New Mathematics and Natural Computation

THE SEPARATION GAME

TRISTAN CAZENAVE
Labo IA, Université Paris 8, 2 rue de la Liberté
St-Denis, 93526, France
cazenave@ai.univ-paris8.fr

The separation game is different from the connection game, but has some similarities with it. In the game of Go, it is often useful because it helps enclosing groups and areas. An evaluation function, moves generation functions and a search algorithm for the separation game are described in this paper.

Keywords: Search; computer go.

1. Introduction

The game of Go is an ancient two-player complete information game. It is yet unknown how to write strong playing programs for the game of Go, despite a lot of efforts [3,6]. The usual approach that some of the best current programs use is based on a decomposition of the game into simpler sub-games. Examples of some of these well-known sub-games are the capture game, the connection game, the eye game or the life and death game. A sub-game that appears less often in the literature is the separation game [1,2,7]. We present a search algorithm, based on generalized threat search [5], for the separation game, as well as optimizations that enable to solve separation problems faster than a naive algorithm.

In section 2 the separation game is described. Section 3 deals with the empty separation game. In section 4 the evaluation function used in the search algorithm is given. Section 5 is about the selection of the interesting separation games. Section 6 details the search algorithm. Section 7 gives experimental results.

2. The separation game

The separation game has already been described by B. Bouzy and M. Mueller [1,2,7]. The existence of the separation game is due to the grid topology of the Go board. In Go, each intersection has at most four neighbors (four in the center, three on the border and two in the corner), the neighbors are horizontal or vertical neighbors. Two intersections are connected when there is a path of stones of the same color that are linked through vertical and horizontal neighbors. We call 4-connections these kinds of connections. For example, the black stone marked B in figure 4 is 4-connected to three other black stones. In Go, sets of 4-connected stones of the same color are called strings.

When instead of only considering the horizontal and vertical neighbors, we also consider the diagonal neighbors, the intersections on the Go board have at most eight neighbors

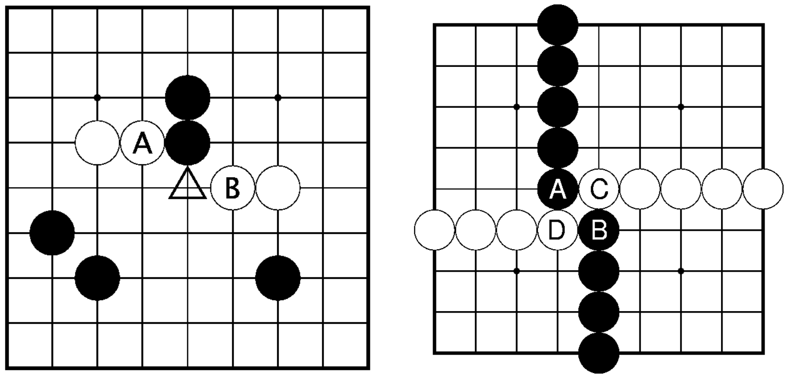


Fig. 1. examples of separations.

instead of four. When there is a path between two stones that goes through the eight neighbors, the two stones are 8-connected. A property of the Go board is that an 8-connection between two stones prevents the opponent to 4-connect through the 8-connection path. It means that 8-connecting two stones creates a separation of the board for the 4-connection.

Formally, two stones form a separation if they are 8-connected. Two stones are unseparated if they cannot be linked with an 8-connected path of stones of the same color (i.e. they cannot form a separation).

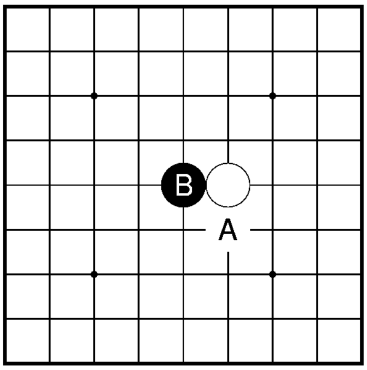
For example A and B in the right board of figure 1 form a separation: C and D cannot be 4-connected since A and B are 8-connected.

The white move at Δ in the left board of figure 1 does not connect A and B: a search for the connection of A and B returns that they cannot be connected. However, after the white move at Δ, A and B form a separation because they are 8-connected.

The black move at Δ in the left board of figure 1 prevents the creation of a separation between A and B. It is not a disconnection move since A and B are already disconnected, it is an unseparation move.

In Go, if two stones are 4-connected, they always form a separation, but a separation is not always a 4-connection, it can also be a 8-connection. Symmetrically an unseparation is a disconnection, but a disconnection is not always an unseparation. To put it another way, a connection is stronger than a separation, and an unseparation is stronger than a disconnection.

It is a property of the 4-connection that the associated separation is the 8-connection. For example in the game of Hex things are different. The separation of a 6-connection in Hex is also a 6-connection: when a player prevents the opponent to connect her two sides at Hex, he connects his own two sides. It means that the separation and the connection game are the same in Hex, but not in Go.



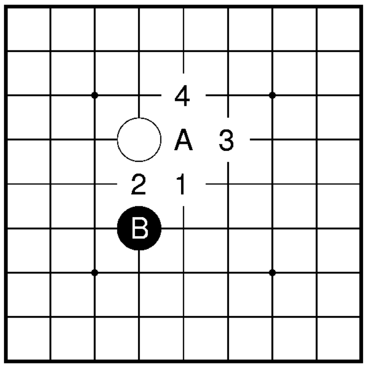


Fig. 2. empty separations.

3. The empty separation game

The empty separation game consists in playing a stone, and checking if it is separated from a given string. Figure 2 gives two examples where playing a black stone at A creates a separation between the newly created black string at A and the black stone at B. In the left board, a black stone at A is not connected to B with a simple connection search, however a black stone at A forms a separation with B. Similarly, in the right board, a black stone at A is not connected to B with a simple connection search. A simple separation search finds that a black stone at A and B form a separation. A possible sequence to find the separation is black A, white 1, black 2, white 3 and black 4. After black 4, A and B are 8-connected and form a separation.

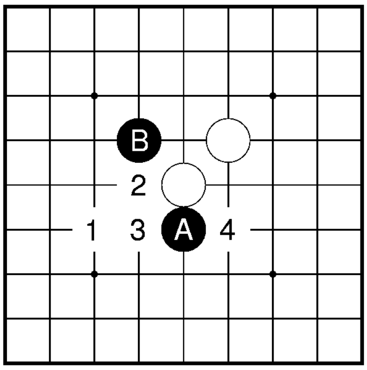
The empty separation game is useful to play destabilizing moves and to enclose areas. To verify the empty separation game, the program plays the move and searches if the separation game is won even if the opponent plays first.

4. Usefulness of the separation game for a Go program

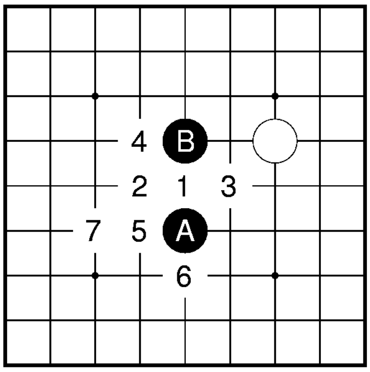
The first use of separations in a Go program is to prefer unseparations to disconnections when both are possible. For example in figure 3(a), a white move at 1 disconnects A and B (white 1, black 2, white 3, black 4 for example). But white 1 does not prevent A and B to form a separation. White 2 is a better move because it disconnects A and B, and also prevents A and B to form a separation. A similar move is the black move at Δ in figure 1 which prevents A and B to form a separation. It cannot be found with a disconnection search since A and B are already disconnected.

Symmetrically, in figure 1 a white move at Δ cannot be found with a connection search since there is no simple way to connect A and B. A simple separation search can find it.

In figure 3(b), A and B can be disconnected (white 1, black 2, white 3, black 4, white 5, black 6, white 7). However disconnecting a tobi is often a bad move even if it disconnects it. A search on the separation of A and B finds that White cannot prevent Black from forming a separation with A and B, even if it can disconnect them. This is an indication that the disconnection is less interesting, since the forming of a separation cannot be prevented.



(a)



(b)

Fig. 3. use of separation.

The empty separation game is also useful for a Go program since it finds attacking and enclosing moves such as the two black A moves of figure 2.

5. The evaluation function

The evaluation function uses the shortest 8-path between the two strings that form the separation, and it also verifies that none of the two strings can be captured. The Max player tries to create a separation, the Min player tries to avoid the separation. The smallest possible value for the evaluation function is *Lost*, and the greatest one is *Won*. The first subsection details the computation of the shortest 8-path. The second subsection deals with the capture avoidance of any of the two strings.

5.1. *The shortest 8-path*

The evaluation function is based on the shortest 8-path between two strings. The length of the path is defined by the minimum number of moves of the same color that are required to 8-connect the two strings. The following basic function verifies if the strings are 8-connected:

```
eightConnected (string1, string2) {
 if (string1 == string2)
  return 1;
 for each stone s in string1
  for each diagonal d of s
   if d is part of string2
```

```
     return 1;
 return 0;
}
```

A more advanced function finds a length one 8-path between the two strings:

```
lengthOne (string1, string2) {
 for each liberty l of string1 {
  if l is liberty of string2
   return 1;
  for each diagonal d of l
   if d is part of string2
    return 1;
  for all neighbors n of l
   if n is a friend string str
    for each stone s of str
     for each diagonal d of s
      if d is part of string2
       return 1;
 }
 for each liberty l of string2 {
  for each diagonal d of l
   if d is part of string1
    return 1;
  for all neighbors n of l
   if n is part of a friend string str
    for each stone s of str
     for each diagonal d of s
      if d is part of string1
       return 1;
 }
 return 0;
}
```

Similar and more advanced functions are used to find paths of length two and three. When the shortest 8-path between the two strings is strictly greater than three, the evaluation function returns Lost.

5.2. *Avoid capture*

The evaluation checks that none of the two strings can be captured. For example, in figure 4, the strings A and B are 8-connected, but White can capture the A string. A general threat capture search [5] is called for each of the two strings, with the (6,5,0) threat. A (6,5,0) threat means that during the verification of the threat, six moves of order one, five moves of order two, and no move of greater order are allowed in the threat search tree. In capture search,

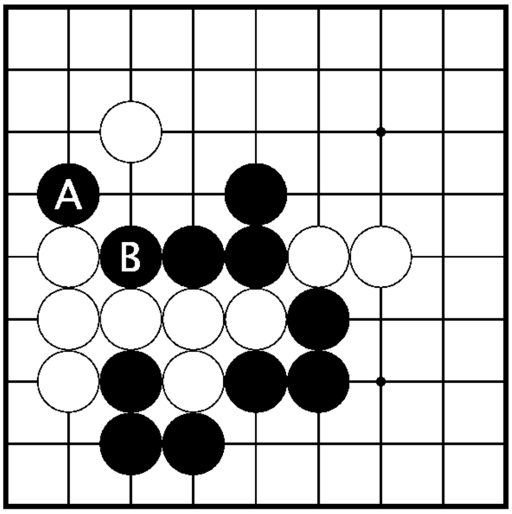


Fig. 4. avoid capture.

a move of order one is a capture, and a move of order two is an atari. The search with a (6,5,0) threat detects that the string A in the figure 4 can be captured. The search is only called when evaluating a position and it is the Min player's turn.

Another concern related to capture is when strings cannot be captured alone, but when one of the two strings can be captured if the other is defended. An example of such a problem is shown in figure 5. A White move at C can capture either A or B, but a capture search finds that B alone is not captured, and that A alone is not captured. To deal with this problem, the evaluation function avoids returning Won when either string A or B has two liberties or less and it is the Min player's turn.

5.3. *Potential unseparating moves*

The evaluation function always records the shortest 8-path. When it is Min's turn, it tries Min moves on the shortest path, and checks after each min move if the shortest 8-path becomes greater than 4, in which case it returns Lost.

6. Selection of the meaningful separations

In a given position, there are many possible separation games. It is useful to compute only some of them. For example in figure 6 a possible separation game is the separation between A and C. But A is connected to B, and B and C form a separation. In order to avoid unnecessary search, groups are built using won connection games before selecting separation games. For example, in figure 6 the program finds a won connection between A and B, and between B and D. So a group containing A, B and D is built. Once groups are built,

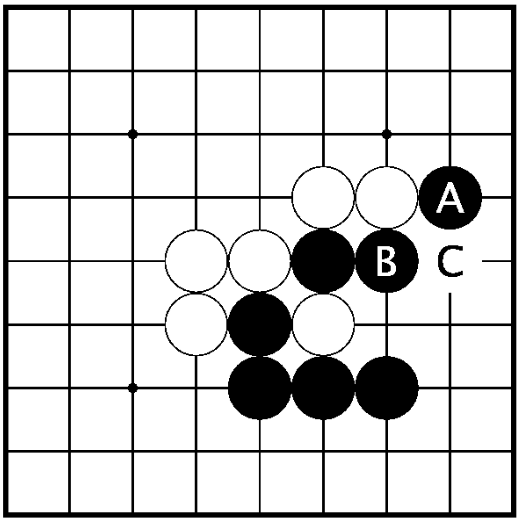


Fig. 5. avoid double capture.

the only separations between two groups that are selected are the ones with the shortest 8-paths. This avoids trying to solve unnecessary complex games, such as the separation between A and C (the shortest 8-path between A and C is of length one, and the shortest 8-path between B and C is of length zero, as A and B are in the same group, the potential separation between A and C is discarded). It prevents the program from finding unsettled complex separation due to a lack of search when the separation is won, and therefore it avoids unnecessary and bad moves. For example in figure 6, a too simple separation search could wrongly find that the separation between A and C is unsettled and try to prevent the separation, playing a bad white move at the left of F.

In order to select empty separations, the program starts with computing the empty connections of groups (i.e. empty intersections that become connected to the group when a stone of the color of the group is played at the intersection). The empty intersections that are at a 8-length less than three from existing strings are selected. The program selects the shortest empty separations between these empty intersections and the group, provided the intersections are not already empty connected to the group (when the same empty intersection can form a separation with two strings of the same group, it only selects the shortest empty separation).

7. Search algorithm

We have used the generalized threats search (GTS) algorithm [5] to solve separation problems. The first subsection gives a naive approach to max moves generation. The seconds subsection explains how to be more selective in max moves generation. The third subsection tells how min moves are selected.

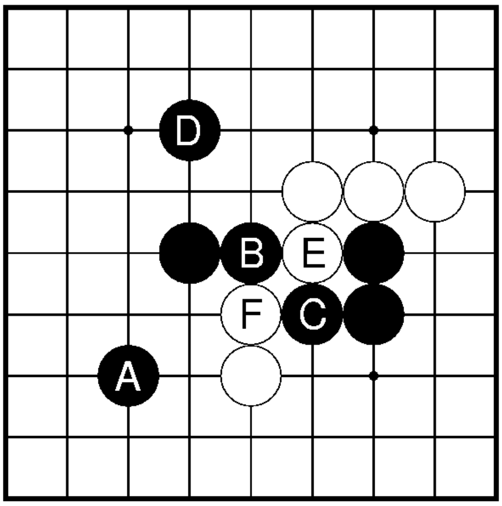


Fig. 6. selection of the simplest separation.

7.1. *Naive max moves*

Naive max moves of order one are the liberties of the two strings. If one of the two strings has less than three liberties, the liberties of adjacent strings that have only one liberty are also tried.

Naive max moves of order 2 are the liberties and the second order liberties of the two strings. If one of the two strings has less than three liberties, the liberties of adjacent strings that have two or less liberties are also tried.

7.2. *Advanced max moves*

GTS uses the max move generator for generating max moves in the threats and in the main search. An important information that can be used to optimize the max moves generation is the order of the position. The order of a position is the minimum number of moves of the same color that have to be played to win the game. The order of the generated moves is available both in the threat and in the main search. The order of the moves in the main search is the order of the threat verified at min nodes plus one.

The length of the shortest path between the two strings is the minimal order of the position. It is an admissible heuristic on the order of the position that can be used to speed up threat verification [4]. When computing the shortest path, the program records all the intersections on the shortest path. If the order of the moves to generate is lower than the length of the shortest path, then no moves are generated. If the order of the moves to generate is equal to the length of the shortest path, then the only moves to generate are the moves on the shortest paths. If the order is greater than the length of the shortest path, then the moves to generate are the moves on the shortest paths, and the naive max moves.

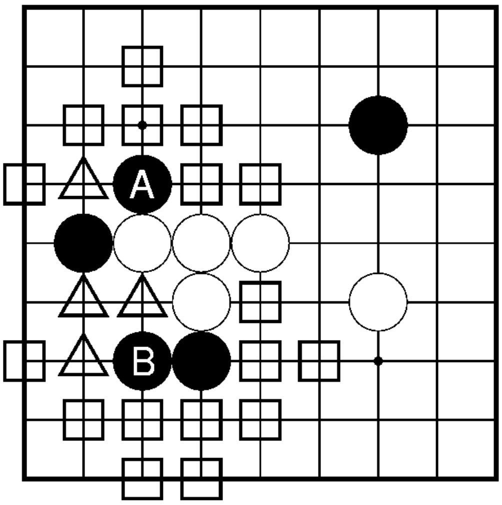


Fig. 7. naive and advanced max moves.

Figure 7 gives an example of the set of advanced max moves of order two (the two triangle intersections) and the set of naive max moves of order two (the square and the triangle intersections), both for the creation of a separation between A and B.

The program also looks for threats to capture one of the two strings. When a threat is verified, it only generates the moves that prevent the threat to capture.

Similar knowledge can be used when generating moves for the connection game using the shortest 4-paths.

7.3. *Min moves*

The set of min moves is selected using the trace of the threat verified at min nodes. If no threat is verified, the node is cut.

8. Experimental results

The generalized threats search algorithm has been tested for the separation game. Experiments include different move generators, different threats and different maximum number of moves per problem. They were performed on a Celeron 1.7 GHz, with 1 Gb of RAM. The test suite consists of 25 separation problems taken from games between Golois and Gnugo.

Table 1 gives the time and the total number of moves for different algorithms. For example the first line is the search algorithm that uses the naive move generator for max moves, the threat number 0 (the (1,0) threat), and that stops search at 10,000 nodes. It solves 20 problems in 250 milliseconds and 81,824 moves. The best algorithm is gts (1) with 100,000 nodes: it uses the (2,1,0) threat and the advanced max moves generator, it solves all 25 problems in 340 milliseconds and 219,357 moves.

Table 1. Comparing algorithms.

$Algorithm$	max	$\#solved$	$time$	$\#moves$
$naivegts(0)$	10,000	20	250	81,824
$naivegts(0)$	100,000	23	270	121,738
$naivegts(0)$	1,000,000	23	270	121,738
$naivegts(1)$	10,000	19	310	101,260
$naivegts(1)$	100,000	23	1,520	467,338
$naivegts(1)$	1,000,000	25	2,950	1,104,474
$gts(0)$	10,000	21	110	54,161
$gts(0)$	100,000	23	130	60,556
$gts(0)$	1,000,000	23	130	60,556
$gts(1)$	10,000	20	190	91,824
$gts(1)$	100,000	25	340	219,357

Table 2 details the time and the total number of moves for gts (1) and naivegts (1) when they are given a sufficient number of moves to solve all the problems. It is clear that gts (1) outperforms naivegts (1).

Figure 8 gives the 25 problems of the test suite. The number of each problem is followed by an s if it is a separation problem, and by an u if it is an unseparation problem. The two strings involved in a separation are marked with a triangle.

9. Conclusion

The separation game and the empty separation game have been defined. The shortest 8-path between the two strings that form a separation is useful both for the evaluation function and for moves generation. A threat based algorithm that uses the shortest 8-paths in order to select the moves of a given order has better results than an algorithm that does not use this information. Selection of the meaningful separation games has also been addressed, as well as the links between the separation and the capture game.

References

1. B. Bouzy. Modélisation cognitive du joueur de go. Phd thesis, Université Paris 6, 1995.
2. B. Bouzy. Le role des concepts spatiaux dans la programmation du jeu de go. *Revue d'Intelligence Artificielle*, 15(2):143–172, 2001.
3. B. Bouzy and T. Cazenave. Computer Go: An AI-Oriented Survey. *Artificial Intelligence*, 132(1):39–103, October 2001.
4. T. Cazenave. Admissible moves in two-player games. In *SARA 2002*, volume 2371 of *Lecture Notes in Computer Science*, pages 52–63, Kananaskis, Alberta, Canada, 2002. Springer.
5. 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, 2002. Springer.

Table 2. Details for the (2,1,0) threat.

	gts(1)		naivegts(1)	
problem	*#moves*	*time*	*#moves*	*time*
1	1	0	15	10
2	1,487	0	2,911	10
3	2,813	10	2,326	0
4	927	0	506	10
5	120	0	114	0
6	9,329	20	4,489	10
7	1,946	0	2,159	0
8	98	0	151	0
9	20	0	23	0
10	12,115	30	15,830	40
11	1,731	0	793	0
12	52	0	52	0
13	84	0	55	0
14	8	0	8	0
15	45	0	131	0
16	44	0	44	10
17	191	0	264	0
18	73	0	86	0
19	44	0	133	0
20	38,230	90	141,147	450
21	15,151	40	13,405	30
22	5,098	20	65,071	390
23	40,946	100	453,197	1,740
24	428	0	601	0
25	10,714	30	43,633	250
total	219,357	340	1,104,474	2,950

6. M. Müller. Computer go. *Artificial Intelligence*, 134(1-2):145–179, 2002.
7. M. Müller. Position evaluation in computer go. *ICGA Journal*, 25(4):219–228, 2002.

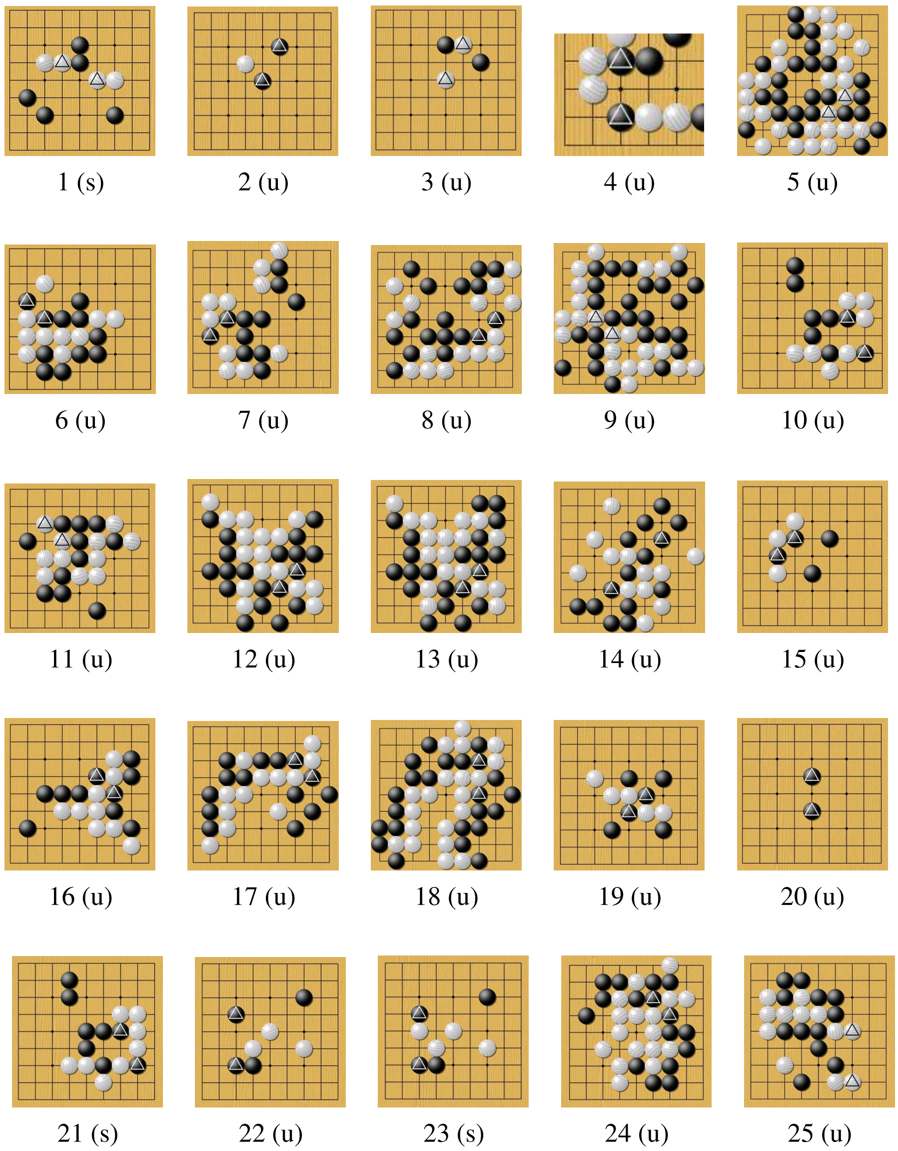


Fig. 8. Problems of the test suite