The Design and Implementation of a Pattern-directed MCTS-based Computer Go Program -- WINGO

Rou-Yu Lai
Department of Information Management,
Chang Jung Christian University,
Tainan, Taiwan, R.O.C.
qwertyuiopyo@gmail.com

Hsin-Hung Chou
Department of Information Management,
Chang Jung Christian University,
Tainan, Taiwan, R.O.C.
chouhh@mail.cjcu.edu.tw

Shun-Chin Hsu
Department of Information Management,
Chang Jung Christian University,
Tainan, Taiwan, R.O.C.
schsu@mail.cjcu.edu.tw

Shi-Jim Yen
Department of Computer Science and Information Engineering, National Dong Hwa University,
Hualien, Taiwan, R.O.C.
sjyen@mail.ndhu.edu.tw

Abstract

In recent years, almost all the top programs in the computer Go tournaments are developed using Monte Carlo Tree Search (MCTS), which brings the top programs having strength to win against the top professional human players on the 9 × 9 board.

In this paper, we introduce a new computer Go program, WINGO. WINGO is a pattern-directed MCTS-based 9X9 computer GO program, written in C++. The pattern set of WINGO adopts the patterns of a traditional knowledge-based computer go program Dragon, including 3X3 and 5X5 patterns. Experiments show that it improves the strength of WINGO with the Dragon patterns adopted.

Keywords: Computer Go; Monte Carlo Tree Search; Patterns.

1. Introduction

While the first computer Go program proposed in the late 1960s, computer Go is always one of the tough challenges in the field of artificial intelligence. Comparing to the other popular board games, the search space for 19X19 Go is quite large, there are $3^{19\times19} \approx 10^{170}$ distinct board states to be considered. There has no simple and reasonable evaluation function been found for Go program so far. The direct way to encode the Go knowledge into a program is using the patterns. Patterns can be applied in all stages of a game, from opening to endgame. Almost all Go programs contain a set of patterns and a pattern matching function. In the early tournaments, the strength of a Go program was relied on the completeness of their pattern set. For example, Taiwanese program Dragon[4], developed by Professor Shun-Chin Hsu and his student Dong-Yue Liu of National Taiwan University, were remarkable for its well-defined pattern set. Dragon contains about five hundred patterns without fixed-size, which describe not only the small area patterns but also the higher-level descriptions of the game states.

In this paper, we introduce a new

computer Go program, WINGO. WINGO is a pattern-directed MCTS-based 9X9 computer GO program, written in C++. WINGO is constructed from an MCTS-based program Cjugo[5], which was developed by Professor Hsin-Hung Chou and ever participated the tournament of JAIST 2010 held in Kazanawa Japan. The pattern set of WINGO adopts the patterns of a traditional knowledge-based computer go program Dragon, including 3X3 and 5X5 patterns. Experiments show that it improves the strength of WINGO with the Dragon patterns adopted.

The rest of this paper is organized as follows: The second section introduces the Monte Carlo Tree Search briefly. The third section describes the pattern encoding scheme of WINGO. The fourth section shows the experimental results. At last, we give conclusion in fifth section.

2. Monte Carlo Tree Search

In recent years, almost all the top programs in the computer Go tournaments are developed in Monte Carlo Tree Search (MCTS)[3], which brings the top programs having strength to win against the top professional human players on the 9X9 board. Recently, the top programs have reached the strength about six dan amateur level on the 19X19 board.

MCTS is a kind of best-first search that tries to find the best move and to keep the balance between exploration and exploitation of all move. MCTS was firstly implemented in CRAZY STONE[3], the winner in the 9X9 Go tournament at the 2006 Computer Olympiad. Together with the emergence of UCT[6], the huge success of MCTS stimulated profound interest among Go programmers.

MCTS uses two main strategies: (1) A simulation strategy gives the explorations on the candidate moves played in the Monte-Carlo simulations. (2) A selection strategy, derived from the Multi-Armed Bandit problem, selects the best move using the results of previous explorations. The more number of simulations, the quality of the move-selection will be more accurate. It derives that the strength of the program would depend on the program parallelism and the power of the hardware. However, the parallelism and the power of the hardware have the limitation. We need the Go knowledge to help the simulation more accurately. In the other words, each move of the simulation is not totally random, but directed by the patterns we built in.

Many enhancements of MCTS have been proposed, such as Rapid Action Value Estimation (RAVE)[7,8] and progressive bias[1], to strengthen its effect. Most of comprehensive studies were also focused on the policy and better quality of the playout[1,2,9].

3. Pattern Encoding

The pattern set of WINGO adopts about 600 3X3 patterns and 280 5X5 patterns of Dragon. For each 3X3 pattern, we represent the content of a board point by 2 bits: EMPTY(00), BLACK(01), WHITE(10), and

BOARDER(11). We encode each 3X3 pattern into an integer by a sequence of 16 bits. The 3X3 pattern sequence is shown in Figure 1. Figure 2 shows a 3X3 pattern instance with code 33963.

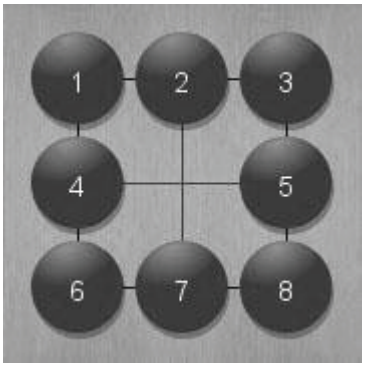

Fig. 1. 3X3 pattern sequence.

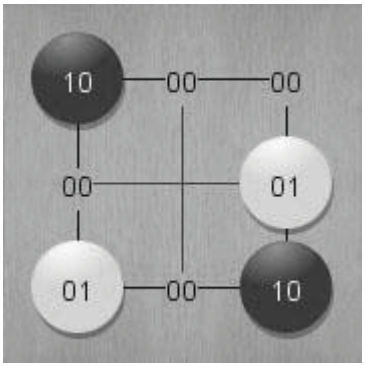

Fig. 2. 3X3 pattern instance.

We encode each 5X5 pattern into three integers each represents 8 points by a sequence of 16 bits. The 5X5 pattern sequence is shown in Figure 3. Figure 4 shows a 5X5 pattern instance with code (18496,0, 128).

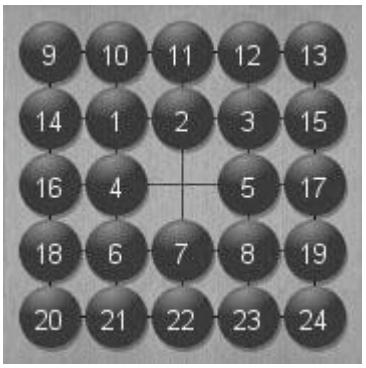

Fig. 3. 5X5 pattern sequence.

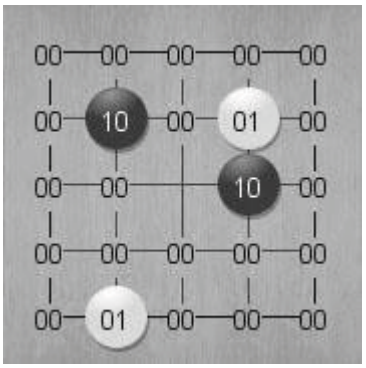

Fig. 4. 5X5 pattern instance.

4. Experiments

We have conducted two experiments to show that the pattern set improves the performance of our program. The first one is that the engine without patterns against GNU GO 3.8 with level 10 and Komi 7.5 through KGS website. The second one is that the engine with patterns against the same opponent. We ran WINGO and GNU GO on the computers with Intel Core Q8200, 4cores, 4G memory. The results are show in Table 1 as follows.

Table 1. Experiment results.

engine\win rate	Black	White	Average
WINGO without pattern	20/60 ≈ 33%	12/40 ≈ 30%	32/100 ≈ 32%
WINGO with pattern	20/42 ≈ 48%	36/62 ≈ 58%	56/102 ≈ 55%

5. Conclusion

Experiments show that it improves the strength of WINGO with the Dragon patterns adopted, increasing the win rate against GNU GO about 23%. In the future, we will try to combine the pattern weight defined in Dragon into our engine to make the simulation more accurate.

Reference

[1] Chaslot, G., Winands, M., Bouzy, B., Uiterwijk, J. W. H. M., and Herik, H. J. van den., *Progressive Strategies for Monte-Carlo Tree Search. Proceedings of the 10th Joint Conference on Information Sciences*, pp.655-661, Salt Lake City, USA, 2007.

[2] Chaslot, G., J-B. Hoock, J.-B, Perez, J., Rimmel, A., Teytaud, O. and Winands, M. Meta, *Monte-Carlo Tree Search for Automatic Opening Book Generation*. Proceedings of the IJCAI'09 Workshop on General Intelligence in Gmae Playing Agents, pp. 7-12, 2009.

[3] Coulom, R., *Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search.* Proceedings of the 5th International Conference on Computer and Games, Vol.4630 of Lecture Notes in Computer Science, pp.72-83, Springer, Turin, Italy, 2006.

[4] Dong-Yue Liu and Shun-Chin Hsu, *Design and Implementation of Computer Go Program*, National Taiwan University, Department of Computer Science & Information Engineering, 1989.

[5] F. Karger and M. Babar, *MYGOFRIEND wins Go 9x9 Tournament*, ICGA Journal, Vol. 33, No. 2, pp.169-179, 2010.

[6] Kocsis, L. and Szepesv'ari, C., *Bandit Based Monte-Carlo Planning.* In J. Furnkranz, T. Scheffer and M. Spiliopoulou (eds.), Machine Learning: ECML 2006, Lecture Notes in Artificial Intelligence 4212,pp.282-293, 2006.

[7] Gelly, S. and Silver, D., *Combining Online and Offline Knowledge in UCT*. Proceedings of the 24th International Conference on Machine Learning, pp. 273-280, Corvallis Oregon USA, 2007.

[8] Gelly, S. and Silver, D., *Monte-Carlo tree search and rapid action value estimation in computer Go.* Artificial Intelligence, Vol. 175, No. 11, 2011.

[9] Hendrik, B., *Adaptive Playout Policies for Monte-Carlo Go.* Master thesis, Institut für Kognitionswissenschaft, Universität Osnabruck, 2010.