JAIST Repository
https://dspace.jaist.ac.jp/

| | |
|---|---|
| Title | Production of Various Strategies and Position Control for Monte-Carlo Go - Entertaining human players |
| Author(s) | Ikeda, Kokolo; Viennot, Simon |
| Citation | 2013 IEEE Conference on Computational Intelligence in Games (CIG): 1-8 |
| Issue Date | 2013-08 |
| Type | Conference Paper |
| Text version | author |
| URL | http://hdl.handle.net/10119/11646 |
| Rights | This is the author's version of the work. Copyright (C) 2013 IEEE. 2013 IEEE Conference on Computational Intelligence in Games (CIG), 2013, 1-8. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. |
| Description | |

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

# Production of Various Strategies and Position Control for Monte-Carlo Go - Entertaining human players

Kokolo Ikeda and Simon Viennot

***Abstract*—Thanks to the continued development of tree search algorithms, of more precise evaluation functions, and of faster hardware, computer Go and computer Shogi have now reached a level of strength sufficient for most amateur players. However, the research about entertaining and coaching human players of board games is still very limited. In this paper, we try first to define what are the requirements for entertaining human players in computer board games. Then, we describe the different approaches that we have experimented in the case of Monte-Carlo computer Go.**

## I. Introduction

In recent years, the strength of computer Go and computer Shogi has increased dramatically, thanks to improvements of the algorithms, for example Monte-Carlo Tree Search (MCTS) [2] based on a Bradley-Terry model in computer Go [3], or consultation algorithms in computer Shogi (Japanese Chess) [4]. In particular, these algorithms can use more efficiently multi-core processors and clusters of computers.

The best programs of Shogi have reached professional level, and the best programs of Go are now only 3 or 4 handicap stones weaker than professionals, which is already a sufficient strength to play even games against most amateur players.

For a long time, the most important problem of computer Shogi or computer Go was to improve the strength. This was the most straightforward goal of programs playing games, and it is still an important domain of research in games like Go where professional strength has not been reached yet. However, now that the strength of the programs have surpassed most human players, a new area of research has appeared where the goal is to entertain and teach human players, instead of only trying to beat them [1].

In areas not directly related to board games like Go or Shogi, programs "with natural behavior" or "able to entertain the player" are a topic of academic research. In particular publications are found every year in the international conference IEEE-CIG (Computer Intelligence and Games) [8]. For example, in FPS (First Person Shooter) games, "human naturalness" is the topic of a Turing contest since 2008, and a cash prize of 7000 dollars was awarded in 2012, for the two first AIs more human-like than the average human player. There are also competitions of "human-like AI" for "Super Mario Bros." or contests for creating game levels with a difficulty that will be best enjoyed by human players.

In the case of the game of Go, there are multiple international tournaments to compete for the strongest program, like the Computer Olympiad, the KGS bot tournament, the UEC-cup or the TCGA tournament. On the contrary, only a small number of tournaments have been organized so far to compete for naturalness of the moves [9] or entertainment [10]. One of the reason why research has been limited until now in this area is the difficulty of evaluating the level of "entertainment" or "teaching" that a program provides to human players.

In this paper, we list first in section II the requirements of a program able to entertain human players. Then, we describe how to implement some of these requirements in the case of a Monte-Carlo Go program: in section III, we describe a "gentle play" algorithm able to control the game and keep it balanced between the two players while avoiding unnatural moves. We evaluate it with various experiments in section IV. In section V, we present some simple methods to play with various strategies, and we evaluate them in section VI with human players. The results are still partial, but we hope that it will fuel further research in this direction.

## II. Requirements for playing entertaining games

Most of the ideas of this article could be applied to a general game, but we will restrict our discussion to the game of Go. It is an ancient board game, where players alternately put a (black or white) stone on the board, trying to enclose the widest possible area of the board. It is possible to capture the opponent stones by encircling them, leading to interesting local fights, called *semeai* and *life-and-death*. The game is particularly famous in Asia with millions of players and several professional leagues. Amateur players are ranked with a system of kyu and dan, with increasing levels of strength as follows: ... $<$ 10 kyu $<$ ... 1 kyu $<$ 1 dan $<$ ... $<$ 7 dan. Dan players are considered as strong players.

The game of Go allows players of very different strength to play games with almost equal chance of winning by the use of handicap stones. However, even with this handicap, it is common that the strongest player does not play for winning, but for teaching and entertaining the weakest player. This skill is in particular required when playing with children, because they need to win a fair number of games to maintain their motivation towards the game. This skill is also fairly different from being just strong at the game. For example, it is frequent that a strong human player is in fact not very good at playing entertaining games with children.

"Teaching games" usually refer to games played by a professional against an amateur player, in the purpose of teaching him the best way to play and improve his strength. Professionals have different approaches to teaching games,

Kokolo Ikeda and Simon Viennot are with Japan Advanced Institute of Science and Technology, email: kokolo@jaist.ac.jp

depending on what they consider as the most important part of the teaching. For example, some of them emphasize the "entertaining aspect" so that winning or losing is not the main goal of a teaching game, while others try to play as close as possible as usual even in teaching games, because there is a risk that the teaching game becomes artificial and loses its value [11].

In this article, we consider mainly players between 10 kyu and 1 dan level, when they play against a stronger amateur player or computer program, with handicaps smaller than what is needed to make the game even (or even no handicap at all). First, we list what we consider as the program requirements in order to play such a kind of "teaching" or "entertaining game". They are based on our experience and also on discussions with a large range of players, from novice to professional players.

### A. Rq-A. Acquiring an opponent model

A strong human player is able to tell roughly the strength of a weaker player with around 20 to 30 moves, including his ability to read the future moves, his precision, his knowledge of shape and understanding of the global position. This ability of strong human players is a kind of "online" information acquirement about the opponent during the game. It is also possible to obtain this information offline, from game records of the player if they are available, or directly from the player, if he knows his kyu or dan rank. This information about the player knowledge is important to implement efficiently some of the other requirements.

A central question in order to play entertaining games is to identify what aspects of a game are entertaining. There are in fact a lot of possible players, which different views on this question:

- Players having fun in winning, or on the contrary players that want to play good moves, without emphasis on the win or loss result
- Players that do not want to play against an opponent not using his full strength
- Players who like peaceful games without fight, or players who find such games boring
- Players who prefer fast games, and players who prefer slow games
- Players who want to play always a particular strategy, and players who want to play a different strategy each time
- There are also players who prefer watching a game, or commenting it, instead of playing

As this list shows, there are a lot of different players, with quite different entertaining needs. Some of them can be guessed from the player moves during the game, but some others need the player to explain in some way what he wants.

In this article, we try to address the case of an average player (without defining it precisely), based on our own experience of the game. An interesting direction of future research would be to address the needs of some specific kind of players.

### B. Rq-B. Controlling the game position

For a lot of amateur players, in particular children, winning is the main motivation towards games. A grandmaster of Shogi said that it is adequate for a professional player to win 2 games and lose 13 ones when playing with children after a child competition [12]. Since the handicap is usually less than it should be to compensate the gap of strength between the professional and the children, it means that professional players use some kind of "gentle moves" in these teaching games.

Figure 1 shows some possible evolutions of the winning ratio in a teaching game, from the stronger player perspective. If crushing the opponent (a) is not recommended, there is also no fun in winning if the stronger player offered no resistance at all (b), or played some clearly bad moves (c) only to balance the game. The stronger player must control the position with slightly under-optimal moves (d), play some risky moves that make the game result unclear, or even take the lead temporarily but with possibilities of comeback for the weaker player (e).

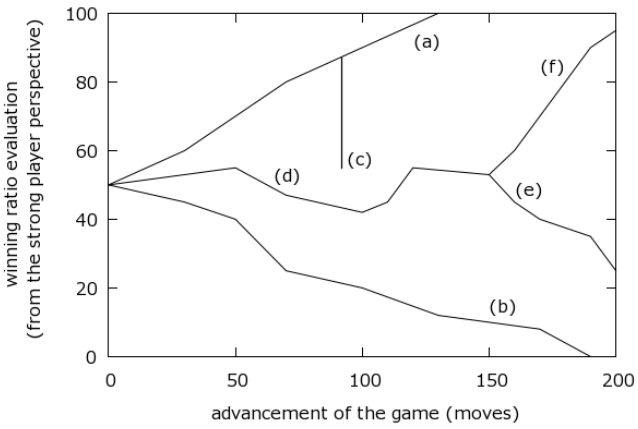


Fig. 1. Possible evolutions of the winning ratio in a teaching game.

The most important part is that the weaker player should not realize that the game is controlled, so that he can think he won because he played well. The stronger player must adapt his moves to the level of the weaker player, and for example win the game (f) if the weaker player made some obvious mistake.

### C. Rq-C. Avoiding unnatural moves

Even when the weaker player know that the stronger player is not using all his strength, the stronger player should avoid unnatural moves. If some unnatural moves are played, the weaker player will not think that he won because he played well, and there is no more fun to win in that case.

The main problem with the concept of "natural" move is that it cannot be defined formally in a mathematical meaning, and it also depends on the player strength. With figure 2, we try to list the main possible reasons why a move can be considered "unnatural".

1) Bad shape move: some bad and good shapes (a local pattern of stones) exist in the game of Go. For

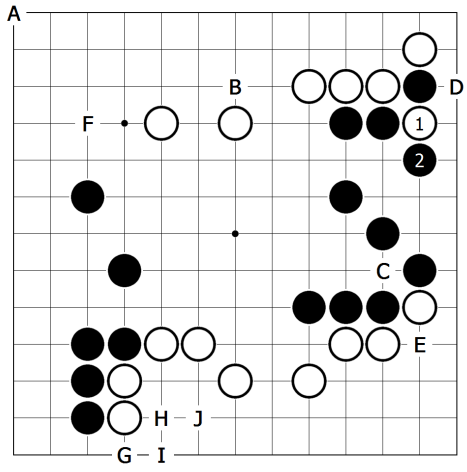


Fig. 2. Example of a game position to describe the problem of "natural moves".

example White A (edge position), B (local shape), or C (suicide move) are all bad shapes, even for novice players.

2) Unnatural move order: if we suppose that White 1 and Black 2 have just been played, White D is the usual continuation. White E is a move of roughly the same value as white D, but playing white E now would not look natural to the average player.
3) Clearly under-optimal moves: on figure 2, White F secures the wide top-left corner, which is much bigger than moves like White E. Playing White E only to make the game close can look unnatural to some players, especially the strongest ones.
4) Too high-level moves: there are also cases where a good move involves advanced knowledge of the game, and cannot be understood by weaker players. On figure 2, after Black G, White H (or maybe White I) would seem natural to most kyu-level players, but in fact, it is better to take a step back and play White J. This is related to the possible follow-up sequences for black and white and whether they keep the initiative or not (*sente* or *gote*). White J will possibly even be judged as an intentional bad move. This problem cannot be solved without an opponent modeling and an evaluation of the level needed to understand the possible moves.

### D. Rq-D. Using various strategies

When a strong player plays many games with the same player, using always the same strategies or style of play can be boring for the other player. To avoid this, in particular in the case of a computer program, it is effective to change the opening moves of the game (fuseki), but also to change the style play, aggressive vs defensive, pessimistic vs optimistic play for example.

In most commercial programs, it is usually possible to set the level of strength, but not the style of play. Some improvements in this area are possible (the possibility of choosing the style of play exists partially for commercial products of some games, like Mahjong or card games, but not so much for the game of Go).

### E. Rq-E. Thinking time and resign timing

The naturalness of the play does not concern only the chosen moves, but also the time used for choosing them and the resignation timing (when losing). Until recently, the resignation timing of computer Go programs was frequently too slow, frustrating the stronger players. This problem has partially improved due to a better understanding of *life-and-death*, *semeai*, or *seki* by the Go programs. However, against intermediate-level players, too early resigns or resigns in a close game are still a source of frustration, because most players of this level want to know the final score of a close game. MCTS based programs are usually set to resign when the expected winning ratio goes under a given value (for example 20%), which makes this problem quite frequent.

Lastly, the time used for choosing the move is an important factor of the fun for the players. Computer programs often use a fixed amount of time per move, which is boring when the next move is clearly obvious. And on the contrary, if the stronger player uses almost no time to play in a difficult position, it can hurt the feelings of the weaker player, who feels like he is not a worthy opponent.

Compared to other requirements, we believe that Rq-E can be implemented more easily, and already is in some programs. For example, it is possible to use the expected final score instead of only the winning ratio to decide the best resignation timing. It is also possible to play quickly obvious moves by checking the search progress at regular intervals, and stopping it earlier when some move is clearly identified as the best for some criteria.

### F. Rq-F. Comments about the game

One more important aspect of a game between real human players is the ability to make comments during or after the game. Even on the internet, where some players prefer to avoid talks with other players, comments after the game are still considered as part of the fun.

For example, a lot of weaker players are eager to hear comments from stronger players about their play, what was good or not, what other variations were possible, etc. It seems achievable to detect the good and bad moves of a player with a program, but it should be noted that in the case of the game of Go, one more difficulty comes from the fact that comments are usually exchanged with a specific vocabulary of shapes and goals (*tsuke*, *hane*, *nobi*, *atari*, ), instead of only the coordinates like (7, 4). Using if-then rules is sufficient to treat some simple cases, but distinction of advanced cases would probably require some machine learning.

In this article, our main interest is to show that three of these requirements can be implemented relatively easily in a computer Go program : controlling the game position (Rq-B), avoiding unnatural moves (Rq-C) and using various strategies (Rq-D).

## III. Position control in the case of Monte-Carlo Go

In this section, we focus on the problem of controlling the game (Rq-B) by keeping the expected winning ratio inside an ideal interval, while avoiding unnatural moves (Rq-C).

There is a tight relationship between controlling the game position and avoiding unnatural moves, which justifies to consider them simultaneously. It would be easy to control the game if we were allowed to play both very good moves and very bad moves, but as discussed in (Rq-C), moves like (c) on Figure 1 should be avoided. What is interesting and difficult is to control the game *without* playing obviously bad or unnatural moves, in the way of (d), (e), (f) on Figure 1.

### A. Computer Go

Since the introduction of Monte-Carlo Tree Search (MCTS) algorithms, and in particular the Upper Confidence Tree in 2006 [2], computer Go has shown remarkable progress. The general idea is to evaluate the leaf nodes of a search tree with random simulations of the game. The *estimated winning ratio* of a node (or a move) is the average percentage of the simulations starting from this node that are winning.

The main enhancement to MCTS is to perform *realistic* simulations of the game instead of random ones. A largely used model is the Bradley-Terry model [3] that allows the program to learn the moves of strong players in game records. The output of this model is the *static selection probability*, which reflects the probability that a move will be played by strong players in a given situation. It is used to perform realistic simulations, but also to prune the legal moves of the search tree [3] or to add some bias in the search [5], [13]. It is computed from a set of features, like "local patterns of stones", "the distance to the edge of the board", "stones in risk of capture", etc.

Our implementation of the MCTS algorithm and the Bradley-Terry model is a program called *Nomitan*. It reached a rank of 2 dan on the KGS server, under the account *nomibot*, thanks in part to the recent improvements of the search bias [13]. KGS is an international Go server with many players from novice to expert level, where programs are allowed to play games against the human players.

### B. A method to control the winning ratio

There are two main possible strategies to play under-optimal moves against a weaker player, with both advantages and drawbacks.

- Play always at a weaker level: for example, we can decrease the tree search time, and always choose the best move. However, this is possible only if we know the strength of the target player (Rq-A).
- Play strongly or gently depending on the overall advantage: for example, if the computer program has already the advantage on the board, play loosely and gently, and on the contrary, if the program is in a losing situation, play as strongly as possible. The difficulty is that the player will find the game strange if the computer plays a very bad move after a sequence of very good ones.

We are interested in the second method, and the goal of this paper is to design an algorithm that respects Rq-C 1) of avoiding bad shape moves (too low static selection probability), but also Rq-C 3), of avoiding clearly under-optimal moves (too low winning ratio compared to the best move). In other words, when we choose under-optimal moves, we need to voluntarily select moves with a not-so-bad shape and a not-so-bad winning ratio. We propose the following algorithm.

I. *Search*. First, we search the game tree with the usual MCTS algorithm existing in the target program. However, we take care to block the program from searching only a small set of moves, or to focus on only the best moves. This is an important characteristic of a program that plays a lot of gentle under-optimal moves. At the end of the search, we sort the potential moves in decreasing order of the estimated winning ratio.

II. *Case of a unique possible move*. If the difference between the winning ratio of the best move and the second best move is greater than a parameter $T_{uniq}$ (for example more than a 10% gap), we play the best move in order to fulfill the Rq-C 3) of avoiding clearly under-optimal moves.

III. *Case of low winning ratio*. If the winning ratio of the best move is under a parameter $T_{min}$ (for example 30%), we play the best move. This prevents the program from losing without any resistance, as required in Rq-B.

IV. *Case of intermediate winning ratio*. If the winning ratio of the best move is above $T_{min}$ but under $T_{max}$ (for example 45%), we restrict the selection to moves with at least a $T_{dif} = 5\%$ gap of winning ratio with the best move, and in these moves, we choose the move with the highest selection probability for the static evaluation. Since the winning ratio is in the target "control range", we choose a not-so-bad move which seems natural.

V. *Case of high winning ratio*. If the winning ratio of the best move is above $T_{max}$, the program is out of the ideal winning ratio control range. In order to decrease the advantage as fast as possible, we select the worst move for the winning ratio, but inside the moves that have a not-so-bad selection probability for the static evaluation. If such a move does not exist, we just play the best move. To keep naturalness while decreasing the winning ratio, we define a policy where a move needs to have a bigger and bigger static selection probability (i.e. being more and more natural) if the winning ratio gap with the best move is bigger. For example:

i. For a winning ratio gap under 3%, the static selection probability must be over 5%
ii. For a winning ratio gap from 3% to 4%, the static selection probability must be over 10%
iii. For a winning ratio gap from 4% to 6%, the static selection probability must be over 20%
iv. For a winning ratio gap from 6% to 8%, the static selection probability must be over 40%

We give an example in Table 1. Since the difference of winning ratio between the best move and the second best

TABLE I. EXAMPLE OF SEARCHED MOVES, WINNING RATIO AND STATIC SELECTION PROBABILITY

| Rank | Move | Winning ratio | Selection probability |
|---|---|---|---|
| 1 | A | 54% | 0.15 |
| 2 | B | 51% | 0.25 |
| 3 | C | 49% | 0.15 |
| 4 | D | 48% | 0.25 |
| 5 | E | 38% | 0.30 |

move is 3%, if $T_{uniq} = 10\%$ the condition II is not fulfilled. If we consider $T_{max} = 60\%$ and $T_{dif} = 5\%$, condition IV would be hit, and we would choose the move with the highest static selection probability inside A, B, C, so B would be chosen. If we consider $T_{max} = 45\%$, condition V is hit, B fulfills conditions i and ii, D fulfills condition iii, C and E do not fulfill any condition. So, we would choose the worst winning ratio between B and D, and D would be chosen.

## IV. EVALUATION OF THE POSITION CONTROL

We try now to evaluate the "gentle play algorithm" presented in section III-B. First, we check that it correctly limits the strength of the program, and then we check that it is doing so without playing unnatural moves.

### A. Evaluation of the strength control

To evaluate the strength control, we have used 5 different program settings.

a) default (strong) program
b) program made weaker by using short thinking time (only 8% of the time of the default settings above)
c) program made weaker by algorithm III-B, mild effect of gentle play
d) intermediate effect of gentle play
e) strong effect of gentle play.

The parameters are $T_{uniq} = 0.08c$, $T_{dif} = 0.03c$, $T_{min} = 0.35$, $T_{max} = 0.55$, and conditions V for the winning ratio gap are respectively $0.03c$, $0.04c$, $0.06c$, $0.08c$, with $c = 0.8$ for mild effect, $c = 1.5$ for intermediate effect, and $c = 2.5$ for strong effect.

For each program settings, around 100 games were played against humans of different strengths on the KGS server, in board size $13 \times 13$, with 5s/move (15s/move for the humans). We present in table II the number of wins and losses for different ranges of players.

TABLE II. RESULTS OF DIFFERENT PROGRAM SETTINGS AGAINST HUMAN PLAYERS (NUMBER OF WINS - LOSSES, WINNING RATE)

| | 2 dan and more | 1 dan - 2 kyu | 3 kyu and less |
|---|---|---|---|
| a | 17-5, 77% | 33-8, 80% | 44-4, 92% |
| b | 4-2, 67% | 24-22, 52% | 32-3, 91% |
| c | **4-10, 29%** | 39-36, 52% | 55-12, 82% |
| d | 1-18, 3% | **17-23, 42%** | 19-12, 61% |
| e | 0-14, 0% | 6-37, 14% | **22-39, 36%** |

When our program plays with its full strength (a), the winning rate[1] is around 77% even against the strongest range of players (more than 2 dan). Some games are lost to weaker players, revealing the weakness of our program in some particular *life-and-death* cases, and also caused by some inaccuracy in KGS player rankings. Next, it is important to note that even if we reduce a lot the search time (b), the winning rate against players weaker than 3 kyu is unchanged. Decreasing even more the thinking time is not reasonable, because it would cause a very shallow reading and some obvious local mistakes would be done.

When we use the proposed method of gentle play with a mild effect (c), our program is able to lose a fair amount of games against 2 dan or stronger players, but it is still winning most games against 3 kyu and less players. The winning rate against this weakest range of players decrease clearly when we make the effect intermediate, reaching even only 36% when the effect is strong. If we know in advance the level of the opponent (as described in Rq-A), it is possible to set the ideal level of gentle play, according to the bold cells of the table. Unfortunately, on the KGS server, the rank of the human opponents is not transmitted to the programs.

### B. Example of game without unnatural moves

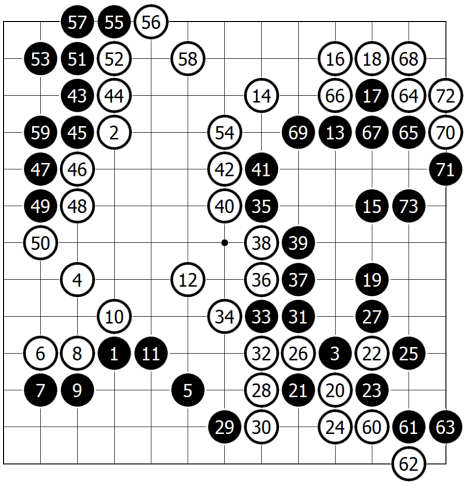


Fig. 3. Black 8k player vs our program playing gently.

Figure 3 shows a game where an 8 kyu player (black) was able to win by 6.5 points against our program playing gently. A lot of under-optimal moves can be found like white 54, but thanks to the proposed "gentle play" method, white played no fatal mistake and no clearly bad moves. In the next section IV-C, we check more thoroughly with questionnaires if the moves seem natural to the average human player.

### C. Evaluation of the naturalness of the moves

First, we have prepared an old and weak version of our program, which will be used as a weak opponent reference. Against the current version of our program playing with its full strength, the winning rate is only 5%. Then we have played $n$ games (limited to 60 moves) between this old weak version and the current version playing the "gentle moves" proposed in section III-B, obtaining a set $A$ of game records. We have also prepared a second set $B$ of game records played between the old weak version and a "naive gentle

[1] not to be confused with the winning ratio of Monte-Carlo algorithms

move" version less sophisticated than the proposed method, in particular the static selection probability is not taken into account.

The human subjects were given one game record from the set $A$ and one from the set $B$ and had 15 minutes to review them freely. They were told in advance that Black is a "weak player" and White a "strong player playing gentle moves", but not that different algorithms were used for the White player of the two games. Then, they were asked to list the white moves that look unnatural in each game record.

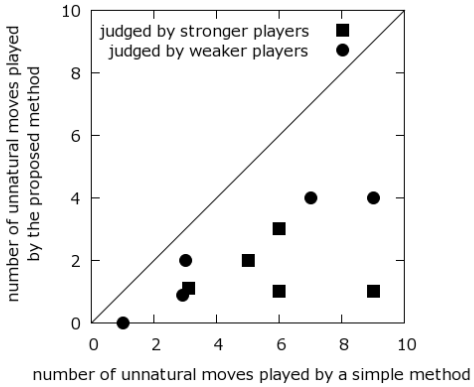


Fig. 4. Comparison of the number of unnatural moves between the proposed method and a simple method.

Figure 4 shows the result of an experiment with 5 strong players (3 kyu to 4 dan), and 5 novice to intermediate players (15 kyu to 7 kyu). Despite the fact that naturalness is something difficult to define and different for each player, all players felt that the program using the proposed method played less unnatural moves. The average value is 1.9 unnatural moves for the proposed method, less than half of the 5.2 average for a naive method.

With the results of Table II, we can conclude that the position control method of section III-B allows the program to limit its strength and lose against weaker players, without playing too much unnatural moves.

## V. Playing various strategies in the case of Monte-Carlo Go

In order to force a program to play with a given strategy, we can learn this strategy from game records, or add some specific features to the program, like it was proposed for the game of Shogi [6]. In this section, we present first two methods that are easy to implement in any Monte-Carlo Go program, one to enhance artificially the importance of the center of the board or of the corners, and one to create an optimistic or pessimistic behavior. Then, we present a third method to play preferentially moves close or far from the opponent last move, but this more sophisticated method can be implemented only in a program already using some kind of static evaluation function.

### A. Territorial versus Influence Play

When we count the points of each player after the simulation has reached the final position, it is of course usual to count 1 point for each open intersection controlled by a given player. But it is very simple to change this rule and add some weight, for example count an intersection in the corner 1.5 point, and an intersection in the center 0.5 point. By doing this, it is possible that what should have been counted as a loss will in fact be counted as a win, for example if a lot of points where taken in the corners. One could think that such a tweak of what is counted as a win or a loss would completely disturb the search algorithm, but in fact the result is that the program simply starts to play a territory-oriented strategy (taking corner territory) or an influence-oriented strategy (taking the center). The details of the method are given below.

I. Set the parameter $\alpha$ that indicates the relative importance of the center, and the parameter $n_{max}$ that limits the influence when the game advances.
II. At the position of move $n$ ($n < n_{max}$), use the following weights when counting the territory result of a simulation:
   i. $1 - \alpha \cdot (1 - \frac{n}{n_{max}})$ from the first line to the third line of the board (usually considered the main place for territory in the game of Go)
   ii. 1 for the fourth line
   iii. $1 + \alpha \cdot (1 - \frac{n}{n_{max}})$ for lines above the fifth one (territory in the center).

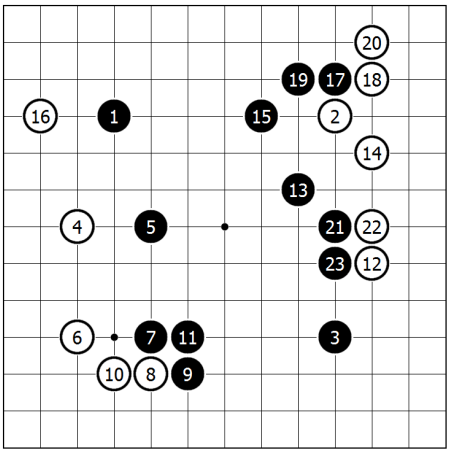


Fig. 5. Example of real game between territorial settings (white) vs influence settings (black).

The first line is the line at the edge of the board, and the second line the line next to it inside the board, etc. In the real game shown in Figure 5, Black uses $\alpha = +0.2$ (center-oriented), white uses $\alpha = -0.2$ (territory-oriented), and $n_{max} = 80$. We can see clearly that black prefers the center area while white prefers the corners and the edges.

Against the open-source Fuego program (version 1.1), the winning rate of our program is around 56% with the usual settings, 58% when using territory-oriented settings, and 46% with center-oriented settings. It shows that any of these settings is sufficiently strong for playing gently against intermediate level human players. A possible explanation for the increase of the winning rate with territory-oriented

settings is that the parameters of the Bradley-Terry model used in our program are learned in size $19 \times 19$, leading to an overestimated importance of the center in the smaller size $13 \times 13$. In fact, we are now considering using territory-oriented settings as our new default configuration in size $13 \times 13$ (for tournaments).

### B. Pessimistic versus optimistic

A lot of human players, whether amateur or professional, have a tendency to be either optimistic or pessimistic. Pessimistic players tend to think that they are losing, even if they are in fact winning, leading sometimes to useless and disastrous invasions to reverse the game. On the contrary, optimistic players tend to believe they are winning, even if they are in fact losing, leading to actions delayed too much, and good chances taken first by the opponent.

One way to reproduce these personality trends in a program is to artificially add (or remove) a virtual komi when counting the final territories. This is similar to the concept of Dynamic Komi that is used for example in Pachi open-source program [7] when the winning ratio of the simulations is too high or too low. The details of the method are given below.

I. Set the optimistic parameter $\beta$, and the parameter $n_{max}$ that limits the influence when the game advances.
II. At the position of move $n$ ($n < n_{max}$), add to the program side $\beta \cdot (1 - \frac{n}{n_{max}})$ points after adding the komi when counting the territory result of a simulation.

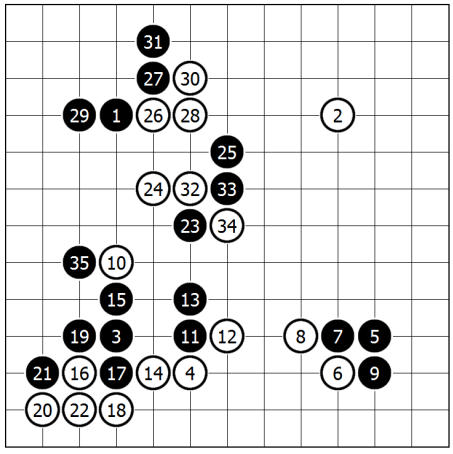


Fig. 6. Example of real game between optimistic settings (black) vs pessimistic settings (white).

Figure 6 shows a real game where Black uses $\beta = +10$ (optimistic settings), white uses $\beta = -10$ (pessimistic settings), and $n_{max} = 80$. White 10, 24, 32, 34 are "pushing forward moves" often seen with the pessimistic settings, and Black 19, 29, 35 are somewhat "slack moves" often seen with the optimistic settings. Against Fuego, the optimistic settings achieve 59% winning rate, pessimistic settings 53%, which are not particularly weaker than the default settings (56%). One possible reason why optimistic settings are better is because *life-and-death* is a weak point of our program, so it is better to be optimistic and avoid starting local fights.

### C. Far from the last move versus near to the last move

As we described in section III.A, a lot of programs, including our program, use the distance to the last move as a feature [3], [5].

The result of the machine learning is that a smaller distance to the last move is better, which gives a bigger static value to moves near the last played move. When this static value is used in Progressive Widening [3] or as a bonus in the UCB formula [5], moves around the last move are searched and selected more.

Interestingly, professional players have different preferences in relation to the distance to the last played move. For example, compared to the classical Japanese style, there is a tendency to play far from the last move in the Chinese and Korean style, with the effect of creating more simultaneous fights.

From this, we can expect that this tendency to play around the last opponent move can be used as a feature to create relatively easily different styles of play. In order to keep the implementation simple, we have not tried to modify directly the machine learning algorithms or the existing features of our program. Instead, we have introduced heuristics like "enhance close moves" or "enhance far moves" as a correction of the learned coefficients.

If we define:

- $f(s, a)$ the value of the evaluation function for a board state $s$, and a legal move $a$
- $last(s)$ the last move played in the board state $s$
- $dist(a_1, a_2)$ the distance between moves $a_1$ and $a_2$ [3]. $dist(a_1, a_2) = |a_{1,x} - a_{2,x}| + |a_{1,y} - a_{2,y}| + max(|a_{1,x} - a_{2,x}|, |a_{1,y} - a_{2,y}|)$
- $\kappa$ the correction parameter

Then, we write the corrected evaluation function as:

$$f_\kappa(s, a) = f(s, a) \cdot (dist(a, last(s)))^\kappa \quad (1)$$

When $\kappa$ is positive, the corrected evaluation is bigger for bigger values of $dist(a, last(s))$ which means that moves far from the last move will be selected more. On the contrary, when $\kappa$ is negative, the search will give (relative) priority to moves close to the last move.

In some preliminary experiments, the use of $\kappa$ in all the nodes of the search tree caused a big loss of strength in the program, in particular positive values (where the search around the opponent last move is limited). To keep the program reasonably strong, we have limited the use of the corrected evaluation function to the root node.

Figure 7 is a self-play game where both players use $\kappa = +3$. We can see that the players have a tendency to play independently of each other, all around the board. With $\kappa = +3$, the winning rate is 39% against Fuego. Our program is a bit weaker with this corrected evaluation function, but still sufficiently strong for the goal of playing against weaker players with various strategies.

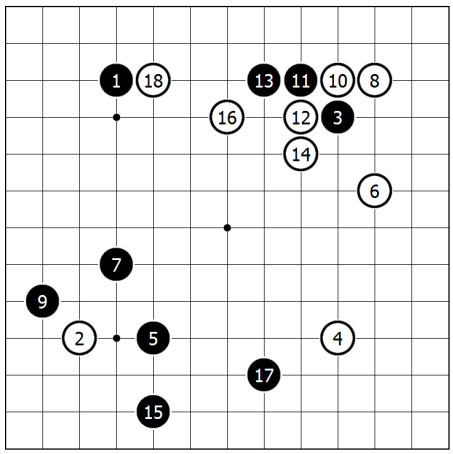


Fig. 7. Example of real self-play game with $\kappa = +3$.

## VI. Evaluation of the styles of play

We have performed the following experiment to check if human players are able to distinguish the difference of styles.

First, we have created a set of game records in size $13 \times 13$, by playing games until 60 moves (3s/move on a fast machine, which implies that the program is of strong amateur level), with:

(a) $\alpha > 0$ vs $\alpha < 0$
(b) $\beta > 0$ vs $\beta < 0$
(c) $\kappa > 0$ vs $\kappa < 0$
(d) default vs default

A positive value of the parameters $\alpha$, $\beta$ or $\kappa$ activates one of the styles of play described in section V, and a negative value activates the corresponding opposite style.

The human subjects were given one game from each set (a) to (d), and 30 minutes to review them freely. They were told in advance that Black and White use exactly the same strategy, or opposite ones, depending on the game. They were not told what are the possible strategies or how many games use opposite ones. Then, they were asked to tell which games were played with opposite strategies, and in that case, to describe what is this difference.

With the same subjects as the experiment of IV-C, we obtained the following results:

- All subject players found that (a) was played with opposite styles of play. Moreover, 8 of the 10 subjects found correctly that the difference was related to "territorial vs influence".
- Only 4 of the 10 subjects (3 of the 5 strongest players, and 1 of 5 weakest players) felt the difference of (b). Since Optimistic vs Pessimistic is an abstract concept, the influence on the game is probably difficult to feel for less advanced players.
- 6 of the 10 subjects felt correctly the difference of style in (c). Some players explained later that they did not expect such kind of style difference. The result will probably be better if players knew in advance what styles are possible.
- 7 players answered correctly that (d) is played with the same style, but 3 players incorrectly answered that they felt a difference "aggressive vs protective". In fact, such a difference can appear naturally in a given game, depending on the game development.

We can conclude that it is possible to force a program to use the proposed style of plays (a) and (c) in a way clearly felt by human players, with only a small loss of strength.

## VII. Conclusion

In this article, we have first listed the different requirements for a program that would entertain human Go players. Then, we have described concrete approaches for three of these requirements, first how to play natural moves while controlling the expected winning ratio of the game, and then how to play with various strategies.

The "gentle play" method described in this paper displayed promising results on the KGS server against human players. In particular, it seemed more adequate than simply decreasing the thinking time, in order to limit the strength of the program. Questionnaires with subjects also showed that this method correctly avoids playing unnatural moves.

An interesting complement to this research could be an experiment to evaluate if human players find the combination of "gentle play" and various strategies more entertaining than a program that takes only the strength into account.

## References

[1] Hiroyuki Iida and K Handa, "Tutoring Strategies in Game-Tree Search", *ICGA Journal*, pp. 191-204, (1995)

[2] Levente Kocsis, Csaba Szepesvari, "Bandit based Monte-Carlo Planning", 17th European conference on Machine Learning (2006)

[3] Remi Coulom, "Computing Elo Ratings of Move Patterns in the Game of Go", *ICGA Workshop*, (2007)

[4] Takuya Obata, Takuya Sugiyama, Kunihito Hoki, Takeshi Ito, "Consultation Algorithm for Computer Shogi: Move Decisions by Majority", Computers and Games, (2010)

[5] Shih Chieh Huang, "New Heuristics for Monte Carlo Tree Search applied to the Game of Go", Ph.D. Thesis, National Taiwan Normal University (2011)

[6] Ryuji Takise and Tetsuro Tanaka, "Development of entering-king oriented shogi programs", 16th Game Programming Workshop, pp. 25-31 (2011)

[7] Petr Baudis, "Balancing MCTS by Dynamically Adjusting the Komi Value", *ICGA Journal*, 2011

[8] IEEE-CIG competitions, http://geneura.ugr.es/cig2012/competitions.html

[9] Jaist Cup 2011, $9 \times 9$ Turing-test competition, http://www.jaist.ac.jp/jaistcup2011/index-e.htm

[10] Jaist Cup 2012, $9 \times 9$ Entertainment Go Contest, http://www.jaist.ac.jp/jaistcup/2012/jc/eng/13roalg-eng.html

[11] Hirofumi Ohashi, professional Go player, personal communication (2012)

[12] Kunio Yonenaga, professional Shogi player, personal communication (2012)

[13] Kokolo Ikeda, Simon Viennot, "Efficiency of Static Knowledge Bias in Monte-Carlo Tree Search", Computer and Games (2013), to be published