About problems in generalizing a tsumego program to open positions

Thomas Wolf
School of Mathematical Sciences
Queen Mary and Westfield College
University of London
London E1 4NS
T.Wolf@maths.qmw.ac.uk

July 12, 1996

1 Introduction

At the first Hakone Game Programming Workshop 1994 a report was given about `GoTools`, a program for investigating closed life/death positions [1]. Work done since that time included

- the speed-up and clean-up of several parts of the code
- an improvement of the interface
- an improved tactics sub-module
- further tests and debugging.

Recently work has been started with the aim to enable `GoTools` to investigate positions with an arbitrarily open boundary to finally make it usable within a game playing program. Problems encountered so far in this approach shall be reported in this paper. The sequence in which they are described is random and does not reflect priorities.

Throughout the paper we will use Black to name the side that wants to kill and White as the side that tries to live.

2 What is a life/death problem and what not?

To apply a life/death module within a game playing program, the first question is to decide for exactly which kind of problems it should be used. So far `GoTools` had only been applied to fully enclosed life/death problems where one could assigne a unique status to each possible first move, which is the status that this move enforces. Differently in real game situations, where at first a conceptual question has to be answered: *What kind of questions should be answered by a life/death module?* Examples are:

- What is the status of a given chain?
- Can all chains of a given group survive?
- Can at least one chain of a group survive?
- What is the best result achievable if the position is so big that at least some chains can survive?

The current position of the author is, to restrict the possible questions to those that can be answered with a YES or NO. Reasons for this restriction:

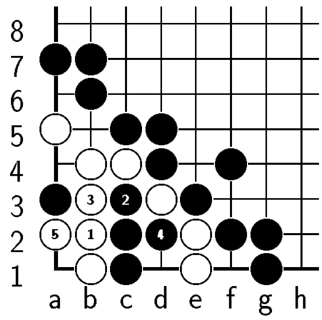
- A binary search that can answer only yes or no is faster than a search with more than two possible outcomes.
- If a big position is given where at least some chains survive for sure, then the question 'What is the best move such that the most chains with the most outside influence survive?' is the most general question that one can ask in Go. Such a question needs to include strategy and is not a tactical question. It is not a question for a specialized and fast module, it is more the question for the complete program to answer.
- This specification is not too much of a restriction. It still allows the first three of the four questions above to be asked and even allows all winning moves to be determined because one could check each possible first move one after another whether it guarantees the optimal outcome or not.

3 How to formulate the question?

In fully enclosed positions which are not too alive like Problem 1) below one can just take all enclosed white chains and treat the group as alive if at least one chain can live.

The difficulty with open problems is that if there is a real chance of escape then what is often asked is not whether some stones can escape but whether the majority of stones can escape. A simple way to do this, is to specify a chain which serves as a criterion of whether a position obtained during tree search is alive or not.

The difficulty with this kind of formulation of a life/death problem is, that the result may depend of which chain is selected to play this role. If, for example, in the problem below the chain with `E1` is selected then the position is dead as this chain can not be brought to life. On the other hand, if `B4` is selected then White lives.



Problem 1)
○ to move first and live.

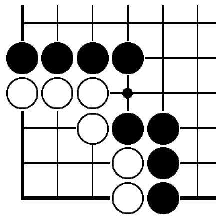
The answer to this problem is probably to repeat the search with slightly different questions. One could at first ask whether the majority of stones can survive, if yes, then perhaps whether all can survive, or if not then whether at least one chain can survive. One might be able to use intermediate results from the first search in subsequent searches to improve their heuristic.

4 Termination of a sequence of moves by death

In the closed problem version of `GoTools` a position is recognized to be dead if

- the white chains have been captured, or
- if the eye-space can not be split into 2 eyes and any single eye is not big enough to live or give seki, or
- if White has only moves left that result in a cycle of moves in which Black captures more stones than White or in which White has to find more external Ko-threats than Black (like double Ko in favour of Black).

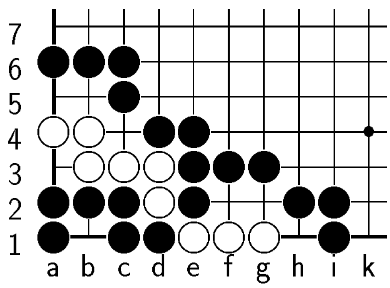
In an open position there still is the possibility of an escape. Therefore, in order to conclude that a position is dead, further moves must be made until all potential escape routes are blocked. That means that data which describe the strength of the boundary must constantly be updated. This does not mean that the boundary has to be closed completely. For example, holes like in problem 2) are not regarded as potential escape routes. The classification of holes does not only depend on the shape and size of the hole but also on the distance to white chains inside and outside of the problem. For this purpose some kind of influence function becomes necessary for open problems.



Problem 2)
● to move first and kill.

5 The selection of moves to be tried

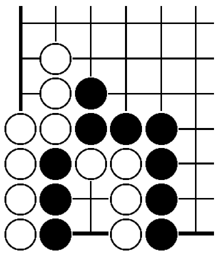
If a position is closed, then the moves to be tried by each side are about all legal moves inside the boundary. To speed up calculation one can exclude what appear to be useless moves but it is hard still to be accurate and not to miss the best variation. An example of how easy it is to disregard the winning move is the following problem which I was shown by Denis Feldmann:



Problem 3)
● to move first and kill.

Black wins by playing on `b1`. If the problem is open, how should the area be bounded out of which moves are to be tried? If the black enclosure has only a few smaller holes then moves in the vicinity of each hole are all that is to be added. This already can lead to serious difficulties as if moves are constantly played slightly too far outside then the problem grows and grows. In more general situations a white chain can penetrate the black enclosure. If this white chain is alive or has many outer liberties then no 'outside' moves to capture this chain make sense. On the other hand, if this white chain is weak then any move anywhere at the board which effects the safety of the white chain is a candidate to be tried. Even if there is no penetrating white chain but black enclosing chains that are not completely alive, then any move on the board effecting their safety is to be considered as well.

To give a very simple example, the program should recognize the following position as a terminal position as there is no point in catching the white chain on the run, but if there would be additional black stones on the `b-` or `c-` line then the search would have to be continued.



Problem 4)
● to move.

6 An increased number of moves

As discussed above, the unavoidable effect of a weak enclosure of the problem is that the number of moves to be considered in a given situation increases and the length of sequences of moves increases too. This can not be compensated by a good heuristics alone. Even if the heuristics is perfect and the winner always tries the best move first, then in order to prove the win all moves of the looser that could have an influence have to be tried, which may be too many and therefore impossible. A different approach would be to investigate the safety of boundary chains or of a white chain on the run separately with tactical modules, but the outcome depends on the status and liberties of neighboring chains and therefore these tactical searches have to be repeated in each node of the tsume go search which is expensive in time. An approach of this kind is currently pursued in `GoTools`.

The essence is that the weaker the boundary of life/death situations becomes, the less clear can tactical search and local live/death search be separated and the more they are dependent on each other.

For such big problems the question then also naturally changes its character. Because it might be obvious that at least some white chains will survive, the question then becomes what the optimal move is to kill/keep alive as much as possible. But a question of this character is in principle not simpler than deciding what the best move in the whole game is and which therefore should not be the task of a specialized life/death module.

7 Open problems and depth-first-search

Doing a tree search on enclosed problems there is a nice way to learn from a terminal leave: The last move of the opponent which won is a good candidate for the looser to try himself before. The program therefore can learn from terminal leaves of the search tree and learn the faster, the earlier a terminal leave is reached, i.e. a depth-first-search is appropriate.

Contrary in open problems. If, for example, a wrong move is tried first in order to block an escape attempt, then the escape tree-search may last very long or forever and the correct blocking move will never be learned and never be found and the problem can therefore not be solved. A breadth-first-search or a mixture of both is more appropriate here.

8 Summary

In summarizing, the following problems arise in writing an open boundary tsume go program and should therefore be answered.

- Which criteria does a position have to fulfil to be effectively investigated by a life/death program? The answer to this depends on what a life/death program is supposed to contain. So the real question to answer is actually more complex: Which kind of positions do allow a quicker investigation than the most general full board positions? The unknowns in this optimization problem are the position *and* the search algorithm.

  The proposal is that the problem should be clear enough to be formulated as a yes/no problem. As binary search algorithms, which answer only a yes/no question, are considerably faster than $\alpha - \beta$ versions with a parameter labeled range of outcomes, the problem should be simple enough to be answered with yes or no. Examples: Can chain x be captured? Can one of the chains $x, y, z, \ldots$ be captured? Can all chains $x, y, z, \ldots$ be captured? Consequences for the position: There should only be a small number of chains whose status is not clear and which are causally dependent on each other. That means either all of them will survive (with negligible few additional stones that may die) or all die, or some will definitely survive and only the future of one major chain is open. In other cases the problem is simply not a yes/no question and has to be investigated with a more complete search algorithm. The program `GoTools` currently classifies Ko-situations in repeated binary searches and seki is treated as alive.

- With the possibility of escapes in open problems a wrong blocking move may have as a consequence that a deep branch of the tree has to be searched. Therefore a mixture of breadth-first-search and depth-first-search is to be prefered against purely depth first search.
- To cope with the increasing search-space of open problems, subroutines for the capture of chains with few liberties and for closing holes in a wall become necessary.
- To have a reliable program it is necessary to test it with as many problems as possible. For that purpose an automatic problem generator of open problems would be useful similarly to the closed problem generator that is available in `GoTools` already for some years.

9 Acknowledgment and two announcements

The author is thankful for the financial help provided by the organizers to attend this workshop.

Having the opportunity to give this talk I would like to add two announcements which might be of interest to one or another from you.

The first is about a program that Bill Shubert (Oregon) wrote. It generates every day a new page 'Tsume Go Of The Day' on the world wide web using two of the problems `GoTools` has generated during the last years. Its address is http://imageek.york.cuny.edu/~wms/index.html .

The second is about a silent version of `GoTools` for enclosed problems. It has a simple file interface and is suitable to run as a life/death subroutine in a full go playing program.

References

[1] T.Wolf, 'The program GoTools and its computer-generated tsume go data base' in Proceedings of the first Game Programming Workshop in Japan '94, p. 84, Hakone 1994.