2005－GI－14（9）
2005／9／5

囲碁の部分問題における両利きの探索

美添 一樹*　　　　今井 浩

概 要

囲碁においては、盤面全体に対する、速く正確な評価関数を作ることは困難である。そのため、小目標ごとのサーチが、囲碁プログラムの間では広く用いられている。ここで問題になるのが小目標間の依存関係である。小目標の勝敗に影響を与える範囲を求めて依存関係を解決するアプローチが研究され始めている。 *relevancy zone* という概念が使われ始めているが、この求め方を改良することを目標としたアルゴリズムを提案する。二つの小目標についてそのような範囲が重なっていれば、そこが両利きの候補となる。

Searching for Double Threats in Subproblems of the Game of Go

Kazuki Yoshizoe *　　　　Hiroshi Imai

Abstract

It is difficult to make a fast and accurate evaluation function for the whole board in the Game of Go. Therefore sub-goal directed search is used widely among Go playing programs. One problem of sub-goal directed search is dependencies between sub-goals. There are several researches which aim to resolve the dependencies by obtaining the area which involves with the result of sub-goals. An idea called *relevancy zone* is being used in some researches. In this paper, we introduce an algorithm which search for an area which would improve *relevancy zone*. The intersection of two such areas will be the candidate for double threat.

1 Introduction

Compared to the programs of chess like games, Go playing programs are weak. One of the reasons is that in the game of Go, existing evaluation functions are inaccurate and/or requires great computational effort.

It is widely believed that it is extremely difficult to make fast and accurate evaluation function for whole board in the Game of Go.

One reason of the difficulty is that the board is almost always “not quiet”. That means, *quiescence search* is always needed before evaluation. Fortunately in chess like games, the moves searched in quiescence search will usually turn out to be good moves.

* 東京大学大学院情報理工学系研究科コンピュータ科学専攻yoshizoe@is.s.u-tokyo.ac.jp
* University of Tokyo, Graduate School of Information Science and Technology

But in Go most of the moves searched in quiescence search will not be actually played. To evaluate a node, we have to extend search depth, and after the search, we have to return to the node where we have started *quiescence search*. It means we have to search further, if we want to evaluate a node. This makes it very time consuming to evaluate a node in the game of Go.

Because of this difficulty, evaluation of the whole board cannot be performed so frequently. Therefore, many Go playing programs use sub-goal directed search. A sub-goal is a local target which is easy to evaluate, such as connecting stones, capturing stones, life and death problems, etc.

There are many sophisticated algorithms for sub-goal directed search. For example, state of the art life-and-death problem solvers such as GoTools [4] and Df-pn based Tsumego Solver [2] are very strong. Their speed and accuracy had overcome the level of professional players. But overall strength of Go playing programs remain at the level of novice players. (It is widely believed that the strongest Go playing program is about 10 kyu in AGA[1] rating.)

To improve Go playing programs, it is necessary to fill in the gap between the strength of sub-goal searches and the strength of overall play. One critical element for this purpose is to resolve the dependencies between different sub-goals. Without this ability, it is difficult to find multipurpose moves, and also difficult to evaluate such moves exactly. For example, it is difficult to find moves which aims to connect or to live locally at the same time.

The paper written by Cazenave and Helmstetter [1] and another paper by Jan Ramon and Croonenborghs [5] are two examples of searching double purpose moves using the idea of *trace* [1] or *relevancy zone* [6].

In this paper we present an algorithm for obtaining an area which we call *inverting threat*. It is an attempt to find a more strict area compared to *relevancy zone* and *trace*.

Related works are described in section 2. The motives for searching *inverting threat* is explained in section 3. An algorithm for searching *inverting threat* is shown in section 4. The result of an experiment is in section 5. Finally, there is the conclusion and future work in section 6.

2 Related Work

The definition of *relevancy zone* is described in the paper about lambda search [6] written by Thomsen. This is an area which possibly inverts the result of local sub-goal search, in which if stones are played. Thomsen used this idea to solve local problems of Go. *relevancy zone* is strictly defined and easy to calculate, but as Thomsen already pointed out in his paper [6], it does not guarantee correctness.

In Transitive Connections [1], Cazenave and Helmstetter defined *trace*. They used *trace* to resolve the dependencies between two connections. *trace* is obtained by adding the intersections that involved with the tests performed during

[1]American Go Association

the local sub-goal search. Unfortunately, the definition of *trace* is not further described in [1].

In the paper, unions and intersections of *traces* are used for searching transitive connections. Unsurprisingly, using union was safer but slow, and using intersection was fast but less safe. This is the first attempt to combine multiple goals into one search using the idea such as *trace*. However, the sub-goals searched in this paper is limited to connections problems.

In a paper written by Ramon and Croonenborghs [5] *relevancy zone* is used for searching compound goals. This can be said to be a generalization of [1]. In this paper, sub-goals are not limited to connections but also includes captures and living. Their algorithm builds compound goals by combining sub-goals using logical AND/OR/NOT. On improving speed for searching compound goals, they use *relevancy zone* obtained from each sub-goal search.

3 The Motive for Searching Inverting-Threat

We propose to call an area *inverting-threat*. *Inverting-threat* has the same purpose as *relevancy zone* and *trace*.

In each sub-goal directed search, there are local winner and local loser. Inverting-threat is a move, or a set of moves which will invert the local winner if winner passes. We also define the order of inverting-threat. 1st order inverting-threat is the set of moves which will invert the local winner to loser if the winner passes 1 time.

The definition of *relevancy zone* is strictly given, and fairly easy to calculate, and also works well in many cases such as described in [6, 5]. But it does not guarantee the correctness. As already pointed out in Thomsen's paper [6], *relevancy zone* can miss intersections which actually affect the result of the sub-goal search. This might occur especially when the liberties strings adjacent to *relevancy zone* are small (maybe less than 2). In the middle game of Go, complicated situations always appears in which many stings are neighboring each other and many of them have small number of liberties. In such cases, the possibility that *relevancy zone* is not correct is considerably high.

We can only guess about *trace* because the definition of *trace* is not strictly given in the [1]. We reckon that *trace* is covering larger area compared to truly relevant area. Covering larger area makes the possibility to overlook the intersections which are really an inverting threats.

In the paper "Transitive Connections" [1] Cazenave and Helmstetter used two kinds of move candidates in searching transitive connections. One is the union of the traces, and the other is the intersection of the traces. Using union could solve more problems right, but was more time consuming. Using intersection was faster, but was less reliable.

By using *relevancy zone*, a program sometimes overlooks an inverting threat. Union of *trace* is large and makes multipurpose search time consuming. Intersection of *trace* is small but unsafe.

Above these facts, our ideal goal is to implement an algorithm to find such areas, which would not overlook an inverting-threat and also does not include false threat. Our idea is that the most important part of such areas are 1st order *inverting-threat*. So in this paper, we focus on finding 1st order *inverting-threat*.

4 Algorithm for searching Inverting-Threat

We have implemented an algorithm for searching 1st order inverting-threat.

Our algorithm follows 3 steps.

1. Do normal local sub-goal search and find the local winner and the local loser.
2. Do second search which finds other winning moves for the local winner.
3. Retrieve 1st order inverting-threat from the search tree of the second search.

4.1 1st step - Normal Search

There is nothing special about the 1st step. This step is for determining local winner and the local loser. We have implemented a df-pn [3] based search algorithm for this purpose.

4.2 2nd step - Searching Other Winning Moves

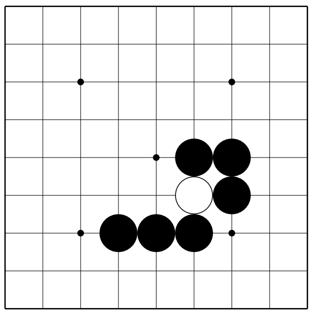

Figure 1: Captured White Stone

Let's consider the case shown in figure 1. In the 1st step, search for capturing the white stone would be done. The search is done twice, one black plays first, the other white plays first.

The result shows that white cannot escape from being captured even if white plays first. In this step, we search for the inverting-threat which turns the loser(white) into the winner.

To search for the inverting-threat, we cannot use the result of the 1st step immediately.

Figure 2 shows two examples of black's winning moves if white played first. Left example shows a capture by a ladder, and right example shows a capture by a loose ladder (or geta). In many search algorithms, it is likely that capturing by a ladder is searched first. In such case, other winning moves such as loose ladder (or geta) are pruned and would never be searched.

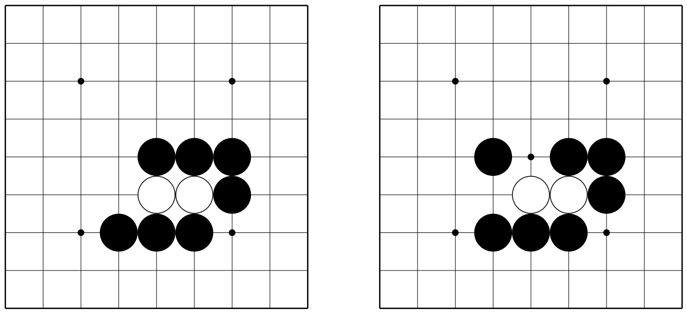

Figure 2: Ladder and Loose Ladder(geta)

In this case, white wants to search the *inverting-threat* which makes the white stone possible to escape (if black does not react).

If only capturing by a ladder was searched, white would misunderstand that ladder breakers are part of the *inverting-threat.* White would play a false ladder breaker, and black will ignore the move, then white tries to escape just to get caught by a loose ladder.

Therefore, ideally all possible winning moves (for the local winner) should be searched. Currently, we are doing a almost mini-max search for this step. Pruning is done only when parent node is a pass move.

4.3 3rd step - Retrieve Inverting-Threat

In this step, 1st order inverting-threat is retrieved from the game tree which is already searched in the 2nd step.

If we have an ideal move candidate generator which generates the candidate moves for *inverting-threat*, all we have to do is to check each move if that is really an *inverting threat.* (We can check it by allowing the loser to play two consecutive moves.) Since we cannot have such a move generator, we have to retrieve the information from the lower nodes in the game tree.

Figure 3 shows Lose in OR node. For OR node to be a Lose, all child nodes must be Lose. If a child node is Lose, there is at least one Lose node among the grandchild nodes. The local loser can choose any losing move. Therefore,

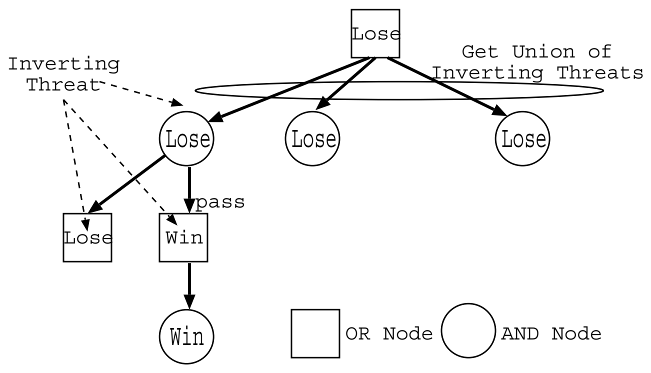

Figure 3: Retrieve Inverting Threat. OR Node.

the *inverting-threat* will be the union of each *inverting-threat* passed from the children.

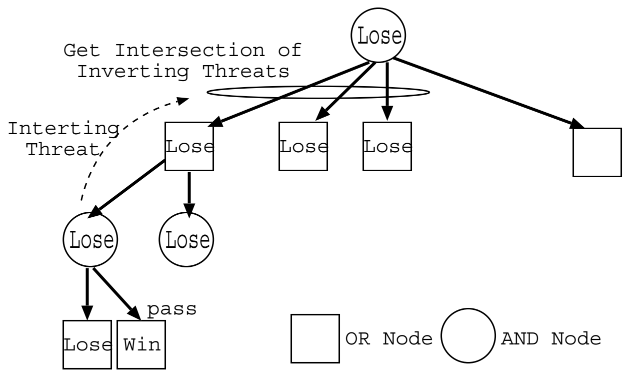

Figure 4: Retrieve Inverting Threat. AND Node.

Figure 4 shows Lose in AND node. For AND node to be a Lose, at least one of the children must be Lose. The local winner can choose best Lose for the winner. Therefore, the *inverting-threat* will be the intersection of each *inverting-threat* passed from the children.

5 Results

Our algorithm successfully finds inverting threat shown in figure 5.

But in the 2nd step of the algorithm, the number of searched nodes are extremely large. In the 1st step, df-pn based capturing search only searched 98 nodes in total. In the 2nd step, our algorithm needed to search more than

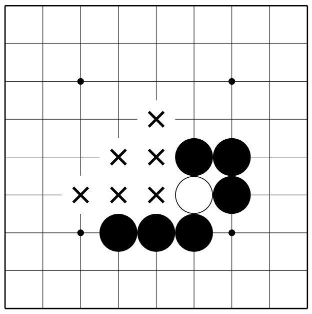

Figure 5: 1st order Inverting Threat

1,000,000 nodes.

There are two reasons for this. One reason is the number of move candidates are increased (forward pruning decreased). This is because the moves which are only candidates of *inverting threats* were added.

The other reason is searching for other winning moves results in less pruning (backward pruning decreased). Searching for other possibilities for other winning moves near the terminal node exponentially increased the number of searched nodes compared to normal search algorithms especially capturing in a ladder.

6 Conclusion and Future Work

We have implemented an algorithm which searches for an area which we call 1st order *inverting-threat*. However, in one of the steps of the algorithm, it consumes computational time which is very close to that of mini-max search.

To improve the speed, a more sophisticated pruning specially designed for inverting threat search should be implemented.

We will use this result to implement a program which solves local problems of Go which includes multiple sub-goals.

References

[1] Tristan Cazenave and Bernard Helmstetter. Search for transitive connections. *Information Sciences*, December 2004.

[2] Akihiro Kishimoto and Martin Muller. Dynamic decomposition search: A divide and conquer approach and its application to the one-eye problem in go. In *IEEE Symposium on Computational Intelligence and Games (CIG'05)*, pages 164–170, 2005.

[3] Ayumu Nagai. *Df-pn Algorithm for Searching AND/OR Trees and Its Applications.* PhD thesis, Department of Information Science, University of Tokyo, Tokyo, 2002.

[4] M. Pratola and T. Wolf. Optimizing gotools' search heuristics using genetic algorithms. *ICGA Journal*, 26(1):28–49, March 2003.

[5] J. Ramon and T. Croonenborghs. Searching for compound goals using relevancy zones in the game of go. In J. van den Herik, Y. Bjornsson, and N. Netanyahu, editors, *Fourth International Conference on Computers and Games*, Ramat-Gan, Israel, 2004. ICGA.

[6] Thomas Thomsen. Lambda-search in game trees - with application to go. *ICGA Journal*, 23(4):203–217, December 2000.