4*4-Pattern and Bayesian Learning in Monte-Carlo Go*

Jiao Wang, Shiyuan Li, Jitong Chen, Xin Wei, Huizhan Lv, and Xinhe Xu

College of Information Science and Engineering Northeastern University
wangjiao@ise.neu.edu.cn

Abstract. The paper proposes a new model of pattern, namely the 4*4-Pattern, to improve MCTS (Monte-Carlo Tree Search) in computer Go. A 4*4-Pattern provides a larger coverage space and more essential information than the original 3*3-Pattern. Nevertheless the latter is currently widely used. Due to the lack of a central symmetry, it takes greater challenges to apply a 4*4-Pattern compared to a 3*3-Pattern. Many details of a 4*4-Pattern implementation are presented, including classification, multiple matching, coding sequences, and fast lookup. Additionally, Bayesian 4*4-Pattern learning is introduced, and 4*4-Pattern libraries are automatically generated from a vast amount of professional game records. The results of our experiments show that the use of 4*4-Patterns can improve MCTS in 19*19 Go to some extent, in particular when supported by 4*4-Pattern libraries generated by Bayesian learning.

1 Introduction

Go is an ancient board game for two players; it originated in China over 2000 years ago. The game still enjoys a great popularity all over the world [11]. Go has long been considered as the most difficult challenge in the field of Artificial Intelligence and is considerably more difficult than Chess [2]. Given the abundance of problems, and the diversity of possible solutions, computer Go is an attractive research domain for Artificial Intelligence.

Computer Go began in the 1960s with the prevailing static method preferred during the early days. This method chooses a handful of appropriate moves combined them with fast localized tactical searches, see, e.g., GNU GO [8]. Recently, some advanced theories led to a breakthrough performance in computer Go [6], e.g., by Monte-Carlo Tree Search and the Upper Confidence bound for Trees (UCT). At present, the best Go programs running on a cluster are ranked as 2 dan-3kyu.

Currently, the research on enhancements of the MCTS implementation mainly focuses on three key areas, i.e., tree search, random simulation games, and machine learning [12]. Some heuristic algorithms and pruning algorithms, as well as the domain knowledge enhancement methods are described in [4]. The formulation and the use of a pattern is a well-known technique in computer Go [14]. An example is the program GNU GO with its handcrafted pattern database for move selection. Patterns

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

H.J. van den Herik and A. Plaat (Eds.): ACG 2011, LNCS 7168, pp. 108–120, 2012.

are also used in MCTS programs to improve the quality of the random simulation games [5], e.g., in MoGo and Fuego. The typical patterns applied in MCTS are handcrafted 3*3-Patterns with many limitations.

A novel 4*4-Pattern model is proposed in this paper. It can be easily implemented in random simulation games and generated by Bayesian learning from professional game records. Experimental results show that the 4*4-Pattern is much better than 3*3-Pattern. We consider it an improvement of MCTS.

The paper is organized as follows. Section 2 analyzes the possibility of 4*4-Patterns and then introduces its basic idea. Section 3 describes the necessary operations of the 4*4-Patterns. Additionally, offline 4*4-Pattern-learning based on the Bayesian method is introduced in Section 4. Section 5 shows some experimental results of the 4*4-Patterns. Finally, the conclusion is presented in Section 6.

2 Motivation

Below we describe the 3*3-Pattern background (2.1) and the possibility of 4*4-Patterns.

2.1 The 3*3-Pattern Background

The 3*3-Pattern is nowadays widely used in the move generator in random games. They can improve the quality of random games to some extent so as to enhance the overall performance of the UCT search. In their implementation, the 3*3-Pattern in MoGo is handcrafted [5], whereas Fuego adopts some hard-coded disciplines [3]. Some examples of 3*3-Patterns are shown below.

Fig. 1. Two examples of a 3*3-Pattern. The left one is the pattern with the move in center of the board. In the right one, the move is on the board edge.

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

2.2 Possibility of 4*4-Patterns

We discuss two items in particular: (1) memory limitations and (2) multiple matching.

- **Memory Limitations**

The major limitation of a 3*3-Pattern and the central symmetry characteristic would suggest expanding the area to a 5*5-Pattern. However, the coverage space of a

5*5-Pattern contains in total 24 points except the central point. Each point has three possible status, i.e., empty point, black point, and white point, so a 5*5-Pattern requires at least 3^{24} bits memory space (approximately 33G byte) to store all the information. The amount of memory demand is not affordable for common computers, so the external disk memory has to be used comparable to an endgame database in Chess [13]. Nevertheless, it is not a preferable way because the high frequency reading may immensely reduce the efficiency of random games, which are executed thousands of times in UCT search [9]. So, we may conclude that a 5*5-Pattern is not applicable on personal computers at present.

Considering the coverage-space defect of a 3*3-Pattern and the memory limitations of the 5*5-Pattern, this paper proposes a new 4*4-Pattern model as a compromising solution. The storage and operations of a 4*4-Pattern is quite special compared to the 3*3-Pattern and the 5*5-Pattern. Once these crucial problems are solved, a 4*4-Pattern can be a desirable improvement in UCT search, which provides a larger area than a 3*3-Pattern and costs less memory than a 5*5-Pattern. In implementations, a single 4*4-Pattern library takes up approximately 14M byte memory, which is acceptable for most common computers.

- **Multiple Matching**

A 4*4-Pattern is not centrally symmetric, thus the traditional mapping method is not applicable for a 4*4-Pattern. To overcome this obstacle, a new method named multiple matching is proposed using multiple templates.

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

Fig. 2. Match template of 4*4-Pattern

The procedure of multiple matching is explained below. First, traverse all the eight points around the last move (the same as in the 3*3-Pattern procedure), and then apply several different templates on every point for matching. Here, the point is named anchor point, which comes from the Go terminology. There are three categories in total, i.e., center pattern, edge pattern, and corner pattern. All of them have several corresponding fixed templates. Every template reflects to a specific coding order of the 15 stones in the 4*4 area except the anchor point. Third, the coded numeric value is used to query the corresponding pattern library. Fig. 2 shows one of four templates with a center pattern. The "!" means the anchor point, and "*" is the point needed to be coded which may be empty or occupied by a black piece or a white piece.

Compared to a 5*5-Pattern, the coverage space of a single 4*4-Pattern is smaller, exactly 9 points less, and thus carries less information. But the multiple matching with several templates would compensate it to a large extent. For instance, all the points of the 5*5-area around the anchor point are taken into account after four templates have matched with a center pattern. Although a 4*4-Pattern cannot fully achieve the effect

of a 5*5-Pattern, the achievement is still impressive with much lower cost. From another point of view, each pattern is an improvement in the move selection in UCT search even if it is not a perfect one. We remark that even a 5*5-Pattern may suffer from exceptional peripheral information and lead to a wrong decision. But in most cases, the information provided by a pattern is correct and meaningful.

3 Operations of a 4*4 Pattern

This section discusses the required operations of a 4*4-Pattern. In 3.1 we introduce the three categories of a 4*4-Pattern based on the position of the anchor point. In 3.2 we briefly describe the compression to be applied. In 3.3 we discuss the storage and data structure of a 4*4-Pattern library. Specific coding sequences of templates are introduced in 3.4. In 3.5 we provide pseudo codes of crucial operations.

3.1 Classification of the 4*4 Patterns

According to different positions of the anchor point, a 4*4-Pattern can be categorized into three types, i.e., center-pattern, edge-pattern, and corner-pattern. The corner-pattern deals with situations where the anchor point is one of the four corner points on the board. The situation where the anchor point is on the edge points, but not in the corner points, is dealt with by the edge-patterns. The center-pattern deals with all the remaining situations, which are the majority in all situations.

Fig. 3. Center-pattern templates

Fig. 4. Edge-pattern templates

Fig. 5. Corner-pattern template

The meaning of the symbols is explained in 2.2. As shown in the Figs 3 to 5, the center-pattern has four templates, whereas the edge-pattern has two and the corner-pattern has only one template.

3.2 Compression

For the edge-pattern and the corner-pattern, the templates are the result after compression. In fact, each of the templates of the edge-pattern includes four situations, i.e., the top edge, the bottom edge, the left-most edge, and the right-most edge. So, there are totally eight templates for the edge-pattern, and it is interesting that some templates are essentially equivalent given some tricks applied on the coding sequence. Fig. 6 shows a compression example of the edge-pattern.

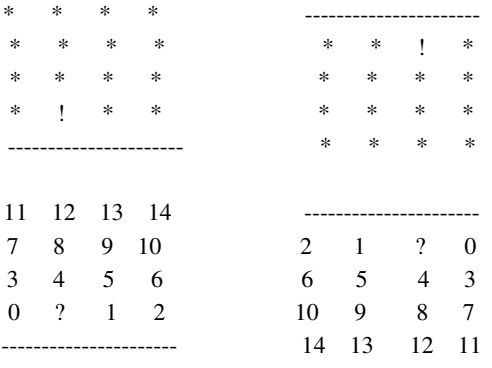

Fig. 6. Example of edge-pattern compression

In the figure, "--" indicates the boundary of the board, and the two templates on the top are essentially equivalent if the viewing angle turns 180 degree. This can be accomplished by imposing restrictions on the coding sequences. See the following two examples. It is possible to compress all the three 4*4-Pattern types, but in the usual implementations, this is not applied in the center-pattern considering the peripheral disturbance nearby the boundary.

A second method of compression is the color-based method. In a 4*4-Pattern matching, every piece is either white or black; so, all the patterns have two copies, and therefore the current playing side is taken into consideration. The pieces are treated as having the same color as the playing side, or just the contrary. So, we do not use anymore Black or White. Thus, a saving of a halve is achieved by the compression ratio method.

3.3 4*4-Pattern Library

As discussed in 3.1, there are four templates in the center-pattern, two in edge-pattern, and one in the corner-pattern, totally seven. Every template has the same memory occupancy. Three two-dimensional arrays are used for storing the templates, as represented below.

$$\begin{aligned} &bool\ CenterTable[4][14348907] \\ &bool\ EdgeTable[2][14348907] \\ &bool\ CornerTable[1][14348907] \end{aligned} \tag{1}$$

In these arrays, the number 14348907 (3^{15}) represents the maximum possible coding value of 15 points. The first dimension of the array indicates the serial number of the templates, and the second dimension is the coding result of 15 points in the 4*4 area except for the anchor point. It needs to be mentioned that the CornerTable is a linear array in practice, represented as a double-dimension so as to keep the formats in accordancy. The type of data stored in these arrays is "bool", where true or false indicates whether the corresponding pattern can be chosen or not. The memory occupancy space is nearly 13.7M byte for every pattern library. So, in total 96M byte memory is required for all the pattern libraries. It is affordable for most contemporary computers.

3.4 Coding Sequence and Lookup Table

The query input of a 4*4-Pattern is composed of the piece distribution information of the 5*5-area on the board. For the sake of compression and distinction, strict regulations are made on the coding sequence of the points. The coding sequence rules of all conditions in the three categories are shown in Table 1.

Table 1. Coding Sequence of 4*4 Pattern

Center-Pattern	Serial	0	1	2	3
	Coding Sequence	8 9 10 11 5 6 7 12 3 ? 4 13 0 1 2 14	11 10 9 8 12 5 6 7 13 3 ? 4 14 0 1 2	5 6 7 14 3 ? 4 13 0 1 2 12 8 9 10 11	14 5 6 7 13 3 ? 4 12 0 1 2 11 10 9 8
Edge-Pattern	Serial	0	0	0	0
	Coding Sequence	11 12 13 14 7 8 9 10 3 4 5 6 0 ? 1 2	2 1 ? 0 6 5 4 3 10 9 8 7 14 13 12 11	0 3 7 11 ? 4 8 12 1 5 9 13 2 6 10 14	14 10 6 2 13 9 5 1 12 8 4 ? 11 7 3 0
Edge-Pattern	Serial	1	1	1	1
	Coding Sequence	11 12 13 14 7 8 9 10 3 4 5 6 0 1 ? 2	2 ? 1 0 6 5 4 3 10 9 8 7 14 13 12 11	0 3 7 11 1 4 8 12 ? 5 9 13 2 6 10 14	14 10 6 2 13 9 5 ? 12 8 4 1 11 7 3 0
Corner-Pattern	Serial	0	0	0	0
	Coding Sequence	11 12 13 14 7 8 9 10 3 4 5 6 ? 0 1 2	14 10 6 2 13 9 5 1 12 8 4 0 11 7 3 ?	? 3 7 11 0 4 8 12 1 5 9 13 2 6 10 14	2 1 0 ? 6 5 4 3 10 9 8 7 14 13 12 11

For the sake of saving repeated computation time in multiple matching, the coding sequences of four templates of the center-pattern are rather special. The code of the eight nearest neighbors around the anchor point is calculated only once and the result

is reused subsequently. The coding sequence may seem a little complex, but it is not hard to implement using preset tables.

3.5 Program Codes for Querying

Some pseudo codes of the key procedure in querying are based on the lookup tables given in this subsection. The input is one coordinate of the eight neighbors around the last move of the opponent. The binary output represents whether the point can be played or not.

Some program codes for a 4*4-Pattern querying.

```
bool Match44Any(SgPoint p)
{
  if (IsCenter(p) > 1)
     return MatchAny44Center(p);
  else if (IsEdge (p) > 1)
     return MatchAny44Edge(p);
  else
     return MatchAny44Corner(p);

  return false;
}

//Multiple matching procedure for center-pattern.
bool MatchAny44Center(const BOARD& bd, SgPoint p)
{
  //Caculate the common code of 8 neighbors.
  int cm = CodeOf8CommonNeighbors(m_bd, p);//common code
  /*Iterate 4 templates, return true if the matched pattern is
  favorable,  otherwise  false.  CodeOfRestNeighbors  is  to
  caculate the codes of the rest 7 neighbors.*/
  if (lookupCenterTable[0][p][0] != INVALID //Table is avaiable
  &&m_44Centertable[0][CodeOfRestNeighbors(m_bd,p,0)+cm]==true)
     return true;
  if (lookupCenterTable[1][p][0] != INVALID
  &&m_44Centertable[1][CodeOfRestNeighbors(m_bd,p,1)+cm]==true)
     return true;
  if (lookupCenterTable[2][p][0] != INVALID
  &&m_44Centertable[2][CodeOfRestNeighbors(m_bd,p,2)+cm]==true)
     return true;
  if (lookupCenterTable[3][p][0] != INVALID
  &&m_44Centertable[3][CodeOfRestNeighbors(m_bd,p,3)+cm]==true)
     return true;

  return false;
}
```

Not all the pseudo codes are shown, such as the functions for edge-pattern and corner-pattern. However, they are quite similar to the ones of the center-pattern, and can be easily implemented.

4 Bayesian Learning of 4*4-Pattern

This section introduces Bayesian learning on 4*4-Pattern, which is a kind of statistical learning. In 4.1 we briefly describe the Bayesian theory and the model designed. In 4.2 we introduce some improvements on the traditional learning process. Then, in 4.3 the learning results are analyzed.

4.1 Bayesian Pattern Learning Model

Bayesian statistics is a classic theory of statistical learning, in which the post probability is calculated from a Bayesian formula combined with the prior probability and conditional probability in discrete condition. The post probability is used for classification instead of the prior probability, because it has more information to reflect the uncertainty of assessing an observation. The Bayesian formula is well known [7]. Bayesian learning has already been adapted in computer Go in recent years. Bruno Bouzy uses Bayesian learning in K-Nearest-Neighbor patterns [1], while David Stern et al. predict the professional moves [10]. An effective offline Bayesian learning model on 4*4-Patterns is proposed according to successful research achievements, by reading every position in professional game records.

$$P_{posterior} = play\_time / match\_time \tag{2}$$

In the formula, play_time stands for the times a certain pattern is played, while match_time represents that pattern occurrence in time. In a static position, many valid patterns probably exist but only one pattern can be executed. So, the match_time of all the valid patterns increases by one, and play_time of the played patterns increases by one providing that the move matches a specific pattern. For a 4*4-Pattern, every point has to count for several templates when traversing all the points on the board.

4.2 The Improvement of Learning Procedure

The 4*4-Pattern can be automatically generated according to the work (see 4.1), but the learning results are not satisfying. More meaningful improvements should be introduced to make the results better. Below we discuss three suggestions: data preprocessing, adjusting the learning process, and filtering bad patterns.

- **Data Preprocessing**

The quality of the professional game records is vital for learning. Dirty data may originate from the unequal matches, or from a weak game procedure. Some restrictions should be imposed to guarantee the data quality. We mention three of them.

(1) Restriction on the players' level. The level of players can be found by analyzing the SGF files; only those game records are acceptable when the grading of the two players is beyond 6 Dan.

(2) Restriction on the game result gap. The professional game records are accepted only if the result of professional games not exceeds 30 moyo.
(3) Restriction on the winning side. Only the moves of the winning side are input into Bayesian learning program.

By the restrictions above, about 20% of the SGF game records are removed from the game records database and even more samples are eliminated from the sample set. At least, so, the quality of learning material is guaranteed.

- **Adjust the Learning Process**

For a single professional game record, all the moves are input and then executed in proper sequence. Not only should a rule judgment be made to guarantee the correctness, but also attention should be given to some special situation that has to be coped with. The typical one is the move taking pieces, and this situation should avoid pattern learning. Since the pattern is essentially tactical, and not meant for an attacking purpose. Threat or attack is already solved before querying the pattern library during the move generation in a random simulation.

- **Filter Bad Patterns**

There are still many unreasonable patterns available even after the two procedures above. Additional filtering procedures are necessary. Below we mention two of them.

(1) Eliminate the patterns with low post probability. Using post probability as the confidence level is the essence of Bayesian statistics. So, these patterns with low post probability are obviously unacceptable. Currently the minimum of post probability is 5%.
(2) Eliminate the patterns with low match_time or play_time. For example, some arbitrary moves from inspiration are executed once they appeared, so play_time and match_time are all equal to 1 and 100% post probability is obtained. Obviously, it is against the original thoughts of Bayesian statistics. Currently, the limitations of the total amount for both are not less than 10.

5 Experiments

The experiments are composed in two parts, i.e., (1) Bayesian 4*4-Pattern learning experiments and (2) the effectiveness experiments presented below.

5.1 Bayesian 4*4-Pattern Learning Experiments

Two experiments are designed to analyze the result of the Bayesian 4*4-Pattern learning. Over 100,000 professional games are collected for the experiments and the setting of the learning restriction was seen in 4.2. In the first experiment, the game records are input into the learning program one by one, and the statistics of the occupancy rate are kept. For a single pattern library, the occupancy rate is equal to the valid patterns number divide 3^15. The experimental results are shown in Fig. 7.

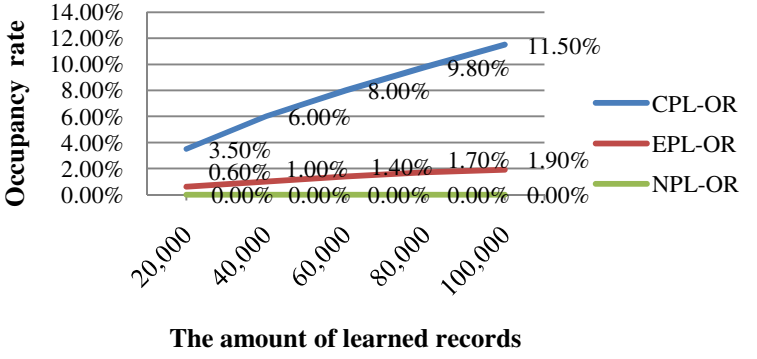

Fig. 7. The occupancy rate of each pattern library

Here CPL-OR means the center-pattern libraries' occupancy rate, while EPL-OR and NPL-OR is the occupancy rate of for edge-pattern libraries and corner-pattern library. It is should be noted that the four libraries of the center-patterns, and also the two libraries in the edge-patterns, share an almost identical distribution of occupancy rate. So, only the typical curves are provided. As can be seen in Figure 7, the occupancy rate goes up while the number of input game records increases, and the CPL-OR reaches 11.50% when all the game records are learned.

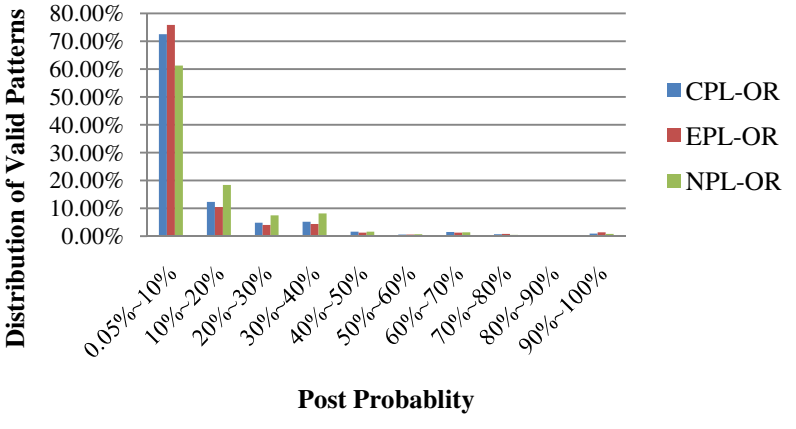

Fig. 8. The distribution of valid patterns according to the value of post probability

The results of the second experiment are shown in Fig. 8. It shows the relationship of post probability and valid patterns. The post probability of the majority of valid patterns is under 10%, and decreases while the percentage range of values rises. Only a few patterns are considered absolutely good, namely that the post probability is 100%. Similar to the former experiment, the sub-libraries also shares an almost identical distribution in the center-pattern and edge-pattern libraries.

5.2 Effectiveness Experiments

In this section, the effectiveness experiments are designed to prove the enhancement of a 4*4-Pattern, and also the correlation with the supporting libraries. The experiments are based on FUEGO (Version 1.1.2), one of the strongest Go programs, which is open-source under GNU license and equipped with a hard-coded 3*3-Pattern in their random simulation. To fit our experiments, the relevant codes of the 3*3-Patterns are removed and the code supporting the 4*4-Patterns is added in FUEGO, together with several sets of all the necessary 4*4-Pattern libraries from the Bayesian 4*4-Pattern learning. Of course, the amount of professional game records is different. To achieve the 4*4-Pattern libraries was easy. They were used in the program by reading external files at their initialization. The 4*4-Pattern program played 1000 games against the original FUEGO, with alternating the playing side. All games used Chinese scoring, 7.5 points Komi and 60 seconds for every move, running on the servers with 4-core Intel i5 2.8Ghz, 4G memory.

The experiments were applied on 19*19 Go and 9*9 Go. Although the 4*4-Pattern libraries were learned from records of 19*19 Go, they still could be used in 9*9 Go. The effectiveness of the experimental results is shown in Fig. 9.

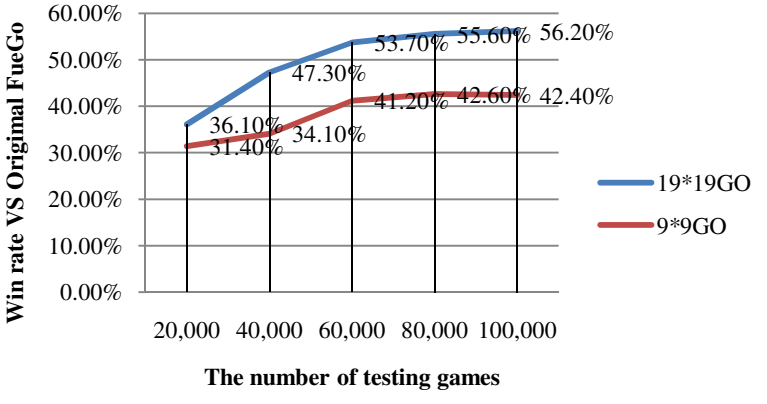

Fig. 9. Effectiveness experiments on 19*19 Go and 9*9 Go with different 4*4-Pattern libraries

As seen in Fig. 9, the playing strength boosts while the amount of learned records increases. However, if the game records for learning are insufficient, the playing strength is unsatisfying due to the low quality of the 4*4-Pattern libraries. The win rate is stable and exceeds 50% once the amount exceeds 60,000. For 9*9 Go, it is amazing that the win rate of the 4*4-Pattern program is always lower than the original FUEGO. There may be two possible reasons. First, the pattern libraries are generated from offline learning by 19*19 Go due to inadequate professional game records of 9*9. Many learned patterns may not be significant in 9*9 Go, because the patterns are more likely to reach the board border. Second, the 19*19 Go is more tactical than 9*9 Go, and a pattern move is mostly a tactical move. So, the effectiveness of the 4*4-Pattern decreases in sharp 9*9 Go games.

A second impact factor of effectiveness is the time limitation. The effectiveness of a 4*4-Pattern is more notable in longer games. The underlying reason is that a

;4*4-Pattern slows down the simulation games to some extent and negatively influences MCTS, but the 4*4-Pattern provides more significant information in a larger coverage space compared to the 3*3-Pattern. Therefore, the contribution of a 4*4-Pattern plays a bigger role than any negative effect. So, the overall effectiveness is positive in the longer games, assuming that the number of simulation games is sufficient.

6 Conclusion and Future Work

In this paper we proposed the 4*4-Pattern model. The details of the implementation are introduced, including design, classification, multiple matching, and coding sequences. In addition, Bayesian learning of 4*4-Patterns and some improvements on the basic method are described. The experimental results show that the 4*4-Pattern is better than the 3*3-Pattern in improving the MCTS in 19*19 Go to some extent, especially in the long games. There are several essential factors for the effectiveness of 4*4-Pattern, i.e., board space, the amount of learned records, time limitation, the effect on different pattern sizes, and the threshold of learning filtration. Some of them are not discussed in this paper, because of the paper length limitation.

Future work should focus on two issues. First, more effective 4*4-Pattern operations require intensive research. In fact, the ideal 4*4-Pattern is not realized unless all the points of 5*5-area around the last opponent move are traversed, and this inevitably costs more time. So, the fast computation and early refutation algorithm are in demand. Second, the learning methods on professional game records should be improved. Bayesian learning is fundamental in statistical learning and the implementation is too straightforward to obtain a convincing gamma value as happened in some top programs. Although the experimental results are satisfying, there is much room for improvement if more appropriate models are adopted.

References

1. Bouzy, B., Chaslot, G.: Bayesian generation and integration of k-nearest-neighbor patterns for 19×19 Go. Computational Intelligence in Games, 176–181 (2005)
2. Bouzy, B., Cazenave, T.: Computer go: An AI oriented survey. Artificial Intelligence 132(1), 39–103 (2001)
3. Fuego Developer's Documentation, http://www.cs.ualberta.ca/~games/go/fuego/fuegodoc/
4. Gelly, S., Silver, D.: Combining Offline and Online Knowledge in UCT. In: ICML 2007: Proceedings of the 24th International Conference on Machine Learning, pp. 273–280. Association for Computing Machinery (2007)
5. Gelly, S., et al.: Modification of UCT with Patterns in Monte-Carlo Go. Technical Report 6062. INRIA,France (2006)
6. Gelly, S., Wang, Y.: Exploration exploitation in go: UCT for Monte-Carlo go. In: On-line trading of Exploration and Exploitation Workshop (2006)
7. Minka, T.P.: A family of algorithms for approximate Bayesian inference. Massachusetts Institute of Technology (2001)
8. Müller, M.: Position Evaluation in Computer Go. ICGA Journal, pp. 219-228 (2002)

9. Silver, D., Tesauro, G.: Monte-Carlo Simulation Balancing. In: Proceedings of the 26th Annual International Conference on Machine Learning, Montreal, Quebec, Canada, pp. 954–852 (2009)
10. Stern, D., Herbrich, R., Graepel, T.: Bayesian Pattern Ranking for Move Prediction in the Game of Go. In: The 23rd International Conference on Machine Learning, pp.873–880 (2006)
11. Stern, D., Graepel, T., MacKay, D.: Modelling Uncertainty in The Game of Go. In: Advances in Neural Information Processing Systems, pp.33–40 (2004)
12. Wang, Y., Gelly, S.: Modifications of UCT and sequence-like simulations for Monte-Carlo Go. In: IEEE Symposium on Computational Intelligence and Games, pp. 175–182 (2007)
13. Wu, R., Beal, D.F.: A Memory Efficient Retrograde Algorithm and Its Application To Chess Endgames. In: More Games of No Chance, vol. 42. MSRI Publication (2002)
14. Zobrist: Feature. Extraction and Representation for Pattern Recognition and the Game of Go. PhD thesis, University of Wisconsin (1970)