International Core Journal of Engineering
ISSN: 2414-1895

Volume 10 Issue 5, 2024
DOI: 10.6919/ICJE.202405_10(5).0048

Automatic Move Program for Two-player Military Chess based on UCB&MCTS Algorithm and Move Evaluation

Jinhua Bian[1, a], Junli Li[2, b], and Gang Wang[3, c]

[1] School of Computer Science and Software Engineering, University of Science and Technology LiaoNing, Anshan 114051, China

[2] School of Materials and Metallurgy, University of Science and Technology LiaoNing, Anshan 114051, China

[3] School of Computer Science and Software Engineering, University of Science and Technology LiaoNing, Anshan 114051, China

[a]17816066868@163.com, [b]939309665@qq.com, [c]purgwg@163.com

Abstract

The two-player military chess is a typical zero-sum game with non-complete information, which has more pieces, more moves, and will not reduce the number of situation branches as the game progresses. Therefore, the basic Monte Carlo Tree Search (MCTS) cannot satisfy the high searching demand of military chess. For this reason, this paper proposed the UCB&MCTS algorithm based on move evaluation and applies it to the automatic move program of military chess, and the experiments prove the feasibility and effectiveness of the improved UCB&MCTS algorithm.

Keywords

Two-player Military Chess; Monte Carlo Tree Search; Move Evaluation; UCB Formula.

1. Introduction

Computer gaming, also known as machine gaming, is one of the most challenging and important topics of research in the field of artificial intelligence.In recent years, with the rise of the artificial intelligence industry,researchers have set off a boom in the research of computer gaming neighborhoods, and have achieved fruitful academic results:In 2016, AlphaZero developed by Google Inc. defeated the Go legend Lee Sedol with a score of 4:1. In 2017, after nine months of self-iterative training, AlphaZero's chess power was once again substantially improved, and beat the world's best Go player Jie Ke with a score of 3:0. Classical computer gaming algorithms include Monte Carlo tree search algorithm, Great Minimization search, Q-Learning reinforcement learning algorithm, etc. These are effective algorithms for searching for information-complete gaming games (such as Go and International Chess). However, two-player military chess is a non-complete information game, which cannot see the specific information of the opponent's pieces when the players are playing the game, which also adds lots of situations to the search tree of the game, and the ordinary Monte-Carlo tree search or minimax algorithm may not be able to adapt to the huge search situation of military chess.

Monte Carlo Tree Search Algorithm (MCTS) is a classical computer game search algorithm, this algorithm proposes four major steps of Selection, Expansion, Simulation and Back Propagation to simulate and score the situation according to certain rules and select the step with the highest score as the next action step. The highest rated step is chosen as the next action step.In recent years, many researchers have also integrated various algorithms with MCTS algorithms,for example,Marc Lanctot combined Monte Carlo simulation with Counterfactual Regret Minimization(CFR) algorithms and proposed Outcome-Sampling MCCFR and External-Sampling MCCFR algorithms[1],Richard Gibson proposed Aggregate-Sampling MCCFR algorithms[2].which focuses on accessing the set of information accessed by the Nash equilibrium strategy,thus improving the efficiency of optimal solution retrieval.

For all chess games, the estimation of the current value of pieces and position is also an important factor that affects the computer's decision-making,and there are many examples of value estimation of the two-player military flag in previous studies,Wei Zhang[3] performed position evaluation by evaluating the value of the pieces as well as the probability of the appearance of the different pieces of the other side on the board,and designed the empirical function,the attack function, and the defense function to perform and select the behavior of the next move,Kun Meng[4]and others designed a position valuation function-pessimistic algorithm to conservatively evaluate the calculation of the score of a particular move,and also designed attack and defend functions to perform the selection of the next operation,which makes the growth of computer chess power.In this paper, we design a move evaluation function,and combine the upper confidence interval formula UCB with MCTS,propose to improve the UCB&MCTS algorithm and implement the automatic move program for the military chess game, we compare the implemented program with the military chess move program implemented by some classical algorithms,and the experiments proved the validity and reasonableness of the improved UCB&MCTS algorithm.

2. Two-player Military Chess Introduction and Definition of Basic Information on Military Chess Programs

2.1 Two-player Military Chess Introduction

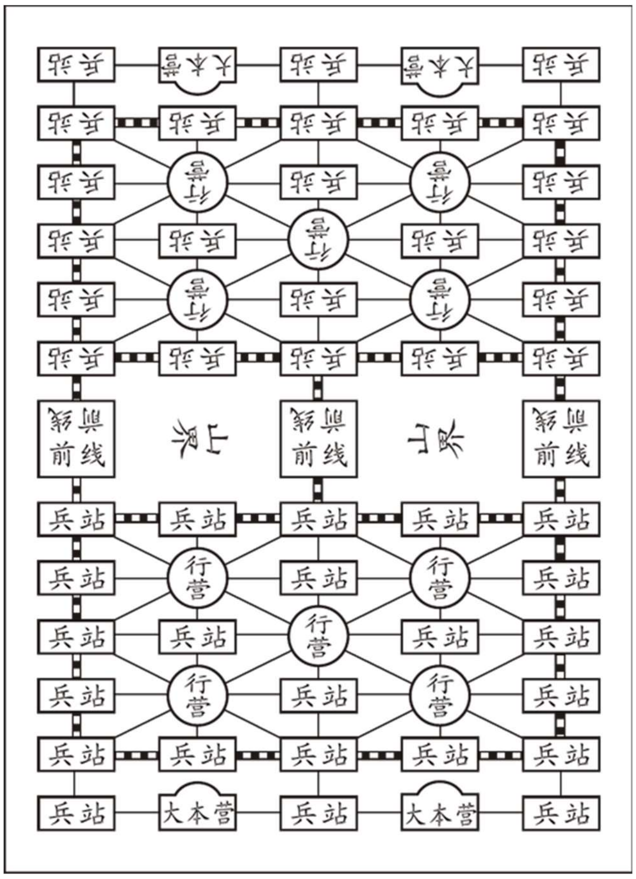

Figure 1. Diagram of military chess board

Two-player military chess consists of a board with 12 rows and 5 columns and 25 pieces for each side,where each side's 25 pieces consist of one Chief Commander, one Army Commander, one Army

Flag, two Division Commanders, two Brigade Commanders, two Regiment Commanders, two Battalion Commanders, one Bomb, and three Company Commanders, three Platoon Commanders, one Engineer, and three Land Mines. If team A successfully attack the opponent's flag, then team A wins. The two-player military chess board is shown in Figure 1.

According to the official website of China University Computer Gaming Competition, the main rules and game flow [5] are as follows: before the game starts, both sides will put their 25 pieces in their pawn stations and base camps according to their own layouts, when the game is in progress, one side can choose a piece and walk along the solid line or the dotted line of the squares on the way, and if there is a piece of the other side in the focus of the walk, then both sides will make a comparison of their pieces, if there is a bomb in both sides, then both sides will die together.then the player with the higher ranked piece wins and eats the other player's piece. Neither side can attack the other side's pieces in the camp(called “行营” in figure 1), both sides' military flags are located in their own base camp(called “大本营” in figure 1), when one side's piece reaches the position where the other side's flag is located, that side wins.

Due to the complexity of the rules of military chess, the paper will not spend a great deal of space describing all the rules of military chess in detail. What can be noticed is that when team A makes a move decision, it does not have full information of the exact distribution of the opponent's pieces, so for every attack move, team A has a certain risk: team A's attacking piece will die if it’s rank is lower than the opponent's attacked piece. This makes the possibilities of different positions in the game so large that there are $7.1*10^{17}$ possible complete-information positions at the beginning of the game, when both sides are laying out their pieces. At this point the size of the search tree of a complete position search algorithm such as the classical Monte Carlo tree search algorithm is unable to return all legal moves. To address this problem, we found that the value of searching deeper levels is much smaller than searching the current position in detail in the 3-4 layers of the game tree, so we limit the upper limit of the game tree extension to 4 layers, and use the UCB formula to search the position more accurately, and use the move evaluation function to score the position so as to simulate the best moves.

2.2 Definition of Basic Information about the Military Chess Program

(1) chess piece definition:

```
struct Chess{
int id;             //Piece id,shown what kind of piece it is
int position;       //Piece position on the board
int level;          //Chess level
double pro[12];   //Storing the probability distribution of enemy pieces
}mychess[12],enemychess[12];
```

(2) chess board definition:

First, each position of the board is numbered, and the position of point {i,j} is numbered as (i-1)*12+j.

int dist[100][100];//dist[i][j]=1 if the position numbered i,j has an edge, otherwise it will be set infinity.

3. Algorithm Design

3.1 Handling of Incomplete Information Situations

For the case that the player cannot see the distribution of his opponent's pieces in two-player military chess game, we first create a probability distribution table of the pieces to record the probability information of each opponent's piece in the current position.For each position on the opponent's board,we build a probability analysis table,after which we can dynamically update the probability table as the position changes and the corresponding information gained from piece attacks.For example,the probability table for all the pieces in a base camp position of the enemy is as follows.

Table 1. Probability distribution table of chess pieces in the initial situation of enemy base camp

Flag	Bomb	Platoon	Landmine	Battalion	Company	Others
0.5	0.15	0.15	0.1	0.05	0.05	0

Due to the special location of the base camp,the pieces that are in the base camp cannot move outward, so the probability that one of them is a high-ranking piece can be regarded as 0.Since the rules stipulate that the flag must be in one of the two base camps,we can initially stipulate that the probability of the flag being in one of the base camps is 0.5,and for the lower-ranking pieces,we can set probabilities based on whether they can move or not, and on their relative values setting.For all other positions,the probability value of each piece can be set by referring to the rules of two-player army chess and the probability analysis method mentioned above.It should be noted that in any position, the probability of all pieces must sum to 1.

In the process of military chess game, it will produce the situation that both sides' pieces attack each other, for the result produced after attacking, we can analyze and obtain the relevant information of surviving pieces, for example, if our rank 4 piece attacks the opponent's piece A, and as a result, our piece dies, we can set all the probabilities that piece A is rank 4 or below to 0. After that, we perform the normalization operation, and set the probability of the sum of the probabilities of the pieces of piece A to 1 from the new one, assuming that the updated probability distribution of piece A is $P=\{p_1,p_2,p_3,...,p_{12}\}$, so that:

$$sum = \sum_{j=1}^{12} p_j \tag{1}$$

$$p_j = \frac{p_j}{sum}, j = 1,2,3,...,12 \tag{2}$$

After the above operations, the update of the piece probabilities is complete.

3.2 UCB & MCTS Algorithms

In the MCTS algorithm,the selection of nodes and the estimation of the situation are the two decisive factors in determining the accuracy of the algorithm,we should consider the probability of winning-times for each possible search node and the number of times the node has been visited when deciding which node will be chosen to search next,we need to consider both the nodes with a high probability of winning-times,and also the nodes with a small number of searches, to achieve a balance between exploration and utilization.The UCB formula[7] successfully balance these two points well,and its formula is expressed as follows.

$$UCB = \frac{w_c}{n_c} + cof \times \sqrt{\frac{2 \times \log n_p}{n_c}} \tag{3}$$

In formula (3), w_c represents the number of winning-times the current node search, n_c represents the total number of times the current node is visited, n_p represents the number of times the current node's parent node is visited, and *cof* is a manually-set coefficient, the former item of the UCB formula utilizes the currently available information and calculates the current node's winning-times rate, which makes the algorithm biased towards selecting nodes with a higher winning-times rate; and the latter item of the UCB formula takes into account the number of visits of each node's number of visits, so that the algorithm is biased towards selecting nodes with fewer visits, and it can be found that when the number of visits to a node is 0, the latter term of the UCB is infinite, which indicates that the UCB formula prioritizes the unvisited nodes and ensures that every node will be visited.

In the selection phase of the MCTS algorithm, we need to select a node from the current situation downward that needs to be expanded most, if there is a node that has not been visited in the current situation, then this node will be selected as the expansion node, if the current situation lead to an end of the game(such as successfully arrived at the location where the enemy flag is located),we can immediately perform the back propagation operation.Otherwise,all feasible operations of the current node have been expanded, at this time we use the UCB formula to select the best expansion node.

After selecting the best node, the algorithm enters the expansion and simulation session, expanding from the selected nodes that have not yet been expanded, and randomly simulating actions to guide the end of the current game situation, at this time, perform the backward propagation operation, updating the results of this simulation to all the nodes on the path from the root node to the current node:The number of visits to all nodes will add one, and if the situation will lead to a victory of the side, then all nodes' winning times will add one.

The four steps of Monte Carlo tree search are depicted in the picture in Figure 2.

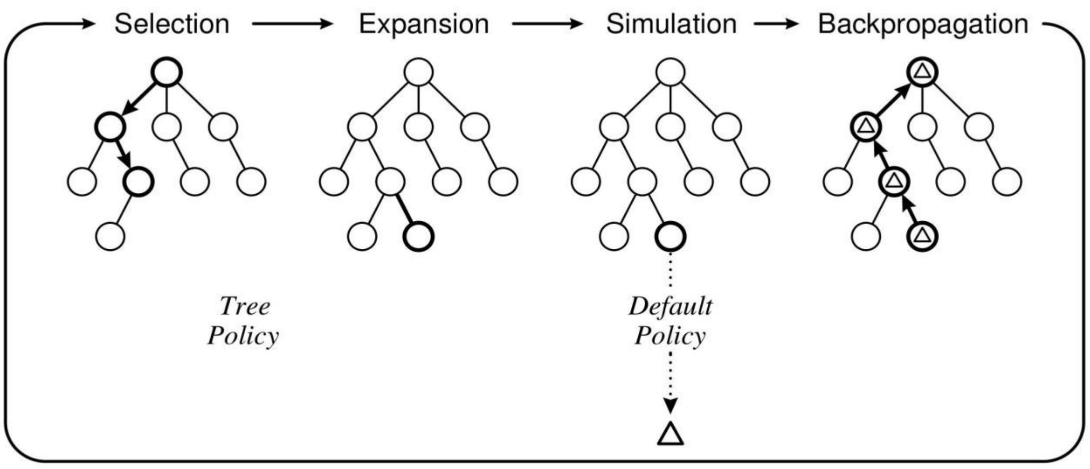

Figure 2. Illustration of Monte Carlo tree search

In order to solve the shortcoming that the MCTS algorithm cannot completely traverse the whole game tree of military chess, we propose a piece value estimation function to simulate the top ranked piece moves from the perspectives of both the value of the piece itself and the danger it can cause, so as to search for the optimal move for the current position in a highly efficient way.

3.3 Piece Move Value Estimation Function

In order to be able to simulate as many good moves as possible in limited time, we need to estimate the value of each move, for the estimation of the value of each move, we evaluate the value of each move from two perspectives, the value of the move itself and the threat degree of the move, in the study, we consider from two perspectives, the rank of the move and the impact of losing the move on the position, and we design and specify each kind of value of a piece type, as shown in the following table.

Table 2. Comparison table of pieces name and piece value

Name	Flag	Chief	Army	Division	Brigade	Regiment	battalion	company	platoon	sapper	landmine	Bomb
Value	5000	100	87	65	40	31	19	8	3	44	39	77

For the estimation of the value of a move, we should not only see the gain of the current attack move, but also need to consider the threat that the opponent may pose to us after our move, if the gain of the offense is greater than the loss which enemy's caused to us, then we choose an attack strategy, otherwise we choose a defend strategy.

3.3.1 Pieces Attack Strategy Design

When a piece walks, it may attack the opponent's piece, at this time, we need to consider whether the piece can defeat the opponent's piece, and the possible threat to our own piece after defeating the opponent's piece, assuming that the value of our piece is *v*, and the probability distribution of the opponent's attacked piece is $P=\{p_1,p_2,p_3,\ldots,p_{12}\}$, then when the opponent's piece is *i*, the expected gain we will get is:

$$Exvalue_i=\begin{cases} p_i \times v_i & if\ lv_{self} > lv_{enemy} \\ -p_i \times v & otherwise \end{cases} \tag{4}$$

Where p_i represents the probability that the enemy piece is *i*, and v_i represents the value of the enemy piece when it is *i*. We can check in Table 2 that when our piece level lv_{self} is larger than the opponent's piece level lv_{enemy}, then we can get the benefit of eating the opponent's piece, otherwise, we will lose the benefit of our own piece. When our piece successfully defeats the opponent's piece, we also need to consider the threat of the enemy's piece to our current piece when our piece is in a new position. Suppose a total of *n* enemy pieces are threatening to our current piece, and the probability distribution of the *i-th* piece is $P=\{p_1,p_2,p_3,\ldots,p_{12}\}$, then the loss value of the threat of this piece to our piece is set as:

$$Exloss_i=\begin{cases} p_i \times v & if\ lv_{self} < lv_{enemy} \\ 0 & otherwise \end{cases} \tag{5}$$

At this point we have a complete consideration of the gains and losses incurred by the attack, and we can calculate the total score Attack_Score for the offense:

$$Attack\_Score=\sum_{i=1}^{12} Exvalue_i-\sum_{j=1}^{n} Exloss_i \tag{6}$$

The pseudo-code for the attack function scoring valuation algorithm is shown in Algorithm 1

Algorithm 1: Chess_Attack

```
Input: mychess,enemychess
Output: Attack_Score
1:  my_level ← mychess.level
2:  i ← 0
3:  Attack_Score ← 0
4:  while i<12 do
```

```
5:        if enemy_level>my_level then
6:          Attack_Score ← Attack_Score + enemychess.pro[i]*enemychess.value
7:          my_pos ← enemychess.position
8:          for j = 1 to 60
9:              if dist[j][my_pos] == 1 and has_enemy_chess then
10:                 for k=0 to 11
11:                   if enemy_level>my_level then
12:                   Attack_Score←Attack_Score - enemychess.pro[k]*mychess.value
13:                 end if
14:               end for
15:             end if
16:         end for
17:       else
18:           Attack_Score ← Attack_Score - enemychess.pro[i]*mychess.value
19:       end if
20:      i ← i + 1
21:  end while
22:  return Attack_Score
```

3.3.2 Piece Defense Strategy Design

Before choosing the attack operation, we should first consider the threat of enemy pieces to our pieces, we can calculate the loss estimation of ordinary pieces refer to formula (5) . Since the survival of the flag determines the result of the game, the design of the defense function should be especially designed for the defense of the flag function, and secondly, our pieces close to the enemy's flag should also be the key object of protection, the defense function designed in this paper includes the protection of three types of pieces: Firstly, the protection of our flag, and then the protection of high-value pieces (e.g., those with high value of their own, and those close to the enemy's base camp or the flag position), and then the protection of other pieces.

For the flag defend function: First, we start from our flag position and use breadth-first traversal (BFS) algorithm to search for the piece closest to our flag. Since the flag cannot move, when an enemy piece is close enough to our flag to pose a threat, we can only defeat the enemy piece by using our nearby pieces or by deploying landmines or bombs near the flag in the initial position. When enemy piece A is about to threaten our flag, we immediately search and move our nearby piece which has the highest probabilities to defeat A, and fix the best move as the current move, thus protecting the flag. The pseudo-code of the flag protection function is shown in Algorithm 2.

Protecting high-value pieces: Assuming that the current high-value piece being protected is A. First, from the position of A, use BFS to search for the closest enemy piece B to A. Since piece A can move, we first consider moving A as far as possible toward the enemy base camp while avoiding enemy piece B. If this is not possible, then we choose to move A back to avoid enemy piece B.

Algorithm 2: Flag_Defend

Input: chess id,chess position
Output:defend_flag; defend_chess

```
1:  flag_pos ← mychess[0].position
2:  pair<int,int> enemy←bfs(flag_pos)
3:  int step←enemy.first,id←enemy.second,enemy_pos←enemychess[id].position
```

```
4:  Boolean defend_flag←false;
5:  **if** step<=4 **then**
6:      **while** defend_chess can't beat enemy **do**
7:          Pair<int,int> defend_chess←bfs(enemy_pos)
8:      defend_flag←true
9:  **return** {defend_flag,defend_chess}
```

4. Experiment

The UCB&MCTS military chess program based on move evaluation proposed in this paper is implemented using CodeBlocks software, C++ language. Through the experimental results, we determined the value of the *cof* parameter in the UCB formula to be 10; and the improved UCB&MCTS program and the two-player military chess program implemented by the ordinary MCTS program played 30 games for first move and second move respectively, and the results of the experiments are shown in the following table.

Table 3. The result table of experiment(No time limit)

	Improved UCB&MCTS	MCTS
first move	25 wins	5 wins
second move	22 wins	8 wins

In order to simulate the situation that there is a time limit during the game, we limit the time for each side to 10 minutes, and for each move, the simulation program will be forced to exit after exceeding 30 seconds. After setting the time limit, we again used the improved UCB&MCTS program and the two-player military chess program realized by the ordinary MCTS program to play 30 games against each other in first move and second move, and the experimental results are shown in the following table.

Table 4. The result table of experiment(10 minutes limit)

	Improved UCB&MCTS	MCTS
first move	29 wins	1 wins
second move	23 wins	7 wins

It can be shown that the improved UCB&MCTS algorithm beats the MCTS algorithm more often, and the program power is effectively improved. When we limit the playing time, the winning rate of the improved UCB&MCTS program becomes higher, which also shows that the improved UCB&MCTS program can find better solutions than the normal MCTS program in a short time.

5. Conclusion

In this paper, a military chess move evaluation function is proposed by introducing the UCB formula for the weakness of the large time complexity of the simulation process of the MCTS algorithm. It enables the simulation process of MCTS algorithm to simulate more optimal nodes in a short time, thus realizing the efficiency improvement. However, the move evaluation function of piece value in this paper is relatively fixed in terms of moves and responses, and the expansion of the game tree is relatively small, which still has more space for improvement compared with those programs combined with deep learning. In the next research, machine learning algorithms such as Q-learning can be introduced to dynamically adjust the parameters of the move evaluation function, so as to improve the program's chess power.

Acknowledgments

This work it was supported by National College Students' innovation and entrepreneurship training program project (2024).

References

[1] Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. MonteCarlo sampling for regret minimization in extensive games. In Advances inNeural Information Processing Systems 22 (NipS), pages 1078-1086, 2009.

[2] Gibson R , Burch N , Lanctot M ,et al. Effcient Monte Carlo CounterfactualRegret Minimization in Games with Many player Actions|C]// NeuralInformation Processing Systems (NlpS). Curran Associates Inc. 2012.

[3] Zhang Wei, Wang Jun, Yan Tong et al. A military chess gaming system based on situation situation[J]. Intelligent Computer and Application,2018,8(01):87-90+94.

[4] Meng Kun,Wang Jun,Yan Tong. An empirical knowledge-based military chess game algorithm design and implementation[J]. Intelligent Computer and Application,2017,7(02):66-69.

[5] Information on http://computergames.caai.cn/.

[6] Jinwen Wang,Yuan Li,Shaobo Qu.Computer Game Case Tutorial(Electronic Industry Press,China 202) p.178-180.(In Chinese).

[7] Lu Mengxuan.Research on the key technology of Einstein chess computer game[D].Anhui University, 2019.