An Approach to the Ko Situation in Life and Death Problems in Computer Go

Chi-Chieh Chiang[1], Shi-Jim Yen[1], Tai-Ning YangP[2P], Jr-Chang ChenP[3P], Shun-Chin HsuP[4], Chen-Kung Wu[5]

[1]Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan.
sjyen@mail.ndhu.edu.tw
[2]Department of Computer Science, Chinese Culture University, Taipei, Taiwan.
tnyang@faculty.pccu.edu.tw
[3]Department of Applied Mathematics, Chung Yuan Christian University, Chung Li, Taiwan.
jcchen@cycu.edu.tw
[4]Department of Information Management, Chang Jung Christian University, Tainan, Taiwan.
schsu@mail.cju.edu.tw
INTERNATIONAL GAMES SYSTEM CO., LTD., Taipei, Taiwan.
slime573@gmail.com

Abstract

Life-and-death problems (tsume-Go) are among the most important issue in computer Go. When we apply solve life-and-death problems use minimax search technique, the circular situation (ko) often occurs and interferences the search process. In this paper, we present a new model to solve this problem. Minimax search technique is processed normally in this model, which solves as many as difficult ko situations. The proposed model for solving life-and-death problems improves the ability for ko-fight in computer Go programs.

Keywords: Computer Go, Minimax Search, Life-and-death problems, Ko

1. Introduction

Life-and-death problems are important issue in computer Go. However, many of the proposed tools for solving this problem cannot deal with complex ko-fight problems, such as one-step-to-ko, hanami-ko and mannen-ko. [6][11,12,13]

The minimax search tree is a general algorithm in computer Go. Because of the large branching factor in computer Go, the global problem must be separated into semi-independent local problems. The life-and-death problem, or "tsume-Go", is one of the most important of these. This article proposes a search tree structure which analyzes the data pointer structure of the ko-fight search tree, identifies their differences and quantifies them in a comprehensible way so that they can be solved. A method of presenting the results of such a search is also described. A life-and-death problem solving model and the aforesaid techniques can improve the capability of computer Go programs to win a complex ko-fight, which has not been achieved thus far.

2. Life-and-death problems

The concept and data structure in this study are as follows: every search tree node corresponds to one board state and node connected with another child node by move generation. Let node ***A*** corresponds to state ***a*** (board state) and node ***B*** corresponds to state ***b*** (board state), state ***b*** can amount by move generation of state ***a***. As mentioned, node ***B*** is the child node of node A. Every node is given a life or death state after search. Two conditions are possible: *L* (life) and *D* (death). The connected expressions for discriminate functions are listed below:

1. If the stone chain value of approximate eye spaces > 1.5, then the condition is *L* (Life).

2. Otherwise, the condition is *D* (Death).

3. If the opponent cannot amount any *Candidates* to kill the player, then the stone chain is *L*.

4. If the player cannot amount any *Candidates* to kill opponent, then the stone chain is *D*.

(If a move can change the life-and-death state, we called "*Candidate.*")

3. The proposed method for Ko fight

The proposed method of solving the ko fight is demonstrated in the following example:State (A), is a simple *ko*-fight problem. The Black becomes alive by playing move at (*a*), and the White can make ko by playing mov at (a), as well. This state corresponds with the tree structure in Fig. 1(a) of node (A) if White plays first. (The lines indicate that if White plays move *a*, so do the others.)

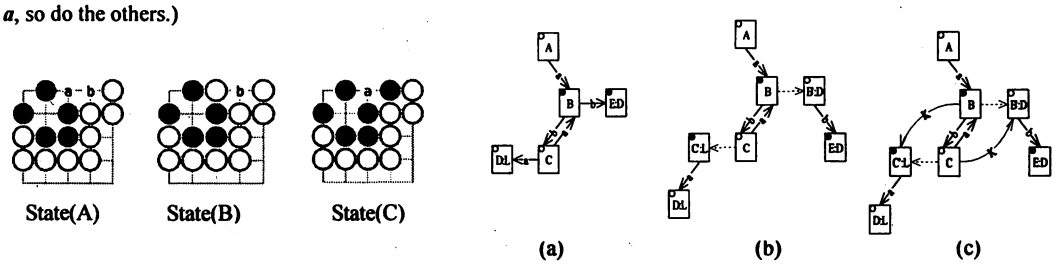


Fig. 1. A simple *ko* board with a search tree

Figure 1(b) can be drawn if the pass move is considered. The dotted line represents the pass move. For example, node *C* has the same board state as node *C'* but with a different turn because White passes and Black can live by playing move at (*a*). The condition *L* is given for node *C* because it can live by playing move at (a). Of course, node *D* is *L*(live) as well. In node *B'*, the Black is captured, so the condition *D* (death) is given for node *B* and node *E* as well. Furthermore, in addition to letting node *B* has information that it can live if win a ko-fight, a pointer *K* is added to node *C'* (and node *C*), and is called the winning-ko-move. This represents the basic ko-fight search tree, and more concepts are introduced in table 1 to solve more complex ko-fights.

If Black and White all have the same ko-threat in the *Go* board, the ko-fight move sequence is important. The player who first takes the ko will win the ko-fight. This concept is applied in the search tree structure.

category of move generation	Point to	The same with the origin board	It will change the value of pointer of condition of node	It will change the return value
Normal move	Opponent board	Yes	No	No
Pass move	Opponent board	No	Yes (if find the seki)	No
Winning-ko-move	Self board	Yes	No	Yes (add information of ko-fight)

Table 1. Concepts of more complex ko-fights

The concept is demonstrated below and then in a simple example.

1. **If Black lives by winning the ko and points to a node with condition of Death (Life), then it return a value of KbD (KbL).**

2. **If White wins the ko and moves the pointer to a node with condition of Death (Life), then it returns a value of KwD(KwL).**

In fig. 2, if White first takes the ko, It will need an addition ko-threat for Black. According to this concept, we know KbL and KwD are different. KbL is well than KwD. If a stone chain want to make alive in this kind of situation, KbL is better than KwD (for Black).

KbL > KwD

Completely life is better than life by ko, and life by ko is better than death. The order is

L > KbL > KwD > D

IF the state of node *E* is also *L*, Black will live even if it passes. Therefore, an additional test is needed to determine if it is really worth taking ko in this situation. This helps to avoid unnecessary ko-fights.

4. Accumulated ko information with life and death

Node state can be represented by ko information according to the last paragraph, *e.g.* KbL, KwD. An important question is how to determine the parent node state if the state of children nodes are represented by ko information. The simplest way is to accumulate information such as KwKwD, KbKwD and KbKbL.

Figure 3 is a ko-fight example. State of node ***D*** is KwD in Fig. 3(c). In Fig. 3(d), Node ***C*** can get KbL, then node ***A'*** gets KbL and node ***B*** gets KwKbL. Finally, node A gets KwKbL in Fig. 3(e). The final result is a one-step-to-ko (approach-move-ko). In that calculation, the comparison was used: **KwKbL > KbL**. Because the one-step-to-ko profits Black in this example.

KbL is better than KbKbL because Black pays less cost (L > KbL). Therefore, KbL > KbKbL, and we get

KwKbL > KbL > KbKbL

KwKwD > KwD > KbKwD.

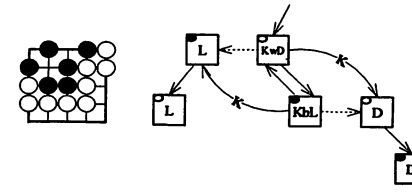
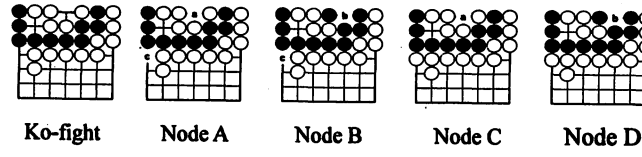
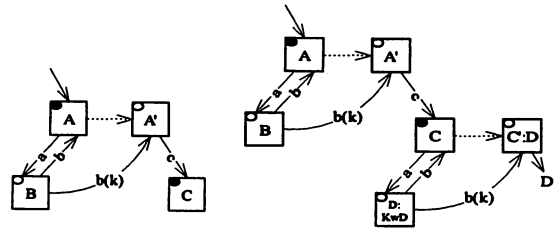


Fig. 2. A ko example with a complete search





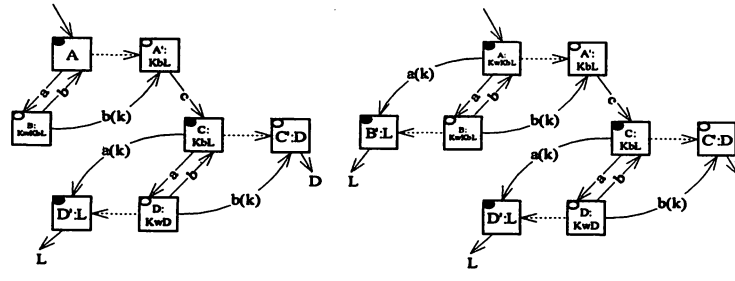


Fig. 3. A complex ko-fight example

The condition of KbKbL and KwKwD must also be considered. The proposed analysis has three steps:

1. **If both Black and White have the some ko threats, KbKbL can live but KwKwD cannot. Therefore, KbKbL > KwKwD.**

2. **If Black has more ko-threats than White, both KbKbL and KwKwD can live, but KwKwD is profitable because White must pay additional costs (lose some ko-threats).**

3. **If Black has fewer ko-threats than White, neither KbKbL and KwKwD can live, KwKwD is profitable because White must pay additional costs.**

The KbkbL and KwkwD states are influenced by number of ko-threats. The final conclusion is:

1. If player and opponent all have the same ko-threats, KbL>KwD, also

KwKbL>KbL>KbKbL>KwKwD>KwD>KbKwD

2. If Black and White has different ko-threats, KbL=KwD, also

KwKbL=KwKwD>KbL=KwD>KbKbL=KbKwD

The reader is encouraged to calculate those examples in Fig. 4.

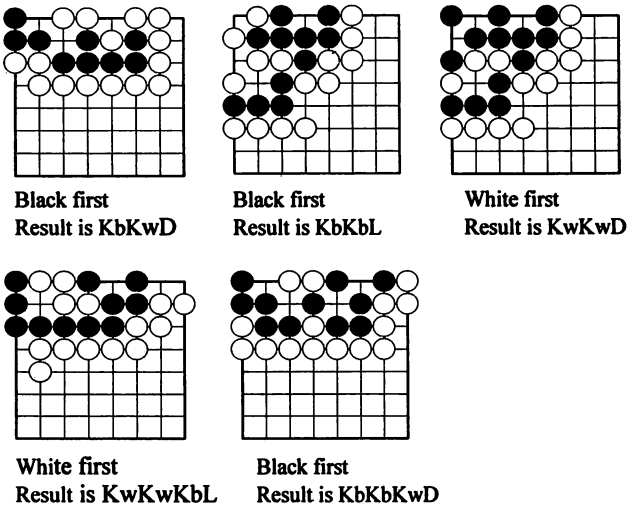


Fig. 4. Five complex ko-fight examples.

5. Summary

This article examined ko-fight problems. If most life-and-death ko problems are solved, the next step is up to globe board not only local life and death. This issue is more complex and requires a good evaluation function for the full board. Using a computer program to identify ko-threats is also an important issue involving evaluation of move generation for the entire board.

6. Reference

[1] D. Dyer, "An Eye Shape library for Computer Go," draft, available from the author's homepage, 1995.

[2] D. Dyer, "Global Evaluation Strategies in Go," draft, available from the author's homepage, 1995.

[3] D. Dyer, "Searches, tree pruning and tree ordering in Go," *Proceedings of the 2nd Game Programming Workshop*, 1995.

[4] H. Landman, "Eyespace values in Go," *Proceedings of the Games of No Chance*, Vol. 29, pp. 227-257, 1994.

[5] T. Cazenave, "Generation of Patterns with External Conditions for the Game of Go," *Proceedings of the advances in Computer Games Conference*, 1999.

[6] M. Muller, "Computer Go: a Research Agenda," *ICCA Journal*, Vol. 22.2, pp. 104-112, 1999.

[7] M. Muller, "Decomposition Search: a Combinatorial Games Approach to Game Tree Search, with Applications to Solving Go Endgames," *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, Vol.1, pp. 578-583, 1999.

[8] M. Muller, "Race to Capture: Analyzing Semeai in Go," *Proceedings of the Game Programming Workshop*, 1999.

[9] Z. Chen, "Programming Technics in Handtalk," http://homepage1.nifty.com/maznaga/IGO/handtalk, 1999.

[10] K. Chen and Z. Chen, "Static Analysis of Life and Death in the game of Go," *Proceedings of the Information Sciences*, Vol. 121, no. 1-2, pp. 113-134,1999.

[11] T. Wolf, "The program GoTools and its computer-generated tsume go database," *Proceedings of 1st Game Programming Workshop*, 1994.

[12] T. Wolf, "About problems in generalizing a tusmego program to open positions," *Proceedings of the 3rd Game Programming Workshop*, 1996.

[13] T. Wolf, "Forward pruning and other heuristic search techniques in tsume go," *Proceedings of the Information Sciences*, Vol. 122, pp. 77-90, 2000.