 and 1.1. When a chessman is in the corner, you can not eat any of opponent's chessmen and will not be out of the corner when the second-line (eg: the smaller arc line) has opponent's chessmen. That is the reason why the value 0.8 is in 4 corners. The position of value 1.0 is a common position. The position of value 1.1 is the key position (eg: larger arc line) linking the second-line with the third-line called points of battle.

v_s is related with the number of chessmen in Surakarta chess, but our PVT is designed very cleverly making each value close to 1, so that our program does not need to evaluate the number of chessmen. Therefore, we no longer distinguish between v_p and $v_{s.}$ We only need a comprehensive valuation. Based on the above, the valuation of Surakarta chess is the valuation of black minus that of white:

$$val = \sum_{i=1}^{12} v_{bi} - v_{wi}$$

3.2.4 Search Strategy in Surakarta chess

The basic search Algorithm is Alpha-Beta, which is extremely popular around all the countries in the world.

When considering generating move generator, our program divides move generator into two parts including "eating"state and"normal"state. We permit the search depth of eating state deeper. Different from VCF in connect 6,the eating move generator in Surakarta does not force the opponent to eat chessmen, but tends to ask the opponent to consider eating chessmen preferentially. We limit search depth in real numbers. The number of layers each search costs is also a real number. In our program ,search depth can reach a 20-layer level which is achieved by controlling depth.

In the search process, the cost depth d which makes the normal move generator search for one layer is ln(2*b_1) rather than 1.b1 represents the number of normal move generator in the current nodes. The cost for the eat move generator is ln(2*b_2).b_2 represents the number of eat move generator in the current nodes.

Giving the global fixed depth, the logic of programming search is as follows:

Step1: search 1: If the search depth is greater than depth / 5, go to the move generator 1. For normal move, it goes to search 1, and for eat move, it goes to search 2 . If the search depth is lower than depth / 5, it only goes to search 1. If the search depth is used up, it returns the valuation.

Step 2: move generator 1: It generates normal move and eating move.

Step 3 :search 2:It is applied for the situation that there is remaining depth. If the search depth is used up, it returns the valuation.

Step 4:move generator 2:It only generates eating move.

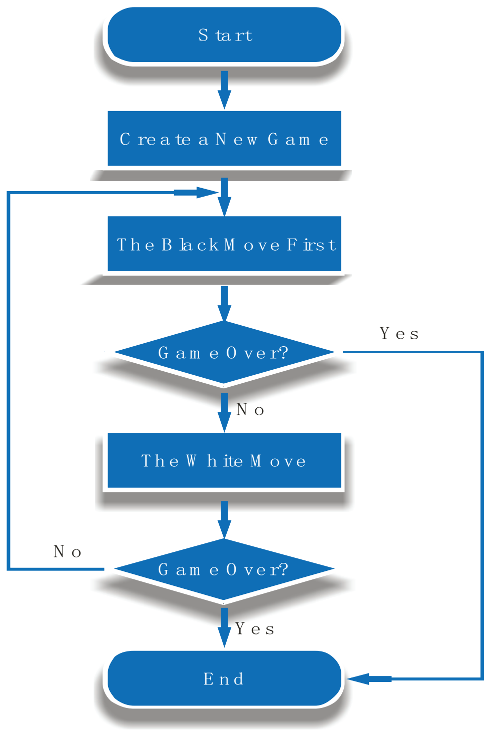

Fig 1. The Program Flowchart of Surakarta Chess

4 THE PROGRAMMING TEST AND EVALUATION OF SURAKARTA CHESS

We set 3 difficulty levels. Level 2 is the best combination of strength and speed of search options, and is also the general difficulty level used to fight against players. If the program is in level 3, a step sometimes may be more than two minutes, which is not convenient for us to test. In the end, we have adopted level 2 with players.

Our tests select players with different ages and background knowledges of game , not improved programs and outstanding programs from around the world. Such test results can expose the comparison with human intelligence, and our researching level compared with other people around the world.

4.1 Tests with Human Players

In 2012, our Surakarta program was exhibited in 60 years' school anniversary due to high achievements .We also played with alumni who have much confidence in games for nearly 30 bureau, but no one player can beat our program in the level2.

Fig2. Game Situations among Players of Surakarta Chess

	Computer Wm	Computer Lose	The Rate of Wm
School Anniversary Test	30	0	100
Invitation Test	200	0	100
The Sum	230	0	100

In May 2013, we invited 85 students to attend our tests, and let students play some rounds with our programs not counting results to make them familiar with Surakarta chess. Making sure that they have been familiar with Surakarta chess, we played many rounds with them, and got 100 sets of data(200 game results).The results are that no one can beat our program. The table below shows all game situations with the players.

4.2 Tests among Other Programs

4.2.1 Tests between Improved Programs and Not Improved Programs

We played 50 rounds with the basic programs utilizing pure alpha-beta algorithm in the situation that all have nearly the same search time. The result is that we won all the games.

Fig3. Game Situations between Improved Programs and Not Improved Programs of Surakarta Chess

	Win
The Improved Program	50
The Not Improved Program	0
The Winning Rate of Improved Program	100%

4.2.2 Performances of Improved Programs in Competition

In 2013 National Undergraduate Computer Game Contest, We win the champion.

5 CONCLUSION AND OUTLOOK

These achieved initial results show that: the proposed hybrid search strategy and improvements in search depth control do have good results. The improved program enables us to have won the champion in the 2013 National Undergraduate Computer Game Competition.

According to the currently known situation, hybrid search combined with controlling search depth used in other board games will also receive a similar effect. We may mainly focus on the Multi-algorithm combination in the future. The core idea of research is to "make search more effective." Adhering to this principle, we can continue to raise other search strategies to improve the level of the computer game.

REFERENCES

[1] Zhang Ming-liang, Wu Jun,Li Fan-zhang Design of evaluation-function for computer gobang game system. Journal of Computer Applications.2012,32(7) : 1969 -1972.

[2] Shannon C E. XXII. Programming a computer for playing chess[J]. Philosophical magazine, 1950, 41(314): 256-275.

[3] Xiaochun Wang. PC Game Programming (Machine Adversarial)[M]. Chongqing University Press, 2002.5, 102-114.

[4] Korf R E, Chickering D M. Best-first minimax search[J]. Artificial intelligence, 1996, 84(1): 299-337.

[5] Korf R E, Chickering D M. Best-first minimax search: Othello results[C]//PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE. JOHN WILEY & SONS LTD, 1995: 1365-1365.

[6] Korf R E. Depth-first iterative-deepening: An optimal admissible tree search[J]. Artificial intelligence, 1985, 27(1): 97-109.