Checking Life-and-Death Problems in Go I: The Program ScanLD

Thomas Wolf and Lei Shen

Brock University, St. Catharines, Ontario, Canada,
twolf@brocku.ca,

Abstract. In this paper we introduce the program ScanLD (built on GoTools) which checks solutions of life-and-death problems for correctness. This is a task for which computer programs are especially useful. Not only can they do the computations, they can also do the handling of data in checking all moves of all solutions for optimality and reporting any errors that occur. Their refutation and their correction would be tedious and error prone if done with a computer, but interactively.
After discussing the different types of checks that are performed and giving some statistics resulting from checking a 500-problem tsume go book, some examples are given. A long list of mistakes that have been found is given in an on-line addendum to the paper.

1 Introduction

The computer Go program MoGo (see [2]), written by Sylvain Gelly and Yizao Wang, is built on the UCT algorithm invented by Levente Kocsis and Csaba Szepesvári ([3]). It has much recent success, for example, by winning the latest (March 2007) computer Go tournament on the KGS server (see [7]) and leading the continuously ongoing tournament on the 9x9 Computer Go Server (see [1]).[1] One might wonder whether there is still any purpose in using specialized Go software, for example, in analyzing life-and-death problems.

Although programs which use probabilistic methods (e.g., by evaluating positions using Monte-Carlo and searching the tree of moves using the UCT algorithm) are relatively strong Go-playing programs, they are not designed to *prove* statements.

In contrast, life-and-death programs like GoTools, give a result which is supposed to be exact (if the program is bug free) and the result consists not only of a best first move but also of a status of the position, including (if the status is ko) the number of ko-threats that one side needs to ignore in order to win. As a small study in [10] shows, the last few % of correctness cost an exponentially growing amount of computation time – for GoTools as a normal tree search algorithm – and we would claim even more so for probabilistic methods.

[1] The success of MoGo is the more remarkable both in that the size of its code is only a fraction of the amount of code that goes into established older programs and in that it scales well, i.e., it still has reserves and will probably be able to make good use of multi-core CPUs as they become more and more widely available.

A contribution which life-and-death programs can make is to pre-compute the status of enclosed areas and to store them in a data base which might be used by full game-playing programs. An example is a database of eyes surrounded by only one chain (monolithic eyes) which was computed recently (see [11]). As a by-product, a classification of strange positions was obtained which might be entertaining for Go players to look at (see also [11]). Other applications of GoTools include a program that generates new life-and-death problems (see [9], [6]) and an Internet application that solves problems online (see [8]).

In this paper GoTools is the workhorse behind the program ScanLD which is used to check thoroughly life-and-death problems including their solutions. This is described in the next section. It is followed by a description of how ScanLD was applied to 500 problems found in a Chinese tsume go book ([4]) which from now on we simply call 'the book'. In section 3 we show how many of these problems were solved by the program and how many gave problems. What might be of most interest to Go-players is section 4. It shows examples of problems where ScanLD uncovered non-optimal play in their solutions or other mistakes in the book. A long list of findings is given in [5]. Go-players can use the examples in this paper and the extensive list in [5] twice, once for solving the given problem and one more time by trying to find out what is wrong with the solution given in the book.

The life-and-death problems discussed in this paper are generally very interesting because at least some aspect of their solution was obviously missed by the authors, one of them being a professional 3-Dan player at the time of writing the book.

Although all move sequences computed from ScanLD in this paper and in [5] should be bug free, they may have 'features' resulting from performing a fast but simplified search in which, for example, seki is considered to be equivalent to life. This and other 'shortcuts' should be kept in mind when inspecting these sequences. For that purpose in the appendix all limitations of the programs GoTools (and thus of ScanLD) are listed for completeness.

2 About ScanLD

Life-and-death problems can be wrong in more ways than the obvious one that the shown winning first move is wrong. The following is a list of tests that are performed by ScanLD. The first 2 tests concern the problem itself, not the solution.

1. After ScanLD solves the problem it tests whether the side moving first can win, or at least reach a ko. If not then this is a less interesting problem.
2. Secondly, it determines the status for all possible first moves and thus finds the set B of all moves that give the same optimal result. If B includes more than one move then all moves in B and their likely follow-up sequences are shown. In that case the problem has no unique solution and is thus not a nice problem. We list them in section 4.3 if in the book no comment was made about the existence of another optimal solution.

3. The next test checks whether the first move of what is provided as the best solution, is among the moves in B. If that is not the case then all moves better than the best move given in the book are shown. Next, the status of the first move of each solution sequence is tested.
4. The program compares the given status of the first move with what the program had found earlier as the status for this move. If the given status of the solution is an unspecified ko[2] and if the program found ko as status too but a specific ko then this specific ko is from now on taken to be the correct status of the given solution, and is used as a reference when checking the next moves in the sequence.
5. Now the test of the solution sequences and the status of what the first move can reach begins. The following checks are performed for each given sequence and within each sequence for each move.
   The first check tests whether the move in the sequence is legal. The second check tests whether the move is optimal in the current situation, no matter whether the current status is the same as the original or whether it has changed due to earlier non-optimal play of any side.
   If a move is not optimal then the better move and the optimal status that it achieves are given. In addition, the program explains why the shown move is not optimal by showing how it can be countered with a likely follow-up sequence.
6. Sometimes a mistake by one side playing a non-optimal move in the solution sequence is compensated by a mistake from the other side afterwards or is followed by more mistakes by the same side. Therefore each solution sequence is checked to the very end to find all mistakes.
7. Finally, the order in which the alternative solution sequences are given in the book is checked and a comment is made if the side moving first achieves more in a later solution than in an earlier solution. This of course is not an error message. It only helps to sort solutions according to what they achieve.

In published problem collections variations of non-optimal moves are shown and discussed. These are of course not mistakes in the publication. Most often it is the wrong first move which is discussed and this would not lead to an error report by ScanLD. But if the discussed error consists of a later move then ScanLD would report this as a mistake. The current format of life-and-death problems as it is used by the programs GoTools and ScanLD does not allow us to mark single moves in a given sequence as known to be wrong. Therefore error reports of ScanLD have to be checked manually to see whether they coincide with the discussion in the publication.

3 Statistics

In this section some statistics for the success rate of ScanLD in finding mistakes in [4] is shown. The outcome of a test falls in general into one of the following categories which are listed again in table 1.

[2] This is mostly the case as very rarely is the precise ko-status given in problem collections, i.e., how many external ko-threats one side has to ignore in order to win.

- a Because ScanLD can solve effectively only problems with a closed boundary, there are problems which can not be closed easily without changing their status (1.8%).[3]
- – For another group of problems ScanLD closes the problems too widely or too narrowly. If the closure is too narrow the solutions are changed, if it is too wide then the problem is too complicated for the program. In these cases the boundary had to be adjusted by hand. The current routine used for closing a problem has still some room for improvement. An idea to be implemented in future is to use the solutions (if they are given) for constructing the enclosure such that the additional stones do not interfere with any move in any solution. On the other hand to make enclosures perfect is ultimately as hard as solving the problems. When ScanLD reports a mistake, it also prints the original position and the position after the closure to allow a quick inspection and if necessary a re-run with a human-improved enclosure if it was too narrow or too wide.
- b Although most problems are fully checked in 10-30 min, others (33.4%) take much longer or are simply too hard to be solved and checked by ScanLD in a reasonable time of currently 1 day. The 500 problems have been divided into 25 groups and each group was run in batch mode on a separate node of a Pentium 4 Beowulf cluster.
- c For another group (48.6%) of problems the program confirms all solutions in the book in the sense that all moves in all solutions belong to the set of optimal moves in each of the situations.
- d The remaining problems where the program identifies a mistake in the book, could be categorized according to whether the mistake changed the status from life to death, or ko to life, etc. but we have them in one category when the status changes to life or death (4.2%) and ...
- e a smaller group (2%) of problems where a ko is overlooked which prevents complete defeat.
- f In a relatively large group (8.4%) of problems the non-optimality of moves is only minor in that the number of necessary external ko-threats changes that are needed to win a ko or the side changes which needs ko-threats. Although being less dramatic, such errors can make a difference between winning or losing a game too. Due to their relatively high number these problems are not included in this paper and not in [5].
- g Finally we list a group (1.6%) of problems in which there is nothing wrong with the provided solutions except that the problem has more than one first move giving the same optimal result and this has not been mentioned in the book.
- – An extra service for which no statistics has been made in this paper concerns problems for which solutions are missed that are not optimal but that are better than the worst solutions discussed in the publication. For example, if

[3] Such problems would need stones added outside the weak boundary to build life outside which is worth to connect to and the whole situation would have to be enclosed by one big boundary which the program can not do automatically.

● has one move that kills and moves that lead to life discussed in the paper but not existing moves that at least give a ko then this leads to a comment by the program.

Table 1. Statistics of the 500 problems in the book

Category	**Problems**	**Percentage**
a. boundary not easily closed	9	1.8%
b. timed out	167	33.4%
c. agreement	245	49%
d. changes status to life/death	20	4%
e. missing ko	10	2%
f. change of # of ko-threats	42	8.4%
g. no bug, more than 1 best move	7	1.4%

Checking the 500 problems from the book put the program to a rigorous test because for each problem the book gives 3-6 solutions and for each solution about 5-20 moves, each leading to a position that had to be solved in order to check whether the move in the sequence is optimal. In this process 3 bugs were found in GoTools that were corrected. More changes were made to the interface of ScanLD, i.e., how it reports discrepancies and moves which counter those on the book.

The numbers in the table reflect the situation as of April 2007. We intend to have a closer look at the relatively large fraction of unchecked ('timed out') problems. We will try give them more time, use more memory to allow a larger hash table and hand optimize the closure of the boundary to reduce their percentage. Findings will be used to update the online available paper [5] of all mistakes that have been found. Other outstanding work includes an improvement of the procedures that enclose an open problem although this is not too urgent as a closure by hand is not a problem and is done quickly.

4 Problems with Problems

As outlined in the previous section, there are three categories of mistakes which will be discussed in the following three subsections. In all cases the first diagram (like diagram 1 below) shows the original problem as given in the book without extra boundary closure as added by ScanLD.

The second diagram (like 2) shows a solution with non-optimal play from the book. If the book assumes that all moves in the sequence are correct (in this book solution 1 for each problem) then the task is to find any error as stated, for example, underneath diagram 2 is written 'Find any error'. If the second diagram discusses a move, say ❸, which is known in the book to be wrong then we discuss this problem only if there is at least one more mistake following ❸ that is not mentioned in the book. In such a case the caption underneath the second diagram would say 'Find error after ❸' with the understanding that it could be move ④ that is wrong or any other move, like ❼. Such mistakes are documented in the follow-up article [5]. Here we have space for only one example in each category.

The third and possibly further diagrams each correct one move from the solution sequence in the second diagram as in diagram 3.

To make the paper self-consistent and useful for the reader, for each problem a solution with correct (optimal) play should be given. If the corrected diagrams do not already include one with optimal play (because one move in the sequence is known in the book to be wrong and then the corrected diagrams will contain this move too) then another (last) diagram with optimal play is added. This extra last diagram can be solution 1 from the book (if it was correct there) or a correct solution which was missed in the book, as in diagram 4. The extra last diagram can also show an additional solution if the problem has more than one best move, as diagram 12.

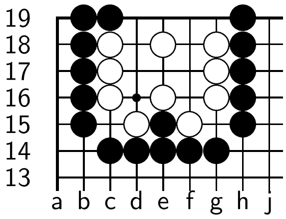
Go players could use this section for practice twofold: first, by covering all diagrams except the first one and trying to solve the problem on the left and a second time by uncovering the second diagram and trying to find the mistake in the solution sequence.

Most often a single diagram can only illustrate but not prove the statements that are made. In order to verify interactively comments in this paper the reader can solve and explore problems online at [8] but has to first close the boundary suitably.

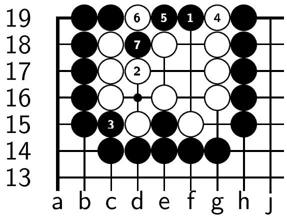
4.1 Mistakes changing the Status to Life or Death

For all problems in the book ● plays first. What we call 'solution 1' below is the first solution in the book which always is supposed to be the solution where both sides play optimally. Later solutions typically discuss non-optimal play.

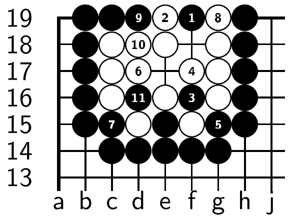
Problem 33, solution 1



● to move.
Diagram 1.



Find any error.
Diagram 2.



⑫ @ ❸
Diagram 3.

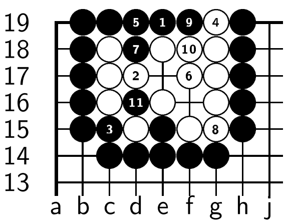


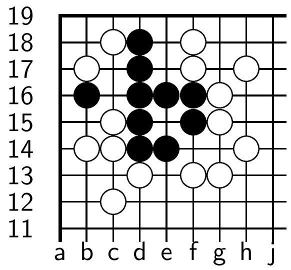
Diagram 4.

In Diagram 2, ○ needs to win a ko at d19 to live. But ○ can live by playing ② on e19 as shown in Diagram 3. The only ❶ that reaches at least a ko is shown in Diagram 4 and is missed in the book too.

4.2 Mistakes changing the Status to Ko

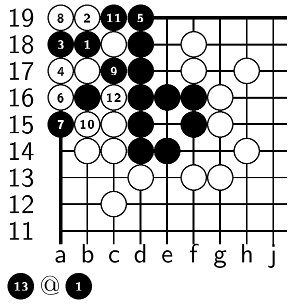
In the problems of this subsection moves are missed which would provide a ko for the side that otherwise would lose. We give one example; more are listed in [5].

Problem 181, solution 1



● to move.

Diagram 5.



Find any error.

Diagram 6.

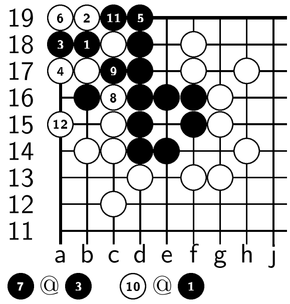


Diagram 7.

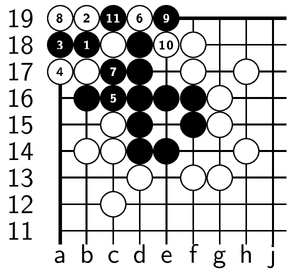


Diagram 8.

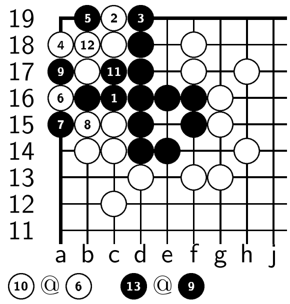


Diagram 9.

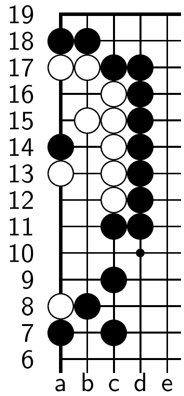
In Diagram 6 the move ⑥ is not optimal as it enables ● to live. Better is ⑥ on a19 as shown in Diagram 7 which kills. One move earlier, ❺ is an error too as it could have prevented death by playing on c16 in Diagram 8 leading to a favourable ko for ●.

A slightly better and optimal play of ● is shown in Diagram 9 where ○ needs one more ko-threat to kill. The solution in the last Diagram is equivalent to solution 3 in the book which due to the above mistakes is described there falsely as non-optimal.

4.3 Problems with more than one best Move

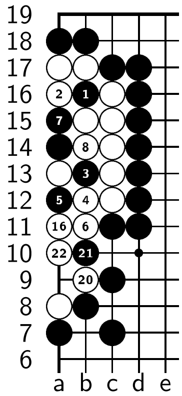
Some problems in the book are somewhat less suitable for a tsume go collection because they have at least two equally good first moves. If there are any mistakes in their solutions in the book then these problems have been discussed in the previous sections. Here we give an example of an error-free problem with non-unique solution. The extra winning move is often a strong threat which has to be answered before the best move from the book is made. Another source of multiple best moves is the situation where the order of ❶ and ❸ can be reversed.

Problem 81



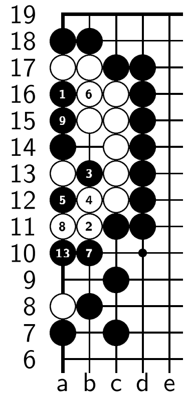
● to move.

Diagram 10.



9 @ 1 10 @ a13 11 @ 3
12 @ b17 13 @ a17 14 @ a13
15 @ 3 17 @ a13 18 @ 2
19 @ a13 23 @ a14

Diagram 11.



10 @ a13 11 @ 5
12 pass

Diagram 12.

In addition to 1 on b16 in Diagram 11, 1 on a16 as in Diagram 12 also reaches the status that ● needs two external ko-threats to win the ko although in Diagram 12 ○ can afford to pass one more time (which the computer program ScanLD does not take into account). The moves 1-10 in Diagram 11 are from the book, the extra moves are from the program to show that ● needs in total two external ko-threats to win.

5 Summary

The emphasis of this paper was not to judge the quality of a specific publication on life-and-death problems but to prove the usefulness of the programs ScanLD and GoTools. Like every text to be published nowadays would always be run through a spell checker first, equally life-and-death problems should be checked by a computer program before being published. Also for readers that are not so strong it is advantageous to have a computer program which provides solution sequences to the very end and which answers any refutation attempt. This is often necessary to realize all the purposes a particular move has, they do not become obvious in a single variation.

The book [4] is the first one we checked. It includes a huge amount of information. The 500 problem diagrams should have not more than one best solution. It also includes 1990 solution diagrams with on average 8 moves, giving 15920 moves, of which about (1990 - 500) are identified in the book as wrong, leaving 14430 moves that could be wrong. If we ignore the problems with multiple best first moves but count multiple errors in one solution sequence then we found 83 wrong moves. As we checked about 65% of the book this gives an error rate of $83/(0.65 \times 14430) = 0.88\%$ for each move which seems very accurate to us.

A Disclaimer

The following is a list of either deliberate restrictions or of non-intentional weaknesses that GoTools, and thus the program ScanLD still have.

- By performing a search with only two possible outcomes (win or loss), GoTools finds a group to be alive if it can not be killed, i.e., it does not make a distinction between unconditional life and seki or a double ko life.
- When comparing the values of different kos and thus the quality of different moves leading to different kos, then two measures are important which characterize the cost for that player, say player A, to win the ko, who would otherwise lose the fight if no external ko-threat were played. One measure is the number of external ko-threats that A needs to have more than B. This number is determined by GoTools. The other measure is the number of passes/tenuki (plays elsewhere) that either side can afford in addition to the fact that A has to play the last move if A wants to win. This measure is not determined by GoTools currently. Ignoring this number would be correct if playing elsewhere had no value, such as at the very end of the game.
- When evaluating a ko position, the number of external ko-threats that is available to the loser is incremented successively until either some limit is reached, or until the loser wins. In the computer runs for this paper this limit is set to 5 because a position in which one side needs to ignore 5 ko-threats may as well be counted as an unconditional loss for that side.
- Open problems have to be enclosed before GoTools can solve them. If the enclosure is too wide then the problem may be too hard to solve. If the enclosure is too narrow then it may affect the status of the different first moves. For example, it happens currently that ScanLD finds additional best moves that kill but a closer human inspection reveals that they are only an effect of a too narrow boundary. Also, if the problem that is checked would have more than one solution but only one is given, and the side moving first is trying to live, then a too narrow boundary may prevent one of the solutions and ScanLD would not find that the problem is unsuitable.
- Sometimes, the variations given in published life-and-death problems do not differ by the first move, but by a later move in the sequence. ScanLD would report any non-optimal second or later moves in a sequence as an error. Obviously this makes it necessary for a human to check whether in the publication this move is mentioned to be non-optimal too. What happens relatively frequently is that the wrong move is indicated to be wrong in the publication too but then more wrong moves appear further down in the sequence which are not recognized to be wrong. Such mistakes in books are less grave but should still not be there and are probably unintended.
- The program GoTools on which ScanLD is based, performs a search with only two possible outcomes: win or loss (apart from ko by repeating a win-loss search with increasing numbers of external ko-threats allocated to one side). A consequence is that there is no measure which could characterize one losing move as being better than another losing move. In order to come up with move sequences in which the loser plays interesting moves as well,

losing moves are chosen which postpone defeat as long as possible. Often this gives the most interesting sequence, but sometimes not.
Another consequence is that search stops for any position within the search tree when a winning move is found, not necessarily the most elegant one. An exception is the original position for which all possible first moves are investigated. This feature is intentional as an $(\alpha - \beta)$ search resulting in a number, not simply a win or loss, would take much longer.
- For large problems GoTools may simply be too slow to solve them. For the current investigations a time limit of one day was applied for evaluating the original problem position or any position that comes up after any sequence of moves in any one of the solutions.

Acknowledgements. The authors want to thank Harry Fearnley for comments on a draft version of the article. This work was made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (SHARCNET: www.sharcnet.ca).

References

1. Dailey, D: CGOS - A 9x9 Computer Go Server: `http://cgos.boardspace.net`
2. Gelly, Sylvain: MoGo home page: `http://www.lri.fr/~gelly/MoGo.htm`
3. Kocsis, L and Szepesvári, C: Bandit-based monte-carlo planning. ECML06, 2006.
4. Li, A and Li, Y: Korean Life-and-Death Problems (in Chinese) Beijing Sport University, ISBN 7811000601, 271 pages (2005).
5. Shen, Lei and Wolf, T: Checking Life-and-Death Problems in Go II: Results. `http://lie.math.brocku.ca/twolf/papers/bugsextra.ps`
6. Shubert, W M and Wolf, T: Tsume Go Of The Day `http://www.qmw.ac.uk/~ugah006/tsumego/index.html`
7. The KGS Go Server: `http://www.gokgs.com` and Computer Go Events: `http://www.computer-go.info/events/index.html`
8. Vesinet, J P and Wolf, T: GoTools online, `http://lie.math.brocku.ca/gotools/index.php?content=go_applet`
9. Wolf, T: Quality improvements in tsume go mass production, in the Proceedings of the Second GO & Computer Science Workshop, Cannes 1993, publ. by INRIA Sophia-Antipolis, 11 pages.
10. Wolf, T: Forward pruning and other heuristic search techniques in tsume go, Special issue of Information Sciences 122 (2000), no 1, pp. 59-76. `http://lie.math.brocku.ca/twolf/papers/jis_nat.ps`
11. Wolf, T and Pratola, M: A library of eyes in Go, II: Monolithic eyes, preprint, 16 pages, accepted for publication in the Proceedings of the International Workshop on Combinatorial Game theory at BIRS (Banff International Research Station), June 2005. `http://lie.math.brocku.ca/twolf/papers/mono.ps`