A Heuristic Monte Carlo Tree Search Method for Surakarta Chess

Guoyu Zuo[1,2], Chenming Wu[1]
1. College of Electronic and Control Engineering, Beijing University of Technology, Beijing 100124, China
2. Beijing Key Laboratory of Computational Intelligence and Intelligent System, Beijing 1000124, China
E-mail: zuoguoyu@bjut.edu.cn, wcm@emails.bjut.edu.cn

Abstract: This paper proposes a heuristic Monte Carlo Tree Search (MCTS) method for a computer game, Surakarta chess. The proposed method uses heuristic knowledge to guide the search procedure heading to a better solution. In our implementation we use a CPU based root parallelization technology which permits many instances running simultaneously. The reliability of our method has been proved to be considerably effective by experimental results.

Key Words: Surakarta, Monte Carlo Tree, Parallelization

1 INTRODUCTION

Computer game is an important and prosperous branch of artificial intelligence. In computer game, search strategy plays an important role on making decisions. Over the last three decades, a lot of search methods have been well studied, which greatly promotes the development of computer game. The most famous historical event is "Deep Blue", a chess-playing computer developed by IBM, defeated a Russian champion in 1997. This event remarked the new techniques will undoubtedly change the future of computer game [1].

Surakarta game is a fascinating game which firstly played by Indonesians. It is a strategic board game [2]. The board of Surakarta chess is composed by 6x6 lines and 8 arcs as shown in Figure 1.

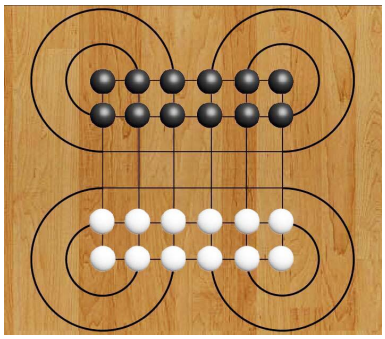

Figure 1: The board of Surakarta chess

The detailed rules of the game are very easy to understand: two players move chess piece one by one, only one of twelve chess pieces (commonly in white or black) can be moved in one round. The piece can only be moved a square vertically or diagonally to a new position where there is no other color piece, or to capture the opponent's piece when your piece goes through at least an arc line to get arrived at the opposing piece's position. To determine the final winner of the game, the referee (commonly is the computer control program) will count the number of chess belonging to each player. The side whose piece number is equal to zero will lose the game, and the side has less number of pieces will lose the game when exceeding the fixed time.

In this paper, a heuristic Monte Carlo Tree Search (MCTS) method for Surakarta game has been proposed. The paper is organized as follows: Section 2 briefly introduces the background of MCTS. Section 3 shows how we describe Surakarta chess and corresponding rules in programes. Section 4 explains three strategies including heuristic energy function, heuristic termination and parallelization. Section 5 shows experimental results and conclusion.

This work is supported by the Scientific Research Common Program of Beijing Municipal Commission of Education (KM201310005005).

2 MONTE CARLO TREE SEARCH

MCTS is a popular and effective approach for making optimal decisions in artificial intelligence problems particularly in computer games [3, 4]. MCTS is a best-first tree search algorithm and it has also been proved successful in some games for which the evaluation function is intractable to estimate. Compared with the Max-Min or Alpha-Beta based search methods, MCTS is much more easy to programme. Even if it stopped early, MCTS still produces better results. It will adopt randomized exploration in the search space and use these explored results to build a tree. This method consists of four phases, and each of them will be well illustrated as follow:

Selection: When we are exploring a new move, a balanced relationship between exploitation versus exploration should be concerned. In other words, it is a trade-off between exploiting the best move so far and exploring the uncertain moves. Many theories and methods to solve this problem such as Upper Confidence Bound Tree (UCT) search [5], All-Moves-As-First (AMAF) Heuristics search [6], and UCT enhanced with Progressive Bias (PB) [7] have been well studied recently.

The UCT algorithm is an application of the Upper Confidence Bound algorithm, which comes from the Bandit

Algorithm Monte Carlo Tree Search

Require: $Leaf$
Require: $Root$
Require: $CurrentState$
  $BestResult \leftarrow \emptyset$
  while $time - remaining \neq 0$ **do**
    $Result \leftarrow \emptyset$
    while $CurrentState \notin Leaf$ **do**
      $CurrentState \leftarrow selection(CurrentState)$
    end while
    while $Simulationnotend$ **do**
      $CurrentState \leftarrow simulation(CurrentState)$
    end while
    while $CurrentState \neq Root$ **do**
      $propagation(CurrentState)$
      $CurrentState \leftarrow CurrentState's\ Parent$
    end while
    if $Result > BestResult$ **then**
      $BestResult \leftarrow Result$
    end if
  end while
  return $BestResult$

problem, a very simple problem based on one-armed bandit. In MCTS, the node with maximum UCT value will be selected.

$$UCT_j = X_j + C\sqrt{\frac{\ln n}{n_j}}$$

where X_j is the score value of node j, n_j is the visited number of j, n is the count of visited number of its parent nodes. C is a real number which needs to be tuned by experiments.

Expansion: After selecting the most promising node, a new node will be added to a randomly chosen child position in the tree. This phase is called expansion.

Play-out: In this phase, a randomly Monte Carlo simulation begins until reached the termination of the game. Uniformly random moving ensures the convergence of the final search result. If we use heuristics knowledge to rule the randomness, a faster speed of convergence can be gained and will benefit the accuracy of decision.

Back-propagation: After the above three phases, the result of simulated play-outs should be propagated to all nodes traversed from the leaves nodes to the root in the search tree. All of the nodes visited have their visited counts incremented. If the player won in the Play-out phase, the win counts should also be updated by adding one.

The search procedure iterates until exceed the time-threshold set before.

3 GAME DESCRIPTION

In computer programming, the chess game is represented by data and codes. The data record the board's details and other useful information guide the search engine toward a better solution. The codes ensure the whole system running correctly and efficiently. Normally a Surakarta chess game program consists of three parts: board depiction, moves generator, search engine. We use a 6-by-6 matrix (1) where positive ones represent black chess pieces and negative ones represent white chess pieces.

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix} \tag{1}$$

The capture judging process is non-trivial because the time-complexity of moves generating and the time-complexity of capture judging are linear correlation. We implemented a recently proposed storage structure [8] to determine capture judging process which provides a better performance in our experiments.

4 SEARCH FRAMEWORK

In general, MCTS, Alpha-Beta pruning search and their variants have been adopted widely since the search results are nice looking and acceptable. In Surakarta chess, the high time-complexity of moves generator weaken the effectiveness of MCTS. The traditional MCTS cannot be adopted directly since the single step time for one player is from 10 to 30 seconds. We use three strategies to improve the simulating quality: 1) We use an energy function formulates the current state to help the selection stage. 2) We compulsively terminate the simulation stage once the state has great disparity or falls into a repetitive loop. 3) We use OpenMP [9] to parallelize our heuristic MCTS method in CPU to increase the number of Game-Per-Seconds [10].

4.1 Strategy 1

We define an energy function $F(s)$ consist of several terms such as position, attack, defense enhances the reliability of simulations.

4.1.1 Position

Estimating the value of position is non-trivial because it is uncertain how each position represents on the board. F_P is the value of position for the current board. We use a map to restrict the distribution of weights:

$$\begin{bmatrix} 0 & 5 & 5 & 5 & 5 & 0 \\ 5 & 10 & 20 & 20 & 10 & 5 \\ 5 & 20 & 10 & 10 & 20 & 5 \\ 5 & 20 & 10 & 10 & 20 & 5 \\ 5 & 10 & 20 & 20 & 10 & 5 \\ 0 & 5 & 5 & 5 & 5 & 0 \end{bmatrix} \tag{2}$$

For an arbitrary distributions on the board we check whether it leads to a high F_P by element-wise multiplying matrix (2) with the board's matrix (1). The value of position is equal to the sum of matrix elements.

4.1.2 Attack

To measure the aggression term, we design a formulation that finds the move which maximizes the potential quality

of attacking. We calculate the number of player's chess pieces which bring out captures as F_A. The calculation observes a principle: "the more captures, the more aggressive".

4.1.3 Defense

We identify the pieces having high defending properties to formulate the defense function. We assume that if the opponent captured your piece and you can capture this opposite piece in return, this your piece is defensible. All of these defensible pieces make the defense term F_D larger.

In our case, we package the three terms as a linear combination as follow:

$$F(s) = \sum (a_k \times F_k) \quad \cdots \quad (k = P, A, D)$$

where a_k is the weight of each component. All results in our experiments were produced with the following weights:

$$a_P = 0.000025 \qquad a_A = 0.03 \qquad a_D = 0.03$$

To combine the MCTS with our heuristic evaluation, we rewrite the selection function in MCTS as follow:

$$UCT_j = \arg\max \left(X_j + C\sqrt{\frac{\ln n}{n_j}} + F(s) \right)$$

4.2 Strategy 2

The number of chess pieces plays a vital role in Surakarta chess game, which is useful to reduce the time-complexity at the stage of simulation. If the number of each side tends to be lopsided, the simulation will be terminated in advance. We simply compare the difference between two side's pieces and if the difference is greater than a threshold $L = 3$, the side who has more pieces will be regarded as the winner.

In our observations, the average steps of a game is about 50. If the steps far exceeds the average, the simulation may fall into a repetitive loop. It occurs frequently in Surakarta chess game. To address this issue, we record the step of each simulation in our method, if the step number is greater than 50, a drawing result will be returned.

4.3 Strategy 3

With the rapid development of parallelism technology, most CPUs are able to execute more than one thread at the same time. The most common parallelization methods for MCTS are root parallelization, leaf parallelization and tree parallelization, respectively [10]. The easiest way to implement is leaf parallelization (Figure 2) but it usually returns bad results since it only simulates one node as many as possible each time which is not suitable in practice. The root parallelization (Figure 3)is more effective than traditional leaf parallelization because this method expands the search space to avoid local optimal solutions. Based on these considerations, we select root parallelization to accelerate our simulation.

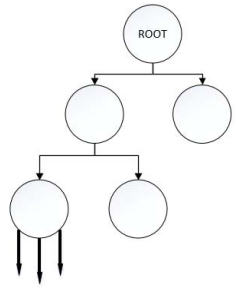

Figure 2: Leaf parallelization

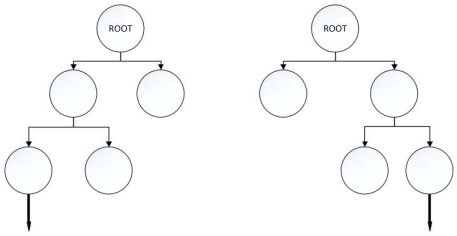

Figure 3: Root parallelization

5 EXPERIMENTS

In order to demonstrate the effectiveness and efficiency of our method, we conducted the experiments comparing a original MCTS with our heuristic MCTS method at a Windows PC with Intel Xeon E5 3.6 GHz CPU and 16GB RAM. Table 1 shows the results of Surakarta chess using the our method against original MCTS (50 games for each time constraints). Figure 4 shows the performance comparison between our method and orginal MCTS. Note that the original MCTS ran only in one thread whereas our method ran in 64 threads. It is clear that our method gives a significant promotion than traditional MCTS. Additionally, our method could gain more improvements by means of increasing the number of CPU's cores or using GPU acceleration.

Table 1: Results of experiments

Time Constraints	Win rates
10s	100.0%
15s	100.0%
20s	84.0%
30s	74.0%

6 CONCLUSION

In this paper, we discuss the MCTS and its variants which are frequently used in Computer Game, and proposed a heuristic search method based on MCTS for Surakarta chess. We also investigate what inspires us to build the heuristic strategies and how to formulate three strategies together. The limitation of our work comes from the intensive computation of heuristic evaluation, which will be our future work.

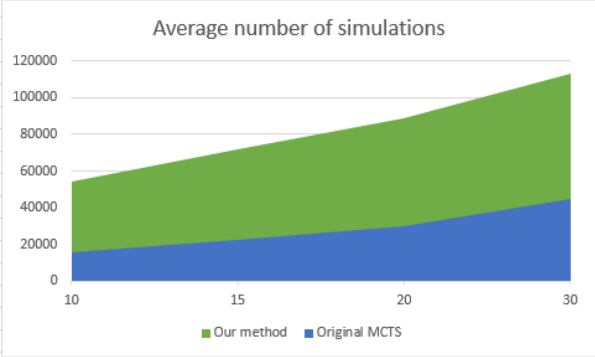

Figure 4: The average number of simulations

REFERENCES

[1] Campbell M, Hoane A J, Hsu F. Deep blue. Artificial intelligence, 2002, 134(1): 57-83.

[2] Pritchard D B. The Family Book of Games. Brockhampton Press, 1994.

[3] Coulom R. Efficient selectivity and backup operators in Monte-Carlo tree search. Computers and games. Springer Berlin Heidelberg, 2007: 72-83.

[4] Brgmann B. Monte carlo go. Syracuse, NY: Technical report, Physics Department, Syracuse University, 1993.

[5] Kocsis L, Szepesvri C. Bandit based monte-carlo planning, Machine Learning: ECML 2006. Springer Berlin Heidelberg, 2006: 282-293.

[6] Helmbold D P, Parker-Wood A. All-Moves-As-First Heuristics in Monte-Carlo Go. IC-AI. 2009: 605-610.

[7] Chaslot G M J B, Winands M H M, HERIK H J V A N D E N, et al. Progressive strategies for Monte-Carlo tree search. New Mathematics and Natural Computation, 2008, 4(03): 343-357.

[8] Zhang Y. A new storage structure design of Surakarta chess based on location. Control and Decision Conference (2014 CCDC), The 26th Chinese. IEEE, 2014: 3961-3964.

[9] Dagum L, Enon R. OpenMP: an industry standard API for shared-memory programming. Computational Science Engineering, IEEE, 1998, 5(1): 46-55.

[10] Chaslot G. Monte-carlo tree search. Maastricht: Universiteit Maastricht, 2010.