Combining Final Score with Winning Percentage by Sigmoid Function in Monte-Carlo Simulations

Kazutomo SHIBAHARA Yoshiyuki KOTANI

Abstract— **Monte-Carlo method recently has produced good results in Go. Monte-Carlo Go uses a move which has the highest mean value of either winning percentage or final score. In a past research, winning percentage is superior to final score in Monte-Carlo Go. We investigated them in BlokusDuo, which is a relatively new game, and showed that Monte-Carlo using final score is superior to the one that uses winning percentage in cases where many random simulations are used. Besides, we showed that using final score is unfavorable for UCT, which is the most famous algorithm in Monte-Carlo Go. To introduce the effectivity of final score to UCT, we suggested a way to combine winning percentage and final score by using sigmoid function. We show the effectivity of the suggested method and show that the method improves a bias where Monte-Carlo Go plays very safe moves when it has advantage.**

I. Introduction

Monte-Carlo Go has achieved great achievements in computer Go during recent years. Since evaluation of a position is difficult in Go, it is difficult to make good evaluation function. Monte-Carlo method uses the winning percentage or final score obtained by random simulation (Play out) for position evaluation. It achieved good evaluation with no special heuristics.

It is said that the use of winning percentage is superior to the use of final score in Monte-Carlo method. Information amount of final score is superior to that of winning percentage, but final score has many outliers. Since the system using winning percentage concerns about winning or losing, the game is likely to be insipid.

In this paper, we investigated the property of using final score in BlokusDuo, and showed its effectivity. Besides, we showed that the use of final score is not effective in UCT algorithm, which is a most famous algorithm in Monte-Carlo Go. So we proposed using sigmoid function to combine final score and winning percentage. It ensures the effectiveness of final score, and eliminates the issues of winning percentage, and achieves good evaluation. We showed that the system using well-calibrated proposal technique is superior to that using winning percentage. In addition, it improves the behavior that the Monte-Carlo system concerns about winning or losing.

A. Monte-Carlo method

Monte-Carlo method is the generic name for a way to get an approximation solution of a problem with random simulation.

K. Shibahara and Y. Kotani are with Department of Computer and Information Sciences, Tokyo University of Agriculture and Technology, Japan (phone: +81-042-388-7492; fax: +81-042-388-7492; email: k-shiba, kotani@cc.tuat.ac.jp).

Brüegmann has applied it to Go[1]. For judgement of a position, Monte-Carlo Go tries to make a move in a random manner from the position repeatedly(random simulation) and tally the number of either winning or final score. Then, it considers the position with the highest value to be the best position. But too many random simulations are needed for high accuracy of position evaluation, it was difficult to use Monte-Carlo in the game where much time has to be spent for making a move.

Since then, Monte-Carlo Go has produced good results, due in part to the performance advances of computer[2]. Monte-Carlo gets an average value of a position. Since game tree search depends solely on only best move, average value is not probably close to an exact value. So several ways incorporated concept of game tree search have been suggested. They include the use of Monte-Carlo at leaf nodes in game tree search[3], and the use of probabilistic framework[4]. The most famous algorithm among them is UCT(UCB for Tree)[5]. It is an extended algorithm of UCB(Upper Confidence Bound)[6] for applying to tree. UCB is effective for n-armed bandit problem, which is similar to game tree search relatively. UCT has been used in MoGo[7] and Crazy Stone[8], which received gold and silver medals respectively at the 12th Computer Olympiad in 2007.

Monte-Carlo method has used in not only Go but also Shogi[9], BlokusDuo[10], Phantom Go[11], etc.

B. BlokusDuo

We dealt with BlokusDuo. It was modified from Blokus, which is four player game. Fig. 1 shows the board of BlokusDuo. It belongs to perfect information two player zero-sum games. There aren't so many people playing the game because it was proposed recently. The base program that we tested in this paper took seventh place out of a population of 16 in 1st Computer BlokusDuo contest in Japan. The strength of base program is probably intermediate grade.

The board has 14 x 14 squares. Each player has 21 pieces which each consists of up to 5 squares and is different from the others. Black places an own piece on a circle of a board. White also places a piece on the other circle. After that, each player continues to place a piece on the board. Each new placed piece must touch other same player's piece, but it can only touch at the corners. Each player gets points equal to the number of squares that the player placed on the board. Winner is a player who has more points than another in the end of the game. Our estimation of the size

978-1-4244-2974-5/08/$25.00 ©2008 IEEE

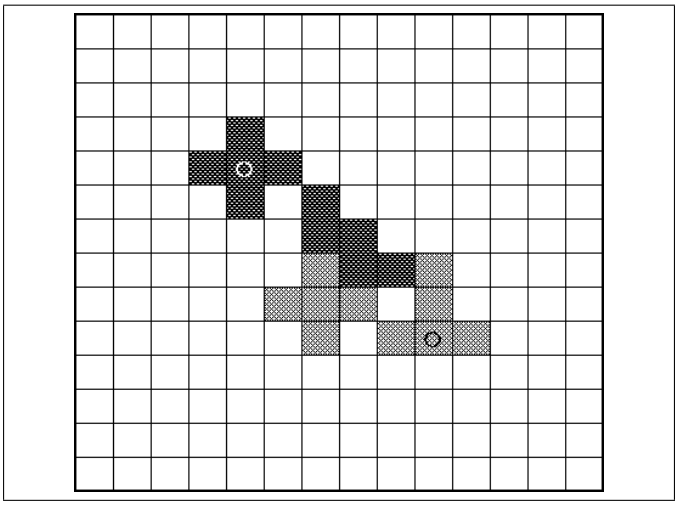

Fig. 1. BlokusDuo

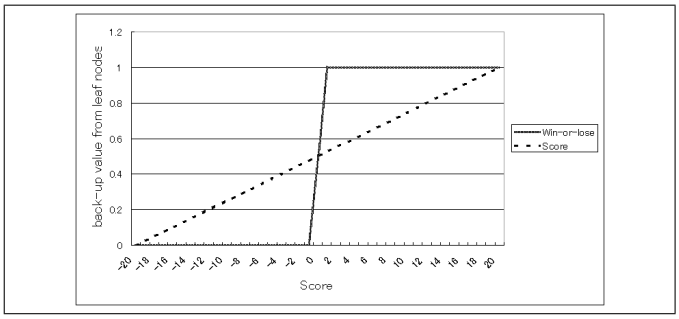

Fig. 2. Win-or-lose and final score

of the game tree is about $10^{70}\sim10^{80}$.[2] There are too many legal moves in the opening game. The number of candidates is sometimes over 1000. On the other hand, there are not many candidates in the ending game. Since there are many positions in the opening game, the search algorithm is not effective. As with Go, it is important to get own area in order to win a game in BlokusDuo. Since the game is similar to Go and the game size is smaller, it is suitable for using in verification of Monte-Carlo method.

C. UCT

UCT is an extended algorithm of UCB for game tree search, which is effective in n-armed bandit problem. UCB eliminates the number of random simulations for obvious bad moves, and spends them for good moves. By doing it, UCB achieves accuracy improvement of Monte-Carlo with concentration of random simulations. UCB also achieves minmax strategy by emphasizing the influence of principal variation.

UCB estimates moves with winning percentage and the number of simulations. Then UCB executes random simulation against the move which has the highest evaluation. There are several computation expressions of UCB. We used the following formula.

$$UCB(i) = \bar{X}_i + \sqrt{\frac{2\log N}{n_i}}$$

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

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

[2]It is yielded by Monte-Carlo algorithm. It is $10^{40}\sim10^{50}$ with random play.

In this paper, for making clearly understandable, winning percentage means $\bar{X}_i$. On the other hand, winning average means the ratio of winning between two programs.

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 didn't use progressive unpruning[12] and first play urgency[13]. The reason is that it is hard to use heuristics because BlokusDuo is relatively new game. In addition, we thought that the performance is not influenced so much by other techniques because the difference between win-or-lose and final score is only the value returned from leaf nodes.

II. Properties of win-or-lose and final score in game tree

A. Win-or-lose and final score

There is a technique for treating quite different information[14], but Monte-Carlo basically uses win-or-lose or final score as a return value of random simulation. Final score is a uniquely-defined value such as a score of the end position. We dealt with win-or-lose as a value. We used win-or-lose value as shown below.

- win:1
- lose:-1
- draw:0

Win-or-lose value is 1 if final score is plus, and it is -1 if final score is minus. Fig. 2 shows win-or-lose value's relationship with final score. We normalized it in this figure.

The difference between win-or-lose and final score probably depends on the number of random simulations. Final score has more amount of information than win-or-lose. However, if the number of random simulations is small, it is strongly affected by precarious results because it has a high dispersion. Past studies showed that win-or-lose is superior to final score in Monte-Carlo Go [15]. Now therefore, Monte-Carlo Go usually uses win-or-lose. Additionally, Monte-Carlo Go using weighting addition of win-or-lose and final score is superior to using only win-or-lose. Thus, the information of final score is beneficial. As far as we know, there are no previous studies that investigated the property of win-or-lose and final score at all.

Crazy Stone, which used win-or-lose, showed a certain feature. It plays safe moves when it is favorable, and plays aggressive and unsafe moves when it is weak. Therefore it usually wins by a neck and loses big. The reason is that it lays disproportionate emphasis on win-or-lose. There are few programs that have this property, thus this is a main reason of strength of Monte-Carlo Go.

B. Comparison of win-or-lose model with final score model

We compared win-or-lose model with final score model in actual game. We limited the number of moves to 16 or later in this experiment because we can execute many random simulations and investigate application in only ending game. Comparative targets are simple Monte-Carlo, called Normal, which executes the same times of random simulations per a candidate move respectively, UCB and UCT. The number of random simulations is from 10^3 to 10^7. However, UCT didn't execute with 10^7 because of restriction of memory and the normal in Monte-Carlo with 10^7 tried about 800 times. In this experiment, the match-up is final score model versus win-or-lose model. For example, UCT using final score model contended with UCT using win-or-lose model. We investigated the transition of the average of winning percentage and final score with increasing random simulations. The number of each match-up is 1000. The final score in BlokusDuo denotes the number of squares occupied by the player's pieces in the end of game. In this experiment, final score is normalized with 89, which is number of occupying squares when player places all pieces. Actually, because existence of usable candidate does not allow pass for a player, no player occupies 89 squares. However, we don't known the maximum score. Fig. 3 shows the result of the winning average of final score model against win-or-lose model, and Fig. 4 shows the result of the average of final score at the end of games. In Fig. 4, if winning average is over 0.5, final score model is superior to win-or-lose model. For example, final score model is superior to win-or-lose model in UCB with 1000 random simulations.

When Normal Monte-Carlo executed random simulations 1,000 times, final score model was obviously inferior to win-or-lose model. The result was possibly produced by lack of statistical stability owing to paucity of random simulations. On the other hand, final score model probably worked so well because UCB concentrate random simulations on good moves. Final score model was superior to win-or-lose model on Normal Monte-Carlo as the number of random simulation increased. When the number of random simulations was 10,000, superiority of final score weakened. And then, many more random simulations emphasized the superiority. Although the superiority was small variation in winning average, it was clear in the average of final score.

When the number of random simulations is small, UCB was not so different from UCT. It is because the behavior of UCT is same as UCB when the number of random simulations is small. UCT is an extension method of UCB for tree structure, thus they do the same treatment for the initial period of time. Unlike the UCB, the increase of the number of random simulations reduced superiority of final score in UCT. UCT possibly came to fluctuate statistically because leaf nodes came to be executed with few random simulations owing to processing based on tree structure. In the result, UCT is not likely to receive benefit of final score. Fig. 5 shows the variances of final score of UCT and UCB. The variance of UCB is less than that of UCT except at 10^6. This result also shows that UCB is stable with final score. At 10^6, the variance of UCB is more than that of UCB. The reason is probably the convergence of Monte-Carlo trial. By comparison, UCT using win-or-lose model was superior to UCB using final score model with 100,000 random simulations. Therefore, though UCT is an effective method, UCT probably can not use the superiority of final score effectively. Consequently, win-or-lose model is superior to final score model with UCT. It is important to introduce final score to UCT in an effective manner.

As shown in II-A, win-or-lose model is obtained from final score mode by reduction of information amount. Final score model gets stability increased by the reduction. However, the experimental result showed that the reduced information include much beneficial information. Thus, it is probably true that combination of win-or-lose model and final score model, that is a restraint of the reduction, probably yield stable and effective information.

III. Combining final score with winning percentage

In order to introduce the superiority of final score to UCT effectively, we propose a method that uses a value gained from combining final score with winning percentage obtained through random simulations. Though a method using simple weighting addition achieved some positive results in past studies, we propose the unified and well-understood method by the use of sigmoid function. In addition, the method improved the feature that Monte-Carlo Go with win-or-lose model sticks to winning or losing.

A. Significance and advantage

We proposed the method of using sigmoid function that combines final score with winning percentage for receiving a result that is similar to win-or-lose model and uses information of final score model. Sigmoid function is also used in logistic regression analysis, which deals with probability. It is suitable for approximation of the probability because sigmoid function has features that are similar to that of probability of winning. It is also used for learning of evaluation function with the features[16]. Fig. 6 shows the shape of sigmoid function.

The formula of sigmoid function is below.

$$f(x) = \frac{1}{1 + \exp^{-kx}}$$

x is final score and k is a constant number. k = 1 in the Fig. 6. As k increases, the gradient becomes intensive and it gets closer to win-or-lose. By using this with final score as x, we can combine final score to winning percentage.

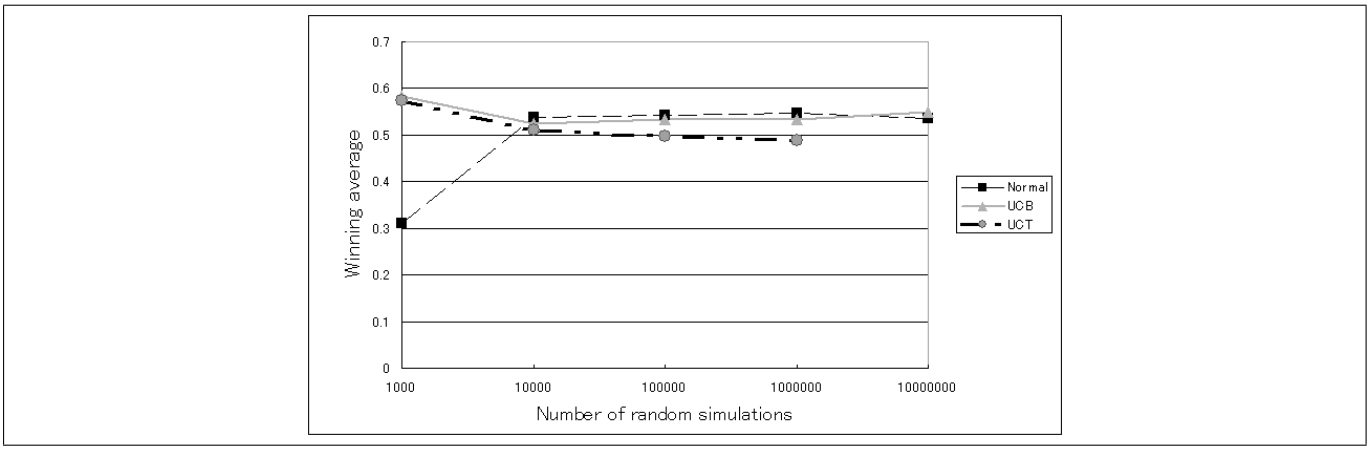

Fig. 3. Transition of the winning average with increasing random simulations (final score model versus win-or-lose model)

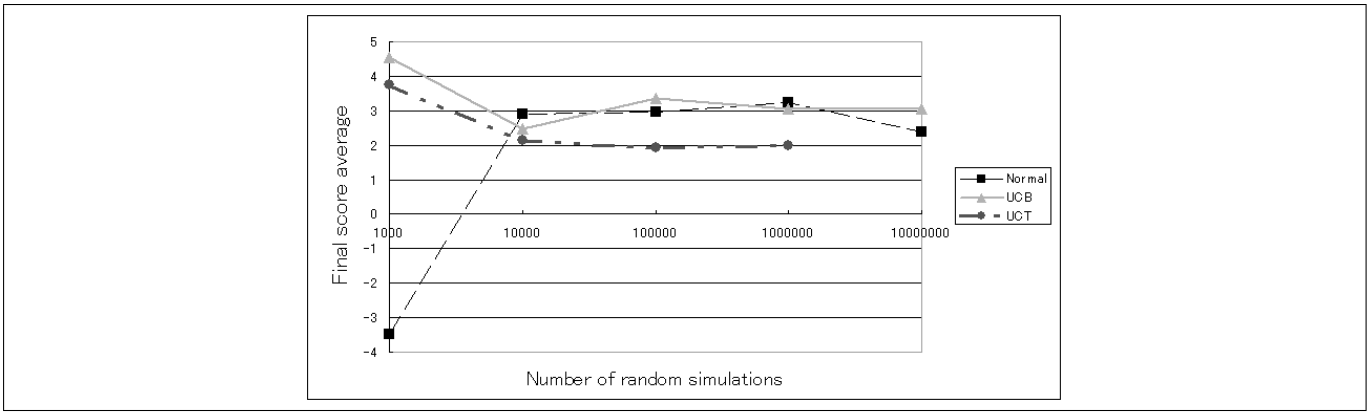

Fig. 4. Transition of the average of final score with increasing random simulations (final score model versus win-or-lose model)

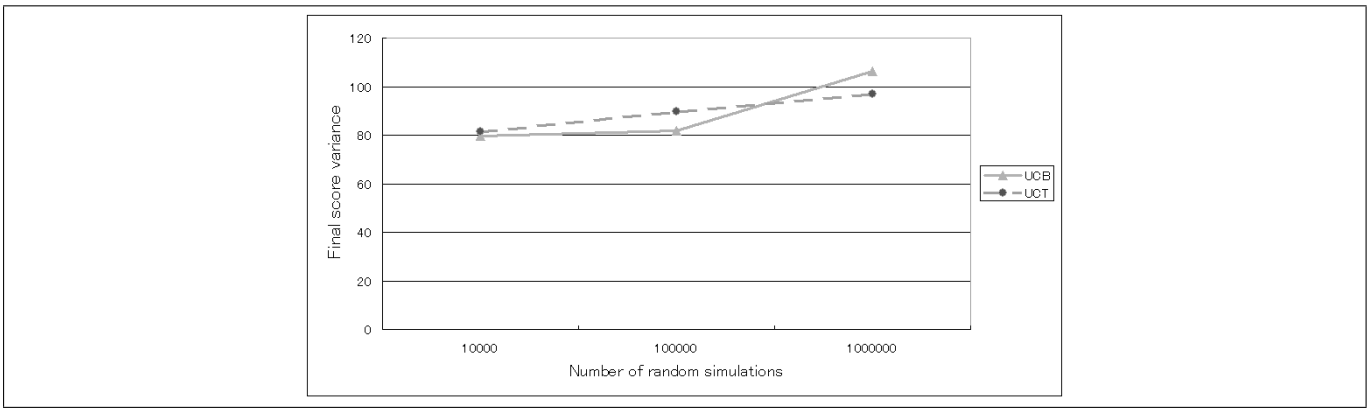

Fig. 5. Transition of the variance of final score with increasing random simulations (final score model versus win-or-lose model)

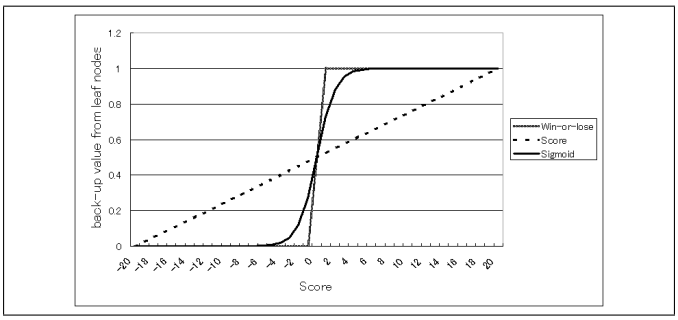

Fig. 6. Sigmoid function

The method provides the well-understood combination of final score and winning percentage. The reliability of final score depends on the number of random simulations. In the game that can try a lot of random simulations, easily applying it by setting k smaller is possible. In the game to which the frequency of the random simulation that can be tried changes easily according to the phase, winning percentage is possibly approximated efficiently only by changing k in proportion to the situation of the stage and the trial frequency of random simulations.

Moreover, it improved features that Monte-Carlo method wins by a nose or comes in nowhere. The difference between a superior position and disadvantageous position is possibly large because the position with a lot of numbers of wins is chosen when doing only by win-or-lose model. Moreover, the error margin's being included easily in winning percentage obtained by Monte Carlo possibly causes a gradual decrease of winning percentage because the difference between superior positions is small. This is possibly a reason why the Monte Carlo method often wins by a nose. Monte Carlo method is a technique for the selection of the hand that leads to building up the advantageousness. Therefore, accurately reflecting minute change and deciding moves probably become important. There is probably a correlation between the final score and easiness to win. The selection of an effective move can be achieved by using the score well. Monte-Carlo system with win-lose probably acts on the defensive in ascendant position, so the game will be uninteresting. On the other hand, it probably behaves recklessly in inferior position, so it will not be able to launch a counterattack perseveringly.

The improvement of bias possibly achieves enjoyable game against a person because the system does not stick to winning or losing.

B. Way of comparative experiment of proposal technique against using final score and win-or-lose model

In the experiments, we compared win-or-lose model, final score model, and the proposal technique by using UCT in BlokusDuo. We obtained the results of the match between final score model and win-or-lose model, and the match between proposal method and win-or-lose model respectively. The number of match-up is 1,000. The number of random simulations is 100,000.

C. Result of comparative experiment of proposal technique against using final score and win-or-lose model

Fig. 7 shows the result with limiting of the number of moves to 16 or later. The limiting is for application to ending game. We defined the winning percentage of win-or-lose model as 0.5 because it played with itself. The result showed that the winning average of the proposal method with k=0.3 is about 56% and showed statistically significant result. Fig. 8 shows mean score when winning or losing. Compared to the simple application of win-or-lose model, the average score when the proposal method is winning is high and the average score when it is losing is slightly low. In addition, because of the change of winning average, the population parameters are different in each value of k. This shows that combining final score with winning percentage with sigmoid function improves a bias where Monte-Carlo Go plays very safely when it has advantage.

Fig. 9 and Fig. 10 show the result with no limiting of the number of moves. The result shows that the winning average of the proposal method with k=0.2 and k=0.7 was about 54% and shows statistically significant result. The graph is crooked though it is similar to the result with limiting. The reason is probably the complexity in the opening and middle game. This also shows the difference between ending game and middle game. The optimal value of k in ending game is probably different from that in middle game.

D. Result of adjustment to progression of game

The results also shows that the final score model becomes effective as the game gets nearer to the ending game. It is a simple application that the ratio of final score increases along with the progress of game. Since Fig. 9 is crooked and the optimal k of sigmoid function probably depends on the stage of game, it is important to use k value appropriate for the stage.

Consequently, we applied sigmoid model depending on progression of game. We assess progression of game by the number of moves because the number of moves correlates strongly with progression of game. It is the best to try a variety of setting, but it spends much time. And so we fixed the setting in advance in a favorable manner. The setting used is as follows.

- 1 to 7(opening game):win-or-lose model
- 8 to 16(middle game):from k=0.5 to 1.0
- over 17(end game):k=0.4

We used sigmoid function with k=0.4 for ending game from the previous results. The reason why we did not use k=0.2 is that we assessed the superiority of k=0.4 from the preparatory experiments. In addition, final score model is not effective in opening game from the preparatory experiments, so we used win-or-lose model in opening game. The number of match-up is 1,000. The number of random simulations is 100,000. Fig. 11 shows the result.

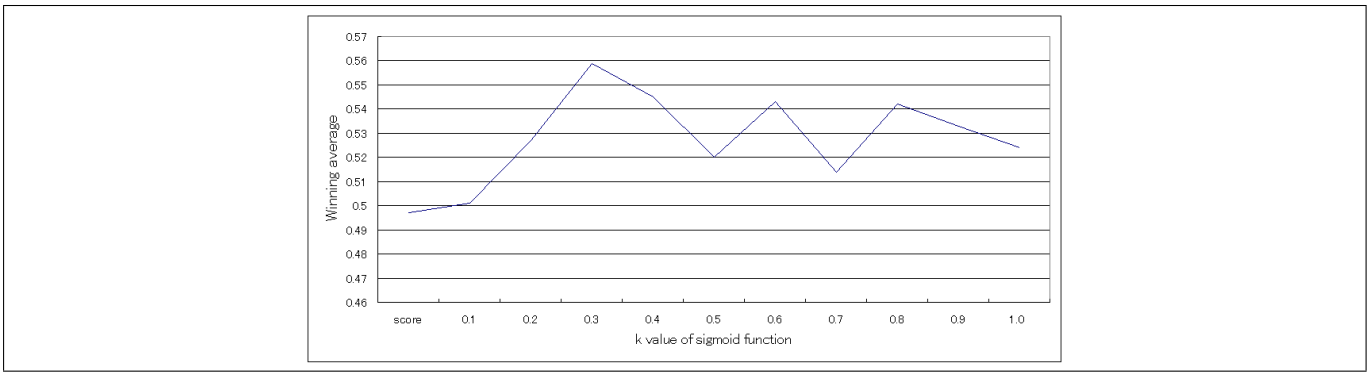

Fig. 7. Result of comparison experiment between the proposal method, win-or-lose and final score model with limiting the number of moves to 16 or later

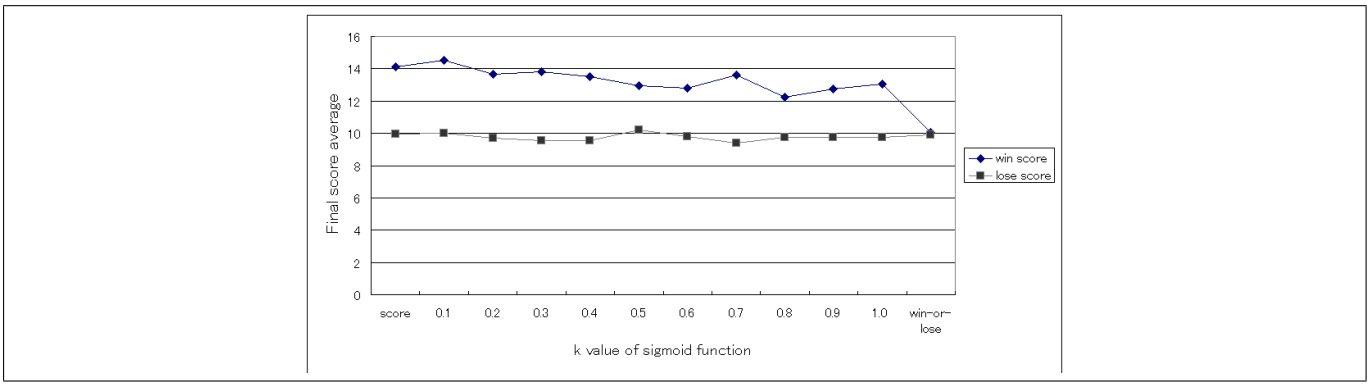

Fig. 8. Result of comparison experiment between the proposal method, win-or-lose and final score model with mean score and limiting the number of moves to 16 or later

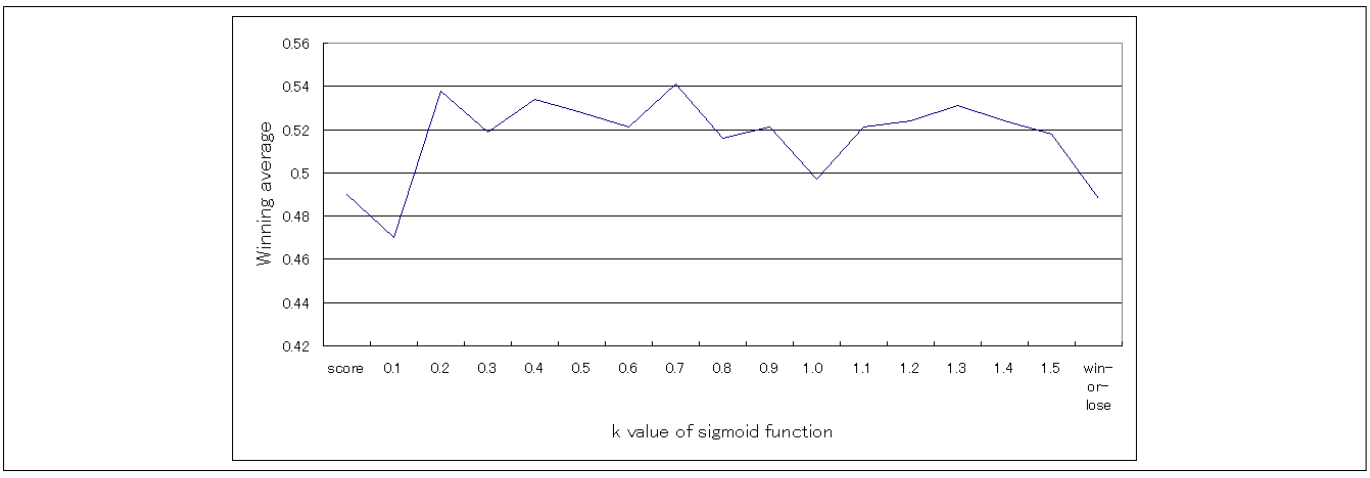

Fig. 9. Result of comparison experiment between the proposal method, win-or-lose and final score model with limiting the number of moves to 16 or later

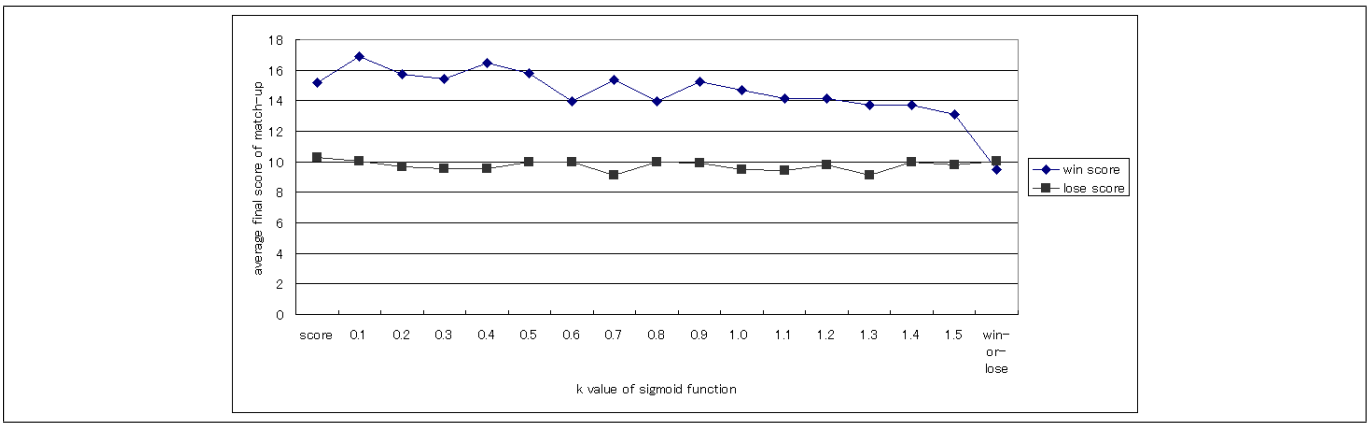

Fig. 10. Result of comparison experiment between the proposal method, win-or-lose and final score model with mean score and limiting the number of moves to 16 or later

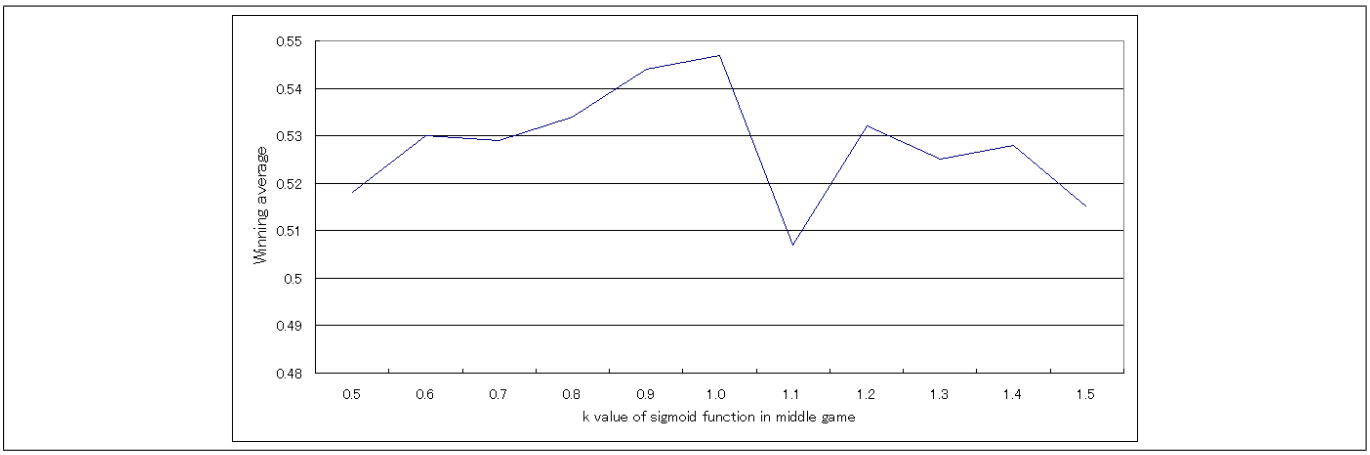

Fig. 11. Result of comparison experiment between the proposal method, win-or-lose and final score model depending on progression of game

The result of experiment achieved 55% winning average tops. k=1.0 is superior to k=0.5(nearer 0.4), so the application depending on progression of game is effective.

E. Questionnaire for comparison between win-or-lose model with final score model

One of an advantage of sigmoid model is achievement of enjoyable game to humans. We sent out questionnaires for verification of it. The examinees were five persons who had played BlokusDuo. There aren't so many people playing the BlokusDuo, it was hard to ensure sufficient numbers. Most of them are beginners, but one of them is able to beat a first BlokusDuo program in the contest.

The way of the experiments is as follows. We showed them twelve game records which are win-or-lose model versus sigmoid model. We used k=1.0 in the sigmoid model because it is equal strength to win-or-lose model. In addition, the game records are non-biased in various points (ex. result of the battle, and total final score and so on). Then we let them judge between win-or-lose model and sigmoid model from three standpoints below.

- Which is more strong?
- Which is more human?
- Which is more amusing?

Of course, the examinees were unapprised of the processing of programs. As a result of the investigation, we obtained the following results.

- Which is more strong? (sigmoid 1 - 4 win-or-lose)
- Which is more human? (sigmoid 3 - 2 win-or-lose)
- Which is more amusing?(sigmoid 3 - 2 win-or-lose)

Win-or-lose model is superior to sigmoid model in the first question, but in other questions, sigmoid model is superior to win-or-lose model. These results shows that "sigmoid model" probably contributes to achieve enjoyable human game.

We show comments on each question.

Which is more strong?

- Win-or-lose model makes less bad moves in opening game.
- Win-or-lose model is probably strong about containing enemy actions accurately, but it is probably weak about taking the lead in attacking.

Which is more human?

- Sigmoid model is probably persevering when it is behind.
- Either models makes unlikely move in human terms at opening game.

Which is more amusing?

- There is not much difference between the two, but sigmoid model makes rough and interesting moves.
- I didn't feel taking it for granted because win-or-lose model didn't make a bad opening move.
- I would enjoy a game with good offense and defense because sigmoid model didn't take strategy of exhaustion. I had expectation of surprising moves.

A examinee described interesting word "rough and interesting moves", but we could not get the detail unfortunately. The ill-regarded point of sigmoid model is bad moves in opening game. We believe the cause is probably difficulty of judgement of aspect in opening game. The results showed that win-or-lose model is probably efficient in opening game. These bad habits are probably a main reason of bad grade and a reason why sigmoid model is inferior to win-or-lose model. There is other case of bad grade of sigmoid model for these bad habits.

On the other hand, examinees appreciated sigmoid model's perseverance, amusingness of moves, element of surprise and skill of attack. There was not these assessments for win-or-lose model, so we believe sigmoid model could bring out amusement of game. Incidentally, a person who could beat first program in the contest chose sigmoid model in all questions.

IV. Discussion

The results of experiments showed that Monte-Carlo algorithm using sigmoid function improved its performance significantly. The method achieved 54% winning average tops. On the other hand, execute time slightly increased because of using only sigmoid function. The method eclipsed the system using win-or-lose or final score model. In addition, the system using final score is effective with many random simulations.

The results also showed that the final score model is effective in ending game. Appropriate applications of sigmoid model depending on the number of moves achieved 55% winning average tops.

In addition, UCT depends on conditions of shallow positions, and optimal k value possibly depends on the condition of the root position. In Monte-Carlo, previous studies proposed the use of heuristics for pruning or selecting moves in random simulation[7], so it is possibly effective to use heuristics for deciding optimal k value.

Systems usually spend much time in important positions. In such cases, the use of optimal sigmoid function in each random simulation possibly achieves effective play.

The result of questionnaire shows that sigmoid model could make amusing moves humanly. However, the result also showed that win-or-lose model is not likely to make definite bad moves in the opening game. Though it is not possible to declare because the number of answers is little, the sigmoid model is probably effective for the implementation of enjoyable game.

V. Conclusion

We showed that the system using final score model is superior to the system using win-or-lose model with many random simulations. Then we showed that UCT is not able to use information of final score effectively. Thus we proposed the method that uses the return value of random simulation in Monte-Carlo through sigmoid function for combining final score and winning percentage. The results achieved significantly 55% as winning average against the system using win-or-lose model. In addition, it improved the winning bias which Monte-Carlo has. The results of questionnaires showed that sigmoid model perhaps makes amusing moves humanly.

Future works include adjustment of gradient of sigmoid function tailored to the number of random simulations and condition of positions, and constructing calculational procedure of the optimal gradient regardless of kinds of game.

References

[1] Bernd Brüegmann: Monte Carlo Go, Technical report, Physics Department, Syracuse University, unpublished, 1993.

[2] Bruno Bouzy and Bernard Helmstetter: Monte Carlo go developments, Advances in Computer Games conference (ACG-10), pp. 159-174, 2003.

[3] Bruno Bouzy: Associating Shallow and selective global tree search with Monte Carlo for 9 × 9 Go, 4rd Computer and Games Conference, CG04, Ramat-Gan, Israel, LNCS 3846/2006, pages 67-80, 2004.

[4] Rémi Coulom: Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search, Proceedings of the 5th Computers and Games Conference (CG 2006), pp.29-31, 2007.

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

[6] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite time analysis of the multiarmed bandit problem. Machine Learning, 47(2-3), pp. 235-256, 2002.

[7] Sylvain Gelly and David Silver: Combining Online and Offline Knowledge in UCT, International Conference of Machine Learning, ICML 2007, Corvallis Oregon USA, pp. 273-280, 2007.

[8] Rémi Coulom: Monte-Carlo Tree Search in Crazy Stone, Game Programming Workshop 2007, pp.74–75 , 2007.

[9] Junichi Hashimoto, Tsuyoshi Hashimoto and Jun Nagashima: A potential application of Monte-Carlo Method in Computer Shogi, Game Programming Workshop 2006, pp.195-198, 2006.

[10] Shugo Nakamura, Makoto Miwa and Takashi Chikayama: Improvement of UCT using evaluation function, Game Programming Workshop 2007, pp. 36–43, 2007.

[11] Tristan Cazenave: A Phantom-Go Program, ACG2005, pp.120-125, 2005.

[12] 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.

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

[14] Tristan Cazenave and Bernard Helmstetter: Combining tactical search and Monte-Carlo in the game of go. IEEE CIG2005, pp. 171-175, 2005.

[15] Bruno Bouzy: Old-fashioned Computer Go vs Monte-Carlo Go, Invited tutorial, IEEE 2007 Symposium on Computational Intelligence in Games, CIG 07, unpublished, 2007.

[16] Kunihito Hoki: Optimal control of minimax search results to learn positional evaluation, Game Programming Workshop 2006, pp.78-83, 2006.