International Journal of Smart Computing and Artificial Intelligence
International Institute of Applied Informatics
2020, Vol. 4, No. 1, 18 – 35

New Applications of the Monte-Carlo Tree Search to Computer Daihinmin

Seiya Okubo [*], Mitsuo Wakatsuki [†], Tasuku Mitsuishi [†],
Yasuki Dobashi [†], Tetsuro Nishino [†]

Abstract

The Monte Carlo tree search is popular in computer programs that play games. In this study, we present two new applications of the Monte Carlo method for the card game Daihinmin: as a method for setting good evaluation values in repeated plays, and as a method for increasing or decreasing the score of a target player.

Keywords: Computer Daihinmin, Game Informatics, Monte Carlo Method

1 Introduction

A game is said to have perfect information if each player is perfectly informed of all previous events. Examples of such games include Shogi and Go. In contrast, in some card games, a player's cards are hidden from the other players (e.g., poker, Daihinmin). Such games are characterized as having imperfect information.

The Monte Carlo tree search is popular in computer programs that play games. This method analyzes the most promising moves, and expands the search tree based on a random sampling of the search space. The method is applied based on many playouts. In each playout, the game is played to its conclusion by selecting moves at random. Then, the final result is evaluated and used to weight the nodes in the game tree, such that better nodes are more likely to be chosen in future playouts. The Monte Carlo method is expected to have many future additional applications.

In this paper, we present two new applications of the Monte Carlo method for Daihinmin, as a method for setting good evaluation values in repeated card plays, and as a method for increasing or decreasing the score of a target player. Furthermore, we conduct several experiments to show the effectiveness of our proposed methods.

[*] University of Shizuoka,Shizuoka, Japan
[†] The University of Electro-Communications, Tokyo, Japan

2 Preliminaries

2.1 Computer Daihinmin

Daihinmin is a card game played mainly in Japan, but similar games are played around the world. It is a multiplayer game of imperfect information that has been studied extensively in recent years [1][2][3] . Computer Daihinmin refers to playing the game on a computer. The UEC Computer Daihinmin Convention (UECda) is an annual competition for computer Daihinmin programs, in which several thousand games are played by computers with extreme computational capabilities. The competing algorithms seek to win over the course of many games, without the influence of an initial hand.

In this study, we adopt the framework utilized by the UECda [4]. Although there are numerous examples of localized Daihinmin rules, the UECda implements the specific rules outlined below.

Game Procedure:
The game is played by five players and uses a total of 53 cards: 13 (ace-king) hearts, clubs, spades, and diamonds; as well as a joker. The cards are ranked in the following: 3, 4, 5,$\cdots$, 10, jack, queen, king, ace, 2; where 3 has the lowest value and 2 has the highest value. At the start of each game, each player is dealt 10 or 11 cards. Players take turns in clockwise order to discard (play) the cards in their hands. The first player to get rid of all their cards is the winner.

Start of the Game:
The game starts with the player who has the 3 of diamonds. The player either plays (discards) his/her card(s), or passes the turn. This process is repeated for each player. If there are no cards on the table (“ the field is lead ”), a player may play any type of card in turn (a single, pair, or kaidan [sequence]). If a previous play is on the table (“the field is follow”), the current player can play card(s) to defeat the previous play.

To Close a Round:
When all players have played their respective turns, the round ends. The last player to play a hand begins the next round without any cards on the board.

Pass:
A player may pass a turn if he/she has no card(s) to play, or would simply prefer to pass. Once a player passes a turn, that player does not have another turn until the round ends.

Eight-ender (8 Rule, 8 Giri):
A round ends when a player plays a hand containing an 8.

3 of Spades:
When the joker is played as a single card, a player may end the round by playing the 3 of spades.

Revolution (Kakumei):
When a player plays a set (pair) of four or more cards with the same number, or a sequence of five or more cards, a revolution occurs and the strengths of all cards are reversed until the end of the game.

Lock (Tight, Shibari):
When a player follows the same suit as the play on the board, the round is tightened by the suit and all players must follow the same suit until the round ends.

Special Titles (Rank, Mibun):
The first player with no cards left is the Daifugo, the second is the Fugo, the third is the Heimin, the fourth is the Hinmin, and the last is the Daihinmin.

Card Change (Card Trade, Despotism):
The Daifugo hands two cards to the Daihinmin and the Fugo hands one card to the Hinmin. The choice of cards to be handed to another player is arbitrary. However, the Daihinmin must yield two of his/her strongest cards to the Daifugo, and the Hinmin must give his/her strongest card to the Fugo.

"Special Titles" and "Card Change" are distinctive rules. are distinctive rules. According to the above rules, the result of one game affects a player's advantages and disadvantages in the next game. This is not found in other imperfect information games, such as mahjong, and results in the game having unique strategies and features.

The UECda framework is used in many studies on Daihinmin, including those that use analytical methods to determine the characteristics of the computer game [5], propose rating algorithms for imperfect information games [3], and study society of mind theory [6].

2.2 Monte Carlo Method

The Monte Carlo method finds approximate solutions by performing many playouts. Here, a playout is a random simulation of a game. In game informatics, playouts are used to evaluate and assign a value to each play. The naive Monte Carlo method is performed as follows:

1. List all card(s) that can be played in a phase.
2. Randomly select one or more cards from the list. Let i represent the selected card(s).
3. From the next phase, which plays cards i, until the end, perform random simulations
4. Evaluate i from the final rank, and update the value $\overline{X_i}$.
5. Repeat steps 2-4 multiple times, playing the card(s) with the highest evaluation value each time.

Here, $\overline{X_i}$ is determined as follows:

$$\begin{aligned} X_i &\leftarrow X_i + V \\ n_i &\leftarrow n_i + 1 \\ \overline{X_i} &\leftarrow X_i / n_i \end{aligned}$$

where n_i is the number of times i is selected, V is the evaluation value in each playout, and X_i is the total evaluation value.

As the number of playouts increases, the evaluations become increasingly accurate. However, because time is limited, a good playout assignment is required. The assignment problem is formulated as a multi-arm bandit problem. Several algorithms have been proposed to solve this problem efficiently. Programs participating in the UECda that use the Monte Carlo method often use UCB1-Tuned [7] [8].

2.3 Daihinmin Program

The many studies on Daihinmin have resulted in the creation of strong programs. Typical programs include the following:

Default
Default performs only standard operations, and the algorithm is simple. The program was used in UECda-2015. It is a heuristic program that plays the weakest card that can be played

Snowl
Snowl, a winning program at UECda-2010 and developed by Mr. Fumiya Suto, was an indiscriminate standard program at UECda-2015. Snowl is a typical example of a program that uses the Monte Carlo method in its implementation of Computer Daihinmin[9]. Because it has been the subject of many studies, its behavior has now become evident and, thus, predictable. As it has been a subject of many studies, its behavior has now become evident and, thus, predictable.

MSM04
MSM04 was developed by Mr. Kiyono and Mr. Watabe. The program does not use special parameters and uses the naive Monte Carlo method.

Wisteria
Wisteria was a winning program in the indiscriminate class of UECda-2015, and was developed by Mr. Katsuki Ohto [10]. This program uses policy gradients and Monte Carlo methods.

Blauweregen(Blau)
Blau was a winning program in the indiscriminate class of UECda-2016, and was developed by Mr. Katsuki Ohto. This is an improved version of Wisteria.

Kou2
Kou2 was a winning program in the lightweight class of UECda-2015, and was developed by Mr. Kozou Tagashira [11][12]. This program use evaluation methods based on human knowledge.

3 Method1: Determining the optimal evaluation value

3.1 Proposed Method

The Monte Carlo method hinges on being able to determine the evaluation value. In this section, we discuss some of the better methods available.

In the naive method, the evaluation value is the score obtained in each game. That is, the evaluation values are 5, 4, 3, 2, 1 in order from the top (hereinafter, this is denoted as (5,4,3,2,1)). However, in Daihinmin, the higher ranks are advantageous in the subsequent game. Therefore, higher ranks are expected to yield more points and, thus, be more valuable. For example, Snowl uses the values (25, 16, 9, 4, 1), that is, the squares of the scores, although the reason for this choice is not described in [9].

Another method obtains a transition matrix between ranks using a computer experiment, and determines a future expectation value from the transition matrix. Wisteria and Blau

use this method. The expected score after n games for each rank is calculated using the transition matrix R, as follows:

$$E_n = R^n \times \begin{pmatrix} 5 & 4 & 3 & 2 & 1 \end{pmatrix}^T \tag{1}$$

That is, the expected value P_n of the score after n games is given by the following formula:

$$P_n = \sum_{i=0}^{n} E_i \tag{2}$$

where the parameter n indicates the number of games to consider. Wisteria uses $n = 14$.

3.2 Computer Experiment

In this study, we evaluate game values using computer experiments based on MSM04, Snowl, and Wisteria. Specifically, we changed the evaluation value in each program and made it play against the original program. The evaluation values obtained from seven methods are as follows:

f0: The evaluation values are (5,4,3,2,1), which are the scores.

sq: The evaluation values are (25,16,9,4,1), which are the squares of the score.

sq-: The evaluation values are (20.75,16,9,4,1), which are obtained by adjusting sq to neglect the Daihugo.

sq+: The evaluation values are (29.50,16,9,4,1), which are obtained by adjusting sq to emphasize the Daihugo.

f1: The evaluation values are P_1.

f2: The evaluation values are P_2.

f5: The evaluation values are P_5.

f15: The evaluation values are P_{15}.

An outline of each evaluation value is shown in Figure1.

Because the value of P_n differs depending on the program, we obtained it in advance using a computer experiment. Tables 1 and 2 show the transition matrices, and Tables 3, 4, and 5 show the values of p_n. Here, Wisteria's p_n values appear in the original program.

A match consists of 12 sets, with 10000 games in each set. The average score was 30000. The combination for the match is each fixed program $\times$ 1, and the f0 program $\times$ 4. The evaluation value is obtained by comparing the program scores.

The experimental results are shown in Figures 2, 3, and 4, where the error bars represent standard deviations. There are no significant differences between many of the patterns. The series sq tends to have a higher score than the series f. In particular, in the case of Snowl and MSM04, sq+ is significantly different from f0. However, Wisteria shows almost no differences between the evaluation values. This change may not be suitable for Wisteria, and may result in unsuccessful playout assignments.

Because it takes time to determine the value of each f, it is appropriate that we use sq+. Note that the opponent is different in each experiment, which may affect the results. Thus, additional experiments are required, which is left to future research.

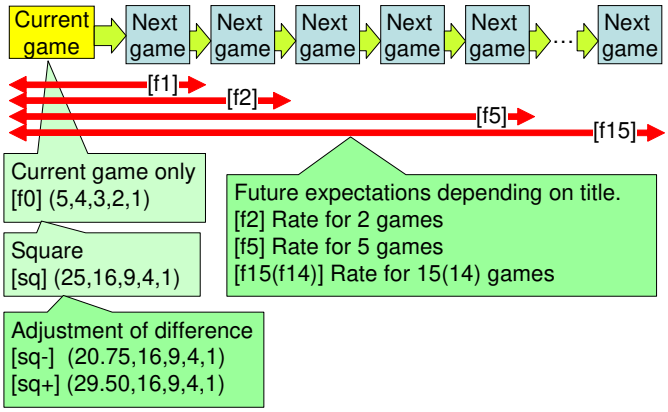

Figure 1: Estimate Values

Table 1: Transition matrix of MSM04

	Daihugo	Hugo	Heimin	Hinmin	Daihinmin
Daihugo	0.467	0.273	0.136	0.064	0.060
Hugo	0.301	0.309	0.198	0.107	0.085
Heimin	0.124	0.200	0.265	0.226	0.187
Hinmin	0.061	0.128	0.228	0.303	0.279
Daihinmin	0.046	0.093	0.173	0.299	0.389

3.3 Comparison of Evaluation Values

In this section, we compare the evaluation values obtained by the computer experiments. Note that the program values have different ranges (e.g., Snowl squares the evaluation values), making a comparison difficult.

However, in the Monte Carlo method, the evaluation values assigned to each rank (mibun) are not as important as the ratios of the values between ranks. For example, the evaluation values (5, 4, 3, 2, 1) from a higher rank have the same meaning as (10, 8, 6, 4, 2) and (7, 6, 5, 4, 3). That is, the ratio of the values between adjacent ranks is important. In this example, the differences between the evaluation values of adjacent ranks are (1, 1, 1, 1), (2, 2, 2, 2), and (1, 1, 1, 1); hence, the ratios are the same. Therefore, in this subsection, we consider the characteristics of the values by comparing the ratios of the differences between

Table 2: Transition matrix of Snowl

	Daihugo	Hugo	Heimin	Hinmin	Daihinmin
Daihugo	0.530	0.267	0.117	0.055	0.031
Hugo	0.295	0.353	0.189	0.104	0.060
Heimin	0.103	0.191	0.290	0.233	0.183
Hinmin	0.040	0.128	0.231	0.331	0.290
Daihinmin	0.031	0.082	0.172	0.279	0.437

Table 3: Value of P_n of MSM04

n	Daihugo	Hugo	Heimin	Hinmin	Daihinmin
0	5.000	4.000	3.000	2.000	1.000
1	9.024	7.636	5.846	4.388	3.106
2	12.563	10.970	8.753	7.062	5.654
3	15.845	14.145	11.703	9.891	8.420
4	18.993	17.236	14.677	12.802	11.298
5	22.070	20.284	17.663	15.756	14.234
6	25.111	23.309	20.656	18.732	17.201
7	28.132	26.323	23.653	21.719	20.184
8	31.144	29.331	26.652	24.713	23.176
9	34.150	32.335	32.651	27.710	26.172
10	37.154	35.338	32.651	33.709	29.170
11	40.156	38.340	35.652	33.709	32.169
12	43.158	41.342	38.652	36.709	35.169
13	46.160	44.343	41.653	39.710	38.170
14	49.161	47.345	44.654	42.711	41.170
15	52.162	50.346	47.655	45.711	44.171

the values of adjacent ranks.

We calculated the differences between the evaluation values of adjacent ranks, and normalized the results.

Table 6 shows the results for Snowl (original), and Tables 7,8 and 9 show the results for Snowl, MSM04, and Wisteria for p1 to p15.

When $n = 1$, all values are $(0.25, 0.25, 0.25, 0.25)$, and the program values vary as n increases. However, they do not differ greatly, and show the same tendency. Specifically, as n increases, the difference between Fugo and Heimin increases, and the difference between Hinmin and Daihinmin decreases. On the other hand, the differences between Daifugo and Fugo and between Heimin and Hinmin remain almost unchanged. The evaluation value can be interpreted as emphasizing the ranking above Fugo, which can receive cards in card changes.

On the other hand, when the squares of the scores are used (Snowl), the differences become larger as the rank increases. These values can be interpreted as emphasizing a higher rank.

These results, along with those of section 3.2, suggest that it is better to play to obtain a higher rank in the current game, than it is to play to obtain higher future evaluation values.

Table 4: Value of P_n of snowl

n	Daihugo	Hugo	Heimin	Hinmin	Daihinmin
0	5.000	4.000	3.000	2.000	1.000
1	9.209	7.720	5.800	4.279	2.9928
2	12.947	11.158	8.6530	6.829	5.415
3	16.394	14.421	11.560	9.555	8.070
4	19.665	17.579	14.503	12.390	10.863
5	22.828	20.674	17.469	15.290	13.739
6	25.927	23.732	20.448	18.230	16.663
7	28.986	26.767	23.435	21.194	19.618
8	32.022	29.788	26.428	24.172	22.591
9	35.044	32.800	29.423	27.159	25.574
10	38.057	35.808	32.421	30.151	28.564
11	41.065	38.813	35.419	33.146	31.558
12	44.069	41.815	38.418	36.143	34.555
13	47.072	44.817	41.417	39.141	37.552
14	50.074	47.818	44.417	42.140	40.551
15	53.075	50.819	47.417	45.140	43.550

Table 5: Value of P_n used in Wisteria

n	Daihugo	Hugo	Heimin	Hinmin	Daihinmin
0	400	300	200	100	0
1	627	472	283	122	0
2	766	576	327	130	0
3	852	640	352	134	0
4	906	679	368	137	0
5	940	704	378	138	0
6	961	719	384	139	0
7	975	729	388	139	0
8	983	735	390	140	0
9	988	739	392	140	0
10	991	741	393	140	0
11	993	743	393	140	0
12	995	744	394	140	0
13	996	744	394	140	0
14	996	745	394	140	0

Table 6: Differences between the Snowl evaluation values (sq)

	Daihugo↔Hugo	Hugo ↔ Heimin	Heimin ↔Hinmin	Hinmin↔Daihinmin
sq	0.38	0.29	0.21	0.13

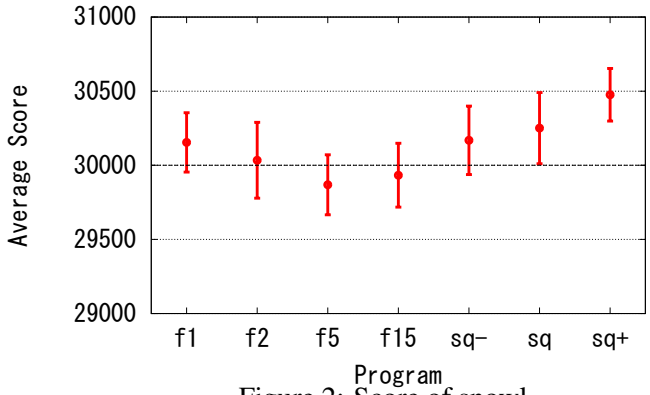

Figure 2: Score of snowl

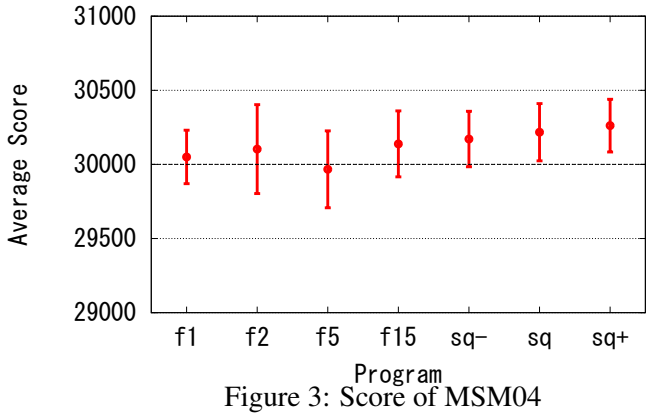

Figure 3: Score of MSM04

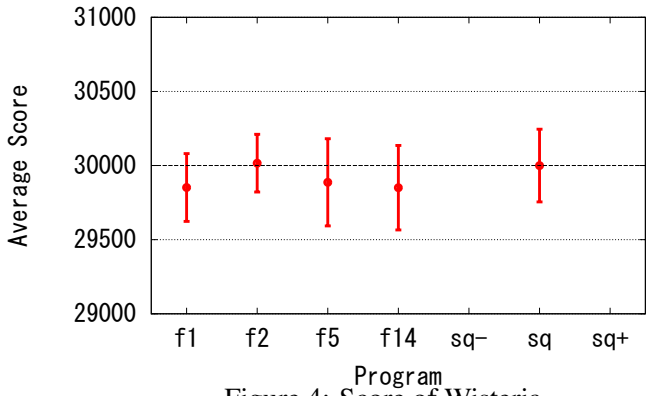

Figure 4: Score of Wisteria

Table 7: Differences between the Snowl evaluation values(P_n)

	Daihugo↔Hugo	Hugo ↔ Heimin	Heimin ↔Hinmin	Hinmin↔Daihinmin
0	0.25	0.25	0.25	0.25
1	0.24	0.31	0.24	0.21
2	0.24	0.33	0.24	0.19
3	0.24	0.34	0.24	0.18
4	0.24	0.35	0.24	0.17
5	0.24	0.35	0.24	0.17
6	0.24	0.36	0.24	0.17
7	0.24	0.36	0.24	0.17
8	0.24	0.36	0.24	0.17
9	0.24	0.36	0.24	0.17
10	0.24	0.36	0.24	0.17
11	0.24	0.36	0.24	0.17
12	0.24	0.36	0.24	0.17
13	0.24	0.36	0.24	0.17
14	0.24	0.36	0.24	0.17
15	0.24	0.36	0.24	0.17

Table 8: Difference between the evaluation values of MSM04

	Daihugo↔Hugo	Hugo ↔ Heimin	Heimin ↔Hinmin	Hinmin↔Daihinmin
0	0.25	0.25	0.25	0.25
1	0.23	0.30	0.25	0.22
2	0.23	0.32	0.24	0.20
3	0.23	0.33	0.24	0.20
4	0.23	0.33	0.24	0.20
5	0.23	0.33	0.24	0.19
6	0.23	0.34	0.24	0.19
7	0.23	0.34	0.24	0.19
8	0.23	0.34	0.24	0.19
9	0.23	0.34	0.24	0.19
10	0.23	0.34	0.24	0.19
11	0.23	0.34	0.24	0.19
12	0.23	0.34	0.24	0.19
13	0.23	0.34	0.24	0.19
14	0.23	0.34	0.24	0.19
15	0.23	0.34	0.24	0.19

Table 9: Difference between the evaluation values of Wisteria

	Daihugo↔Hugo	Hugo ↔ Heimin	Heimin ↔Hinmin	Hinmin↔Daihinmin
0	0.25	0.25	0.25	0.25
1	0.24	0.30	0.25	0.21
2	0.24	0.33	0.25	0.19
3	0.25	0.33	0.25	0.18
4	0.25	0.34	0.25	0.17
5	0.25	0.34	0.25	0.17
6	0.25	0.34	0.25	0.17
7	0.25	0.34	0.24	0.17
8	0.25	0.34	0.24	0.16
9	0.25	0.34	0.24	0.17
10	0.25	0.34	0.24	0.16
11	0.25	0.34	0.24	0.16
12	0.25	0.34	0.24	0.16
13	0.25	0.35	0.24	0.16
14	0.25	0.35	0.24	0.16

4 Method2: Algorithm to target a specific player

4.1 Proposed Algorithm

In multiplayer games, it is difficult to realize a play that affects a specific target player. In this paper, we propose an algorithm to increase or reduce the score of a target player using the Monte Carlo method.

Monte Carlo simulations can be used to obtain a player's own evaluation values, as well as those of other players, for each player's set of card(s). However, existing programs that use the Monte Carlo method seek to maximize the player's score by playing the cards with the highest evaluation values, based only on the player's own evaluation value. Therefore, the values of other players are not used.

The proposed algorithm focuses on the evaluation values of the target player, obtained from the simulation results, and plays those card(s) with the highest or lowest values. Three algorithms are proposed. The first reduces the score of the target player by as much as possible, the second increases the score of the target player by as much as possible, and the third reduces the score of the target player, while considering the player's own score. When executing the Monte Carlo method, the proposed algorithm calculates an evaluation value $\overline{X_i'}$, where the score of the target player is V, and selects the card(s) to play based on this value. The three algorithms are given below.

[*Proposed Algorithm I*] This algorithm reduces the score of the target player by as much as possible. The player's own score is not considered.

1. List all card(s) that can be played in a phase.
2. Randomly select one or more cards from the list. Let i represent the selected card(s).
3. From the next phase, which plays the cards i, until the end, perform random simulations.

4. Evaluate i from the final rank of the target program, and update the value $\overline{X_i}'$.
5. Repeat steps 2-4 multiple times, playing the cards with the highest evaluation values.

Compared with the naive Monte Carlo algorithm, this algorithm is expected to reduce both the player's score and that of the target player.

[*Proposed Algorithm II*] This algorithm increases the score of the target player by as much as possible. The player's own score is not considered.

1. List all cards that can be played in a phase.
2. Randomly select one or more cards from the list. Let i represent the selected card(s).
3. From the next phase, which plays the cards i, until the end, perform random simulations
4. Evaluate i from the final rank of the target program, and update the value $\overline{X_i}'$.
5. Repeat steps 2-4 multiple times, playing the cards with the lowest evaluation values.

Compared with the naive Monte Carlo algorithm, this algorithm is expected to reduce the player's score and increase the score of the target player.

[*Proposed Algorithm III*] This algorithm reduces the score of the target player, while also considering the player's own score.

1. List all cards that can be played in a phase.
2. Randomly select one or more cards from the list. Let i represent the selected card(s).
3. From the next phase, which plays cards i, until the end, perform random simulations.
4. Update the evaluation value $\overline{X_i}$ of the card(s) i from the player' s final rank, and update $\overline{X_i}'$ of the card(s) i from the final rank of the target program.
5. Repeat steps 2.4 multiple times, playing the card with the lowest evaluation value for the target program from the player's two cards with the highest evaluation values.

When selecting cards with low evaluation values for the target player, there are only two options. Therefore, although the score of the target player is higher than that of the Proposed Algorithm I, we estimate that it is lower than that of the naive Monte Carlo algorithm. We further estimate that, although the player's own score is higher than that of the Proposed Algorithm I, it is lower than that of the naive Monte Carlo algorithm.

4.2 Comparison of Algorithms

In this section, we explain the differences in the behavior of each algorithm by comparing them.

The flowchart for each algorithm is shown in Figure 5. The bolded areas in the figure are the differences between the algorithms. The differences signify the ways in which the evaluation values are obtained, and the ways in which the play is selected. These differences affect plays under same situations.

Table 10: Example of Algorithm Results

i	Average of my rank	Average of target's rank	$\overline{X_i}$ in the Naive Monte Carlo method	$\overline{X_i}$ in the Algorithm I and II	$\overline{X_i}-\overline{X_i'}$ in the Algorithm III	Algorithm
0	4.1	3.2	4.1	3.2	0.9	Normal
1	3.4	1.5	3.4	1.5	1.9	Algorithm III
2	1.5	1.2	1.5	1.2	0.3	Algorithm II
3	2.4	3.5	2.4	3.5	-1.1	Algorithm I
4	2.8	3	2.8	3.0	-0.2	

An example of the execution of the naive Monte Carlo algorithm and proposed algorithms are shown in Table 10. The table shows evaluation values and choices of plays in a given situation. In this example, there are five possible plays from 0 to 4. The average rankings of players and their targets for each play in multiple playouts are also shown. There are a variety of ways to get a rating from a rank. In this example, the evaluation value is assumed to be equal to the rank. The rightmost column of the table shows the play submitted by each algorithm.

In the naive Monte Carlo method, play 0 is chosen as it has the highest evaluation value. In Algorithm I, the play with the highest evaluation value of the opponent is selected; therefore, play 3 is selected. In Algorithm II, the play with the lowest evaluation value of the opponent is selected; therefore, play 2 is selected. In Algorithm III, the play with the largest difference in the evaluation value between the player and the target is selected; therefore, play 1 is selected. Thus, even in the same situation, each algorithm chooses a different play.

4.3 Computer Experiment

The effectiveness of the proposed method is evaluated using computer experiments. We created Snowl-I, which is an implementation of the Proposed Algorithm I on Snowl; Snowls, which is an implementation of the Proposed Algorithm II on Snowl; and jsnowl, which is an implementation of the Proposed Algorithm III on Snowl. The performance of each program is compared with that of the original Snowl.

In each experiment, the match combination is as follows: each fixed program $\times$ 1, and one program $\times$ 4. Here, Default, Kou2, Snowl, and Blau are used as the match program, because the results may differ depending on the combination of match programs. The match cards are shown in Table 11. Here, P1 is a target program, and P5 is one of Snowl, Snowl-i, Snowl-s, and Jsnowl. Matches 1-3 consisted of 100 sets, with 3000 games in each set, and matches 1-4 consisted of 30-50 sets, with 3000 games in each set. It is difficult to compare algorithms in the same situation because the algorithms using the Monte Carlo method have ¡random behavior. We considered the average of 100 or 50 games to obtain an unbiased result. Because Default is the most basic action, Kou2 and Blau, winners of the lightweight and indiscriminate classes, respectively, were chosen as opponents. Blau is the strongest of the programs, followed in order by Snowl, Kou2, and Default.

The experimental results for Snowl (called "Base Exp.") are shown in Table 12. In this case, the opponent's score has not changed; in other words, Snowl does not affect specific opponents.

The experimental results for Snowl-i (called " Exp.1 ") are shown in Table 13. Here,

Table 11: Match Cards

	P1	P2	P3	P4	P5
Match1	default	default	default	default	Proposed Algorithm
Match2	Kou2	Kou2	Kou2	Kou2	Proposed Algorithm
Match3	snowl	snowl	snowl	snowl	Proposed Algorithm
Match4	Blau	Blau	Blau	Blau	Proposed Algorithm

Table 12: Averages of score in the experiment using Snowl (Base Exp.)

	P1	P2	P3	P4	Snowl
Match1(Default)	7954	7942	7950	7943	13208
Match2(Kou2)	8930	8927	8890	8959	9292
Match3(Snowl)	8994	8984	9023	8984	9012
Match4(Blau)	9406	9447	9433	9373	7338

the P1 scores are lower than the P2-P4 scores, regardless of the opponent's program. Thus, Algorithm I reduces the score of the target player, as intended. The P5 score is low.

The experimental results for Snowl-s (called "Exp.2") are shown in Table 14. In this case, the P1 scores are higher than those of P2-P4, regardless of the opponent's program. Thus, Algorithm II increases the score of the target player, as intended. The P5 score is again low.

The experimental results for Jsnowl (called "Exp.3") are shown in Table 15. Here, the P1 scores are lower than those of P2-P4, regardless of the opponent's program. Therefore, Algorithm III reduces the score of the target player, as intended. The P5 score is low.

Table 16 shows the average P1 and P5 scores in each experiment. Both the P1 and the P5 scores rank highest to lowest in Base, Exp.3, and Exp.1. This is because of the difference in the way Algorithms I and III treat their respective scores. Algorithm III attempts to increase its own score, while reducing the score of the target player (P1). Therefore, it is less efficient than both the original Snowl and Algorithm I.

To examine the algorithm, we calculated the percentages of the ranks of Snowl-i (P5) in each game (see Table 17), and the average score of each of P1 and P2 for every Snowl-i (p5) rank (see Table 18). As evident from the low scores, Snowl-i does not have a higher rank. Moreover, the value of P2-P1 is large when a player's rank is Heimin. There are several possible reasons for these results. 1) If Snowl-i has a high rank and strong cards, then it has no effect. 2) If Snowl-i has a high rank and strong cards, it must play the strong cards, in which case, the opponent cannot do anything. 3) If Snowl-i has a low rank and weak cards, it cannot control the game. Consequently, we estimate that Heimin, with a wide range of cards, is most able to serve this purpose. These results show that each algorithm achieves

Table 13: Averages of score in the experiment using Snowl-i (Exp.1)

	P1	P2	P3	P4	Snowl-i
Match1(Default)	7392	9013	8986	9023	10584
Match2(Kou2)	8537	10026	10022	10032	6380
Match3(Snowl)	8798	10096	10076	10078	5949
Match4(Blau)	9021	10242	10256	10208	5270

Table 14: Averages of score in the experiment using Snowl-s (Exp.2)

	P1	P2	P3	P4	Snowl-s
Match1(Default)	11284	9628	9639	9608	4840
Match2(Kou2)	11043	10044	10044	10019	3850
Match3(Snowl)	11065	10118	10139	10130	3548
Match4(Blau)	10964	10144	10108	10161	3623

Table 15: Averages of score in the experiment using Jsnowl (Exp.3)

	P1	P2	P3	P4	Jsnowl
Match1(Default)	7639	8379	8373	8393	12214
Match2(Kou2)	8684	9452	9422	9427	8012
Match3(Snowl)	8855	9535	9523	9538	7548
Match4(Blau)	9195	9782	9805	9804	6411

its purpose.

5 Discussion

In Sections 3 and 4, we examined two new methods. In both cases, the objectives were achieved by changing only the method used to calculate the evaluation value.

This process may be more effective if we change other parts of the algorithm. Snowl, Wisteria, and Blau use evaluation functions and a log of parameters to control their behavior by learning in advance. The pre-learned parameters may depend on the evaluation value used in the Monte Carlo method. Therefore, in Method1, it is possible to obtain a more effective evaluation value by incorporating parameter relearning. Furthermore, in Method2, the simulation is based on the pre-learned parameters after the first hand. Therefore, P5 is acting on the first play and, thus, has a different purpose. By reflecting these differences in purpose in the evaluation function and parameters, the games may be more effective.

Lastly, the card exchange algorithm has not changed. In particular, in Method2, it is possible to determine which type of card exchange is best by examining which play is effective. If such a card exchange method becomes evident, the actions of Daihugo may be more effective than those of Heimin.

Table 16: Average score of P1 and P5

	P1			P5		
	Base	Exp.1	Exp.3	Base	Exp.1	Exp.3
Match1(default)	7954	7392	7639	13208	10584	12214
Match2(Kou2)	8930	8537	8684	9292	6380	8012
Match3(Snowl)	8994	8798	8855	9012	5949	7548
Match4(Blau)	9406	9021	9195	7338	5270	6411

Table 17: Ratio of ranks of Snowl-i (P5)

	Match1 (Default)	Match2 (Kou2)	Match3 (Snowl)	Match4 (Blau)
Daihugo	32.95	4.3	4.57	1.87
Hugo	23.83	10.39	9.35	6.32
Heimin	17.48	17.47	14.29	12.24
Hinmin	14.31	29.52	23.36	24.79
Daihinmin	11.41	38.29	48.4	54.76

Table 18: Average score of P1 and P2 according to ranks of Snowl-i

	Default		Kou2		Snowl		Blau	
Own Rank	P1	P2	P1	P2	P1	P2	P1	P2
Daihugo	2.41	2.78	2.58	2.94	2.62	2.95	2.72	3.01
Hugo	2.35	2.94	2.44	3.12	2.49	3.12	2.52	3.23
Heimin	2.41	3.15	2.53	3.36	2.56	3.38	2.60	3.44
Hinmin	2.56	3.24	2.85	3.39	2.88	3.42	2.97	3.43
Daihinmin	2.75	3.23	3.13	3.38	3.17	3.41	3.18	3.43

6 Conclusion

In this study, we proposed two new applications of the Monte Carlo method: a method for setting the evaluation value in repeated games, and a method for increasing or decreasing the score of a target player. Our experiments focused on specific situations. Therefore, further experiments are required, for other situations. The Monte Carlo method is expected to continue providing new applications. Furthermore, future research should clarify the detailed conditions under which play becomes effective.

References

[1] M. Wakatsuki, Y. Dobashi, T. Mitsuishi, S. Okubo, and T. Nishino, "Strengthening Methods of Computer Daihinmin Programs," *Proceedings of the CAINE 2017*,ISCA, pp.229–236, 2017.

[2] S. Okubo, T. Aayabe, and T. Nishino, "Cluster Analysis using N-gram Statistics for Daihinmin Programs and Performance Evaluations," *International Journal of Software Innovation (IJSI)*, vol. 4, Issue 2, pp. 33–57, 2016.

[3] S. Morita and K. Matsuzaki, "Proposal of Rating Algorithms Considering Inhomogeneity of Initial Hand in Daihinmin," *GI*, vol. 2014-GI-31, no. 14, pp. 1–5, 2014.

[4] T. Nishino and S. Okubo, "Computer Daihinmin(<Special Issue>Mind Games)," *Journal of Japanese Society for Artificial Intelligence*, vol. 24, no. 3, pp. 361–366, May 2009.

[5] M. Konishi, S. Okubo, M. Wakatsuki, and T. Nishino, "Decision Tree Analysis in Game Informatics," *5th International Conference on Applied Computing & Information Technology (ACIT2017)*, 2017.

[6] M. Wakatsuki, M. Fujimura, and T. Nishino, "A Decision Making Method Based on Society of Mind Theory in Multi-player Imperfect Information Games," *International Journal of Software Innovation (IJSI)*, vol. 4, Issue 2, pp. 58–70, 2016.

[7] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, vol. 47, no. 2, pp. 235–256, May 2002.

[8] D. Silver and G. Tesauro, "Monte-carlo Simulation Balancing," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML '09. ACM, 2009, pp. 945–952.

[9] F. Suto, K. Narisawa, and A. Shinohara, "Development of Client "Snowl" for Computer Daihinmin Convention," *Computer DAIHINMIN Symposium 2010*, 2010.

[10] K. Ohto and T. Tanaka, "Supervised Learning of Policy Function Based on Policy Gradients and Application to Monte Carlo Simulation in Daihinmin," *GI*, vol. 2016-GI-35, no. 10, pp. 1–8, Mar 2016.

[11] K. Tagashira and Y. Tajima, "Heuristics Implementation and Evaluations for Computer Daihinmin," *IPSJ Journal*, vol. 57, no. 11, pp. 2403–2413, Nov 2016.

[12] K. Tagashira, Y. Tajima, and G. Kikui, "Heuristics for Daihinmin and their Effectiveness," *International Journal of Computer and Information Science*, vol. 17, no. 2, pp. 7–14, Jul 2016.

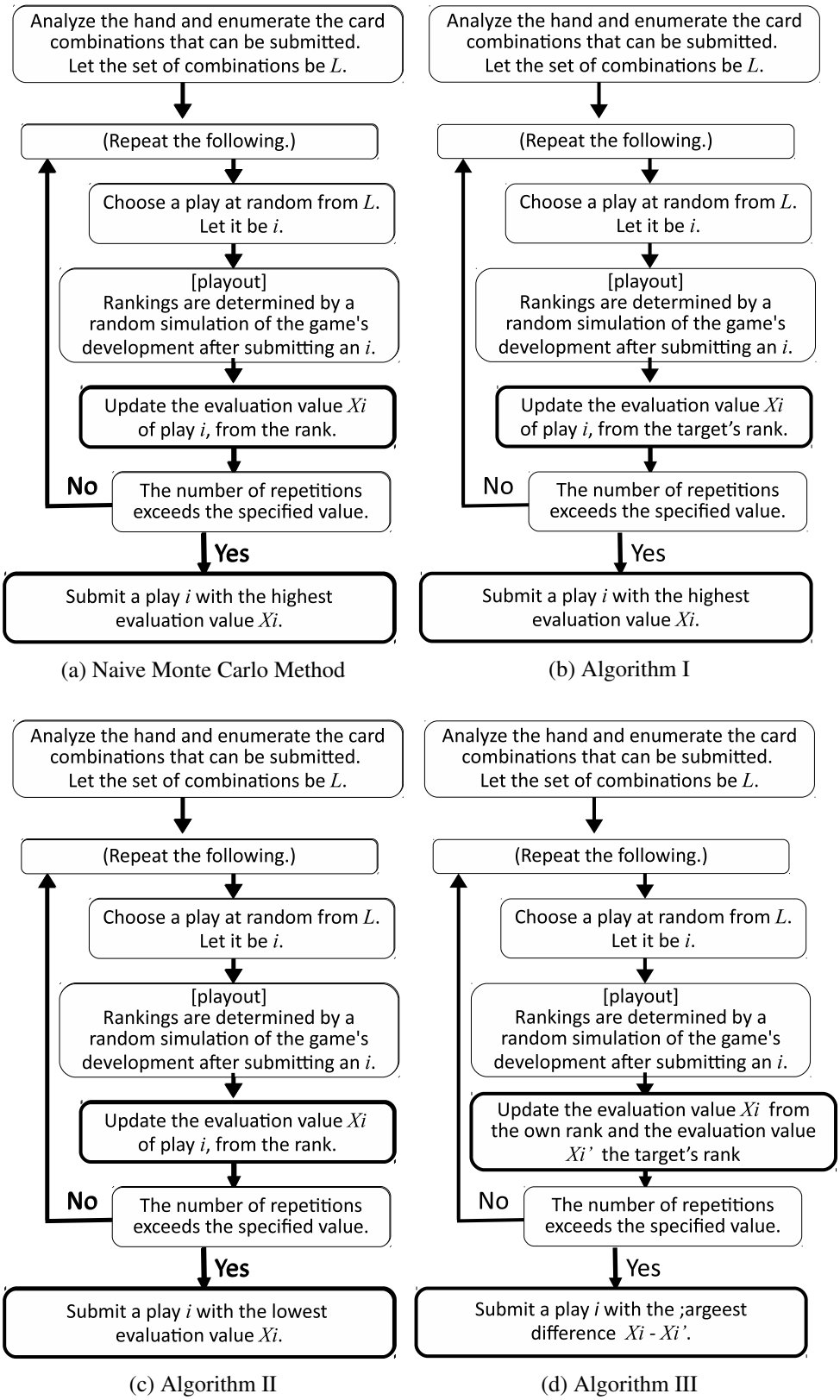

Figure 5: Flowchart of Each Algorithm