Journal of the Faculty of Engineering, Shinshu University, No. 65, 1989
信州大学工学部紀要　第65号

An Algorithm for Solving Alive Situation Puzzles in GO Game

Yasushi FUWA,* Yatsuka NAKAMURA,** and Yoshihiro KOSHISAKA***

(Received October 31, 1988)

For the process of deciding whether the situation of stones in GO game is alive or not, an improved algorithm was devised, and a program for solving alive situation puzzles in GO game was constructed with the algorithm and put into test uses. Conventional algorithms employ methods based on pattern matching and require a lot of processing time and large databases. A new algorithm first partitions stones into groups and then analyze the situation. Every group enclosed by opposite stones is pruned, and if at least one group remains, the situation is considered to be alive. The program has succeeded in solving alive situation puzzles reasonably rapidly.

1. Introduction

In general, it is very difficult to find efficient algorithms for playing strategic games. A great progress in the art how to program a computer to play a strategic game has been made since 1955 when the first computer chess program appeared[2]. In 1976 a Northwestern University chess program named CHESS 4.5 defeated several Experts and Class "A" players. This program is based on the full width search by the MIN-MAX principle. But there have been no such successful programs for GO game. The main reason for it is that GO game has higher complexity than chess. Statistically, the average of possible moves is 40 in chess, but 200 in GO. Moreover, the analysis of static situations in GO game is more complicated and difficult than in chess.[1),3),4)] Here, this paper presents a new simple algorithm for recognizing alive groups of stones on the GO board, which has been successfully applied to solving alive situation puzzles of GO game. In section 2, the notion of alive situation is explained. The algorithm is described in section 3. In section 4, a program for solving Tsume GO with applications of the algorithm and the possibility of speeding up the algorithm are discussed.

* Assistant, Department of Information Engineering.
** Professor, Department of Information Engineering.
*** Student of master course.

2. Alive situation

In a GO game stones are placed on intersections of horizontal and vertical lines. Two stones are called **neighbors** if they are positioned on two neighboring intersections. A set of stones of the same color on the GO board is called a **group** if each stone X in the set has at least one neighbor in the set and all the neighbors of the stone X belong to the same set. Thus, the group is a set of stones closed under the relation of **neighborhood**. A single stone without neighbors is the minimal possible group. The **boundary** of a group is composed of the stones of the group each of which has at least one neighboring intersection unoccupied by stones of the same group. The **neighborhood of a stone** in a boundary is all its vacant neighboring intersections. The **neighborhood of a group** is the union of the neighborhoods of all stones in the boundary. The neighborhood of a group consisting of a single vacant intersection is called the **single intersection neighborhood** of the group. If a group has a single intersection neighborhood and the opponent now has the turn to play, the opponent can **capture** this group by placing his stone on this single vacant intersection. If this intersection is a single intersection neighborhood of more than one group, all these groups are captured. Naturally at most four groups can share a single intersection neighborhood. The group is said to be **surrounded** if there are no intersections in its neighborhood.

The black stones in Figs. 1 (A) (B) form one group, and in Fig. 1 (C) there are two black and two white groups. The basic form of capture is the capture of one stone (Fig. 2). A capture of a group of three stones is shown in Fig. 3.

Consider the situation shown in Fig. 4. The white stones form one group and its neighborhood consists of two intersections. Assume that a black stone has been placed on one of them (triangular mark in Fig. 4). The white group cannot be captured because there still remains one vacant intersection in its neighborhood. The black stone has become a group with no neighbo-

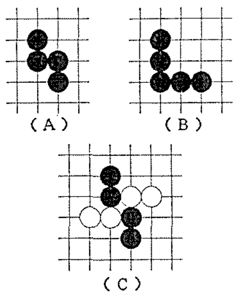


Fig. 1 Relation of stones.

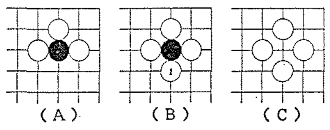


Fig. 2 Capture of one stone.

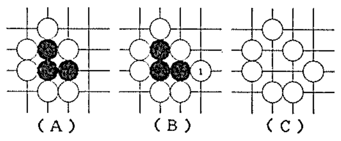


Fig. 3 Capture of stones.

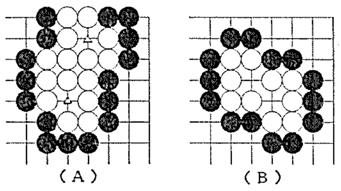


Fig. 4 Alive situation.

rhood on the board. To avoid this strange situation, the rules of GO game do not allow this move, and thus such a group of white stones can never be captured. We will call such a group alive.

Now consider the situation shown in Fig. 5(A). The white stones form two groups. Assume that a black stone has been placed on intersection (1). Although it is placed on an intersection entirely surrounded by white stones, it is alive because the effect of this move leads to the capture of the white group of three stones. These white groups are not alive because they can be captured as shown in Fig. 5 (B)

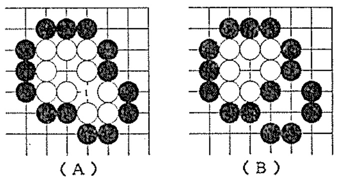


Fig. 5 Not-alive situation.

The intersection in the neighborhood of a group will be called the **eye of the group** if it is surrounded by stones of the group's color. Eyes can be shared by groups. The eye of an alive group will be called the **perfect eye** if it is surrounded by stones of only this group or if it is shared with only alive groups. The eye which is not perfect will be called the **imperfect eye**. It is easy to understand that the alive group should have at least two perfect eyes.

The aim in GO game is to secure territory and to surround as many vacant intersections as possible at the end of game. The territory is the region of GO board completely enclosed with groups which are alive Therefore, the ability to determine which groups are alive is of basic importance for the success in GO game.

3. Algorithm

The function of this algorithm is to analyze the static situation on the GO board. It can determine whether or not there is at least one group alive on the board according to our preceding criterion that a group is alive if it has at least two perfect eyes. The algorithm (see Fig. 6) first partitions all stones of a given color into groups and marks all eyes of these groups. Then it makes sequentially analyses of all the groups with respect to the

```
Alive(plate,color)
{
      captured = TRUE;
      set_of_groups = divide_into_groups(plate,stone);
      while( (captured==TRUE) && not_empty(set_of_groups) ) {
            mark_all_eye(plate,color);
            group = first_of(set_of_groups);
            captured = FALSE;
            while( not_empty(group) ) {
                  if( count_eye(plate,group)<2 ) {
                        remove_from(set_of_groups,group);
                        remove_eyes(plate,group);
                        captured = TRUE;
                  }
                  group = next_of(group);
            }
      if( not_empty(set_of_groups) )
            return( is_alive );
      else
            return( is_not_alive );
```

Fig. 6 Algorithm.

number of their eyes. If a group does not have at least two eyes, it is pruned from the board. If at least one group has been pruned from the board, analysis of all the remaining groups is repeated. The algorithm ends if no groups can be pruned. If at least one group has remained on the board, it may be concluded that on the board there is at least one group with two perfect eyes, and so the result of analysis is that the situation of the stones of the given color on the board is alive.

It will be explained how the algorithm recognizes alive groups in the simple examples shown in Figs. 7, 8, and 9. Suppose that the stones on the

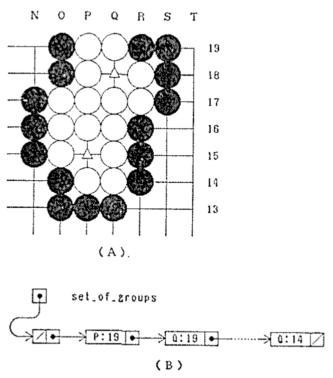


Fig. 7 Process to solution.

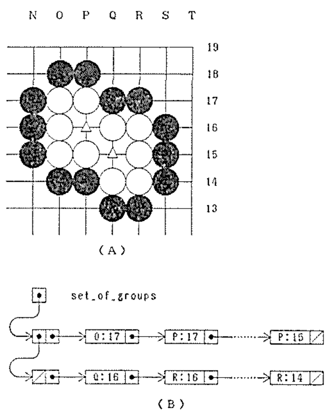
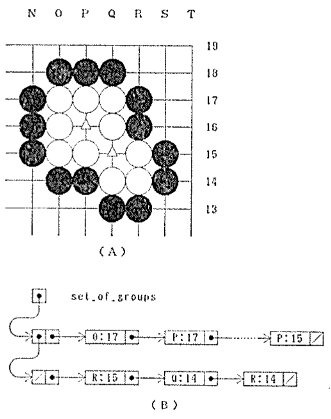


Fig. 8 Process to solution.



figures are only the ones on the board and that we are to analyze the situation of the white stones. For the white stones in Fig. 7, the algorithm will find only one group and its two eyes, so the analysis ends with an affirmative answer after the first iteration since there are no groups with less than two eyes. The group is alive and so is the situation of white stones. For the white stones in Fig. 8, the algorithm will find two groups and their two eyes. As in the preceding case, the analysis ends with

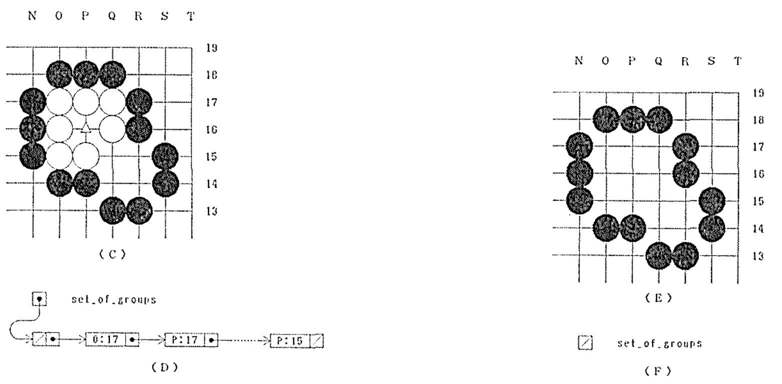


Fig. 9 Process to solution.

an affirmative answer after the first iteration. For the stones in Fig. 9, the algorithm will find two groups, but the group of three stones is found to have only one eye. Therefore. in the first iteration, this group and its eye (see Fig. 9 (B)) is pruned from the board and in the second iteration the group of seven stones is found to have only one eye. Thus, it is removed too, and the algorithm ends with a negative answer because no white groups have remained on the board.

4. Application

This algorithm applies to a program for solving alive situation puzzles in GO game. This puzzle is called **"Tsume GO"** in Japanese. "Tsume GO" is an element of GO game which appears anytime and anywhere on the GO board in course of game. The program which uses this algorithm and includes other functions adopt the **MIN-MAX principle.** To analyze a game, the

MIN-MAX principle makes a game tree. The game tree indicates the process of the game. Fig. 10 shows a sample of game tree to analyze the puzzle. Each point indicates a situation, and each line indicates a move. In this case, the best move for white, i. e., the answer for the puzzle, corresponds to the bold line, because as a result of this move the white player may expect that white stones are alive after all. This method is called the **full width search** by the MIN-MAX principle, and it requires so much time as not to be feasible. We have tried to speed up the analysis. For this purpose, the authors use three methods, i. e., the **alpha-beta pruning method**, the **secondary search method**, and the **register method** which registers all situations, evaluations, and a part of game tree in memory. An example of alpha-beta pruning is shown in Fig. 11. The broken lines which indicate moves of situation are not

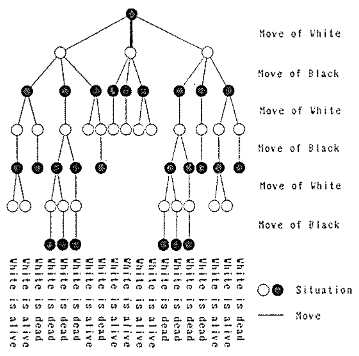


Fig. 10 Game tree.

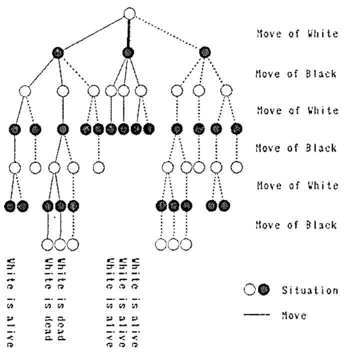


Fig. 11 Alpha-beta pruning.

searched actually. The number of moves to be searched actually is decreased from 51 to 19 by this method. An example of secondary search is shown in Fig. 12. We decide the level of boundary, and this method searches such a situation as has an evaluation of depth less than the boundary. No results of this searching gives the best move, and then a deeper situation is searched. The number of moves to be searched is decreased from 51 to 16 by alpha-beta pruning combined with this method. An example of register method is shown in Fig. 13. This method is such that all situations, and evaluations and a part of game tree above some level are registered and that if the same situation as has been registered before is found, the associated evaluation and part of game tree are copied from memory. In this case, when the situation of move (E:1)—(G:1)—(D:1) is searched, the same situation (D:1)—(G:1)—(E:1) which has been searched before is found, and thus the evaluation and the part of game tree are copied.

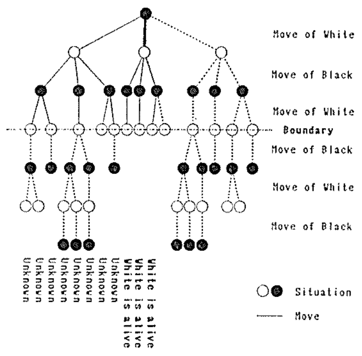


Fig. 12 Secondary search.

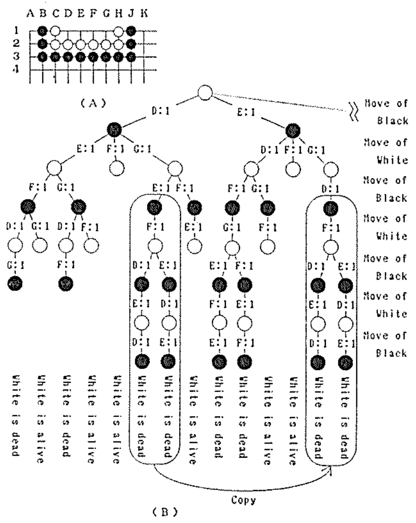


Fig. 13 Register method.

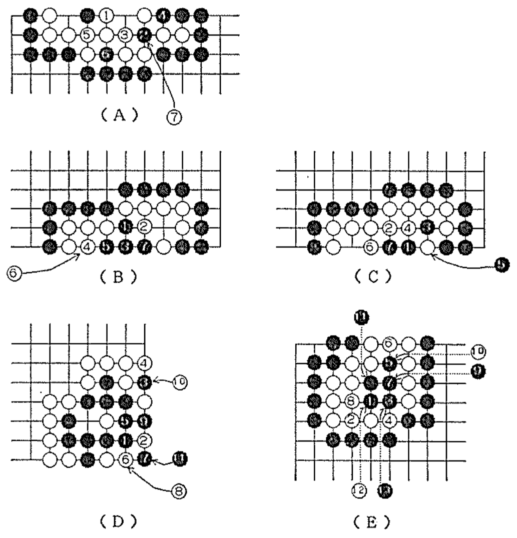


Fig. 14 Results of execution.

The authors have installed this algorithm on a mini-super computer (CO-NVEX Cl-XP) by the C language, and solved some problems of "Tsume GO" game. Fig. 14 shows results of this execution. To solve the problems, Fig. 14 (A) has needed 108 minutes, Figs. 14 (B) and (C) two minutes, Fig. 14 (D) 233 minutes, and Fig. 14 (E) 24 minutes.

5. Conclusions

This algorithm makes it possible to evaluate the situation of stones and to solve alive situation puzzles within realistic time. It is possible to apply the method to solving the GO game. The authors are investigating a hardware of Parallel Processings that can execute this algorithm.

6. Acknowledgment

The authors would like to express our gratitude to Mr. Pavel Tvrdik for reading the manuscript.

References

1) Reitman, W. & Wilcox, B., "Pattern recognition and pattern directed inference in a program for playing GO", in D. Waterman and F. Hayes-Roth (Eds.), Pattern-directed inference Systems, New York Academic Press, pp. 503–523 (1978).

2) Berliner, H., "A chronology of computer chess and its literature", *Artificial Intelligence* 10, 201–214 (1978).

3) Sanechika, Ohigashi, Mano, Sugawara & Torii, "Notes on Modelling and Implementation of the Human Player's Decision Processes in the Game of GO", *Bul,* of *ETL* 45, 1–2 (1981).

4) Mano, Y., "An approach to Conquer Difficulties in Developing a GO Playing Program", *Journal of Information Processing,* Vol. 7 (1984).