The separation game

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

Abstract

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.

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 lot of efforts [1; 2]. 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 [3; 4; 5]. We present a search algorithm 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. Müller [3; 4; 5]. The existence of the separation game is due to the grid topology of the Go board. In other games such as Hex, the separation game is the same as the connection game due to a different board topology. In Go, each intersection has four neighbors two horizontal neighbors and two 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 the figure 3 is 4-connected to three other black stones. In Go, sets of 4-connected stones of the same color are called strings.

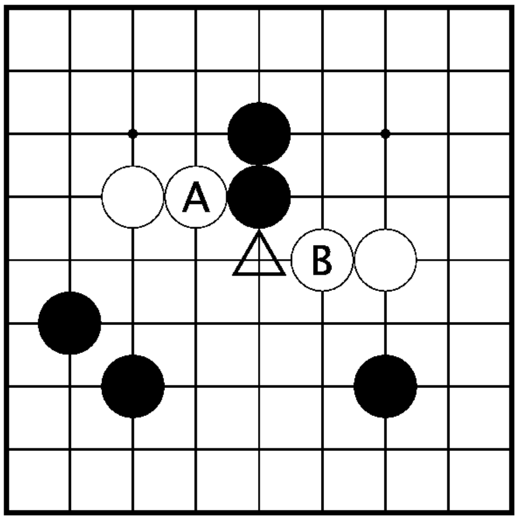

Figure 1: separation and unseparation.

When instead of only considering the horizontal and vertical neighbors, we also consider the diagonal neighbors, the intersections on the Go board have eight neighbors 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 separates the board for the 4-connection.

It is a property of the 4-connection that the 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.

In Go, a connection is a separation but a separation is not always a 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.

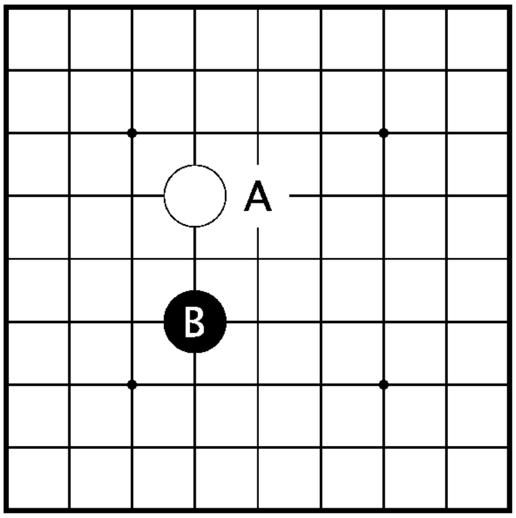

Figure 2: an empty separation.

The black move at Δ in figure 1 unseparates the two strings A and B. A and B are disconnected but not unseparated, Black has to add the Δ move for unseparating them.

The white move at Δ in figure 1 does not connect A and B, but it separates them.

3 The empty separation game

The empty separation game consists in playing a stone, and checking if it is separated from a string. Figure 2 gives an example where playing a black stone at A separates the newly created black string at A from the black stone at B.

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 The evaluation function

The evaluation function uses the shortest 8-path between the two strings to separate, and it also verifies that none of the two strings can be captured. The Max player tries to separate, the Min player tries to unseparate. 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 separating strings.

4.1 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 basic function verifies if the strings are 8-connected:

```
eightConnected (string1, string2) {
 if (string1 == string2)
  return 1;
 for each stone in string1
  for each diagonal of each stone
   if diagonal 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 of string1 {
  if liberty is liberty of string2
   return 1;
  for each diagonal of liberty
   if diagonal is part of string2
    return 1;
  for all neighbors of liberty
   if neighbor is a friend string
    for each stone of friend string
     for each diagonal of stone
      if diagonal is part of string2
       return 1;
 }
 for each liberty of string2 {
  for each diagonal of liberty
   if diagonal is part of string1
    return 1;
  for all neighbors of liberty
   if neighbor is a friend string
    for each stone of friend string
     for each diagonal of stone
      if diagonal 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 separating strings is strictly greater than three, the evaluation function returns Lost.

4.2 Avoid capture

The evaluation checks that none of the two separating strings can be captured. For example, in figure 3, the strings A and B are 8-connected, but White can capture the A string. A general threat capture search [6] is called for each of the two separating strings, with the (6,5,0) threat. The search with a (6,5,0) threat detects that the string A in the figure 3 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 separating string can be captured if the other is defended. An example of such a problem is shown in figure 4. 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

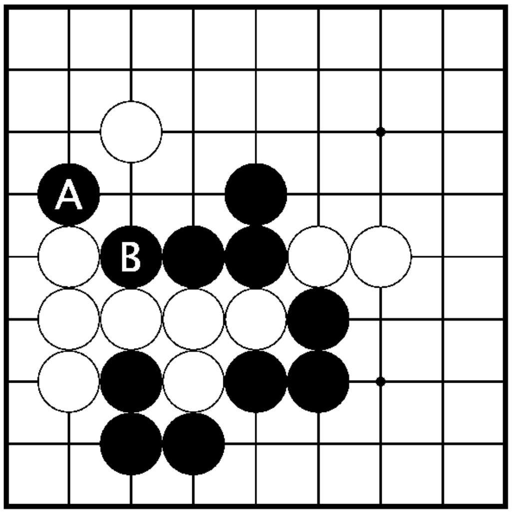

Figure 3: avoid capture.

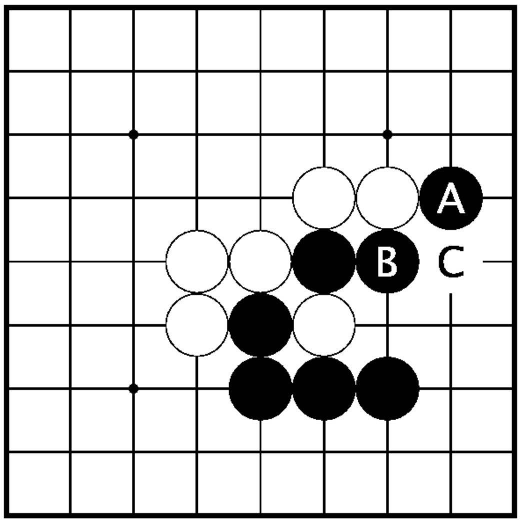

Figure 4: avoid double capture.

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.

4.3 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.

5 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 5 a possible separation game is the separation between A and C. But A is connected to B and B is separated from C. In order to avoid unnecessary search, group are built using won connection games before selecting separation games. Once groups are built, 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. 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.

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 separate to two strings of the same group, it only selects the shortest empty separation).

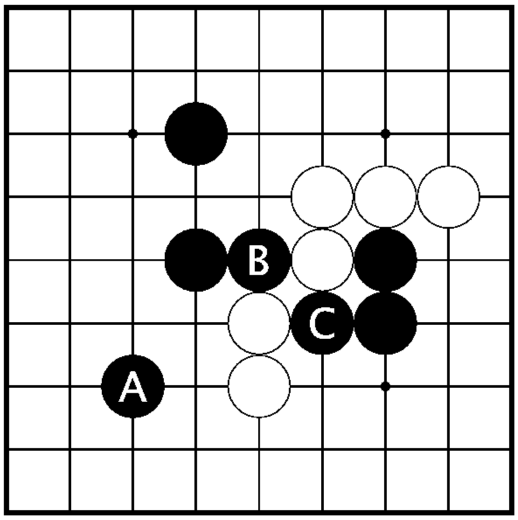

Figure 5: selection of the simplest separation.

6 Search algorithm

We have used the generalized threats search (gts) algorithm [6] 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.

6.1 Naive max moves

Naive max moves of order one are the liberties of the two separating strings. For the separating strings that have 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 separating strings. For the separating strings that have less than three liberties, the liberties of adjacent strings that have two or less liberties are also tried.

6.2 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. 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.

The program also looks for threats to capture the strings to separate. 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.

6.3 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.

7 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 33 separation problems taken from games between Golois and Gnugo.

Table 1: comparing algorithms.

$Algorithm$	max	pb	$time$	$\#moves$
$naivegts(0)$	10,000	12	15.99	693,591
$naivegts(0)$	100,000	14	48.06	2,103,193
$naivegts(1)$	10,000	12	16.85	668,465
$naivegts(1)$	100,000	13	60.53	2,682,075
$gts(0)$	10,000	16	3.35	150,116
$gts(0)$	100,000	16	4.94	241,267
$gts(1)$	10,000	17	5.48	260,224
$gts(1)$	100,000	17	14.56	687,923

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 12 problems in 15.99 seconds and 693,591 moves. A better algorithm is gts (1) with 10,000 nodes: it uses the (2,1,0) threat and the advanced max moves generator, it solves 17 problems in 5.48 seconds.

8 Conclusion

The separation game and the empty separation game have been defined. The shortest 8-path between the two strings to separate 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 is also a concern that has been addressed, as well as the links between the separation and the capture game.

References

[1] Bouzy, B., Cazenave, T.: Computer Go: An AI-Oriented Survey. Artificial Intelligence **132** (2001) 39–103

[2] Müller, M.: Computer go. Artificial Intelligence **134** (2002) 145–179

[3] Bouzy, B.: Modélisation cognitive du joueur de go. Phd thesis, Université Paris 6 (1995)

[4] Bouzy, B.: Le role des concepts spatiaux dans la programmation du jeu de go. Revue d'Intelligence Artificielle **15** (2001) 143–172

[5] Müller, M.: Position evaluation in computer go. ICGA Journal **25** (2002) 219–228

[6] 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