2010 International Conference on Technologies and Applications of Artificial Intelligence

Monte Carlo Go using Previous Simulation Results

Takuma TOYODA
Graduate School of Engineering
Tokyo University of Agriculture and Technology
2-24-16 Naka-cho, Koganei-shi, Tokyo, 184-8588, Japan
50010646130@st.tuat.ac.jp

Yoshiyuki KOTANI
Institute of Engineering
Tokyo University of Agriculture and Technology
2-24-16 Naka-cho, Koganei-shi, Tokyo, 184-8588, Japan
kotani@cc.tuat.ac.jp

***Abstract*—The researches on Go using Monte Carlo method are treated as hot topics in these years. In particular, Monte Carlo Tree Search algorithm such as UCT made great contributions to the development of computer Go. When Monte Carlo method was used for Go, the previous simulation results were not usually stored. In this paper, we suggest a new idea of using previous simulation results (PSR heuristic) and try to improve the method. Experimental results showed that this heuristic is very effective in blitz and on larger Go board.**

***Keywords*-artificial intelligence; computer Go; heuristic; Monte Carlo; UCT;**

I. Introduction

Go is a perfect information game which is played all over the world. The object of the game is to surround a larger portion of the board than the opponent and there is few draws because of *Komi* (compensation points which compensate the second player for the first move advantage of his opponent).

In recent years, computer Go made much progress in using Monte Carlo method[1]. This method plays large number of random games from current position to the end and evaluates each move based on random game results. In particular, Monte Carlo Tree Search algorithm such as UCT is used by state of the art Go programs (Crazy Stone[2], MoGo[3], etc) and MoGo succeeded to defeat professional Go players on 9×9 board.

When Monte Carlo method was used for Go, the previous simulation results were not usually stored. In this paper, we suggest a new idea of using previous simulation results (PSR heuristic) and try to improve the method.

The rest of this paper is organized as follows. Section II describes Monte Carlo Go. In Section III, we explain how to use PSR heuristic for Monte Carlo Go. Section IV presents our experimental results and discusses the advantage of the heuristic. Section V presents our conclusions and describes future work.

II. Monte Carlo Go

This section gives a detailed description of Monte Carlo Go (crude Monte Carlo method, UCB1 algorithm, UCT algorithm).

A. Crude Monte Carlo Method

The simplest way to apply Monte Carlo method to computer Go is to run same number of random games (called playout) for each legal move and chooses a move which has the best winning percentage. One playout process is as follows.

1) Choose the move randomly among the legal moves.
2) Repeat 1) until both players pass.
3) Compute the score and return the result (win or lose).

However, if the moves were chosen completely at random, the game has no ending. Therefore, programs must recognize the moves filling own eyes as illegal moves. Additionally, it is shown that the program performance is improved by introducing heuristics so that the playout will more closely approximate a realistic game[4][5][6].

For example, if 100 playouts were run for each legal move, the move b is chosen (Figure 1).

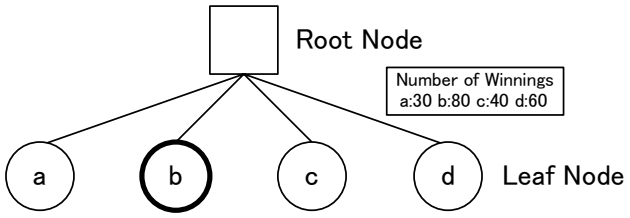

Figure 1. Example of crude Monte Carlo method

B. UCB1 Algorithm

Crude Monte Carlo method ran the same number of playouts for each legal move. This method is not so efficient, because there are limited in time to run playouts. So it is necessary to allow hopeful moves to run many playouts. Using UCB1[7] which is effective in n-armed bandit problem is one of the techniques for doing it. The value of UCB1 is defined by the following equation.

$$UCB1(i) = \bar{X}_i + C\sqrt{\frac{\log N}{n_i}} \quad (1)$$

$\bar{X}_i$ is a winning percentage of the move i with playouts. n_i is the number of playouts for i. N is the total number of

978-0-7695-4253-9/10 $26.00 © 2010 IEEE
DOI 10.1109/TAAI.2010.39

playouts of each n_i. The constant C controls the balance between exploration and exploitation (When C is big, it behaves like crude Monte Carlo Method). And default value is infinite.

There are several ways to decide a move. We used the move which has not maximum mean value $UCB(i)$ but maximum mean value $\bar{X}_i$. UCB1 algorithm is as follows.

1) Run playout for a move which has the maximum value of UCB1.
2) Update the value of UCB1 of each move.
3) Repeat 1)-2) within the time limit.
4) Choose a move which has the best winning percentage.

C. UCT Algorithm

UCT[8] is an extended algorithm of UCB1 for game tree search. It is applied in state of the art Go programs such as Crazy Stone and MoGo. The tree grows in an asymmetric manner based on simulations. So it can explore more deeply the good moves. There are various extension methods such as the way to limit node expansion to hopeful nodes in order to achieve effective evaluation. In this paper, we used simple UCT algorithm, so we did not use progressive unpruning[9] and first play urgency[10]. UCT algorithm is as follows.

1) Expand the root node.
2) Search for child node which has the maximum value of UCB1.
3) If a node has visited, expand the node. Otherwise run playout and update the value of UCB1 of each node.
4) Repeat 2)-3) within the time limit.
5) Choose a node which has the best winning percentage.

III. PSR Heuristic

This section describes PSR heuristic and explain how to use it for Monte Carlo Go. PSR heuristic is an idea of using previous simulation results. More precisely, it initializes the node based on the last search tree (Normally, default values of playout and winning are zero). Go has a property that evaluation of the moves does not change easily when own turn has came again compared with Chess and Othello. (Grandfather heuristic[11] also utilizes such property, but it is different from PSR heuristic. It initializes the node based on the grandfather node in UCT algorithm shown in Figure 3. So it is unable to use grandfather heuristic on depth=1 and 2 nodes.) The position of those games is changeable according to pieces-work or reverse of stones. Therefore, the last good moves are probably not good in the current position in those games but in Go it is not so.

For example, in opening game like Figure 2, candidates a, b, c, d is usually good moves. Those moves still remains high evaluation value when own turn has came again unless the opponent set the stone near the candidates or played some force moves. Therefore, previous simulation results can be used as default values in Monte Carlo Go and it can make the algorithms efficiency.

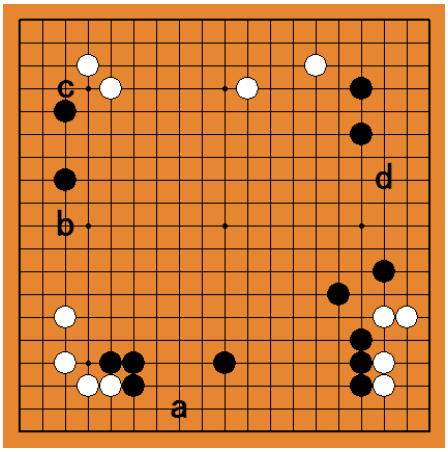

Figure 2. Example of opening game (white to move)

A. Apply to Crude Monte Carlo Method

PSR heuristic for crude Monte Carlo method is shown on Figure 4. For example, there were legal moves a, b, c, d in last own position. If the move b was chosen and opponent selected the move c, remaining legal moves are a and d. The move e is the new legal move because of *Ko* or capture moves. Then, the moves a and d can inherit previous simulation results. The move e is initialzed normally (The value of playout and winning is zero).

But if we use them directly for default value, it will be hard for the move which is undefeated in the simulations of current position to outweigh the move which has high default winning percentage. Therefore, we have to discount them. The default value of playout and winning is calculated from the following formula.

$$n_i = rn'_i \tag{2}$$

$$w_i = rw'_i \tag{3}$$

r ($0 \leq r \leq 1$) is the discount ratio. n_i is the number of playouts for i. n'_i is the number of last playouts for i. w_i is the number of winnings for i. w'_i is the number of last winning for i. After the default value is given, decide the move using crude Monte Carlo method.

B. Apply to UCB1 Algorithm

Using the heuristic for UCB1 algorithm is almost same as the case of crude Monte Carlo method. The default value of UCB1 is calculated based on eqs. (2) and (3).

C. Apply to UCT Algorithm

There are several possible methods to use previous simulation results for UCT algorithm such as the way to inherit last search tree in whole. In this paper, we outfit the default search tree with the depth=1 nodes of last search tree (except illegal moves). Because the move is decided depending on winning percentage of depth=1 nodes and it is easy to implement. These are summarized in Figure 5.

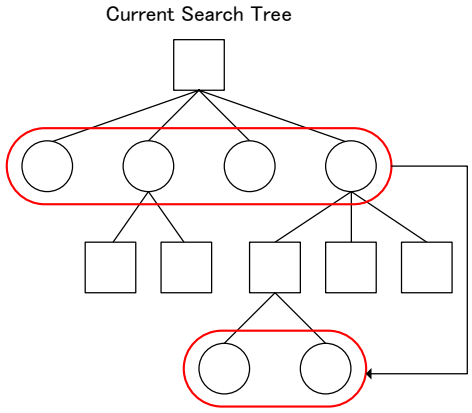

Figure 3. Apply grandfather heuristic to UCT algorithm

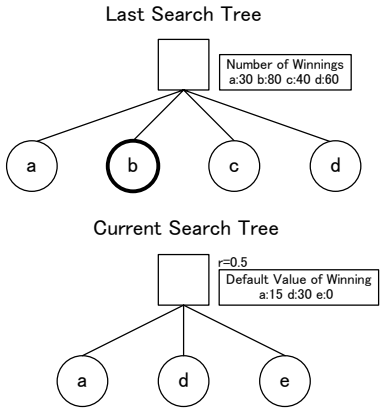

Figure 4. Apply PSR heuristic to crude Monte Carlo method

IV. Experimental Results

This section gives the results of experimental games between the program using PSR heuristic and the program without using (X_{PSR} and X_{std}). Experiment environment is as follows.

- *Komi* are set to 7.5 points on 9×9 and 19×19 board.
- The programs have 1 second per move (The programs can run about 20,000 or 3,000 playouts per move).
- The number of matches is 600 times (Playing half of the games as Black and the other half as White).

A. Crude Monte Carlo Method Results

Figure 6 and 7 shows the crude Monte Carlo method results. As Figure 6 shows, MC_{PSR} won MC_{std} significantly at $0.125\leq r\leq0.6$ on 9×9 board. Maximum winning rate is $62.5\pm3.9\%$ at r=0.25. MC_{PSR} has low winning rate at $0.8\leq r\leq1.0$. Probably this is because bad moves in current position which has high default winning percentage were probably chosen as described in Section III-A. As r is smaller, MC_{PSR} will be similar to MC_{std} and winning rate converges to 50%.

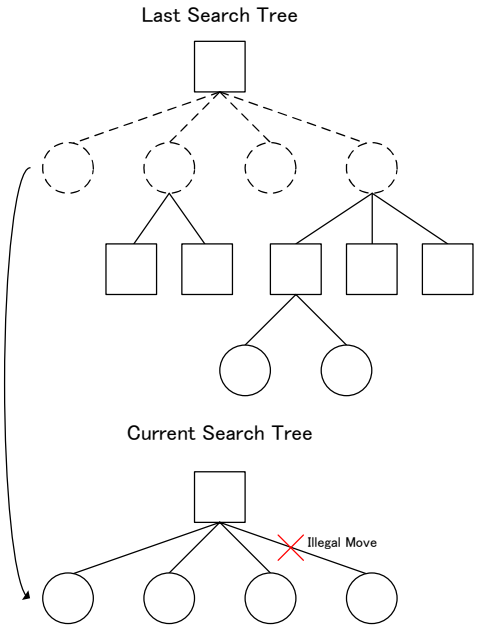

Figure 5. Apply PSR heuristic to UCT algorithm

As Figure 7 shows, amazingly, MC_{PSR} is overwhelming effective on 19×19 board. MC_{PSR} was defeated only once out of 600 matches at r=0.9. There are many candidates in 19×19 board compared with 9×9 board. Thus, PSR heuristic worked more effectivelly on 19×19 board than 9×9 board. As r is smaller, winning rate is expected to converges to 50% like 9×9 board results.

B. UCB1 Algorithm Results

Figure 8 and 9 shows the UCB1 algorithm results. As Figure 8 shows, UCB_{PSR} won UCB_{std} significantly at $0.125\leq r\leq0.7$ on 9×9 board. Maximum winning rate is $59.2\pm4.0\%$ at r=0.7. Winning rate of UCB_{PSR} is low compared with MC_{PSR} at r=0.9 or 1.0. This is because UCB1 algorithm runs many playouts for hopeful moves and few playouts for unhopeful moves (It is necessary to achieve more winnings to turn the game around).

As Figure 9 shows, UCB_{PSR} also has high winning rate on 19×19 board. Winning rate at r=1.0 is very low because of using total number of playouts to calculate the value of UCB1. Total playouts diverges to infinity. So the value of UCB1 will be odd.

C. UCT Algorithm Results

Figure 10 and 11 shows the UCT algorithm results. As Figure 10 shows, UCT_{PSR} won UCT_{std} significantly at $0.125\leq r\leq0.8$ on 9×9 board. Maximum winning rate is $69.8\pm3.9\%$ at r=0.7. There are certain winning rates at r=0.8 or 0.9 because UCT search tree grows to explore better moves based on simulations

As Figure 11 shows, UCT_{PSR} also has amazing winning rate on 19×19 board.

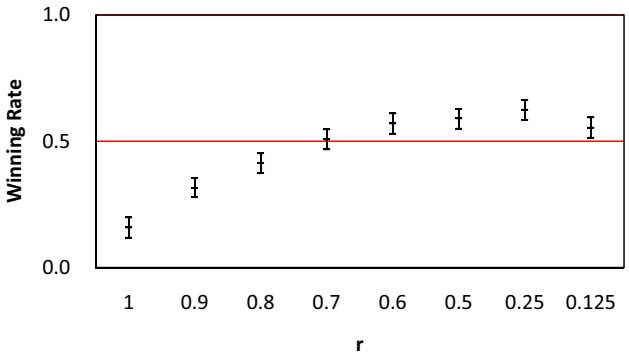

Figure 6. Winning rate of MC_{PSR} against MC_{std} on 9×9 board

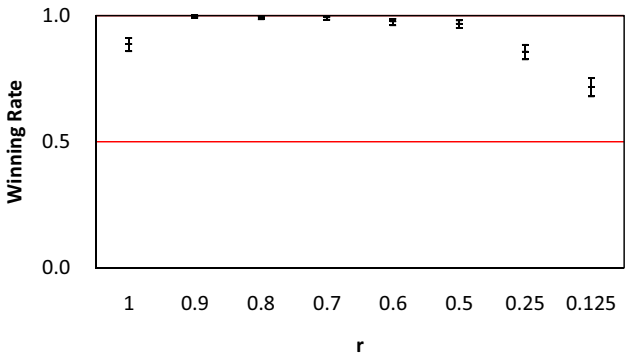

Figure 7. Winning rate of MC_{PSR} against MC_{std} on 19×19 board

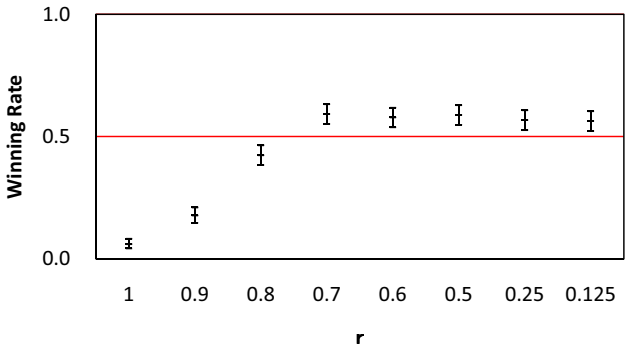

Figure 8. Winning rate of UCB_{PSR} against UCB_{std} on 9×9 board

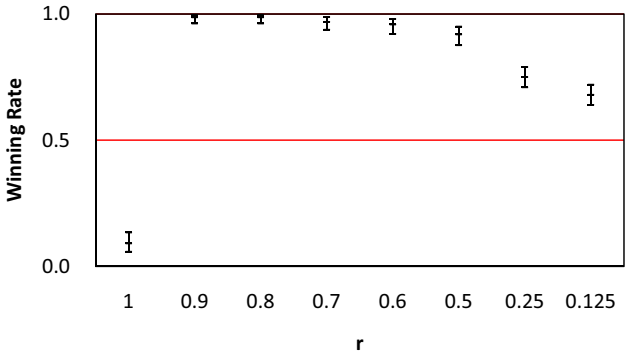

Figure 9. Winning rate of UCB_{PSR} against UCB_{std} on 19×19 board

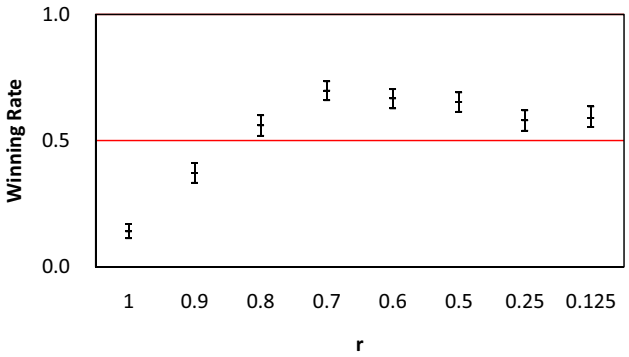

Figure 10. Winning rate of UCT_{PSR} against UCT_{std} on 9×9 board

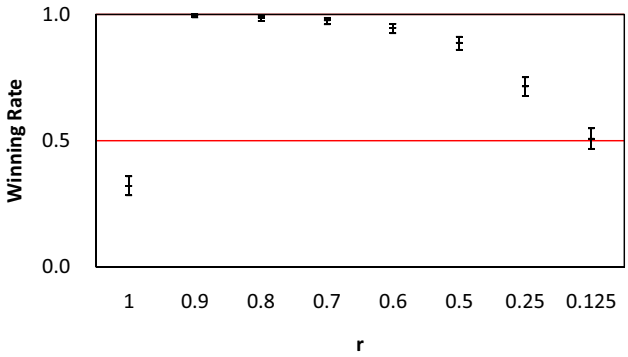

Figure 11. Winning rate of UCT_{PSR} against UCT_{std} on 19×19 board

V. Conclusions and Future Work

In this paper, we suggested a new idea of using previous simulation results in Monte Carlo Go. Experimental results showed the program using PSR heuristic won significantly against without using program in three algorithms:crude Monte Carlo method, UCB1 algorithm and UCT algorithm. Maximum winning rates on 9×9 board were MC_{PSR}:62.5±3.9%, UCB_{PSR}:59.2±4.0%, UCT_{PSR}:69.8±3.9%. On 19×19 board, amazingly, the programs using PSR heuristic were hardly defeated.

Therefore, the effectiveness of PSR heuristic in Monte Carlo Go was shown according to these results. In particular, the larger the board is, the more the availability of PSR heuristic increases. Moreover, in limited time like blitz, PSR heuristic is very effective because it needs few computational complexity.

An example of further work would be changing the method of using previous simulation results for UCT. Another possibility is to examine the performance of PSR heuristic when the program ran tuned-playout (adding domain-specific knowledge) not random-playout.

REFERENCES

[1] Bruno Bouzy and Bernard Helmstetter, "Monte Carlo Go Developments," Advances in Computer Games conference (ACG-10), pp.159-174, 2003.

[2] Rémi Coulom, "Computing Elo Ratings of Move Patterns in the Game of Go," In Proceeding of the 6th International Conference on Computers and Games, pp.113-124, 2007.

[3] Sylvain Gelly, Yizao Wang, Rémi Munos and Olivier Teytaud, "Modification of UCT with Patterns in Monte-Carlo Go," Technical Report No.6062, INRIA, 2006.

[4] Peter Drake and Steve Uurtamo, "Move Ordering VS Heavy Playouts: Where Should Heuristics be Applied in Monte Carlo Go?," In Proceedings of the 3rd North American Game-On Conference, 2007.

[5] Peter Drake and Steve Uurtamo, "Heuristics in Monte Carlo Go," In Proceedings of the 2007 International Conference on Artificial Intelligence, 2007.

[6] Tristan Cazenave, "Playing the Right Atari," International Computer Games Association Journal Vol.30 No.1, pp.35-42, 2007.

[7] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer, "Finite time analysis of the multiarmed bandit problem," Machine Learning, Vol.47, pp.235-256, 2002.

[8] Levente Kocsis and Csaba Szepesvári, "Bandit-based monte-carlo planning," In 15the European Conference on Machine Learning (ECML-15), pp.282-293, 2006.

[9] Guillaume Chaslot, Mark Winands, H. Jaap van den Herik, Jos Uiterwijk and Bruno Bouzy, "Progressive strategies for Monte-Carlo tree search," In Joint Conference on Information Sciences, 2007.

[10] Sylvain Gelly and Yizao Wang, "Exploration exploitation in Go: UCT for Monte-Carlo Go," Twentieth Annual Conference on Neural Information Processing Systems (NIPS2006), 2006.

[11] Sylvain Gelly and David Silver, "Combining online and offline knowledge in UCT," In Proceedings of the 24th International Conference on Machine Learning (ICML2007), pp.273-280, 2007.