

Einstein's Chess Based on Monte Carlo Tree Searching

Gengyun He¹, Junli Li², Gang Wang³

¹School of Computer Science and Software Engineering, University of Science and Technology Liaoning, Anshan114051, China, hgynumber@126.com

²School of Materials and Metallurgy, University of Science and Technology Liaoning, Anshan114051, China, 939309665@qq.com

³School of Computer Science and Software Engineering, University of Science and Technology Liaoning, Anshan114051, China, purgw@163.com

Abstract

Monte Carlo Tree Search (MCTS) is an intelligent decision tree search algorithm for solving problems such as board games. It uses random sampling and statistical principles to find optimal solutions in incomplete information and complex environments without traversing all possible paths. The core steps include selection, expansion, simulation and back propagation. In the selection phase, the algorithm selects potential nodes for expansion through a strategy that balances known and unknown information using an "upper confidence bound (UCB)" algorithm. The expansion phase adds new child nodes to the selected nodes, representing possible scenarios under the current decision. The simulation phase starts with the newly expanded node through random or policy-based simulation until the termination condition is satisfied. The back-propagation phase updates the node statistics along the path, such as the number of victories and visits, based on the simulation results to guide the next round of the selection process.

By performing these steps iteratively, MCTS gradually finds relatively good solutions to perform the search in an efficient manner within a given time. This makes MCTS a powerful tool for solving complex decision problems, especially in contexts where the search space is inexhaustible.

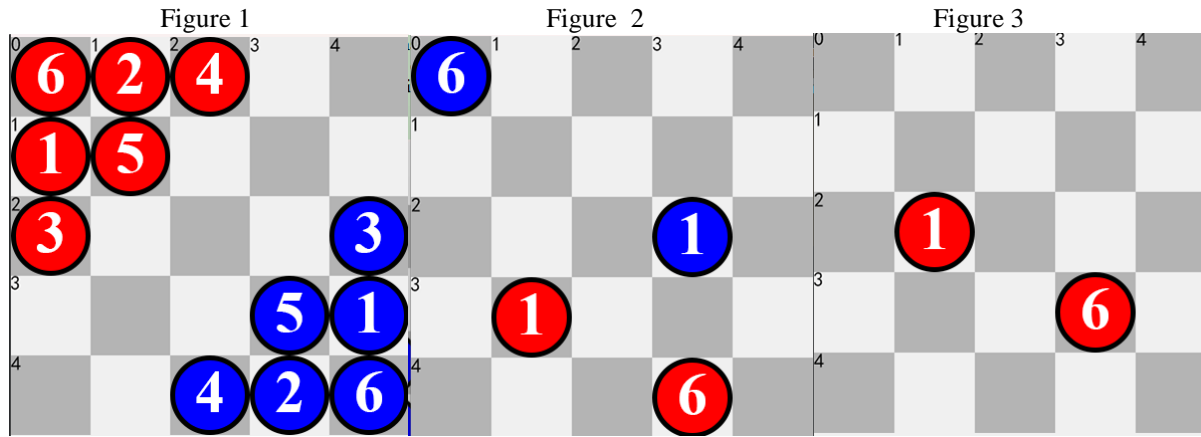
Keywords: Monte Carlo Tree Search, Einstein Chess, Dynamic Adjustment Strategies for Upper Confidence Interval Parameters

1. Introduction

Einstein Chess is a two-player non-complete information game of 5×5 squares, with each side having six chesses numbered 1 to 6. The goal of the game consists of moving to the opponent's diagonal point or annihilating the opponent's pieces without producing a draw. (Figure 1 shows the initial board; Figure 2 shows that Blue reaches Red's corner square first, and Blue wins; Figure 3 shows that Red destroys all Blue flags, and Red wins.)

MCTS (Monte Carlo Tree Search) is commonly used to solve such problems with complete information but uncertainty. However, MCTS faces challenges including high computational complexity, slow convergence, sensitivity to adversarial attacks, trapping in local optimal solutions, difficulty in dealing with continuous action space, and dependence on state evaluation. To optimise MCTS, improvement directions such as increasing computational efficiency, accelerating convergence speed, enhancing resistance to adversarial attacks, adapting to continuous action space, and improving state assessment accuracy can be explored to enhance its performance and adaptability in solving complex problems. In previous academic research, there are many studies for the optimisation of MCTS algorithms, Liu Guoqing [1] et al. proposed the optimal action recognition algorithm DLU based on relative entropy for Monte Carlo Tree Search (MCTS) abstract model under fixed budget, which improves the tree node valuation accuracy through the pure exploratory method of relative entropy confidence interval, and is validated in the artificial tree model and the Tic-Tac-Logic Validation in scenarios shows higher accuracy and performance advantages in complex models; Ji Hui[2] et al. proposed an improved Monte Carlo tree search for computational efficiency in complex decision-making processes, which improves accuracy and efficiency in two-player gaming games by combining a great-minimum algorithm and a pruning strategy; Xiao-chuan Zhang[3] et al. introduced parallel computation, valuation factor (WINK), and suboptimal node balancing factor (UCTK) to improve the UCT algorithm successfully applied to the Einstein Chess game system, and Yingjian Song [4] et al. verified the practicality of the algorithm by using dynamic valuation of pieces and stalemate optimisation through the improved algorithm based on the MCTS search and the dynamic hybrid position evaluation function, which significantly improves the win rate and the search efficiency of the Stan Chess AI.

The concept of upper confidence intervals is applied in the paper for a more subtle strategy selection, aiming to improve the overall effectiveness of Monte Carlo tree search. Through the flexible use of confidence intervals, an attempt is made to guide the algorithm to explore candidate solutions more intelligently in the face of uncertainty and complex search space scenarios. The subtlety of this strategy lies in the fact that it is expected to effectively cut down the computational complexity and accelerate the search convergence speed as well as improve the algorithm's adaptability in terms of adversarial attacks, locally optimal solutions, and continuous action spaces while maintaining the comprehensiveness of the search. The dependence on state evaluation can also be further optimised through the clever integration of upper confidence intervals, thus more comprehensively enhancing the performance of the Monte Carlo tree search algorithm in Einstein chess.



2. Algorithm Design

2.1 Basic principles of Monte Carlo tree search

The principle of Monte Carlo tree search is divided into the following steps:

Selection of strategy: A suitable strategy is chosen to decide which node in the search tree to expand. The Upper Confidence Bound (UCB) algorithm is usually used, which is very effective in balancing the known information (number of visits) with the unknown information (confidence in the value of the node).

$$UCB_i = \frac{w_i}{n_i} + c \sqrt{\frac{\ln N}{n_i}} \quad (1)$$

(UCB_i is the UCB value of the i -th node, w_i is the cumulative reward value of the i -th node, n_i is the number of visits to the i -th node, N is the total number of visits to the parent node, and c is a tuning parameter that controls the balance between exploration and exploitation.)

Expansion Strategy: Adds new child nodes to the selected nodes that represent possible ways of going or states under the current decision. This step is designed to expand the search space, allowing the algorithm to better explore unknown areas.

Simulation Strategy: In the next step of the node expansion strategy, the algorithm performs a simulation. This simulation process simulates a possible decision path that extends to the end state of the game, either randomly or according to a certain strategy. The aim is to evaluate the state of the situation in the current search path in order to get a more complete picture of the possible outcomes under this decision sequence. This simulation process is a key step in Monte Carlo tree search, and by simulating different moves, the algorithm is able to get a fuller picture of what might happen on the current decision path, which helps to optimise the subsequent search strategy.

Evaluation Strategy: Based on the results of the simulation, the number of victories and visits of the simulated paths are counted, and this information is used to guide the selection process in future cycles.

Backpropagation Strategy: The results of the simulation are backpropagated to the root nodes of the search tree, updating the statistics of the nodes visited. Typically, this updating behaviour involves increasing the number of visits and adjusting the number of

victories according to whether the game is won or lost. As for the stopping condition, it involves the decision of when to end the search behaviour, which is often constrained by the characteristics of the problem itself and by time/resource constraints. The Monte Carlo tree flowchart is shown in Figure I. The pseudo-code of the Monte Carlo tree search algorithm is shown in Algorithm 1.

Algorithm 1: Monte Carlo tree search

```

function UCB( $s_0$ )
  Create root node  $v_0$  with state  $s_0$ 
  while Calculate expenses in the budget do
     $V_i \leftarrow \text{Select}(v_0)$  //选择
     $date \leftarrow \text{simulate}(s(v_i))$  //模拟
     $\text{Backpropagation}(v_i, date)$  //反向传播
  return  $a(\text{bestChild}(v_0, 0))$ 
function Select( $v$ )
  while  $v$  is a non-terminal node do
    if  $v$  is not fully extended then
      return  $\text{expand}(v)$ 
    else
       $v \leftarrow \text{bestChild}(v, c)$ 
  return  $v$ 
function  $\text{expand}(v)$ 
  choose  $a \in$  untried actions from  $A(s(v))$ 
  add a new child  $v_l$  to  $v$ 
  with  $s(v_l) = f(s(v), a)$ 
  and  $a(v_l) = a$ 
  return  $v_l$ 
function  $\text{bestChild}(v, c)$ 
  return  $\arg \max \frac{Q(v_i)}{N(v_i)} + c \sqrt{\frac{2 \ln N(v)}{N(v_i)}}$  ( $v_l \in$  children of  $v$ )
function  $\text{defaultPolicy}(s)$ 
  while  $s$  is non-terminal do
    choose  $a \in A(s)$  uniformly at random
     $s \leftarrow f(s, a)$ 
  return reward for state  $s$ 
function  $\text{backpropagation}(v_i, date)$ 
  While  $v$  is not null
     $N(v) \leftarrow N(v) + 1$ 
     $Q(v) \leftarrow Q(v) + date$ 
     $v \leftarrow \text{parent of } v$ 

```

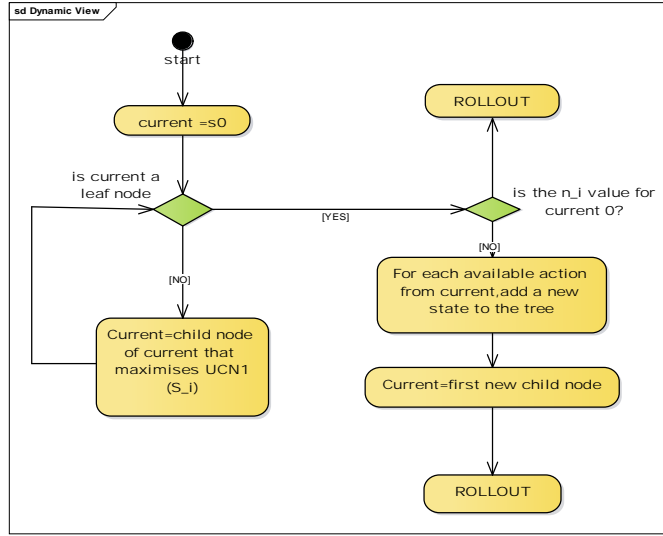


Fig. 1 Flowchart of the Monte Carlo tree algorithm

3. Dynamic tuning of UCB parameters using sliding window averaging, adaptive learning rate

The comprehensive strategy for dynamically adjusting UCB parameters aims to smooth out historical changes, sensitively adapt to performance changes, and make the UCB parameters more adaptive during the iteration of the algorithm. Specifically, by combining both sliding window averaging and adaptive learning rate, we aim to achieve the following objectives:

Sliding Window Averaging: Smoothing of historical changes in UCB parameters is achieved by calculating the average of the performance metrics over a recent period of time. This helps to maintain the relative stability of UCB parameters within a certain time frame.

Adaptive learning rate: dynamically adjusts the learning rate according to the real-time performance of the algorithm performance. When the performance drops, the learning rate is moderately reduced to carefully adjust the UCB parameters; when the performance improves, the learning rate is moderately increased to adapt to the changes more quickly.

Combining the two: At each update of the UCB parameters, a sliding window averaging is first performed to compute the average of the most recent performance metrics. Then, the UCB parameters are adjusted based on the calculation of the average performance and the adaptive learning rate. This combined strategy allows the UCB parameters to be flexibly adjusted to the most recent performance while taking into account historical trends.

Iterative process: The above combined strategy is nested in the iterative process of the algorithm. In each iteration, the performance is measured and then a dynamic tuning function is called which performs sliding window averaging, adaptive learning rate calculations, and updates the UCB parameters. Such an iterative process allows the UCB parameters to adaptively change in different states, better balancing exploration and exploitation.

Such a dynamic adjustment strategy provides the algorithm with an adaptive and stable performance mechanism, making the adjustment of UCB parameters more real-time and flexible. In specific applications, the parameter settings can be fine-tuned according to the characteristics of the problem and experimental results to obtain the best algorithmic performance.

Algorithm flow: we want to smooth out the historical changes of UCB parameters based on the average performance over a period of time to make it more stable. At the same time, we also want to dynamically adjust the learning rate based on the performance of the algorithm to adapt more flexibly to performance changes. To achieve these two goals, we introduce two key ideas. First, we maintain a "window" that can store the performance metrics for the recent period. Each time the UCB parameters are updated, we compute the average of the performance metrics within this window to smooth out parameter changes. Second, we use an adaptive learning rate approach. Based on the current performance and the learning rate decay factor, we dynamically calculate the learning rate. When the performance decreases, we decrease the learning rate to carefully tune the UCB parameters; when the performance improves, we increase the learning rate to

adapt to the changes faster. At each update of the UCB parameters, we first perform a sliding window averaging to compute the average performance. Then, in conjunction with the calculation of the adaptive learning rate, we adjust the UCB parameters. This process, which combines the two approaches, smoothes out historical changes and adjusts the learning rate based on the latest performance, making the UCB parameters more adaptive. Finally, we nest this dynamic tuning process within the iterative process of the algorithm. In each iteration, we measure the performance and then call the dynamic tuning function. This function performs sliding window averaging, adaptive learning rate calculations, and updates the UCB parameters. Such an iterative process allows the UCB parameters to flexibly and adaptively change in different states. The pseudo-code of the algorithm for dynamic tuning of UCB parameters using sliding window averaging, adaptive learning rate is shown in Algorithm 2.

Algorithm2 :Sliding window average, adaptive learning rate to dynamically adjust UCB parameters

```

Ucb_Parameter←initial_Ucb-Parameter
recent_Performance[end]←New_PerformanceMetric
If recent_Performance.length()>window.size()
    recent_Performance.pop(0)
    average_Performance←sum(recent_Performance)/recent_Performance.length()
    current_Learning_Rate←1.0/(1.0+recent_Performance.length()*learning_Rate_Decay)
    Ucb_Parameter+=current_Learning_Rate*(Performance_Metric-avg_Performance)
For iteration in range(numIterations)
    Performance_Metric=measure_Performance()
    Ucb_Parameter=update(Performance_Metric)

```

4. Computational Results

In this paper, we propose a Monte Carlo Tree Search algorithm and the use of sliding window averaging, and adaptive learning rate to dynamically adjust the UCB parameters applied to the Einstein Chess game. We conducted experiments with Monte Carlo Tree Search (MCTS) on the Einstein Chess game. A 5x5 board was used with each player holding 6 pieces. In the experiments, we set the parameters of the MCTS algorithm to be fixed or adjusted against dynamic parameters, including the number of simulations and search time. We compared the MCTS algorithm with random strategies and traditional game algorithms, and also compared the MCTS algorithm (UCB parameters fixed) and MCTS (UCB parameters dynamically adjusted). The results are shown in Tables I and II. It shows that MCTS is significantly better than random strategy with traditional algorithms (DFS, BFS and pruning algorithms) in terms of victory rate, while MCTS algorithm (UCB parameter dynamically adjusted) is significantly better than MCTS algorithm (UCB parameter fixed) in terms of victory rate with short search time. The experimental results show that MCTS performs well in the Einstein chess game. Its higher victory rate may stem from its effective exploration of incomplete information and complex decision spaces. The superiority of MCTS in dealing with Einstein's Chess, a complete information but uncertainty problem, is observed.

Through our experiments, we conclude that MCTS has potential advantages in the Einstein chess game. However, we also realise that the performance of MCTS may be affected by factors such as the opponent's strategy and the number of pieces.

This study provides support for the application of MCTS algorithms in the field of gaming, especially in contexts dealing with uncertainty and complex decisions. This may have a positive impact on the development of gaming AI.

Table 1: Comparison between MCTS (UCB parameter fixed) Stochastic algorithm and Traditional algorithms

Number of games played between two players	search time (s)	MCTS (UCB parameters fixed)		Stochastic algorithm		Traditional algorithm	
		win	lose	win	lose	win	lose
39	1	50	28	30	48	37	41
	4	60	18	25	53	32	46
	8	71	7	17	55	29	49
50	1	70	30	20	80	60	40
	4	80	20	15	85	55	45
	8	88	12	10	90	52	48
80	1	100	60	60	100	80	80
	4	120	40	48	112	72	88
	8	150	10	28	132	62	98

Table 2: Comparison between MCTS (UCB parameters fixed) and MCTS (UCB parameters dynamically adjusted)

Number of games played between two players	search time (s)	MCTS (UCB parameters fixed)		MCTS (dynamic adjustment of UCB parameters)	
		win	lose	win	lose
10	1	2	8	8	2
	4	4	6	6	4
	8	5	5	5	5
20	1	5	15	15	5
	4	8	12	12	8
	8	11	9	9	11
30	1	10	20	20	10
	4	12	18	18	12
	8	16	14	14	16

5. Conclusion

This paper demonstrates the superiority of Monte Carlo Tree Search (MCTS) based algorithms in dealing with incomplete information and complex decision spaces. Experimental results show that MCTS significantly outperforms random strategies and traditional algorithms (e.g., DFS, BFS, and pruning algorithms) in the Einstein Chess game in terms of the victory rate, verifying its strong performance in solving complex decision problems. The introduction of dynamic adjustment of UCB parameters plays a key role in improving the performance of the algorithm, especially through the combined strategy of sliding window averaging and adaptive learning rate, which makes the UCB parameters adapt more intelligently to different states, thus effectively improving the search efficiency. This study provides empirical support for the application of the MCTS algorithm in the field of game AI, and provides a powerful and flexible tool for dealing with problems with uncertainty and complex decisions.

Acknowledgments

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

References

- [1] LIU Guoqing, QIAN Yuhua, ZHANG Yayu, WANG Jiting. Monte Carlo tree search for optimal action recognition algorithm based on relative entropy confidence interval under a given budget[J]. Computer Research and Development, 2023, 60(8): 1780-1794
- [2] Ji Hui, Ding Zejun. Improvement of Monte Carlo tree search algorithm in two-player game problems[J]. Computer Science, 2018, 45(1): 140-143
- [3] ZHANG Xiaochuan, LI Qin, NAN Hai, PENG Lirong. Improved UCT algorithm in Einstein chess[J]. Computer Science, 2018, 45(12): 196-200
- [4] Y.J. Song, R.X. Hou, J.R. Sun, G.G. Shi. Application of MCTS algorithm for dynamic mixed position evaluation in Einstein chess[J]. Journal of Shenyang Engineering Institute (Natural Science Edition), 2022, 18(03): 72-76. DOI: 10.13888/j.cnki.jsie(ns).2022.03.013.
- Gengyun He** is currently reading in School of Computer Science and Software Engineering, University of Science and Technology Liaoning, Anshan 114051, China. He is an undergraduate in the third year.
- Junli Li** is currently working as a Professor at School of Materials and Metallurgy, University of Science and Technology Liaoning, Anshan 114051, China.
- Gang Wang** is currently working as a Professor at the School of Computer Science and Software Engineering, University of Science and Technology Liaoning, Anshan 114051, China.