Goal threats, temperature and Monte-Carlo Go

Tristan Cazenave

LIASD
Université Paris 8, Saint Denis, France
cazenave@ai.univ-paris8.fr

Abstract. Keeping the initiative, i.e. playing sente moves, is important in the game of Go. This paper presents a search algorithm for verifying that reaching a goal is sente on another goal. It also presents how goals are evaluated. The evaluations of the goals are based on statistics performed on almost random games. Related goals, such as goals and associated threatened goals, are linked together to form simple subgames. An approximation of the temperature is computed for each move that plays in a simple subgame. The move with the highest temperature is chosen. Experimental results show that using the method improves a Go program.

1 Introduction

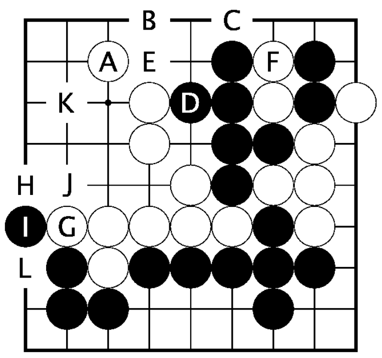


Fig. 1. Examples of connections and connection threats

In figure 1, if White plays at H, G and H are connected. Playing at H also threatens to connect H and L. However the connection between G and H is not sente on the connection between H and L: if White plays at H, Black answers at L, and White has to play at J to keep G and H connected. We say the connection between G and H is not sente on the connection between H and L.

On the contrary, for Black, the connection between I and J is sente on the connection between J and K. If Black plays at J, I and J are connected. Playing at J also threatens to connect J and K. The connection (I,J) is sente on the connection (J,K) because whatever White plays after Black J, either it does not threaten the connection between I and J, or it threatens it, but Black has an answer that both connects I and J, and keeps the threat of connecting J and K.

Moreover, all these connections are related. If White connects G and H, it prevents Black from connecting I and J. If Black connects I and J, it prevents White from connecting G and H. We aggregate them in a single structure in order to evaluate the White move at H and the Black move at J. If $Val_{(G,H)}$ is the evaluation of the connection (G,H), $Val_{(I,J)}$ the evaluation of the connection (I,J), and $Val_{(I,J),(J,K)}$ the evaluation of connecting both (I,J) and (J,K), we approximate the temperature of the White move at H with the temperature of the subgame $\{Val_{(G,H)}||\{Val_{(I,J)}\}|\{Val_{(I,J),(J,K)}\}\}$.

A common approach to Go programming is to compute the status of tactical goals. Examples of tactical goals are connecting two strings, capturing a string or making a group live. The status of a tactical goals is assessed using heuristic search. Once unsettled goals are found, they are evaluated and the one with the highest evaluation is played. Recently, I have shown how to evaluate unsettled goals using a Monte-Carlo approach [6]. It consists in evaluating an unsettled goal with the average of the random games where it has been reached. I build on this approach in this paper.

Besides finding the moves that play unsettled tactical goals, an important aspect of Go is to also find the tactical goals that are threatened by each move. In order to do this a program needs an algorithm that verifies a goal is sente on another goal. This algorithm is presented in this paper as a search algorithm. It uses search at each node of the main search to assess the statuses of the goal and of the associated threatened goal.

The evaluation of a threat uses the Monte-Carlo method, it consists in computing the average of all the random games where the goal and the threatened goal have been reached. Once the goals and the associated threats have been evaluated, they are aggregated in a single structure that is used to approximate the temperature of moves.

The rest of the paper is organized as follows. The second section discusses related work. The third section details the tactical goals used and how the program computes their status. The fourth section gives a search algorithm that verifies if reaching a goal is sente on another goal. The fifth section explains how goals are evaluated with a Monte-Carlo algorithm. The sixth section details the evaluation of moves given the evaluation of goals. The seventh section gives experimental results.

2 Related Work

The use of search for assessing dependencies between goals in the game of Go [7] is related to the search algorithm we present.

The evaluation of goals with Monte-Carlo Go [6] is related to the evaluation of goals and threatened goals in this paper.

Related goals are aggregated in a structure. The structure and the evaluation of goals are used to build a combinatorial game. Thermography [1] can be used to play in a sum of combinatorial games. In Go endgames, it has already been used to find better than professional play [10], relying on a computer assisted human analysis. A simple and efficient strategy based on thermography is Hotstrat, it consists in playing in the hottest game. Hotstrat competes well with other strategies on random games [3]. Another approach used to play in a sum of hot games is to use locally informed global search [9, 8]. In this paper, we use Hotstrat to evaluate the subgames built with goals evaluations.

3 Tactical goals

This section deals with tactical goals of the game of Go and the related search algorithms that compute the statuses of the goals. Examples of tactical goals are connecting two strings, or capturing a string. Traditional Go programs spend most of their time searching tactical goals. The search finds moves that reach the goals, and these moves are then used by Go programs to choose the best move according to the evaluation of the associated goals.

3.1 Possible goals

Goals that appear frequently in a Go game are the connection, the separation [5], the capture and the life. Goals are associated to evaluation functions that take values in the interval [Lost,Won]. Usually Won is a large integer and returning Won means the goal is reached, Lost is the opposite of Won and returning Lost means the goal cannot be reached. We make the distinction between positive and negative goals. A positive goal is well defined and when the associated evaluation function returns Won, it is certain that the goal is reached. Connection, separation, capture and life are positive goals. For example, the evaluation function for connections returns Won when the two stones to connect are part of the same string. Negative goals are the opposite of positive goals. The opposite of connection is disconnection, the opposite of separating is unseparating, the opposite of capturing is escaping and the opposite of living is killing. Negative goals are often ill-defined: when the evaluation returns Won for a negative goal it is not sure that it is reached. For example, the evaluation function returns Won for disconnections when the two strings to disconnect have a distance greater than four. However, there are cases when strings have a distance greater than four and can still be connected. Symmetrically, when the evaluation function returns Lost for a positive goal, it is not sure it cannot be reached.

It can be noted that positive and negative goals can also be mixed. For example for the connection goal, our algorithm also verifies that the string that contains the two intersections to connect cannot be simply captured.

The empty connection goal consists in finding if an empty intersection can be connected to a string.

3.2 Finding relevant goals

For each possible goal, the program assesses if it has chances to be reached. For connections, it selects pairs of strings that are at a distance less than four. For empty connections, for all strings, it selects liberties of order one to four, as well as liberties of adjacent strings that have less than three liberties. Once the goals are selected, the program uses search to assess their status.

3.3 Searching goals

The algorithm we use to search goals is the Generalized Threats Search algorithm [4]. It is fast and ensures that when a search with a positive goal returns Won the goal can be reached. For all possible goals, the program first searches if the color of the positive goal can reach the goal by playing first. If it is the case the program performs a second search to detect if the color of the positive goal can still reach the goal even if the opposite color starts playing. If both searches return Won or if the first search returns Lost, the goal is not unsettled and it is not necessary playing it. If the first search returns Won and the second search returns Lost then the goal is unsettled.

3.4 The traces

Two traces are associated to each search. The positive trace is a set of intersections that can possibly invalidate the result of a search that returns Won. The negative trace is a set of intersections that can possibly change the result of a search that returns Lost.

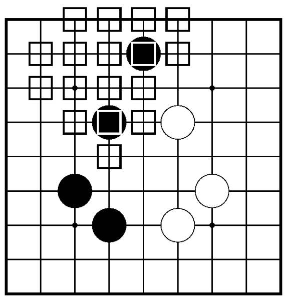


Fig. 2. Positive trace of a won connection

Figure 2 gives an example of a positive trace: the trace of the search where White tries to disconnect the two black stones but Black succeeds connecting them.

3.5 Finding all moves related to unsettled goals

In order to evaluate the threats associated to the moves that reach the goal, a program has to find all the moves that reach an unsettled goal, and all the moves that prevent from reaching it. The program currently uses heuristic functions to generate a set of moves that can possibly reach the goal. It could also use the negative trace of the search that returns Lost when the opposite color of the goal plays first. It uses the positive trace of the search that returns Won when the color of the goal plays first, in order to generate all the moves that can possibly prevent from reaching the goal. For each of these moves, it plays it and then performs a search to verify if it reaches the goal for moves of the goal color, or prevents from reaching it for moves of the opposite color.

3.6 Finding threatening moves for goals

It is interesting to look for threats in two cases. The first case is when the positive goal search returns Lost when the goal color plays first. In this case, all the moves of the color of the goal on the intersections of the negative trace are tried. For each move, it is played and a search for the goal with the color of the goal starting first is tried. If the search returns Won, then the move is a threat to reach the goal. The second case is when the goal search returns Won when the opposite color of the goal plays first. In this case all the moves of the opposite color of the goal, on the intersections of the positive trace are tried. For each move, it is played and a search for the goal with the opposite color playing first is performed. If the search does not return Won then the move is a threat to prevent from reaching the goal.

4 A search algorithm for verifying threats

This section describes how the potential threats are found and details the search algorithm used to verify that reaching a goal threatens another goal.

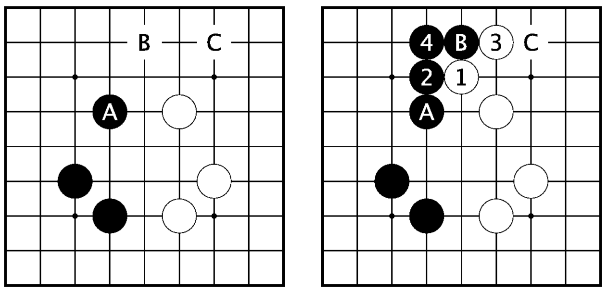
4.1 Finding moves and threats to search

For each move that reaches a goal and that is also a threat on another goal, a search for a threat is performed. The move is played and then the search algorithm for verifying threats is called at a min node.

For example in the problem fourteen of figure 5, the move at E reaches the goal of connecting A and B, and it is also a threat to disconnect C and D. So the White move at E is played and the search algorithm is called at a min node to verify that the connection of A and B threatens the disconnection of C and D. This is not always the case. For example in the problem seven, the Black move at B connects A and B, and the move at B threatens to connect B and C. However the connection of A and B is not sente on the connection of B and C, because when Black plays B, White answers at C both threatening to disconnect A and B, and removing the threat of connecting B to C.

4.2 The search algorithm

A simple idea that comes to mind when trying to model threats is to play two moves in a row and use search on a double goal [7] to verify if both goals are achieved even if the opponent moves first. However it does not work as can be seen in figure 3. In this figure, if Black plays at B and C in a row, White cannot prevent him from connecting A, B and C. However playing at B, connecting A and B, is not sente on the connection



An empty connection problem   White prevents the threat in sente

Fig. 3. Black B does not threaten Black C

of B and C. If Black plays at B, White can prevent the threat in sente as in the right diagram of figure 3.

As the search algorithm on double goals does not work, we have designed a new search algorithm that verifies that when a first goal can be reached, it also threatens a second goal. The first goal is called the *goal*, the threatened goal is called the *threat goal*.

The search algorithm is a selective Alpha-Beta where verifications of the status of goals are searched at each node.

The pseudo-code for max nodes is given below:

```
1.  MaxNode (int alpha, int beta) {
2.    search goal with max playing first;
3.    if (goal cannot be reached)
4.      return Lost;
5.    search goal with min playing first;
6.    if (goal can be reached with min playing first)
7.      return Won;
8.    if (threatGoal is Lost)
9.      return Lost;
10.   for max moves m that reach the goal {
11.     try move m;
12.     tmp = MinNode (alpha, beta);
13.     undo move;
14.     if (tmp > alpha) alpha = tmp;
15.     if (alpha >= beta) return alpha;
16.   }
17.   return alpha;
18. }
```

At max nodes, the first important thing is to verify that the goal can be reached (lines 1-4). If not, the search stops as it is necessary to reach the goal for the algorithm to send back Won. If the goal can be reached if min plays first, then Min has just played a move that does no threaten the goal, and has therefore lost the initiative. So Max has reached the goal and has kept the initiative (lines 5-7). In this case the algorithm returns Won because the goal has been reached keeping the initiative and keeping threatening the threat goal as it has been verified in the upper min node (cf. MinNode pseudo-code). If the goal is not reached and the threat goal is lost, then Min has succeeded in preventing Max from threatening the threat goal, so the algorithm returns Lost (lines 8-9). The only Max moves to try are the moves that reach the goal, so the algorithm tries them and calls the MinNode function after each of them (lines 10-16).

The code for the MinNode function is as follows:

```
1.  MinNode (int alpha, int beta) {
2.    search goal with max playing first;
3.    if (goal cannot be reached)
4.      return Lost;
5.    search goal with min playing first->traceMin;
6.    if (goal cannot be reached with min playing first)
7.      return Lost;
8.    search threatGoal with max playing first->traceMax;
9.    if (threatGoal cannot be reached)
10     return Lost;
11.   if (intersection of traceMin and traceMax is empty)
12.     return Won;
13.   for min moves m in traceMin and traceMax {
14.     try move m;
15.     tmp = MaxNode (alpha, beta);
16.     undo move;
17.     if (tmp < beta) beta = tmp;
18.     if (alpha >= beta) return beta;
19.   }
20.   return beta;
21. }
```

At min nodes, the program starts verifying Max can reach the goal if it plays first (lines 2-4), then it verifies that he can reach the goal even if Min plays first (lines 5-7). The positive trace of this search is memorized in traceMin as it will be useful later to select the Min moves to try. Then the program verifies if the threat goal can still be reached by Max in order to verify that Max keeps threatening it (lines 8-10). The positive trace of the threat goal is memorized in traceMax. The only moves that Min tries are on the intersection of TraceMin and TraceMax (lines 13-19).

For negative goals, the search is performed for the opposite positive goal, and the algorithm takes the opposite of the result of the search. For example, if the threat goal is to disconnect, at min nodes, the algorithm searches the connection goal with the disconnecting color playing first. If the search does not return Won, it considers the disconnection can be reached.

5 Evaluation of goals

This section deals with the approximate evaluation of how many points reaching a goal can gain. We use Monte-Carlo simulations to evaluate the importance of goals.

5.1 Standard Monte-Carlo Go

Standard Monte-Carlo Go consists in playing a large number of random games. The moves of the random games are chosen randomly among the legal moves that do 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. At 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 of highest value.

5.2 Evaluation of unsettled goals

The only simple goals that need to be evaluated are the unsettled ones. For each of these, the program computes the mean score of the games where it has been reached during the game. It also computes the mean score of the games where it has not been reached. The difference between the two means gives an evaluation of the importance of the goal.

5.3 Evaluation of threatened goals

For each move of each unsettled goal, the program searches for all the possible associated threat goals. For each combination of a goal and a threat goal, the program computes the mean value of the random games where they have both been reached.

For a combination of two positive goals, things are simple. The program tests at each move of each random games if both goals are reached. If it is the case the game counts. If it is not the case at the end of the game, the game does not count.

For goals that are a combination of a positive goal and of a negative goal we have a special treatment. For example for connection moves that threaten a disconnection, there are four intersections to take into account. The intersections to connect are s_1 and s_2, the intersections to disconnect are s_3 and s_4. The following tests are performed at each move of each random game :

- if s_1 or s_2 are empty or of the opposite color of the connection, the random game does not count.
- if s_1 and s_2 are in the same string, and if s_3 or s_4 are empty or of the color of the connection, the random game counts.
- if s_3 and s_4 are in the same string, the random game does not count.

At the end of a random game, if the two strings s_1 and s_2 have been connected, and s_3 and s_4 have not, the random game counts.

6 Evaluation of moves

This section explains the different values computed for each move, details how the values are computed for the empty connection goal and for the connection goal, and eventually gives the evaluation of the moves based on these values.

6.1 Values computed for each move

There are basically four values that may be computed for a move. The value of reaching the associated goal (the FriendValue), the value for the opposite color of preventing from reaching the goal (the EnemyValue), the value of reaching both the goal and the best associated threat goal (the FriendThreatValue), and the value for the opposite color of preventing from reaching the goal and reaching another enemy threat goal (the EnemyThreatValue).

6.2 Values for empty connection moves

For each empty connection move, the program finds the set of empty connections of the opposite color that are invalidated by the move. The EnemyValue is set to the value of the highest invalidated enemy empty connection. The EnemyThreatValue is set to the highest threat associated to the selected enemy empty connection. The FriendValue is set to the evaluation of reaching the empty connection, and the FriendThreatValue is set to the value of the highest threat goal associated to the empty connection. The only empty connection threats computed for a friend empty connection goal, are the threats of empty connection to an intersection that is not already empty connected to a friend group. Similarly, enemy empty connection threats do not empty connect to intersections already empty connected to enemy groups.

6.3 Values for connection moves

For connection moves, the FriendValue is the evaluation of reaching the connection, the EnemyValue is the value of not reaching the connection, the FriendThreatValue is the best evaluation among all the threat goals associated to the connection move, and the EnemyThreatValue is the best threat associated to the disconnection.

6.4 Evaluation of moves given their associated values

Once the four values are computed for each move of each unsettled goal, an approximation of the temperature of the moves can be computed. The computation of an approximate temperature given these values is based on the computation of the temperature of the game $\{\{\text{FriendThreatValue} \mid \text{FriendValue}\} \mid\mid \{\text{EnemyValue} \mid \text{EnemyThreatValue}\}\}$.

The thermograph is exact for connection values since all the enemy's options have the same EnemyValue, and all the friend's options have the same FriendValue. However, for empty connections there are different options for the enemy that lead to different EnemyValue, instead of reflecting this in the thermograph, we only take the best the subgame with the best EnemyValue. So we only compute an approximation of the temperature for empty connections.

When there are no friend threats, FriendThreatValue is set to FriendValue, and when there are no enemy threats, EnemyThreatValue is set to EnemyValue. The code for computing the approximate temperature given the four values is as follows (ABS is the absolute value):

```
temperature (EnemyThreatValue, EnemyValue,
             FriendValue, FriendThreatValue) {
  tempEnemy = ABS(EnemyThreatValue - EnemyValue)/2;
  tempFriend = ABS(FriendThreatValue - FriendValue)/2;
  width = ABS(FriendValue-EnemyValue);
  if (tempFriend - tempEnemy > width)
    return tempEnemy + width;
  else if (tempEnemy - tempFriend > width)
    return tempFriend + width;
  else
    return ABS(FriendThreatValue/4 + FriendValue/4 -
               EnemyValue/4 - EnemyThreatValue/4);
}
```

When the program does not use threats, it only sends back ABS(FriendValue - EnemyValue)/2.

7 Experimental results

We have measured the speed and the correctness of the search algorithm that verifies if goals are sente on other goals. We have also measured the benefits a program gets when taking into account the values of threats, and the average time it takes to compute threats.

For the experiments we have only used the connection goal and the empty connection goal. The machine used is a 3.0 GHz Pentium 4. The search algorithm that detects threats has been programmed using templates for the goals. It means that the same search code is used for any combination of goals. The search uses iterative deepening, transposition tables, two killer moves and the history heuristic.

7.1 The test suite for threats

The problems used to test the algorithms are given in figure 5. In the first eleven problems, an empty connection between A and B threatens (or not) an empty connection between B and C. Problem number twelwe is an example of a problem where an empty connection between A and B threatens a connection between A and C. Problem thirteen shows an empty connection between A and B that threatens to disconnect C and D. In problem fourteen, the black move at E connects A and B, and threatens to disconnect C and D.

Table 1 gives the number of moves played in the search algorithm and the time it takes for each of the problems of the test suite. All the problems are correctly solved by the search algorithm.

Table 1. Nodes and time for the threats problems.

Problem	$moves$	$time(ms)$
1	2,095	10
2	34,651	50
3	59,704	90
4	6,063	30
5	68,209	90
6	2,522	30
7	24	10
8	4,982	20
9	91,774	120
10	2,083	10
11	17	20
12	263	10
13	10,529	30
14	1,341	20
total	284,405	540

7.2 Integration in a Go program

The use of connection threats has been tested in a Monte-Carlo based program which evaluates goals. The program is restricted to the connection and the empty connection goals. The threats used are only the connection threats: the empty connections that threaten empty connections, the empty connections that threaten connections, the empty connections that threaten disconnections and the connections that threaten disconnections.

The experiment consists in playing one hundred 9x9 games against Gnugo 3.6. Fifty games as Black and fifty games as White. At each move of each game, the program plays ten thousand random games before choosing its move. Table 2 gives the mean score, the variance, the number of won games and the average time per move of the program with and without connection threats against Gnugo 3.6.

Using connection threats enables to gain approximately nine points per 9x9 game, and sixteen more games on a total of one hundred games. This is an encouraging result given that the program only uses connections and connection threats. An interesting point is that the mean of the games where the Monte-Carlo based program was black, is -6.9 including komi, and the mean with white is -19.1 also including komi. The program is much better with black than with white. Looking at the games, a possible explanation

Table 2. Score and time for the Go program with and without connection threats.

Algorithm	*mean*	*std deviation*	*won games*	*time (sec)*
Without threats	-22.33	24.80	11/100	5.7
With connection threats	-13.02	25.06	27/100	11.9

is that the Monte-Carlo program does not use any life and death search, it looses eight games by 75.5 as white, and only one game by 86.5 as black. Many of the games it looses completely are due to lack of life and death search and evaluation.

7.3 Over-evaluation of threats

The values computed for goals with the Monte-Carlo algorithm already take into account, to a certain amount, the threats associated to the goals. In some cases, it can be misleading for the program as it over-evaluates the value of some threats. An example is given in figure 1. White B threatens White C, and invalidates Black E. The connection (A,B) is evaluated to -2.2, (D,E) to 12.4, (A,B) and (B,C) to -14.2. So the temperature of White B is evaluated to 10.3. The connection (G,H) is evaluated to -5.8, (I,J) to 8.1, (G,H) and (H,L) to -9.4, (I,J) and (J,K) to 15.5. So the temperature of White H is evaluated to 9.7. The program prefers White B to White H, which is bad.

One problem here is over-evaluation of the threat for White of connecting (A,B) and (B,C). In the random games where both connections are reached, one half of the games also have C and F connected, which makes a big difference in the final score. However, if Black plays well, C and F never get connected, and the value of the threat is much lower. A possible solution to the problem of the over-evaluation of threats could be to make the program play better during the Monte-Carlo games [2].

8 Conclusion and future work

An algorithm to verify if reaching a goal threatens another goal has been described, as well as its incorporation in a Go program. It has also been shown how the unsettled goals and the associated threat goals, found by this algorithm, can be evaluated in a Monte-Carlo Go framework. An approximation of the temperature has been used to evaluate moves given the related goal evaluations. Results on a test suite for threats have been detailed. The use of connection threats in a Go program improves its results by approximately nine points for 9x9 games against Gnugo 3.6.

There are many points left for future work. First, it would be interesting to test the algorithm with other goals than connection ones. Second, the program currently often overestimates the values of threats. An improvement of the Monte-Carlo algorithm in order to make it play less randomly could address this point. Third, the interactions between goal threats and combination of goals are also interesting to explore. Fourth, the program currently evaluates the value of playing threats but does not take into account

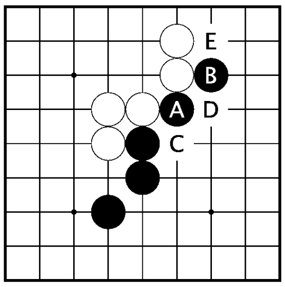


Fig. 4. Moves can avoid threats

the value of playing moves that invalidate threats: for example, in figure 4, the Black moves at C and D both connect A and B, Black D invalidates the threat of White E, but Black C does not invalidate the threat of White E. Eventually, work remains to be done to take into account the different options of the enemy when building the thermograph for empty connections.

References

1. E. Berlekamp, J. H. Conway, and R. K. Guy. *Winning Ways*. Academic Press, 1982.
2. B. Bouzy. Associating domain-dependent knowledge and monte carlo approaches within a go program. *Information Sciences*, 175(4):247–257, November 2005.
3. T. Cazenave. Comparative evaluation of strategies based on the value of direct threats. In *Board Games in Academia V*, Barcelona, Spain, 2002.
4. T. Cazenave. A Generalized Threats Search Algorithm. In *Computers and Games 2002*, volume 2883 of *Lecture Notes in Computer Science*, pages 75–87, Edmonton, Canada, 2003. Springer.
5. T. Cazenave. The separation game. In *JCIS 2005*, Salt Lake City, USA, 2005.
6. T. Cazenave and B. Helmstetter. Combining tactical search and monte-carlo in the game of go. In *CIG'05*, Colchester, UK, 2005.
7. T. Cazenave and B. Helmstetter. Search for transitive connections. *Information Sciences*, 175(4):284–295, 2005.
8. M. Mueller, M. Enzenberger, and J. Schaeffer. Temperature discovery search. In *AAAI 2004*, pages 658–663, San Jose, CA, 2004.
9. M. Mueller and Z. Li. Locally informed global search for sums of combinatorial games. In *Computers and Games 2004*, LNCS, Ramat-Gan, Israel, 2005. Springer.
10. W. Spight. Go thermography - the 4/21/98 jiang-rui endgame. In R. Nowakowski, editor, *More Games of No Chance*, pages 89–105. Cambridge University Press, 2002.

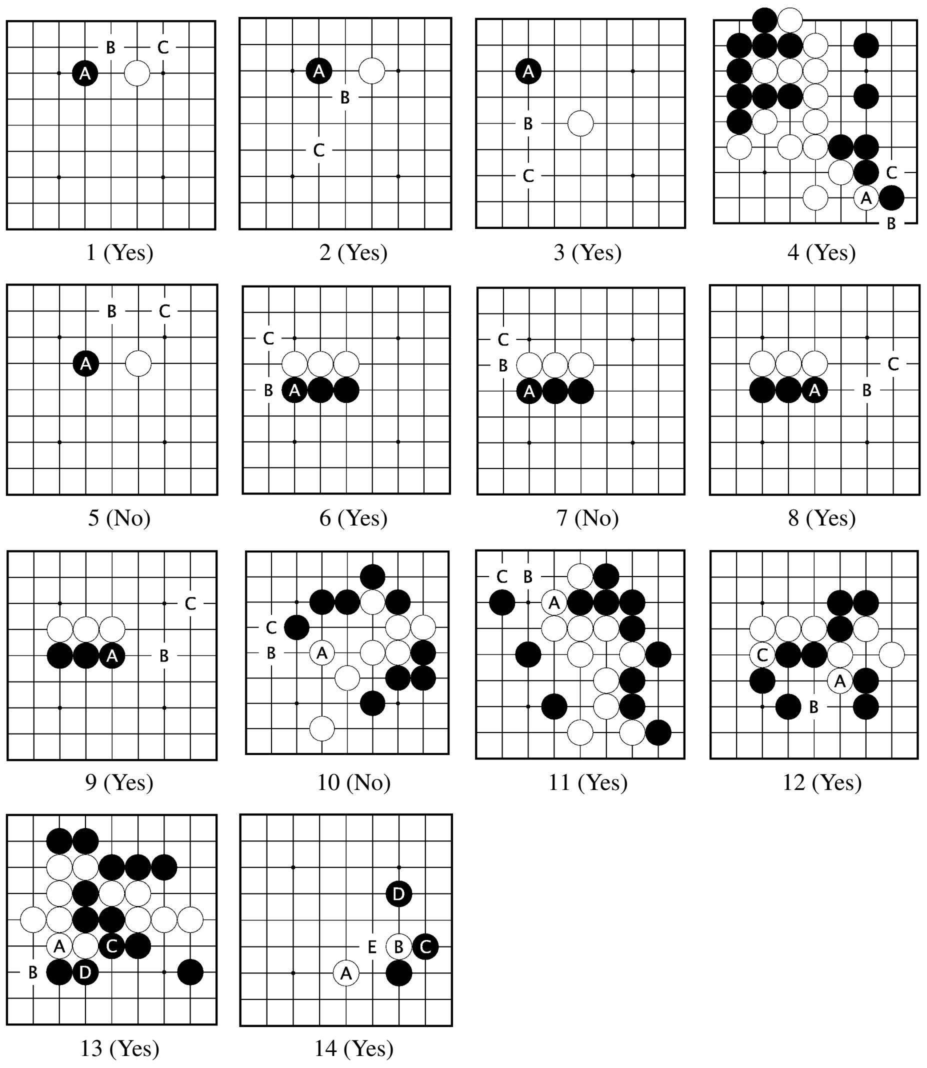


Fig. 5. The test suite