The Research of Pattern Symmetry Problem in Learning in Computer Go

Wei Xin[1], Sun Yinglong[1], Yang Hui[1], Wang Jiao[1](Corresponding Author)

1. College of Information Science and Engineering Northeastern University, Shenyang, 110004
E-mail: wangjiao@ise.neu.edu.cn

Abstract: Pattern is a very effective approach to improve the Monte-Carlo tree search. Due to the limitation of affordable memory space, the two typical pattern sizes are 3*3 or 4*4. Pattern libraries may be constructed by hand-craft or machine learning, which are all suffered from pattern symmetry problem. This paper elaborates and classifies the pattern symmetry problem in 3*3-pattern and 4*4-pattern, and introduces the solution for solving it in learning procedure. The experimental results show that the solution is effective and the learning results are improved through solving pattern symmetry problem.

Key Words: Pattern, Monte-Carlo Go, UCT, Learning.

1 INTRODUCTION

Go is an ancient game which has a long history about 2000 years and now the game still attracts many people all over the world. As Artificial Intelligence began to appear, the computer games have been studied as relevant application fields. After the great successes are achieved in Chess and many games, Go is the new challenging topic in field of computer games [1].

Recent years, the old fashioned static methods are replaced by some advanced theories in computer Go, such as Monte-Carlo Tree search (MCTS) [2]. The MCTS improves the quality and accuracy of the global tree search by focusing the search in the most promising regions of the search space. The MCTS can be divided into four stages. The progress can be described as Fig. 1.

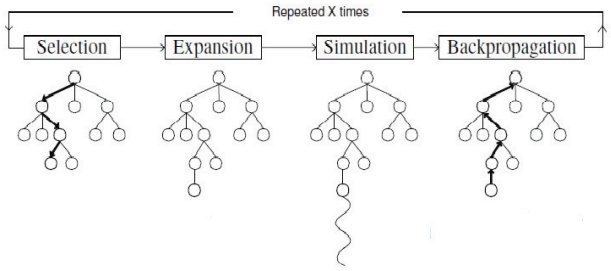

Fig. 1 The Monte-Carlo Tree Search.

From the picture above it can be deduced that through these search progress we can get best node that presents the best move. The progress is like that: Firstly the selection stage is to apply the selection strategy recursively until an unknown position is found. Secondly expand the node and simulate the game from the node's child. At last in the backpropagation the tree is updated by the results of this simulation game. Among many kinds of MCTS, the Upper Confidence Trees (UCT) are the most straightforward choices [3]. Many Go programs such as Mogo, Fuego which are based on the UCT have achieved great success.

Nowadays, the enhancement research which lead to the biggest performance gain in MCTS implementation mainly focus on the simulation strategy and the selection strategy. In selection strategy research field, some heuristic and pruning algorithms are put forward. And in the simulation strategy field, pattern is a well-known technique in computer Go for a long time [4]. It is applied by many famous Go programs such as GNU Go with the handcrafted pattern database for move selection. Many programs based on MCTS also used pattern to improve the quality of random simulation games, such as MoGo and FueGo. Now the method of the pattern proves to contribute a lot to improving MCTS.

This paper is organized as follows. Section 2 introduces 3*3-pattern and 4*4-pattern. Section 3 analyzes pattern symmetry problem. Furthermore, solving pattern symmetry problem in learning is described in Section 4. Finally, some experimental results are shown and the conclusion is presented in Section 5.

2 PATTERN IN MONTE-CARLO TREE SEARCH

The random simulation games can be improved by combining domain knowledge, and pattern is a very important algorithm to significantly improve the performance. Pattern has many formats of templates, and the template within a rectangular region is the most favorite in MCTS. This section describes two kinds of rectangular patterns.

2.1 3*3-Pattern

3*3-Pattern is widely used in move generator in random games, which can improve the quality of random games to some extent so as to enhance the overall performance of UCT search. In implementation, the 3*3-Pattern in Mogo is handcrafted [5], where as Fuego adopts some hard-coded disciplines [6]. Some examples of 3*3-Pattern are shown as follows.

Wang Jiao is the corresponding author. The material in this paper is based upon work supported by the NSFC-MSRA Joint Research Fund under Grant 60971057.

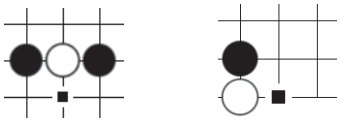

Fig. 2 Two examples of 3*3-Pattern. The left one is the pattern with the move in center of the board, and the move is on the board edge in the right one.

As can be seen in Fig. 1, 3*3-Pattern is very simple. However, the coverage space of 3*3-Pattern is limited thus the information provided is deficient, considering the huge board space. For example, some classic situations such as jump or diagonal move are inextricable by 3*3-Pattern due to space limitation.

2.2 4*4-Pattern

4*4-Pattern is put forward by our team in [7], which provides larger coverage space and more essential information than the original 3*3-Pattern. 4*4-Pattern covers area of 5*5 in comprising way because 5*5-Pattern is unaffordable for most computers.

Due to deficiency of central symmetry, it takes greater challenges to fulfill 4*4-Pattern comparing with 3*3-Pattern. The two key points are classification and multi-mapping.

(1) Multi-mapping

4*4-Pattern is not centrally symmetric, thus the traditional mapping method is not applicable in 4*4-Pattern. To solve this, a new method named multiple matching is put forward using multiple templates.

```
* * * *
* * * *
* ! * *
* * * *
```

Fig. 3 Match template of 4*4-Pattern.

Fig. 2 shows one of four templates in center patter. The procedure of multiple matching is explained below. Firstly, traverse all the eight points around the last move (the same as 3*3-Pattern), and several different templates is applied on every point for matching. "!" is the anchor point which comes from Go terminology, and "*" represents the point needed to be coded which may be taken up by black piece or white piece or just empty. Every template reflects to a specific coding order of the 15 stones in the 4*4 area except anchor point. Finally, the coded numeric value is used to query the corresponding pattern library.

(2) Classification

```
* * * *    * * * *    * * * *    * * * *
* * * *    * * * *    * ! * *    * * ! *
* ! * *    * * ! *    * * * *    * * * *
* * * *    * * * *    * * * *    * * * *
```

Fig. 4 Center-pattern templates.

```
* * * *    * * * *
* * * *    * * * *
```

```
* * * *    * * * *
* ! * *    * * ! *
```

Fig. 5 Edge-pattern templates.

```
* * * *
* * * *
* * * *
! * * *
```

Fig. 6 Corner-pattern template.

As seen in above figures, 4*4-Pattern is classified into three templates, named center-pattern, edge-pattern and corner pattern, differing in the position of the anchor point and all having several corresponding fixed templates. Edge-pattern has two templates and corner-pattern has only one template, while the center-pattern has four templates, which accounts for the most majority.

It should be noted that the border point or outboard point can be coded into "*", and leads to no need classification but takes more memory space. We prefer Classification because it is simple and needs no more computation.

2.3 Coding and Coding Sequence

For both 3*3-pattern and 4*4-pattern, the distribution of the stones in the rectangle area should be coded into a unique number. Each point has three possible statuses, i.e. empty point, black point and white point, so the maximal possibility is 3n where n is the point number counted. The coding representing the distribution of the stones in the pattern is calculated as formula 1, where p_i is the status of a specific point.

$$coding = \sum_{i=0}^{7} p_i * 3^i \qquad (1)$$

To simply the illustration, the intersections around the anchor point are indexed from 0 to 7 to identify the different templates, the number being strictly prescribed to avoid repetition. Fig. 4 shows an example of coding sequence for one of center-patterns.

```
11 12 13 14
 7  8  9 10
 4  !  5  6
 0  1  2  3
```

Fig. 7 Coding sequence example.

3 PATTERN SYMMETRY PROBLEM

Pattern symmetry means several patterns in different extern presentations are essential equivalent. It is a crucial characteristic for the pattern generation, for both the hand-craft and machine learning. This section describes and analyzes the pattern symmetry problem for patterns in different sizes.

3.1 The Symmetry Problem in 3*3-pattern

3*3-pattern is centrally symmetric, which has three possible symmetries, i.e. rotational symmetry, reflex symmetry and color symmetry.

- **Rotational symmetry**

Rotational symmetry means rotate the templates 90 degree each time, so totally four possibilities exist.

5 6 7	0 3 5	2 1 0	7 4 2
3 ? 4	1 ? 6	4 ? 3	6 ? 1
0 1 2	2 4 7	7 6 5	5 3 0

Fig. 8 Four rotational symmetric templates for center-pattern.

5 6 7	0 2 5	1 ? 0	7 4 1
2 3 4	? 3 6	4 3 2	6 3 ?
0 ? 1	1 4 7	7 6 5	5 2 0

Fig. 9 Four rotational symmetric templates for edge-pattern.

5 6 7	? 2 5	1 0 ?	7 4 10
2 3 4	0 3 6	4 3 2	6 3 0
? 0 1	1 4 7	7 6 5	5 2 ?

Fig. 10 Four rotational symmetric templates for corner-pattern.

- **Reflex symmetry**

There are two feasible symmetries, i.e. horizontal symmetry and vertical symmetry, so there are totally twelve templates considering rotational symmetry and reflex symmetry. It is an interesting fact, however, that four vertical symmetries are negligible because of repeated redundancy. So four templates remained according to reflex symmetry are as follows.

0 1 2	2 4 7	7 6 5	5 3 0
3 ? 4	1 ? 6	4 ? 3	6 ? 1
5 6 7	0 3 5	2 1 0	7 4 2

Fig. 11 Four relectional symmetric templates for center-pattern.

0 ? 1	1 4 7	7 6 5	5 2 0
2 3 4	? 3 6	4 3 2	6 3 ?
5 6 7	0 2 5	1 ? 0	7 4 1

Fig. 12 Four relectional symmetric templates for edge-pattern.

? 0 1	1 4 7	7 6 5	5 2 ?
2 3 4	0 3 6	4 3 2	6 3 0
5 6 7	? 2 5	1 0 ?	7 4 10

Fig. 13 Four relectional symmetric templates for edge-pattern.

- **Color symmetry**

Given a template for black playing side, color symmetry is represented by inverting the color of all stones when evaluating a white move. Therefore every template in rotational symmetry and reflex symmetry has two legal copies.

In conclusion, each template has totally 16 equivalent templates by rotational symmetry, reflex symmetry and color symmetry.

3.2 The Symmetry Analysis of 4*4 Pattern

Unlike 3*3-pattern, 4*4-pattern is not centrally symmetric, but still has symmetry characteristic based on the geometric center even if it is not a legal point on the board.

Fig. 14 Geometric center of the 4*4-pattern.

The symmetry problem of 4*4-pattern, according to the geometric center, is still discussed in rotational symmetry, reflex symmetry and color symmetry.

- **Rotational symmetry**

11 12 13 14	0 4 7 11	3 2 1 0	14 10 6 3
7 8 9 10	1 ? 8 12	6 5 ? 4	13 9 5 2
4 ? 5 6	2 5 9 13	10 9 8 7	12 8 ? 1
0 1 2 3	3 6 10 14	13 12 11	11 7 4 0

Fig. 15 Four rotational symmetric templates for one of the four center-patterns.

It can be found that the four rotation operations leads to four templates of center-pattern introduced in Section 2.2, that is why they coming from. Consequently, every 4*4-pattern has four equivalent patterns according to rotational symmetry.

11 12 13 14	0 3 7 11	2 1 ? 0	14 10 6 2
7 8 9 10	? 4 8 12	6 5 4 3	13 9 5 1
3 4 5 6	1 5 9 13	10 9 8 7	12 8 4 ?
0 ? 1 2	2 6 10 14	13 12 11	11 7 3 0

Fig. 16 Four rotational symmetric templates for edge-pattern A.

11 12 13 14	0 3 7 11	2 ? 1 0	14 10 6 2
7 8 9 10	1 4 8 12	6 5 4 3	13 9 5 ?
3 4 5 6	? 5 9 13	10 9 8 7	12 8 4 1
0 1 ? 2	2 6 10 14	13 12 11	11 7 3 0

Fig. 17 Four rotational symmetric templates for edge-pattern B.

As seen in Fig. 6 and Fig. 7, the two kinds of edga-pattern aslo has four equivalent patterns according to rotational symmetry. But unlike center-pattern, the locations in two edge-patterns are unique, so they can be recorded in single pattern library.

- **Reflex symmetry**

4*4-pattern are also horizontally symmetric and vertically symmetric. The four templates remain according to reflex symmetry as follows, because vertical symmetries are negligible just like 3*3-pattern.

0 1 2 3	3 6 10 14	14 13 12 11	11 7 4 0
4 ? 5 6		10 9 8 7	12 8 ? 1
7 8 9 10	2 5 9 13	6 5 ? 4	13 9 5 2
11 12 13 14	1 ? 8 12	3 2 1 0	14 10 6 3
	0 4 7 11		

Fig. 18 Four relectional symmetric templates for the first center-pattern.

0 ? 1 2	2 6 10 14	14 13 12 11	11 7 3 0
3 4 5 6		10 9 8 7	12 8 4 ?
7 8 9 10	1 5 9 13	6 5 4 3	13 9 5 1
11 12 13 14	? 4 8 12	2 1 ? 0	14 10 6 2
	0 3 7 11		

Fig. 19 Four relectional symmetric templates for the second edge-pattern.

0 1 ? 2	2 6 10 14	14 13 12 11	11 7 3 0
3 4 5 6		10 9 8 7	12 8 4 1
7 8 9 10	? 5 9 13	6 5 4 3	13 9 5 ?
11 12 13 14	1 4 8 12	2 ? 1 0	14 10 6 2
	0 3 7 11		

Fig. 20 Four relectional symmetric templates for edge-pattern.

- **Color symmetry**

The color symmetry of 4*4-pattern is same as in 3*3*-pattern. Every template in rotational symmetry and reflex symmetry has two legal copies.

Overall, 4*4-pattern has 16 equivalent templates. Although 3*3-pattern and 4*4-pattern have different sizes and symmetry characteristics, both have the exactly same equivalent templates number.

3.3 Analysis of Symmetry Problem

3*3-pattern and 4*4-pattern are rotationally symmetrical, reflex symmetrical and color symmetrical, both having sixteen equivalent patterns and differing in their symmetry center. Central symmetry in 3*3-pattern is more intuitive, nevertheless the symmetry according to geometric center in 4*4-pattern is obscure.

The color symmetry wastes memory beacause every pattern has two copies. This can be solved through making restriction on playing side, in which the stone is not black or white, but the same as playing side or not.

The other two symmetries have no impact on pattern usage, but may influence the learning procedure a lot. Without considering symmetry problem, the statistical does not reflect the priorities of patterns because they have several substantially equivalent patterns.

4 SOLVE PATTERN SYMMERY PRBLEM IN LEARNING

In practice, the pattern library is generated by machine learning and the results may be affected by pattern symmetry problem. This section discusses solving pattern symmetry problem in the learning procedure. Section 4.1 introduces the two most popular learning approaches. Solving symmetry problem in 3*3-pattern and 4*4-pattern are respectively discussed in Section 4.2 and Section 4.3.

4.1 Learning Methods

There are several popular methods in pattern learning. The two most common ones are Bayesian learning and ELO rating, which have been proved effectiveness in local rectangle pattern learning.

- **Bayesian learning**

Bayesian Learning is a classical theory of statistical learning, in which the post probability is calculated by priory probability and conditional probability. The post probability is used for classification instead of the prior probability, because it has more information reflecting the uncertainty of assessing an observation. An effective offline Bayesian learning model on pattern learning is put forward by reading every position in professional game records.

$$P_{posterior} = play\_time / match\_time \tag{1}$$

In the formula, the play_time stands for the times a certain pattern being played when the pattern appears. The match_time stands for a certain pattern appears. In a static position, many valid patterns are available but only one of them can be executed. So the match pattern of all the patterns increase by 1, and the play_time of the played pattern increase by 1, which means a move matches a specific pattern.

- **ELO rating**

ELO rating system is a good rating system used for rating the candidates. To introduce the ELO rating system better, introducing the Bradley Terry model is essential. The Bradley Terry model is used for computing the individual's strength based on the result of each two of the many participants .For a situation of m participants, the Bradley Terry model can be expressed as following.

$$P(i_{win}) = \frac{\gamma_i}{\sum_{j=1}^{m} \gamma_j} \tag{2}$$

In the formula, γ_i is a positive number stands for the strength of individual i, which need to be estimated in this model. The ELO rating of individual i is $r_i = 400\log_{10}(\gamma_i)$. Usually, the exact strength γ_i of the candidates are unknown and have to be estimated from the individuals' former performance.

The strength of each candidate γ_i can be estimated by iteratively updating the formula below, after some necessary derivation process:

$$\gamma_i \longleftarrow \frac{W_i}{\sum_{j=1}^{N} \frac{C_{ij}}{E_j}} \tag{3}$$

If all the parameters are initialized to 1, and each competition has the same number of participants, the first iteration computes the winning frequency of each individual. Therefore, in some way, this formula provides a Bayesian justification of frequency-based pattern evaluation. Running more iteration improves parameter further. The C_{ij} indicates the strength of i's teammates during competition j, and E_i is the total strength of all the participants.

4.2 Combining the Learning Results

As described in Section 4.1, the pattern libraries learned by the Bayesian learning are binary, while ELO rating has gamma values. On the effect of pattern symmetry problem, the learning results have many equivalent elements. In other words, some elements having different coding are substantially the same. We do not try to identify the pattern symmetry in reading professional records, because it is very time-consuming to calculate all the symmetric coding for each move. However, the additional processing on learning results to eliminate the effect is more, named combining. The processing is executed on temporal learning results after reading all the professional records, i.e. retrieving the play_time and match_time in Bayesian learning, or getting all gamma values in ELO rating.

For each pattern in learned pattern libraries, its symmetrical patterns should all be found and at most eight coding are retrieved. All the values in those elements should be

summed and then the sum is used to update the corresponding elements. This processing should be executed on every element, so the traverse the entire libraries are in demand. There are two important issues require attention in this procedure. Firstly, the repeated computations should be avoided, which can be settled by setting an identifier indicating whether the element has been summed. Secondly, some patterns are self rotationally symmetrical or reflex symmetrical. Therefore, the coding of the symmetrical pattern may be equal to the original one, and these elements should be removed in combining.

In traversing the temporal learning results, all the symmetrical codes should be calculated based on the current coding. For both the 3*3-pattern and 4*4-pattern, the current coding has to be factorized to get all the particular information of the surrounding stones. This is the inverse operation of coding.

```
//n is 8 for 3*3, 15 for 4*4
For  i=0…n
    stone[i] = 0;
i=0
Do stone[i] = code mod 3
       code = code / 3;
       i++;
While i<n
```

Algorithm 1: Pseudo-code of factorization.0018

After factorization, the elements in stone[n] are recombined to generate the coding of symmetrical patterns. As discussed in Section 3, the recombination is not a linear or linear operation and the array for recombination is required.

- **Recombination Table for 3*3-pattern**

```
RecombinationTable[8][8]={
            {0,1,2,3,4,5,6,7},
            {2,4,7,1,6,0,3,5},
            {7,6,5,4,3,2,1,0},
            {5,3,0,6,1,7,4,2},
            {5,6,7,3,4,0,1,2},
            {0,3,5,1,6,2,4,7},
            {7,6,5,4,3,2,1,0},
            {7,4,2,6,1,5,3,0}
}
```

The table is used to calculate the sum of the symmetrical codes as shown as following.

```
Iterate the pattern library to get each pattern Pattern[i]
  begin
    //The element is Pattern[i], i is the code
    if (Pattern[i].isReviewed==false)
        continue;
    //Factorize the code then get the distribution
    factorize (stone[8], i);
    //Get the sum of symmetrical patterns
    for j=0 to 6
        begin
            n = 1;
            for k=0 to 7
                begin
            code+=stone[RecombinationTable [j][k]] * n;
                n *=3;
                end
            codes[j] = code;
        end
    //Iterate codes and eliminate equivalent elements
    removeCoincide(code[j]);
    sum = i;
    for j=0 to 6
        begin
            //valid data
            if (codes[j] != INVALID)
                sum += Pattern[codes[j]].val;
        end
    //update sum value to the symmetrical patterns, and
  //set each review signal
    update(code[j], sum)
  end
```

Algorithm 2: Pseudo-code of combining 3*3-pattern.

The combining procedure of 3*3-pattern learning results as described in Algorithm 2. All the symmetrical patterns are updated with the same value which is the sum of them. Note that the algorithm is based on ELO rating learning, some modification applied on sum calculation would make it adapt the Bayesian learning.

- **Recombination Table for 4*4-pattern**

```
Center_RecombinationTable[4][8][15]={
        {
            {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14},
            {0,3,5,1,6,2,4,7,14,13,12,11,10,9,8},
            {5,3,0,6,1,7,4,2,14,13,12,11,10,9,8},
            {2,1,0,4,3,7,6,5,8,9,10,11,12,13,14},
            {2,4,7,1,6,0,3,5,14,13,12,11,10,9,8},
            {5,6,7,3,4,0,1,2,8,9,10,11,12,13,14},
            {7,6,5,4,3,2,1,0,8,9,10,11,12,13,14},
            {7,4,2,6,1,5,3,0,14,13,12,11,10,9,8}
        },
        ……
    }
```

Similar to combing procedure in 3*3-pattern using RecombinationTable, 4*4-pattern library uses Center_RecombinationTable for combining. It is no need to apply the combining on all the four center-libraries, just on serial one using the provided table is enough.

5 EXPERIMENTS AND CONCLUSION

To explain the potential affect of pattern symmetry problem in center-pattern libraries in machine learning, two experiments are conducted based on the 100,000 professional game records we collected. For simplification, Bayesian learning is adopted as the learning model in the experiments, in which the minimal value of use_time is 3, the minimal value of match_time is 5, while the threshold of post probability is 5%.

Firstly, an experiment is designed to reveal the effectiveness of the recombination by solving the pattern symmetry problem.

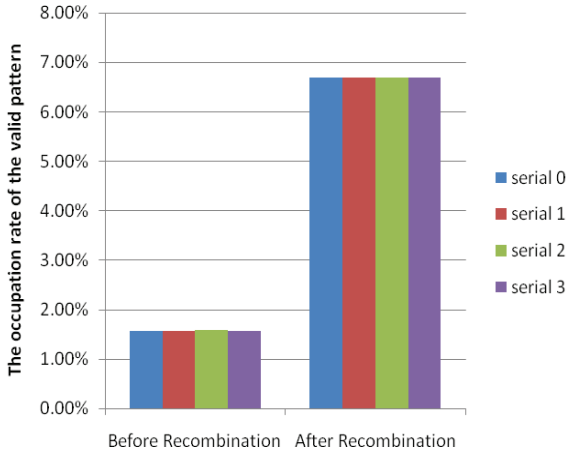

Fig. 21 The.result of learning 100000 records.

As shown in Fig. 21, the occupancy rate of valid pattern in center-pattern libraries has almost reach a three times increase by preceding the pattern symmetry process. The original occupancy rate is about 1.77%, whereas reaches 6.80% after recombination.

In order to show the distribution of learned results, the second experiment is designed.

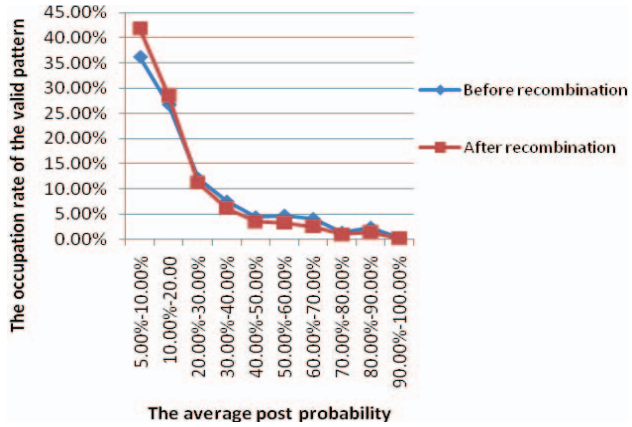

Fig. 22 The distribution of valid patterns in center-pattern libraries.

As seen in Fig. 22, the distribution of post probability is not so far different after recombination. The most notable difference in distribution, 5.6%, is present when the post probability is bigger than 5% and fewer than 10%.

In summary, this paper discusses pattern symmetry problem, furthermore the combination method to get rid of its negative influence in machine learning is put forward. The experimental results show that the solution is reasonable and effective, and the quality of pattern libraries generated by machine learning are much improved.

REFERENCES

[1] Stern, D., Graepel, T., and MacKay, D. Modelling Uncertainty in The Game of Go. Advances in Neural Information Processing Systems pp.33-40. (2004)

[2] B. Bouzy and T. Cazenave. Computer go: An AI oriented survey. Artificial Intelligence, 132(1), pp.39–103. (2001)

[3] Gelly, S.,Wang, Y. Exploration exploitation in go: UCT for Monte-Carlo go. On-line trading of Exploration and Exploitation Workshop. (2006)

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

[5] Gelly, S. and Silver, D. Combining Offline and Online Knowledge in UCT. In ICML'07: Proceedings of the 24th International Conference on Machine Learning, pp. 273-280. Association for Computing Machinery. (2007)

[6] Fuego Developer's Documentation: http://www.cs.ualberta.ca/~games/go/fuego/fuegodoc/

[7] Wang Jiao. 4*4-Pattern and Bayesian Learning in Monte-Carlo Go. Advances in Computer Games 13 Conference. (2011)

[8] Bouzy B, Chaslot G. Bayesian generation and integration of k-nearest-neighbor patterns for 19×19 Go. Computational Intelligence in Games, pp.176-181. (2005)

[9] D. Stern, R. Herbrich, and T. Graepel. Bayesian Pattern Ranking for Move Prediction in the Game of Go. the 23rd International Conference on Machine Learning, pp.873–880. (2006)

[10] Rémi Coulom. Computing Elo ratings of move patterns in the game of Go. Computer Games Workshop 2007. (2007)