Combining Tactical Search and Monte-Carlo in the Game of Go

Tristan Cazenave
Labo IA, Université Paris 8
2 rue de la liberté
93526, St-Denis, France
cazenave@ai.univ-paris8.fr

Bernard Helmstetter
Labo IA, Université Paris 8
2 rue de la liberté
93526, St-Denis, France
bh@ai.univ-paris8.fr

Abstract- We present a way to integrate search and Monte-Carlo methods in the game of Go. Our program uses search to find the status of tactical goals, builds groups, selects interesting goals, and computes statistics on the realization of tactical goals during the random games. The mean score of the random games where a selected tactical goal has been reached and the mean score of the random games where it has failed are computed. They are used to evaluate the selected goals. Experimental results attest that combining search and Monte-Carlo significantly improves the playing level.

1 Introduction

Monte-Carlo Go has been invented in 1993 [1]; it is a simple way to program a decent computer Go program using very little knowledge. It has been recently the subject of renewed interest [2, 3, 4]. The combination of Monte-Carlo with traditional Go programming techniques is promising and gives good results, as can be seen from recent computer Go events. In this paper we show that an original combination of Monte-Carlo methods with tactical search outperforms Monte-Carlo alone. The resulting program is about 50 points above standard Monte-Carlo on the 9x9 board, and 26 points above the previous version of Golois.

The program starts with performing searches for each possible tactical goal and for each color starting first, in order to find unsettled problems. Examples of tactical goals are capturing a string, connecting two strings or making an eye. In a second phase, the program selects interesting goals related to unsettled problems. In a third phase, it computes statistics on the selected goals.

In standard Monte-Carlo Go, the means of the random games where an intersection has been played first by a player are computed for each intersection. What is done for the intersections can also be done for tactical goals. Therefore, we define the following unification of the notion of a goal: a goal can be either related to an empty intersection (in which case the success of the goal depends only on who has played first on the intersection), or it can be related to a tactical goal. We handle these two different classes of goals in a similar way: we compute the mean of the results of the random games where the goal has been reached and the mean of the results of the random games where it has failed. The value of the goal is the difference between the two means. We choose the goal of highest value. If this goal is an intersection goal we play at the intersection; if it is a tactical goal we play a move that reaches the goal. In case several moves reach the goal, we choose the one having the highest intersection value.

The second section presents Monte-Carlo methods for games; the third section details the different search algorithms used in our program; the fourth section presents the statistics our program computes in the random games; the fifth section deals with the combination of Monte-Carlo and search; the sixth section presents experimental results.

2 Monte-Carlo methods and Games

Monte-Carlo methods in games use statistics on more or less random games in order to find the best move. The first application of Monte-Carlo methods to Go was written by B. Bruegmann [1]. Recently, other Go programs have started using it, and improved it, simplifying the method and proposing basic improvements [2] or combining it with a knowledge based program that selects a few number of moves that are later evaluated by the Monte-Carlo method [3].

There are several slightly different ways to write a Monte-Carlo Go program [2]. In this paper, we call standard Monte-Carlo Go the following algorithm. The program plays a large number (usually 1,000 to 10,000) of random games starting at the current position. The moves of the random games are chosen almost randomly among the legal moves, except that they must not fill the player's eyes. A player passes in a random game when his only legal moves are on his own eyes. The game ends when both players pass. In the end of each random game, the score of the game is computed using Chinese rules (in our case, it consists in counting one point for each stone and each eye of the player's color, and subtracting the player's count to its opponent count). The program computes, for each intersection, the mean results of the random games where it has been played first by one player, and the mean for the other player. The value of a move is the difference between the two means. The program plays the move with the highest value.

Monte-Carlo simulations have also been used in other games such as Bridge [5], Poker [6], Tarok [7] and Scrabble [8] for example. In the games of Bridge and Tarok, a combination of Monte-Carlo and search is usual. Statistics are performed on open deals that are solved by search. However, in our approach, searches are performed only once, independently of the random games, and the tactical problems that are solved by search are used to choose the statistics that will be computed during the random games.

Our approach of combining search and Monte-Carlo is new and orthogonal to the previous approaches used in Go: it is very likely that it also improves their performances.

3 Search

We use search algorithms to solve tactical problems such as capturing/saving a string or connecting/disconnecting two strings. In this section, we present five tactical search algorithms that are used in our program. The first one is a capture search, the second one is a connection search, we follow with search for the connection between a string and an empty intersection, eye search and life and death search.

3.1 Capture Search

For each string on the board a capture search is tried. If the capture search fails, no other search is performed. If a capturing move is found, a search that tries to save the string, by playing first a move of the color of the string, is performed. If none of the possibly saving moves works, the string is captured even if the color of the string plays first, and capture searches for this string are stopped. On the contrary, if a saving move exists, other searches are performed. The program tries to find all the possible capturing moves, and all the possible saving moves.

At the end of the process, for each string on the board, its capture status and its save status are known, and for strings that can both be captured by one player and saved by the other, multiple capturing and saving moves are found when possible.

3.2 Connection Search

For each string on the board, the program looks for the strings that can be connected to the first string by playing at most four moves in a row. Then, for each pair of strings, it searches for a connection. When a connecting move is found, it also searches for a disconnection. When a disconnecting move is also found, it searches for all the connecting and disconnecting moves.

After connection search, the program has a list of possible connections, and for each connection the connection status and the disconnection status, as well as multiple connecting and disconnecting moves for strings that can both be connected by one player and disconnected by the other.

3.3 Empty Connection Search

The empty connection goal is based on the connection goal. It involves a string and an empty intersection. The goal of the game is to connect the empty intersection to the string. In practice, in order to find unsettled empty connection problems, the program plays a move of the color of the string on the empty intersection, then the disconnection search is called for the string containing the played empty intersection and the string to connect to. If they cannot be disconnected the empty connection problem is unsettled and the associated move consists in playing on the empty intersection.

3.4 Eye Search

For each intersection on the board, it searches if an eye can be made on the intersection or on any of the direct neighbors.

3.5 Life and Death Search

Life and death search uses Generalized Widening [9] for non enclosed groups. Life and death search is called for each group of strings. Groups are built using the results of the connection search between strings.

4 Statistics on random games

In this section, we describe the different statistics that are computed during random games. Statistics are related to goals. We compute the mean of the results of the random games where a goal has been reached and the mean of the results of the random games where it has failed.

Among the unsettled goals found by search, the program chooses some interesting ones and computes statistics on them.

The different types of goals we use for the statistics are :

- The goal of playing first on an intersection. It is the only goal used in the standard Monte-Carlo approach.
- The goal of owning an intersection at the end of a game.
- The goal of capturing a string. The goal is considered to be lost as soon as the string has more than four liberties.
- The goal of connecting two strings. It is possible that the two strings are captured after being connected in a random game, but we still consider this as a success for the connection.
- The goal of connecting a string and an empty intersection.
- The goal of making an eye on an intersection or on any of its neighbors.

5 Combining Search and Monte-Carlo

There are usually different problems in the initial position, i.e. capture, connection, empty connection, eyes and life and death problems. In this section we detail the combination of search and Monte-Carlo. We start explaining why all the problems are not taken into account and how we select the problems that will be used to compute statistics on. We follow with the statistics collected during the random games. We describe how a move is chosen. Then, we discuss on the usefulness of collecting statistics on unsettled problems, and we define positive and negative goals.

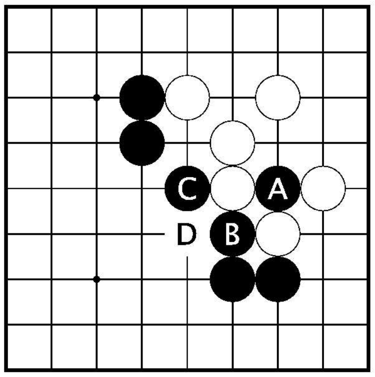


Figure 1: Choosing the simplest connection

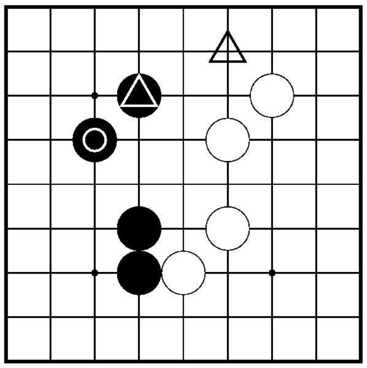


Figure 2: Empty connection

5.1 Selecting problems

In order to avoid playing bad moves and overestimating the importance of a problem, the program needs selecting among the unsettled problems the ones that will be used to compute statistics.

Strings that cannot be disconnected are amalgamated in groups. Groups are used to select the unsettled connection problems that will be used to gather statistics.

It can happen that a string A is connected with a complex search to string B, that string B is connected with a complex search to string C, and that A and C are connected but that the search to find it is too complex. In this case the program can think there is an unsettled connection between A and C and add a useless move. In order to avoid this behavior, complex unsettled connection problems between two strings of the same group are not taken into account.

Furthermore, when there are multiple connection problems that connect the same groups, only the simplest connection problem is retained. The simplicity of a connection problem is measured by the number of moves in a row that are needed to connect the two string in the initial position. For example, in the figure 1 the strings B and C belong to the same group. The connection between A and B can be prevented by White, capturing A for example. The connection between C and A can also be prevented by capturing A, but the white move at D also prevents the connection. However, D is not the kind of move that we want the program to consider to disconnect the black group from A. Here the connection between A and B is simpler than the connection between A and C, so the program will only retain the connection between A and B for gathering statistics. The problem of only retaining the simplest connection is not only a problem of avoiding moves like D, it is also a matter of correctly evaluating the value of the connection between two groups. The mean of the games where A and C have been connected is larger than the mean of the games where A and B have been connected. So keeping only the simplest connection avoids overestimating connections.

Empty connections between strings and empty intersections follow a similar pattern. In the figure 2, the triangle empty intersection can connect to the triangles string. However, it can also connect to the circled string. The mean of the games containing the connection to the circled string is higher than the mean of the games with the connection to the triangles string. In order to avoid overestimating empty connections, the program only selects the simplest empty connection when there are multiple empty connections of the same color to the same empty intersection.

Empty connections are also used to find long distance connections between groups that are too complex to be found by the connection search. If an empty intersection is connected to two different groups, and that there is no unsettled connection problem between these two groups, then a new unsettled connection problem is created that joins the two groups, playing on the common connected empty intersection. There are some exceptions to this rule that can be found using a search for transitivity of connection [10].

5.2 Gathering statistics on selected problems

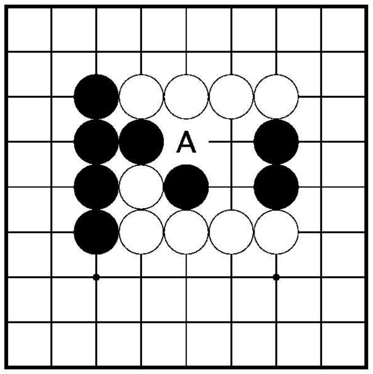
After the program has selected the interesting goals, it plays many random games and for each selected goal, it computes the mean score of the random games where the goal has succeeded, and the mean score of the random games where it has failed.

We call *raw mean* of a move the value of the intersection goal associated to the move.

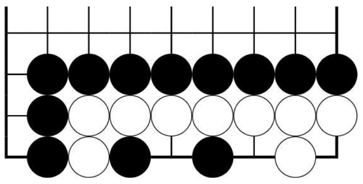
The statistics on the final color of an intersection are used to evaluate the importance of playing moves related to life and death. The mean score of a life problem related to a string is the mean score of the games which ended with an intersection of the string keeping its original color. The mean score of the associated death problem is the mean score of the games which ended with the intersection of the other color.

5.3 Choosing a move

For the selected goals, we compute the value of the goal which is the difference between the mean score of the games where it succeeds and the mean score of the games where it fails. The program chooses to play the goal with the highest value. Among all the moves associated to the selected goal, the program chooses the one with the highest raw mean.



useful connection



useless connection

Figure 3: connections

5.4 Why are statistics on unsettled problems useful?

It is useful to compute statistics in the random games on goals related to unsettled problems. The raw mean of a move is different from the mean of the games where the goal associated to the move has been reached. Figure 3 explains the difference between the two means for the connection goal. We see in the *useful connection* diagram that the move at A connects the two strings. In the random games where a black move at A has been played first, the two strings will only be connected three times out of four. But the program knows, with the connection search, that it is possible to always connect the two strings. The mean of the results of the games where the two strings have been connected give an evaluation of the move at A, which is better than the raw mean of the move at A because it takes into account the fact that the two strings are connected after A.

Another example is given in the *useless connection* diagram of figure 3. This time the games where the two black strings have been connected have a mean which is less than the mean of all the games since the connection move is always useless. Therefore the mean associated to the connection is less than the mean of the best move, and the program does not play the connection. It finds out by itself that it is a useless connection.

Figure 4 shows the symmetric example. The two white strings are disconnected three times out of eight, and the disconnecting move results in a disconnection of the two strings only three times out of four in the random games. On the contrary, the search tells us that they can always be disconnected. To evaluate the disconnecting move, it is more accurate to use the mean of the games where the two strings have been disconnected, than to use the mean of the games where the disconnecting move has been played first.

5.5 Positive and negative goals

We introduce the notion of positive and negative goals. Positive goals are goals we have confidence they can be reached if the search algorithm returns so. For example, when the search algorithm finds a capturing move, we have confidence that the string can be captured. Examples of positive goals are capture, connection and life. On the contrary

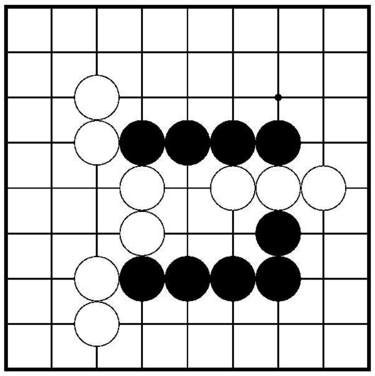


Figure 4: disconnection

negative goals are not sure. For example, when the capture search finds a move that saves a string, we are not assured that the string cannot be captured latter: the string might have gained five liberties but may be completely surrounded by alive enemy groups and therefore be bound to capture anyway. Example of negative goals are saving a string, disconnecting two strings or killing a group.

For negative goals the statistics tend to over-estimate the interest of playing in the related problems. For example, statistics on disconnection measure the mean of the games where two strings have not been connected. It means the program does not count the games where the strings have been disconnected according to the search at a time in the game, but have been connected later in the game because of the capture of a common adjacent string that had temporarily gained five liberties. In order to avoid this behavior we have used the connection game evaluation function to detect the disconnection found by the search at any time in the random games. This way we count for the disconnection mean the games were the evaluation function sends back lost in the game even if at the end of the game the two strings end being connected. Similarly, for the save goal, our program counts all the games where the string had more than four liberties during the game.

6 Experimental results

In order to test our combination of tactical search with Monte-Carlo, we have played 9x9 games between two programs. The first program is the standard Monte-Carlo algorithm. The second program combines tactical search and Monte-Carlo. They both play 10,000 random games before choosing a move. Each program played twenty 9x9 games against the other: ten games with black and ten games with white. The games were scored using Chinese rule.

The capture, connection and eye search use Generalized Threats Search [11]. The threat used is the (6,3,2,0) threat. Life and death search uses Generalized Widening [9] for non enclosed groups.

The program that combines search and Monte-Carlo wins on average by 52.1 points against the standard Monte-Carlo method on the 9x9 board, the standard deviation being 34.2 points. On a Celeron 1.8 GHz, the standard Monte-Carlo algorithm plays a move in five seconds on average for 10,000 9x9 random games. The combination of search and Monte-Carlo plays a move in ten seconds for 10,000 9x9 random games.

In order to compensate for the additional time used by the combination of Monte-Carlo and search, we played the combination program with 1,000 random games against the standard Monte-Carlo with 10,000 games. The combination is then twice as fast as the standard Monte-Carlo. Still, the combination program wins on average by 24.6 points on the 9x9 board, the standard deviation being 40 points.

The combination of Monte-Carlo and search with 10,000 random games has also been tested against Golois. Golois uses exactly the same tactical search algorithm as the program based on Monte-Carlo and search. Golois uses a depth one global search and hand tuned heuristics to evaluate the strength of groups and the moves. Forty 9x9 games have been played between the two programs and the combination of search and Monte-Carlo wins on average by 26 points. Given that the two programs have the same tactical Go knowledge, it appears that the use of Monte-Carlo to assess the importance of tactical goals is a promising alternative to hand tuned evaluation knowledge.

7 Conclusion and Future Work

We have presented a way to integrate search with the Monte-Carlo method in Go. Our program computes statistics during the random games on the goals searched in the initial position, in order to improve the accuracy of the evaluation of the moves related to the goal. The resulting program improves the average result of Monte-Carlo methods against a standard Monte-Carlo Go program by more than 50 points in 9x9 games.

Future work includes using search and statistics on other goals and combinations of goals. It may be possible to better handle the different results of the search for negative goals, and to improve the evaluation of these goals. We could also improve the confidence in the statistics on the random games, by taking into account the moves that threaten the won tactical goals, and replying them inside the random games so as to keep the important tactical goals won. This would prevent the program to overestimate the value of threats and it would result in a better evaluation of the position.

Bibliography

[1] Bruegmann, B.: Monte Carlo Go. ftp://ftp-igs.joyjoy.net/go/computer/mcgo.tex.z (1993)

[2] Bouzy, B., Helmstetter, B.: Monte Carlo Go developments. In: Advances in computer games 10, Kluwer (2003) 159–174

[3] Bouzy, B.: Associating domain-dependent knowledge and monte carlo approaches within a go program. In: Joint Conference on Information Sciences, Cary (2003) 505–508

[4] Bouzy, B.: Associating shallow and selective global tree search with monte carlo for 9x9 go. In: 4th Computer and Games Conference, Ramat-Gan (2004)

[5] Ginsberg, M.L.: GIB: Steps toward an expert-level bridge-playing program. In: IJCAI-99, Stockholm, Sweden (1999) 584–589

[6] Billings, D., Davidson, A., Schaeffer, J., Szafron, D.: The challenge of poker. Artificial Intelligence **134** (2002) 210–240

[7] Lustrek, M., Gams, M., Bratko, I.: A program for playing tarok. ICGA Journal **26** (2003) 190–197

[8] Sheppard, B.: Efficient control of selective simulations. ICGA Journal **27** (2004) 67–80

[9] Cazenave, T.: Generalized widening. In: ECAI 2004, Valencia, Spain, IOS Press (2004) 156–160

[10] Cazenave, T., Helmstetter, B.: Search for transitive connections. Information Sciences (2005)

[11] Cazenave, T.: A Generalized Threats Search Algorithm. In: Computers and Games 2002. Volume 2883 of Lecture Notes in Computer Science., Edmonton, Canada, Springer (2002) 75–87