Search for transitive connections

Tristan Cazenave, Bernard Helmstetter
Labo IA, Université Paris 8
2 rue de la Liberté, 93526, St-Denis, France
e-mail: {cazenave,bh}@ai.univ-paris8.fr

Abstract

We present an algorithm that detects non transitive connections in the game of Go. An optimized Alpha-Beta search is used on top of two Generalized Threats searches, one for each of the two connections. It deals with full board situations such as the ones encountered in real games. Our program is able to solve problems such as the double monkey jump or the double keima on the second line. Even if the results are not theoretically perfect, they are pretty reliable given the results on a test suite.

1 Introduction

In this paper, we explore a way to reduce the complexity of a search on a double connection by searching both connections separately as much as possible. In the best case the two connections are independent, and a complex search that would cost $O((2p)^{2d})$ can be reduced to two searches of complexity $O(p^d)$, the variable p being the average number of possible moves in a connection game, and d the depth of the search for solving one of the connection game. In practice, the two connections are seldom perfectly independent. Even when they are not independent, it is useful to search the two connections separately as it helps finding sets of relevant moves.

Programs are currently bad at finding complex non transitivities as can be seen in tournament games [Wedd, 2000]. Some programs handle non transitivity using hand-coded patterns, but there are too many cases of non transitivity for this approach to be efficient.

The second section describes the problem of the non transitivity of connections. The third section outlines the adaptation of Generalized Threats Search to the connection game. The fourth section details the evaluation function and the selection of moves used in our transitive connection search algorithm. The fifth section gives experimental results. Eventually, the last section concludes and outlines future work.

2 Non-transitivity of connections

In this section we define the problem of the non transitivity of connections. Even the best computer Go programs have problems dealing with non transitivity. This problem is a special case of the more general problem of the dependence between two or more different sub-goals.

Let s_1, s_2, s_3 be three stones of the same color, C_1 the connection between s_1 and s_3, C_2 the connection between s_2 and s_3, and C the transitive connection between s_1 and s_2. We call max player the player who wants to connect and min player the player who wants to disconnect. In the starting position, the connections C_1 and C_2 must be won; otherwise connection C would not be won a fortiori.

The easiest case is when the connections C_1 and C_2 are independent. Then connection C is won. Figure 1 shows an example.

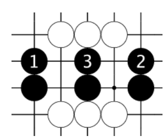

Figure 1: Two independent connections

The interesting cases are when the connections are not independent, that is to say when there is at least a move by min player which threatens to break both connections. Then connection C may or may not be won. Figure 2 shows an example where connection C is lost if min player moves first: the move that disconnects is white $\triangle$. Now figure 3 shows an example of two connections that are not independent but which make a transitive connection nonetheless. The connections C_1 and C_2 are not independent because a white move at A threatens to break both; but then a black move at B would repair both connections.

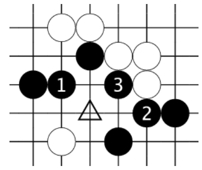

Figure 2: Non transitive connections

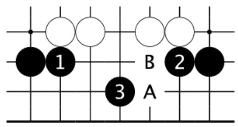

Figure 3: transitive connections

3 Generalized threats connection search

In order to find transitive connections, we have to solve the problem of finding direct and single connections. This section is about the single connection game. We have used Generalized Threats Search [Cazenave, 2002] to solve the single connection game. The threat used for these experiments is the (8, 5, 2, 0) general threat. It is not mandatory to use Generalized Threat Search; however, whatever algorithm we use to find direct connections, it has to send back a trace that contains all the intersections that may change the result.

The evaluation function returns:

- Lost if one of the two strings to connect is captured in a ladder,
- Won if the two stones to connect are in the same string
- Unknown otherwise

We use specialized functions to find the max moves. The order of a threat is the number of moves in a row the max player has to play in order to win the game [Cazenave, 2002]. The functions called for finding the max moves depend on the order of the threat. We have different specialized and heuristic functions for finding possible moves that connect in one move, in two moves or in three moves. When there is a possibility for one of the two strings to be captured, the only max moves considered are the moves that save the threatened string. Concerning the min moves, the Generalized Threat Search algorithm uses the trace of the verified threat to find the relevant min moves.

4 Search for transitivity

We use an Alpha-Beta algorithm with transposition tables, two killer moves and the history heuristic. The game specific functions of our Alpha-Beta are: the evaluation function, and two functions *minMoves* and *maxMoves*, which return sets of relevant moves for the players min or max.

4.1 Evaluation function

The evaluation function searches the connections C_1 and C_2 in isolation, and tries to deduce from this the status of the transitive connection. The situation is different depending on who is to play.

We consider first the case where max player is to play. The connections C_1 and C_2 are first searched with max player playing first, then with min player playing first. Besides the results, the searches also return the traces of all intersections on which the results depend. There are two cases where we can be sure of the status of the transitive connection:

- If C_1 or C_2 is lost, assuming max player plays first, then the transitive connection is lost.
- If one of the connections, say C_1, is won (assuming min player plays first), if the other, C_2, is winnable (*i.e.* it can be won if max player plays first), and if the traces on which those two results depend are disjoint, then the transitive connection is winnable. Indeed, in order to win it suffices for max player max to play the winning move in connection C_2.

We now consider the case where the player min (*i.e.* the player to disconnect) plays first. There are again two cases where we can be sure of the status of the transitive connection:

- If C_1 or C_2 is lost, assuming min player plays first, then the transitive connection is lost.
- If both connections C_1 and C_2 are won, assuming min player plays first, and if the traces on which those two results depend are disjoint, then the transitive connection is won.

4.2 Choose of min moves

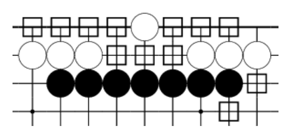

Figure 4: Min moves

In case the evaluation function can't decide the status of the transitive connection we have to continue the main Alpha-Beta search which deals with the transitive connection as a whole. Hopefully the searches that have been made on the connections C_1 and C_2 can give valuable information to find a relevant set of moves.

We take as set of min moves (moves for min player) the moves that threaten to break either connection C_1 or C_2. This is the union of the traces of the two searches that have shown that the connections C_1 and C_2 are won when min player plays first. An example of a set of relevant min moves found by our program is given in figure 4.

We may wonder why we could not use the intersection of the traces rather than the union. However this not safe at all, as can be seen for instance in problem 13 of our test suite (figure 7).

Taking the union of the traces, as we do, is in fact not perfectly safe either. It has worked well in the problems of our test suitem but we know one artificially built transitivity problem (figure 5) where it misses a move. The move white A does not threaten either connection C_1 or C_2, but it does break the transitive connection because black cannot defend against both white B and white C. One can note that white C would have directly worked too, so even in this problem we would find at least one disconnecting move.

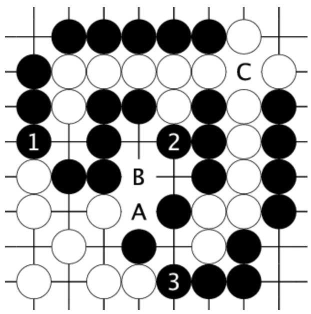

Figure 5: Pathological position

4.3 Choose of max moves

The set of max moves depends on the order at which we want to search the transitive connection. We have chosen to take as set of max moves the union of the sets of moves of order up to ordermax in each connection C_1 and C_2. The variable ordermax equals at most the order of the maximum threat for the connection search plus one. An example of a set of max moves found by our program is given in figure 6. In this case it is obviously far from perfect, because our program selects moves of order 3 although the two connections are of order 2. Using iterative widening on the order of the connections would make it more selective.

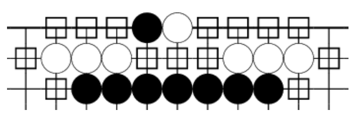

Figure 6: Max moves

5 Experimental results

In our experiments, the maximum threat used for the Generalized Threats connection search has been set to (8, 5, 2, 0). We only choose moves of order less or equal to three for the connections C_1 and C_2 in order to find the max moves in the transitive search. The maximum number of moves in each of the two Generalized Threats searches is limited to 4,000.

We have built a test suite of 21 problems. Half of the problems are taken from Golois games, and the other half are classical problems. All the problems except problems 3 and 17 are solved by our program. Problem 3 is not solved because in one of the forced lines a black cuting string gains 4 liberties and is therefore considered stable. In problem 17, our algorithm finds the good move but fails to see that it depends on a ko.

In table 1, for each problem, the number of moves played in the search and the elapsed time used for the search are given. The experiments were run on a 1.7 GHz Pentium with 100 Mb of RAM. Representative examples of our test suite can be found in figure 7.

Table 1: Nodes and time for the transitivity problems.

Problem	$moves$	$time(ms)$
1	20833	140
2	39497	280
3	83256	480
4	1425	10
5	139018	850
6	9070	70
7	76075	630
8	5179	40
9	4042	30
10	2933	20
11	26408	180
12	19080	130
13	283492	2110
14	23841	100
15	120248	560
16	6523413	33190
17	121936	620
18	148069	670
19	54049	310
20	1460301	10550
21	7763	70

Some problems that human find relatively easy such as the double keima on the second line in problem 16 are difficult for our program, while some problems that humans find relatively difficult such as problem 21 are easy for our program.

6 Conclusion

We have described an algorithm to detect non transitive connections in the game of Go. An optimized Alpha-Beta search is used on top of two Generalized Threats searches. Our program is able to solve problems such as the double monkey jump or the double keima on the second line. It deals with full board situations such as the ones encountered in real games. Even if the results are not theoretically perfect as can be seen on a pathological position, they are pretty reliable given the results on our test suite.

The program is currently too slow for problems such as the double keima to be used in a Go program. However, a simplified and more limited version could be useful to detect more simple cases. Future work includes making it faster by using optimizations such as iterative widening in the overall search [Cazenave, 2001], and extending it toward a more general search program for combinations of sub-goals.

References

[Cazenave, 2001] T. Cazenave. Iterative Widening. In *Proceedings of IJCAI-01, Vol. 1*, pages 523–528, Seattle, 2001.

[Cazenave, 2002] T. Cazenave. A Generalized Threats Search Algorithm. In *Computers and Games 2002*, Lecture Notes in Computer Science, Edmonton, Alberta, Canada, 2002. Springer.

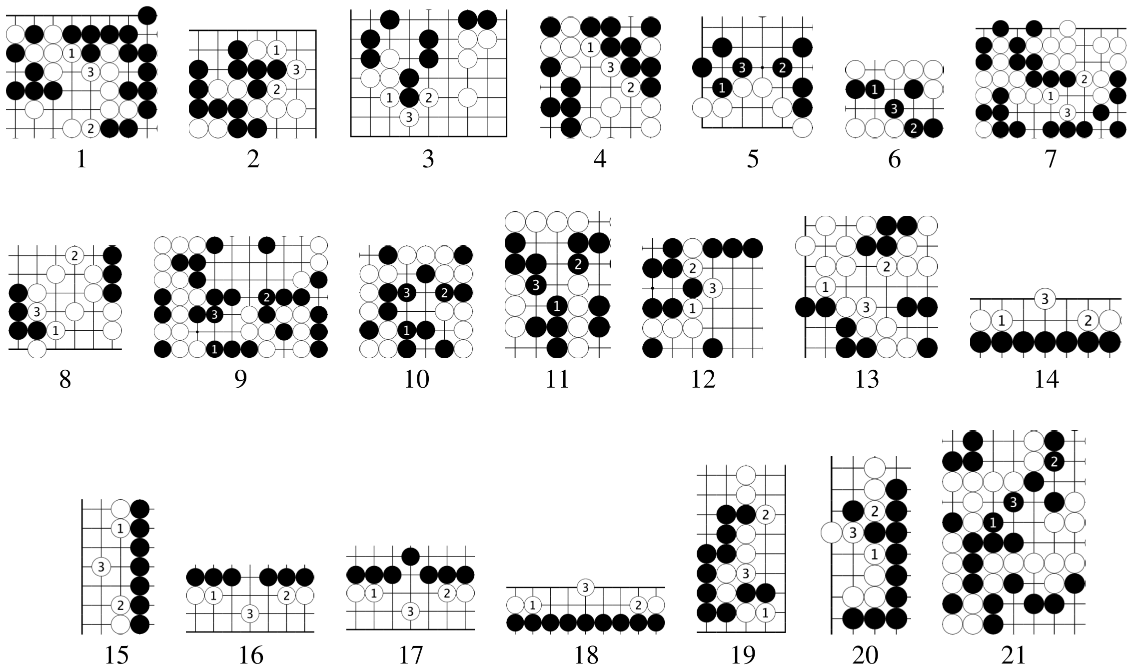

Figure 7: Problems of the test suite

[Wedd, 2000] N. Wedd. Goemate wins go tournament. *ICGA Journal*, 23(3):175–178, September 2000.