The Research on Patterns and UCT Algorithm in NoGo Game

Yuxia Sun, Cheng Liu, Hongkun Qiu
Engineering Training Center, Shenyang Aerospace University, Shenyang, 110136, China
syxa@163.com

Abstract: NoGo game is similar to Go game, that means they have similar notion of group, killing, stones and chessboard. However, their rulers are completely different, or opposite, like the first player suicides, kills a group, or has no more legal moves, the opponent will win the game. In this paper, the patterns of NoGo are expounded in detail. The pattern values, the pattern sizes and the pattern formats are discussed. At the same time, an optimizing pattern matching algorithm is put forward to find out the best move for NoGo game. In addition, if the best move has not been found by the matching pattern algorithm, a modified UCT algorithm is also applied to play the NoGo game. The experimental results show that the proposed approaches are workable for NoGo.

Key Words: NoGo, Computer Games, Patterns and UCT algorithm

1 INTRODUCTION

The NoGo game is a completely new computer game and has been invented by the organizers of the Birs workshop on Combinatorial Game Theory 2011[1]. It has the syntaxes similarity with Go game such as notion of group, killing, black and white stones and chessboards. In addition, the two players confront each other and take turns to play the stones, so the both sides not only know the opponent's previous moves, but also can estimate the opponent's next moves. A two-person information symmetrical game such as Go or NoGo can be typically represented by a special game tree, and near optimal solutions of such game are usually computed by searching its game tree [2].

The NoGo game lacks the concept of pieces and each stone has exactly the same value as other pieces, which made it difficult to define the feature elements for the assessment of situations. Another problem is the fact that the search space is complex and vast, which made it difficult to utilize static evaluation functions to develop NoGo programs. Consequently, the traditional Minimax search algorithms are hardly suitable for NoGo.

Recently, Monte-Carlo Tree Search (MCTS) has been developed to find optimal decisions in a given domain by taking random samples in the decision space and building a search tree according to the results [3]. The Upper Confidence bounds for Trees (UCT) algorithm, as the most popular algorithm of the MCTS family, has contributed to a significant improvement in the game of Go [4]. During simulation, the UCT algorithm treats the selection of moves as a multi-armed bandit problem, using the UCB algorithm [5] for selection, i.e. an application of the UCB algorithm in tree search. In other word, the UCT algorithm is the idea that in the limited time, many tree traversals are played from the root to leaf nodes with evaluations and each evaluation result is propagated back to the root of the tree along the traversal path. Many Go game programs based on the UCT algorithm have achieved great success, such as MoGo or FueGo.

In the simulation strategy field, the pattern is also a well-known technique in computer Go for a long time [6] And the famous Go program, GNU GO, has applied the handcrafted pattern database for move selection. For NoGo game, a new game, the research results is rather immature and there have been less successful manmade intelligent gaming programs. Because NoGo is very similar to Go, but not same, we can partly refer to the method and theory of Go, apply patterns and UCT algorithm to NoGo.

In this paper, an optimizing pattern matching algorithm is put forward to find out the best move for NoGo. If the best move has not found by pattern matching, a modified UCT algorithm will be carried out in NoGo game, which is expounded at the latter part of this paper.

The rest of this paper is organized as follows: section 2 introduces the rules of NoGo. Section 3 describes the patterns for NoGo game. Section 4 deals with UCT algorithm. Section 5 proposes a modified UCT algorithm for NoGo game. Section 6 shows and analyzes the experimental results. Finally, Section 7 gives the final conclusion of this paper.

2 NOGO GAME

Unlike Go game, the players can use the 19-by-19 standards board or other the 13-by-13 board etc. NoGo game just uses the 9-by-9 NoGo board. The two players, White and Black, alternately place their own stones on the empty crossings of the board, if the first player who suicides, kills a group, or has no more legal moves, he will lose the game, not like Go game that the winner is determined by territory considerations at the end of the game [7]. The two players share 81 legal moves in the beginning of the game, and both legal moves are changing during the process of playing the game. As a result, it is very helpful for the player that the own eyes and the own groups with one liberty are more in NoGo. In other word, the number of legal moves for the opponent becomes less, and the winning opportunity for the opponent gets lower. Consequently, the player should let the liberties of own stones become fewer, and the opponent legal move positions decrease, which is exactly opposite of Go.

3 THE PATTERNS OF NOGO GAME

The patterns play a very important role in some computer games such as Go [8-9], not only for computers, but also human players use various of patterns when they play. In consideration of NoGo's syntactic similarity with Go game, the patterns should be very effective for NoGo game. Moreover, comparing with abstract rules, the patterns can make NoGo-knowledge more intuitive and convenient to represent and use. On the other hand, the patterns can be directly used to generate moves to guide playing, reduce search range and improve search efficiency and accuracy. A pattern of NoGo game is often defined one free intersection where one move is supposed to be played and its surrounding. The patterns for NoGo are illustrated in follows.

3.1 Pattern size

In practice, the adequate pattern size should be defined to avoid too large size to reduce the matching rate, and too small size to miss some contexts of surroundings. In order to make the patterns more accurately for the NoGo board, the unfixed boundaries and various kinds of regions should be extracted. Furthermore, according to the patterns of Go game, the pattern size of NoGo game is usually from 3-by-3 to 5-by-5. Because the 3-by-3 pattern can easily describe an empty position and its 8 neighboring intersections, the most pattern size is 3-by-3.

3.2 Pattern value

In general, each point on the board whether stone or empty intersection represents different importance for NoGo game. That is, the importance of each crossing in NoGo is not only determined by itself, but also affected by its surrounding. Because the urgency of a move generated by the different pattern is different, the urgency of a pattern is represented by its value, which corresponds to the urgency of playing the move. The pattern values of NoGo game are usually from 20 to 60. Generally, the greater the value of a pattern is, the more the urgency of the move generated by it is. In pattern matching, the patterns with big value are chosen preferentially.

3.3 Pattern classification

According to the patterns suitable for different positions of the board, the patterns of NoGo are divided into three categories, i.e. corner pattern, edge pattern and center pattern. Because a pattern represents different meaning for Black and White, some patterns are suitable for Black and some are suitable for White in NoGo. Consequently, the patterns of NoGo are also separated into tow types for Black and White. Some most frequent patterns for White are listed in Fig. 1, where the star-marked stones are the good locations for White's next move. in Fig. 1, the (a), (b) and (c) are corner patterns and their values are 40, 35 and 40, respectively; the (d), (e) and (f) are edge patterns and their values are 25, 50 and 40, respectively; the (g)~(k) are center patterns and their values are 50, 60, 50, 25and 25, respectively. The (a) and (b) are very useful for the opening of NoGo. The (e), (g) and (h) are the attacking patterns and can destroy opponent's eyes. The (i) can generate player's eyes.

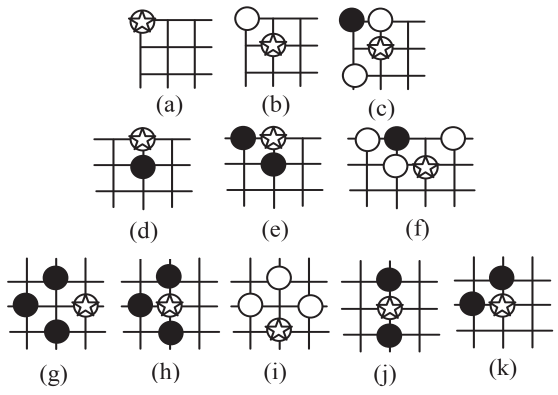
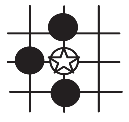


Fig. 1 Some most frequent patterns for White player.

3.4 Pattern format

The patterns are very intuitional layout on the board like in Fig. 1. But it is very complex that the patterns are stored in computer and dealt with by programs. If the patterns in form are coded, the coded patterns can be quickly matched. However, the conversions of the coded patterns need to take time in pattern matching and the shapes of the coded patterns are regular. As a result, the coded patterns are suitable for the games with a large number of patterns, like Go game. Nevertheless, for NoGo game, the patterns are not mature and its number is not very many. So the patterns of NoGo can be directly represented. In Fig. 2, the pattern for White shown in the left can be stored a two-dimensional array, which indicates the pattern stored format, four matching points, and 60 of the pattern value. The location of each matching point in patterns is signified by the relative coordinates of the star-marked point. Moreover, each position in patterns can be black, white, empty and outside the board which are represented by P_BLACK (0001B), P_WHITE (0010B), P_EMPTY (0100B), P_OFF (1000B), respectively. Through rotation and reflection, one pattern can have 8 different variations on the board of NoGo.



```
PATTERNS_W[]=
   {  PATTERN, 4, 60,

      -1, 0, P_BLACK,
      1, 0, P_BLACK,
      0,-1, P_BLACK,
      0,1, P_EMPTY,
        …..
```

Fig. 2 The format of a pattern for White player.

3.5 A pattern matching algorithm for NoGo

According to the above description and analysis, the pattern and the NoGo board are usually represented by two-dimensional arrays. So the patterns needn't convert in matching. An optimal pattern matching algorithm for NoGo is proposed and processed along following steps.

```
Step 1. Choose the pattern for Black or White according to the current player.
Step 2. Search a legal move location on the board of NoGo.
Step 3. Recognize the patterns at the chosen location on the board. The process of matching pattern is described as follows.
Let PS represents the points of the matching pattern and RS represents the relative points around the move location on the board. FLAG is flag variable.
Compute the following statements.
FLAG=1;
For every point in the matching pattern
 {RT = PS AND RS;
If RT = = FALSE
FLAG=0; break; // the matching fail
}
If FLAG= =1    the matching success
Repeat Step 2~3 until all the legal move locations are scanned.
Step 4. Find the best move by comparing the values of the matching successful patterns.
```

The matching algorithm is workable for NoGo, especially at the opening period. If the best move has not been found by pattern matching, a modified UCT algorithm will be used to find the best move. Because of using pattern matching, the range of the UCT search decreases and the searching efficiency obviously improves.

4 UCT ALGORITHM

UCT (Upper Confidence Bounds for Trees) is an algorithm by means of UCB (Upper Confidence Bound) formula and the result of Monte-Carlo simulation to expand search state. UCB is generated from solving K-armed bandit problem, which is a slot machine assumed to have K handles and allows to pull down one of its handle thus to win a certain amount of money, that is the reward in relation to the handle(action). The objective of the gambler is to maximize the collected reward sum through iterative plays. It is classically assumed that the gambler has no initial knowledge about the arms, but through repeated trials, he can focus on the most rewarding arms. There is an exploitation-exploration dilemma in reinforcement learning. Precisely, exploitation is to select the current best arm according to the accumulated knowledge, while in case to select only exploited handles rather than other ones, some with higher reward may be missed. Therefore, the unexploited handles should be chosen suitably, which is called as exploration. UCB tries to solve the contradiction between exploitation and exploration to find the equilibrium point between them [10].

The simplest UCB policy proposed by Auer et al. [11] is called UCB1.Based on known knowledge, the UCB value of a handle is the average reward of the handle so far plus the value of the adjustment item which is used to equilibrate exploitation and exploration in UCB. Each time the handle with the maximum UCB value is selected, UCB formula is as follows:

$$UCB = \overline{X}_i + C\sqrt{\frac{2\ln n}{n_i}} \qquad (1)$$

Where $\overline{X}_i$ is the average reward from arm i, n_i is the number of times arm i was played and n is the overall number of plays so far. The reward term $\overline{X}_i$ encourages the exploitation of higher-reward choices, while the rest term encourages the exploration of less visited choices.

UCT is the extension of UCB to tree search. The idea is to consider each node as an independent bandit, with its child-nodes as independent arms. Instead of dealing with each node once iteratively, it plays sequences of bandits within limited time, each beginning from the root and descending the game tree until encountering a leaf node. When arriving at a leaf node, if it is not a terminal state, a new node is added to the tree. The reward value of UCB formula is the winning rate of the state, which is judged by the result of simulation gaming according to Monte-Carlo sampling. According to the result, UCT will update the reward values of all nodes on the path from the leaf node to the root node.

In the process of descending the game tree, a child node with the highest UCB value will be chosen. The UCB value can be calculated using the following equation for node i.

$$UCB(i) = \begin{cases} \infty & (V_i = 0) \\ \frac{W_i}{V_i} + C\sqrt{\frac{\log N}{V_i}} & (V_i > 0) \end{cases} \qquad (2)$$

Where V_i represents the number of times the node i has been visited, W_i represents the winning times of the node i during the V_i times visits, N denotes the total visited times so far and $C > 0$ is a constant. It is generally considered that $V_i = 0$ yields a UCB value of ∞, so that previously unvisited children are assigned the largest possible value, which can ensure all children of a node to be visited at least once before any child is expanded further. When all children of a node have been explored, UCT will tend to re-explore the most promising ones, which is controlled by a constant C. The higher the constant C is, the more UCT will explore unpromising nodes. At the limit, when the whole game tree has been explored, UCT favors the better branches and tends to converge to the same choice as the Minimax exploration [12].

5 UCT FOR NOGO GAME

UCT is an important algorithm for tree searching, which has been proven very powerful search algorithm for the lack of domain-specific knowledge in Go Game. Like Go game, NoGo game has too much branching factors to formulate the useful heuristics. So UCT algorithm is also applied to NoGo.

In NoGo, if the player put his stones at the eyes or the groups with one liberty, he probable loses the game immediately. As a result, in the process of UCT simulation, the losing points, which are the empty points in the eyes or the groups with one liberty, should be removed when the nodes of the tree are created and Monte-Carlo simulation is executed at the leaf node. In the UCT simulations, each node of the tree represents a NoGo board situation and child nodes represent next situations after corresponding move. The pseudocode of the UCT algorithm for NoGo is shown as following.

```
Node * UCTSearch(int Color)
{
    Create root node Root and initialize it
    CreateChildren(Root);
   while (within the limited time)
   {      BoardCopy();
        UCTSimulation(Root,Color);  }
  retrun(getBestChild(Root));
}
int UCTSimulation (Node *n, int color)
{
    if (n->child == NULL && n->visits < num_visits)
         int randresult = UCTplayRandomGame(color);
   else
   {     if (n->child == NULL)
          CreateChildren(n);
         Node *GoodNode = UCTSelect(n);
         MakeMove(GoodNode,color);
         int res = UCTSimulation (GoodNode,color);
         randresult = 1 - res;      }
        UpdateNode(1-randresult, n);
   return randresult;
}
Node* UCTSelect(Node *nd)
 {
   for (each child of nd, next)
   {  if ((*next).visits > 0) {
        winrate=(double)(*next).wins / (*next).visits;
        uctvalue = winrate +UCTK * sqrt( log((*nd).visits) /
                   (*next).visits );       }
    else   uctvalue =∞;
    Get max uctvalue node in all children of nd, MaxNode}
  return MaxNode;
 }
int UCTplayRandomGame(int color)
{
   while (true)
       {if(empty_num<=0) break; /*empty_num is the number
            of EMPTY except for the losing points*/
       Generate a random value (num) in [0, empty_num-1].
       CoordTransfor( temp, num);
       temp_board[temp.x][temp.y] = color;
       empty_num--;
       empty_void[num] =empty_void[empty_num];
       FindEyeLost();
       color = color%2+1;    }
   return score();} /* 1 or 0*/
```

The functions of UCTSearch() and UCTSimulation() are to execute one sequence continuously within the limited time.

After each sequence, the values of all nodes on the path are updated iteratively from the leaf node to the root node by the result of Monte-Carlo simulation. When the program terminates, the best move, corresponding to the child node with the highest value, will be returned, The UCTSelect() function is to choose one arm with the max UCB value by the formula of (2). In order to ensure each child node to be selected once before further exploration, the UCB values of non-visited nodes are set to infinity.

At the leaf node, if its visited number is less than the threshold value, a Monte-Carlo simulation will be executed by the UCTplayRandomGame() function. The pure random simulation is to play stones on the NoGo board uniformly randomly. But if the simulation is executed by the pure random simulations, the simulation efficiency will be reduced and the most simulation time will be wasted. Therefore in UCTplayRandomGame(), the losing points, which are the empty points in the eyes or the groups with one liberty, are removed in the simulation. Furthermore, because the range of random simulations becomes narrow, the search speed will becomes fast. Accordingly, the branching factor in tree search can reduce and the search depth of the tree can increase. The simulation result is determined by the number of the losing points. If the number of the first player is less the opponent's, the score is 1, or the score is 0. The modified UCT can improve the search efficiency.

6 EXPERIMENTAL RESULTS

In this study, two versions of NoGo programs, A and B, are constructed to show the effect on the winning rate under different the given simulation time (equally the number of simulations) for each move. The program A can play by basic UCT algorithm for NoGo game, but the program B is implemented using the methods in the former section in this paper, i.e. the combined application of the pattern matching and the modified UCT algorithm. The winning rates of the B program are showed in Table 1, and the given simulation time (seconds) per move on the board for Black and White are set to 5, 10, 20, and 50, respectively. From the experimental results, we can see that the winning rates of the B program were higher, and especially the given simulation time per move increases. The results indicated that the combined simulation mechanism is very useful for NoGo game.

Table 1 The winning Rates of B vs. A by changing the simulation time per move.

Seconds per move	Winning Rate of B for Black	Winning Rate of B for White	Total Winning Rate of B
5	70%	68%	69.4%
10	75%	76%	75.5%
20	81%	82%	81.7%
50	89%	86%	87.6%

7 CONCLUSIONS

In this paper, the patterns for NoGo are described in detail, and the pattern's layout and value, which denotes urgency, are also fully discussed. A very helpful pattern matching algorithm for NoGo is put forward and is effective. Additionally, the modified UCT algorithm is supplemented to infer the position of the best move for NoGo if a best move was not found out by the pattern matching algorithm. The experimental results show that the winning rates of B program were higher, especially increasing the simulation time per move, and also confirm that the combined application of the pattern matching and the modified UCT algorithm is more valuable for NoGo comparing with the basic UCT algorithm.

REFERENCES

[1] C. W. Chou, O. Teytaud, and S. J. Yen, Revisiting Monte-Carlo tree search on a normal form game: NoGo. The main European events on Evolutionary Computation (Evo* 2011), Torino, Italy, Apr. 27-29, 2011.

[2] J. Huang, Z.Q. Liu, A Persistent Game Graph in Computer GO. Chinese Control and Decision Conference (CCDC), 2518-2521, 2011.

[3] C. Browne, E. Powley, D.Whitehouse, A Survey of Monte Carlo Tree Search Methods. IEEE Trans. On Computational Intelligence and AI in Games, Vol. 4, No. 1, 1-43, 2012.

[4] S. Gelly, D. Silver, Achieving master level play in 9 x 9 computer go, in Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, 1537–1540, 2008.

[5] P. Auer, N. Cesa-Bianchi, Y. Freund, R. Schapire, The nonstochastic multiarmed bandit problem, SIAM Journal on Computing, vol. 32, 48-77, 2002.

[6] Wang Y., Gelly S., Modifications of UCT and sequence-like simulations for Monte-Carlo Go. IEEE Symposium on Computational Intelligence and Games, 175-182, 2007.

[7] C. S. Lee,, M. H. Wang , Y. J. Chen, H. Hagras, Fuzzy Markup Language for Game of NoGo, IEEE International Conference on Granular Computing, 374-379, 2011.

[8] B. Bouzy, and T. Cazenave. Computer Go: An AI oriented survey. Artificial Intelligence 132, 39-103, 2001.

[9] Y.T. Lee, Pattern Matching in Go based on Patricia Tree. M.Sc. Department of Computer Science and Information Engineering. National Taiwan University, Taiwan, 2004. (in Chinese).

[10] S. Gelly, Yizao Wang, Exploration exploitation in Go: UCT for Monte-Carlo Go, NIPS-2006, 225-236, 2006.

[11] P. Auer, N. Cesa-Bianchi, and P. Fischer, Finite-time analysis of the multiarmed bandit problem, Machine Learning, vol. 47, no. 2/3, 235–256, 2002.

[12] Jean Méhat,Tristan Cazenave, Combining UCT and Nested Monte Carlo Search for Single-Player General Game Playing, IEEE Trans. on Computational Intelligence and AI in Games, Vol. 2, No. 4, 271-276, 2010.