A Brief Introduction to Game Tree Algorithms

Hao Wang, 1727405048, *Soochow University*

***Abstract*—As an important branch of artificial intelligence research, machine games can be studied extensively. Introduce the current mainstream game tree algorithms in computer games, and combine them organically to give a main frame of search to provide inspiration and reference for game tree researchers.**

***Index Terms*—Alpha-Beta Pruning, Game Tree, Monte Carlo Tree Search.**

I. Introduction

GAME problems are ubiquitous, ranging from children's games and disputes, bargaining in various occasions, to competition from businesses, emergency treatment of various emergencies (terrorism, disasters), national diplomacy, bloody and bloodless wars , As long as there is some conflict of interest between the two parties in the game, the Game Theory becomes a way to express and solve the contradiction Game Theory will become the focus of research in a class of intelligent systems. Chess games with the characteristics of "zero-sum,2 players, sequential with perfect information" have been the focus of research for a long time. It is zero-sum because one player wins always means the other player loses. It is sequential because two players move alternately. It has perfect information means that each player is aware of the state of the board all over the time.In the field of artificial intelligence, chess games are always regarded as one of the most challenging research directions.

In 1958, IBM704 became the first computer capable of playing chess with human, called "Thinking", with a thinking speed of 200 steps per second. By the mid-1960s, Delves asserted that the computer would not be able to defeat a 10-year-old chess player.

In 1973, CHESS 4.0 was developed by B. Slate and Atkin and became the basis of future programs.

In 1979, chess software 4.9 reached the expert level.

In 1981, CRAYBLITZ's new supercomputer had special integrated circuits, and it was predicted that it would defeat the world chess king in 1995.

In 1983, Ken Thompson developed the chess hardware BELLE, which reached the level of master.

In the mid-1980s, Carnegie Mellon University began to study the world-class chess computer program-"Thinking."

In 1987, "Thinking" first appeared at a thinking speed of 750,000 steps per second. Its level is equivalent to a chess player with an international rating of 2,450.

In 1988, "Thinking" defeated the Danish grand master Larson.

In 1989, "Thinking" already had 6 information processors, and the thinking speed reached 2 million steps per second, but in the "human-machine war" with the world chess king Kasparov, it lost by 0-2.

In 1993, the second generation of "Thinking" defeated the Danish national team and won the confrontation with the world's best female chess player Judit Polgár.

In 1995, IBM's "Deep Blue" update program, the new integrated circuit increased its thinking speed to 3 million steps per second.

In 1996, "Deep Blue" lost by 2- 4 in the challenge to Kasparov.

In 1997, the "Deep Blue" group composed of an international super master and four computer experts developed a "Deeper Blue", which has a more advanced "brain", through 256 installed on RS/6000S mainframe computers The dedicated processing chip can calculate 200 million steps per second, and stores 1 billion sets of chess records of the world's top chess players in the past 100 years. Finally, "Deeper Blue" defeated Kasparov 3.5 to 2.5. Kasparov requested a rematch, but did not receive a response.

After the "Deeper Blue" defeated Kasparov, after 18 years of development, the artificial intelligence Go program with the highest chess power only reached the level of amateur 5-step Go players. And without handicap, it still can't beat professional chess players.

In 2012, the Zen program running on four PCs defeated Japanese nine-playe Takemiya Masaki twice under the conditions of 5 handicaps and 4 handicaps.

In 2013, Crazy Stone defeated the Japanese nine-step chess player Ishida Yoshio with 4 handicaps,so the occasional result was already a rare result.

AlphaGo' s research plan was conducted in 2014, and since then has shown a significant improvement compared to the previous Go program. In the 500 games with other Go programs such as Crazy Stone and Zen, the stand-alone version of AlphaGo only lost one game. In the subsequent games, the distributed version of AlphaGo won all 500 games and had a 77% win rate against AlphaGo running on a single machine. The distributed computing version of AlphaGo in October 2015 used 1,202 CPUs and 176 GPUs.

In October 2015, AlphaGo defeated Fan Hui and became the first computer Go program to defeat Go professional players on the 19-way board without handicap. It was published in Nature in January 2016.

In March 2016, through self-playing thousands of games to strengthen practice, AlphaGo defeated the top professional chess player Lee sedol 4:1 in a five-play chess game, becoming the first to defeat the Go professional nine-segment without handicap. After the fifth inning, the Korean Chess Academy awarded AlphaGo the first honorary career in the ninth stage.

From December 29, 2016 to January 4, 2017, the re-strengthened AlphaGo used "Master" as the account name. Without revealing its true identity, it used informal online fast Go games to test and challenge the first-rate master of China, South Korea and Japan, 60 wins at the end of the test.

At the Wuzhen Go Summit from May 23 to 27, 2017, the latest enhanced version of AlphaGo and the world's first chess player Ke Jie competed, and cooperated with eight squad players to fight against five top nine squad players. Winning a three-to-zero victory record, the team battle and the team battle also won. This time AlphaGo uses Google TPU to run, coupled with the rapidly evolving machine learning method, computing resources consume only one-tenth of Lee sedol's version. After the game with Ke Jie, the Chinese Go Association awarded AlphaGo the title of 9-stage professional Go.

II. Game Tree

Let one of the two parties in the game be Party A and the other party be Party B.During the game, a party currently has multiple action plans to choose from.At the right time, he always chooses the best for himself and the least for the other profitable action plan. At this time, if we stand on the side of Party A, there is an "or" relationship between the several action options available to Party A, because then the initiative is in Party A's hands, and he can choose these action options Any action plan. However, if Party B also has several action plans to choose from, for Party A, these action plans are in an "AND" relationship, because Party B plays the game. The power is in the hands of Party B. Any of these alternative action plans may be selected by Party B. Party A must consider the occurrence of the most unfavorable situation for itself. The above game process is represented by a graph, and the result is an "and/or" tree, see Figure 1. It should be particularly pointed out here that the "and/or" tree is always derived from the standpoint of a certain party (such as party A), and the "and/or" tree is different for different moves. We call this AND/OR tree describing the game process the game tree.Among them, the square node represents the turn of party A's move, and the round node represents the turn of party B's move. In most gameplay discussions, the format used is slightly different. The two players are named Max and Min, where all values are given from the perspective of Max.

III. MiniMax

The MiniMax algorithm simply traverses each Tree node to get the value of the game.You can see the algorithm in Algorithm 1.The example game tree from above can be filled in with this algorithm (Figure 2).

Unfortunately, this form of Minimax is not particularly useful. It is completely accurate, but for any game that is more complicated than "Tic Tac Toe", it is impractical to search the end status of all possible games to make a decision. Doing so will result in extremely high time

algorithm 1 MiniMax

```
1:  function MiniMax($Node, flag$)
2:      if $node\ is\ leaf$ then
3:          return $Node.value$
4:      end if
5:      if $flag\ ==\ 1$ then
6:          $v \leftarrow -\infty$
7:          for do $traverse\ each\ child\ of\ Node$
8:              $v \leftarrow$ Max($v$, MiniMax($child, -flag$))
9:          end for
10:         return $v$
11:     else
12:         $v \leftarrow \infty$
13:         for do $traverse\ each\ child\ of\ Node$
14:             $v \leftarrow$ Min($v$, MiniMax($child, -flag$))
15:         end for
16:         return $v$
17:     end if
18: end function
```

and space costs. Instead, the game program uses heuristic evaluation. We need to design a constant time function that represents the estimated value of the first player's revenue. How to design this function is beyond the scope of this article. We can limit the depth of the search and return the heuristic evaluation directly when the specified depth is reached.You can see the algorithm in Algorithm 2.

algorithm 2 MiniMax2

```
1:  function MiniMax2($Node, depth, flag$)
2:      if $node\ is\ leaf$ or $depth\ ==\ 0$ then
3:          return $Node.heuristic - value$
4:      end if
5:      if $flag\ ==\ 1$ then
6:          $v \leftarrow -\infty$
7:          for do $traverse\ each\ child\ of\ Node$
8:              $v \leftarrow$ Max($v$, MiniMax2($child, depth - 1, -flag$))
9:          end for
10:         return $v$
11:     else
12:         $v \leftarrow \infty$
13:         for do $traverse\ each\ child\ of\ Node$
14:             $v \leftarrow$ Min($v$, MiniMax2($child, depth - 1, -flag$))
15:         end for
16:         return $v$
17:     end if
18: end function
```

This pseudocode handles all situations perfectly, but the two branches that are logically separated make it annoying. We will use Algorithm 3 instead.

Please note that these two functions are not the same. Minimax always returns the node value for the first player, while Negamax returns the node value for the current

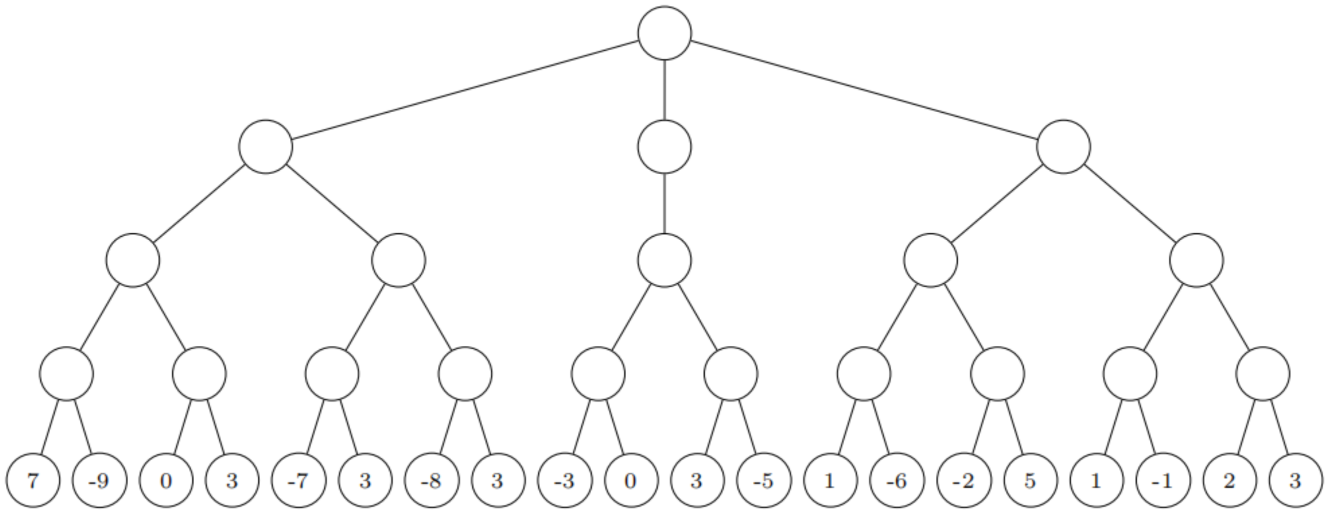

Fig. 1: Game Tree

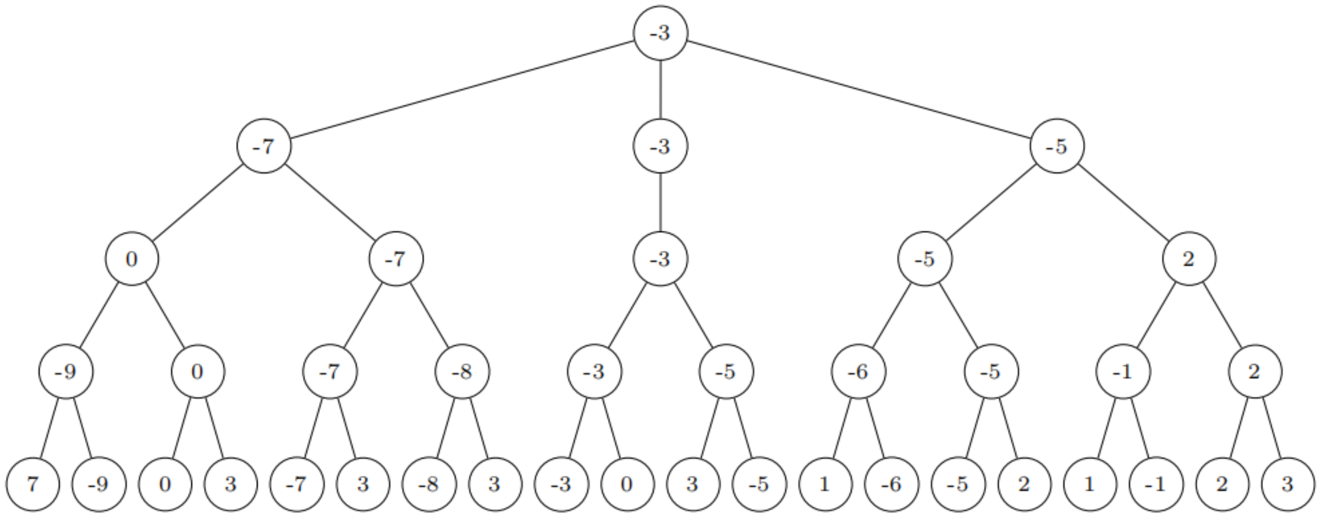

Fig. 2: A game tree that has been filled with a Minimax procedure

algorithm 3 NegaMax

```
1: function NegaMax($Node, depth, colour$)
2:     if $node\ is\ leaf$ or $depth\ ==\ 0$ then
3:         return $colour * Node.heuristic - value$
4:     end if
5:     $v \leftarrow -\infty$
6:     for do $traverse\ each\ child\ of\ Node$
7:         $v \leftarrow$ Max($v$, -NegaMax($child, depth - 1, -colour$))
8:     end for
9:     return $v$
10: end function
```

player. That is, a high Minimax value indicates that the first player is in a good position, and a high Negamax value indicates that the player who is about to move is in a good position. This relationship can be more accurately described as:

$$\textbf{Negamax}(node, depth, 1) = \textbf{Minimax}(node, depth, 1) \tag{1}$$

$$\textbf{Negamax}(node, depth, -1) = -\textbf{Minimax}(node, depth, -1) \tag{2}$$

If we use one of Minimax and Negamax to fill the game tree, the only difference is that the nodes of the second player are opposite to each other. From now on, we will fill the game tree in Negamax style.

IV. Alpha-Beta Pruning

Considering the game tree in Figure 3, it can be shown that no matter what the number at the question mark is, it will not affect the final result. So there is no need to know what the value at the question mark is. To prove

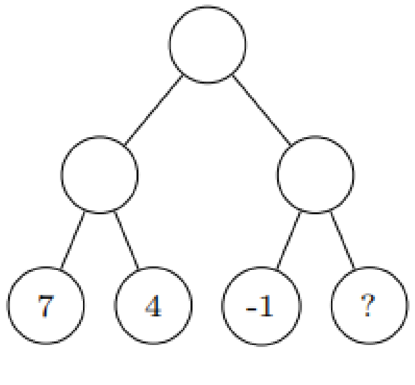

Fig. 3

this, fill the second layer with known information(Figure 4).

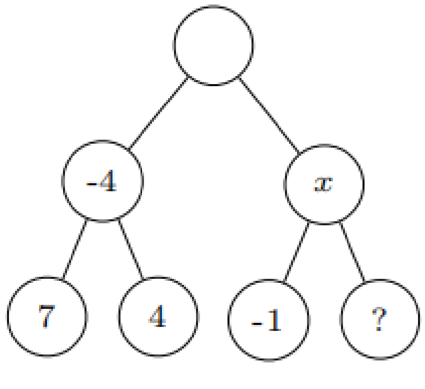

Fig. 4

$x = -\textbf{min}(-1, ?) \implies x \geq 1$, So no matter what is at the question mark, at the root node will always choose the left child since $-4 < x$.

If we can apply this optimization to our Negamax program, then we can evaluate fewer branches in the game tree without any change in the final value. We will record two values, α and β, which represent the best scores that the largest and smallest players can guarantee. We specify the following behavior(Equation 3)

Although not strictly necessary, we will impose the prerequisite of $\alpha < \beta$.This simplifies the proof of correctness to a certain extent, and it should be so in any sensible implementation. The proof is through the induction of depth. In the basic case of $depth == 0$ or when the node is a leaf, the specification can be easily met.Otherwise, please consider entering the loop. We will show that the following invariants adapted from [1] are maintained:

$$m = \max(\alpha \cup -N(p) | p \textit{ is a child that has been visited}) \tag{4}$$

$$m < \beta \tag{5}$$

Where we define $N(p) = \text{Negamax}(p, depth - 1, -colour)$. These invariants are trivially initialized (recall

algorithm 4 Alpha-Beta

```
 1: function ALPHA-BETA($Node, depth, colour, \alpha, \beta$)
 2:     if $node\ is\ leaf$ or $depth == 0$ then
 3:         return $Node.heuristic - value$
 4:     end if
 5:     $v \leftarrow \alpha$
 6:     for do $traverse\ each\ child\ of\ Node$
 7:         $v \leftarrow$ Max($v$, -Alpha-Beta($child, depth - 1, -colour, -\beta, -v$))
 8:         if $v \geq \beta$ then
 9:             return $\beta$
10:         end if
11:     end for
12:     return $v$
13: end function
```

that (5) is simply the precondition). To show that they are maintained, we must be careful about the **return** statement in the loop. We can split up our reasoning into three cases:

1. $N(child) \leq -\beta$. Then, by our induction hypothesis, we have Alpha-Beta$(child, depth - 1, -colour, -\beta, -m) \leq -\beta \Rightarrow$ -Alpha-Beta$(child, depth - 1, -colour, -\beta, -m) \geq \beta$ So our new value of m is also at least β, and we return β. Note that by the definition of Negamax, $N(child) \leq -\beta$ also implies that Negamax$(node, depth, colour) \geq \beta$, so we are correct to return β.

2.$\beta < N(child) < -m$. Then, by our induction hypothesis, we have that Alpha-Beta$(child, depth - 1, -colour, -\beta, -m) = N(child)$ and the invariant follows.

3.$N(child) \geq -m$. Then, by the induction hypothesis, Alpha-Beta$(child, depth - 1, -colour, -\beta, -m) \geq -m \Rightarrow -Alpha-Beta(child, depth-1, -colour, -\beta, -m) \leq m$ So m is not updated. But since $-N(child) \leq m$, we do not want m to be updated and the invariant is maintained.

If we exit the loop, then we have processed every child. So by (4) and (5) we have $m = max(\alpha, Negamax(node, depth, colour)) < \beta$ which, after a moment of thought, we can see is a valid return value. So Alpha-Beta does indeed match the specification.

Figure 5 shows the tree from Figure 1 filled using alpha-beta pruning.

Table 1 shows the number of nodes accessed when using the Negamax and Alpha-Beta pruning algorithms, respectively. All the game trees are all full binary trees.

It can be seen that the effect of alpha-beta pruning is quite obvious.

V. Monte Carlo Tree Search

Although Alpha-Beta pruning can greatly reduce the number of nodes in the game tree search and can defeat the super master when used in chess programs, its effect on Go is still very poor.

The reason is as follows.

First of all, there are so many branching factors (selectable moves for each step) of Go, which can reach more than 200. As a comparison, chess only has about 50.

$$\textbf{Alpha-Beta}(...,\alpha,\beta) = \begin{cases} \le \alpha & \text{if } \textbf{NegaMax}(...) \le \alpha \\ \textbf{NegaMax}(...) & \text{if } \alpha < \textbf{NegaMax}(...) < \beta \\ \ge \beta & \text{if } \textbf{NegaMax}(...) \ge \beta \end{cases} \tag{3}$$

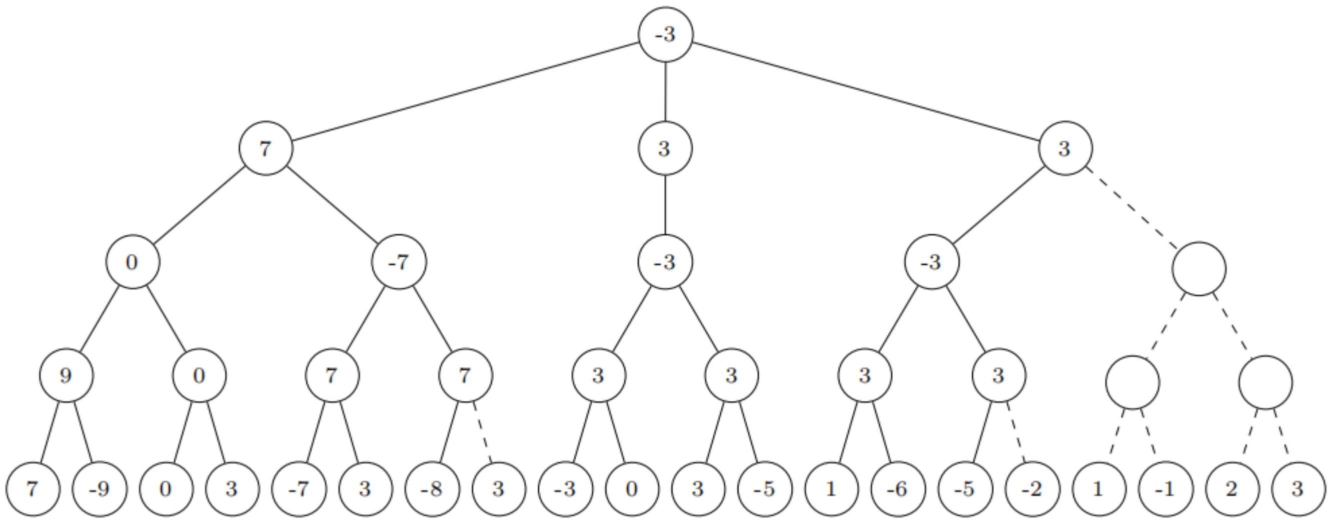

Fig. 5: A game tree filled using an alpha-beta pruning procedure

TABLE I: Number of node visits under different algorithms and data

method / *visits*		Alpha-Beta				NegaMax
		ascend	descend	rand1	rand2	
Tree height	21	43009	33792	184829	175851	4294303
	22	67585	47104	294300	385755	8388607
	23	94209	73728	552554	511948	16777215
	24	147457	102400	607157	1102531	33554431
	25	204801	159744	1842752	1273529	67108863
	26	319489	221184	1573976	2748010	134227727

Secondly, chess will become more and more simplified as the game progresses, which is not the case with Go, and it may even become more complicated.

Finally, it is difficult to evaluate the heuristic value of Go in a state.Moving one piece of a chess piece may cause a huge change in the situation.

In 2006, Rémi Coulom described the application of the Monte Carlo method to game-tree search and coined the name Monte Carlo tree search[2]The focus of MCTS is to analyze the most promising moves, expanding the search tree based on random sampling in the search space. The application of Monte Carlo Tree Search in the game is based on many playouts (also called roll-outs). In each play, the game is played to the end by randomly selecting moves. Then, the final game result of each play is used to weight the nodes in the game tree, so that it is more likely to select better nodes in future plays. The most basic way to use playouts is to apply the same number of playouts after each legal move of the current player, and then choose the move that has the highest win rate. The efficiency of this method (called "pure Monte Carlo game search") usually improves over time, and more game results are assigned to actions that often lead to the current player winning based on previous game results. Each round of Monte Carlo tree search includes four steps:

Selection: Start from the root and select consecutive child nodes until reaching the leaf node L. The root is the current game state, and the leaf is any node from which simulation (playout) has not yet been started. The following section details a method of biasing the selection of child nodes. This method expands the game tree to the most promising actions. This is the essence of Monte Carlo tree search.

Expansion: Unless L decisively ends the game (for example, win/loss/draw) for any player, create one (or more) child nodes and select node C from one of them. Child nodes are any effective movement starting from the game position defined by L.

Simulation: Complete a random playout from node C. This step is sometimes referred to as playout or rollout. A knockout game can be as simple as choosing uniform random moves until the game is determined (for example, in chess, the game wins, loses, or draws).

Backpropagation: Use the results of the playout to update the information in the nodes on the path from C to R.

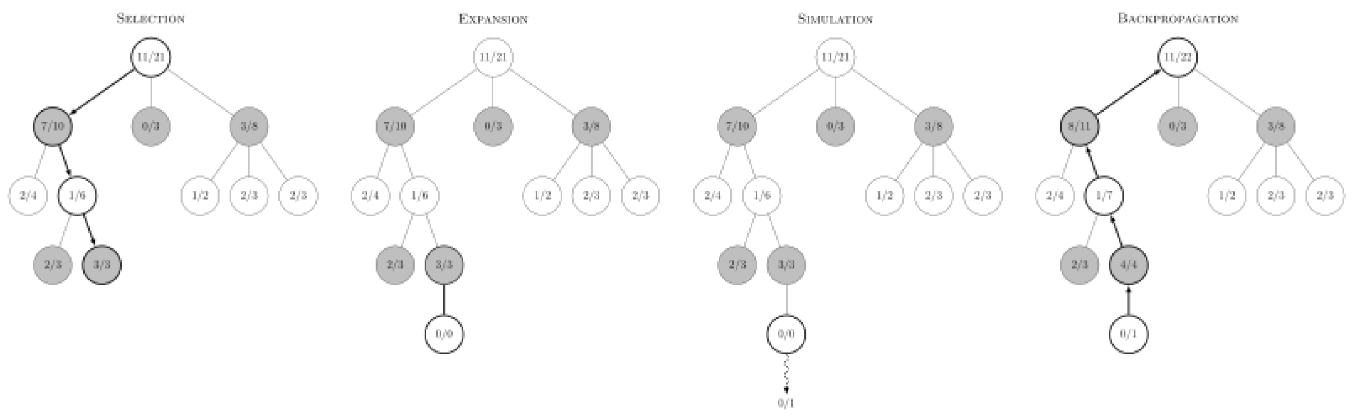

Fig. 6: Step of Monte Carlo tree search

Rounds of search are repeated as long as the time allotted to a move remains. Then the move with the most simulations made (i.e. the highest denominator) is chosen as the final answer. Monte Carlo tree search can effectively reduce the amount of calculation and get a better solution in a very short time

VI. Conclusion

The key technologies in the game tree algorithm are briefly introduced above.Of course, there are some algorithms that have not been introduced, such as the Transposition Table algorithm.As we mentioned before, the original Alpha-Beta Pruning can only used in ”zero-sum,2 players, sequential with perfect information” games. So, some researchers are studying how to use it for games with simultaneous moves[3].Some others are studying how to use it for multiplayer games[4-6]. Also, how to optimize Alpha-Beta Pruning is an important research topic[7-10]. As for Monte Carlo tree search, some are studying how to assist it with memory-based algorithm. Some are studying how to run Monte Carlo tree search in parallel. Monte Carlo Tree Search has been used to schedule elective admissions of patients to the hospital [11].Guided Monte Carlo Tree Search for Planning in Learned Environments]Game tree processing algorithms can provide suggestions for us to actually deal with problems, and scientific researchers will continue to expand their application fields.

References

[1] Silver, David, et al. ”Mastering the game of Go with deep neural networks and tree search.” *nature* 529.7587 (2016): 484.
[2] Coulom, Rémi. ”Efficient selectivity and backup operators in Monte-Carlo tree search.” *International conference on computers and games*. Springer, Berlin, Heidelberg, 2006.
[3] Saffidine, Abdallah, Hilmar Finnsson, and Michael Buro. ”Alpha-beta pruning for games with simultaneous moves.” *Twenty-Sixth AAAI Conference on Artificial Intelligence*. 2012.
[4] Korf, Richard E. ”Multi-player alpha-beta pruning.” *Artificial Intelligence* 48.1 (1991): 99-111.
[5] Fridenfalk, Mikael. ”N-Person Minimax and Alpha-Beta Pruning.” *NICOGRAPH International 2014, Visby, Sweden, May 2014*. 2014.
[6] Sturtevant, Nathan R., and Richard E. Korf. ”On pruning techniques for multi-player games.” *AAAI/IAAI* 49 (2000): 201-207.
[7] Fishburn, John P. ”Another optimization of alpha-beta search.” *ACM SIGART Bulletin* 84 (1983): 37-38.
[8] Björnsson, Yngvi, and T. Anthony Marsland. ”Multi-cut alpha beta-pruning in game-tree search.” *Theoretical Computer Science* 252.1-2 (2001): 177-196.
[9] Nasa, Rijul, et al. ”Alpha-beta pruning in mini-max algorithm–an optimized approach for a connect-4 game.” *Int. Res. J. Eng. Technol* (2018): 1637-1641.
[10] Pearl, Judea. ”The solution for the branching factor of the alpha-beta pruning algorithm and its optimality.” *Communications of the ACM* 25.8 (1982): 559-564.
[11] Zhu, George, Dan Lizotte, and Jesse Hoey. ”Scalable approximate policies for Markov decision process models of hospital elective admissions.” *Artificial intelligence in medicine*61.1 (2014): 21-34.
[12] Knuth, Donald E., and Ronald W. Moore. ”An analysis of alpha-beta pruning.” *Artificial intelligence* 6.4 (1975): 293-326.
[13] Fuller, Samuel H., John G. Gaschnig, and J. J. Gillogly. *Analysis of the alpha-beta pruning algorithm*. Department of Computer Science, Carnegie-Mellon University, 1973.
[14] Benjamin, Katherine. ”Alpha-beta pruning and its place in computer game play.” (2019).
[15] Ginsberg, M., and Alan Jaffray. ”Alpha-beta pruning under partial orders.” *More Games of No Chance* (2002): 37-48.
[16] Singhal, Shubhendra Pal, and M. Sridevi. ”Comparative study of performance of parallel Alpha Beta Pruning for different architectures.” *arXiv preprint arXiv:1908.11660* (2019).
[17] Ferguson, Chris, and Richard E. Korf. ”Distributed Tree Search and Its Application to Alpha-Beta Pruning.” *AAAI*. Vol. 88. 1988.
[18] 徐心和, and 王骄. ” 中国象棋计算机博弈关键技术分析.” 小型微型计算机系统 27.6 (2006): 961-969.
[19] 岳金朋, and 冯速. ” 博弈树搜索算法概述 .” 计算机系统应用 9 (2009).

Hao Wang Studying at Suzhou University, pursuing a bachelor's degree in engineering.