Two Applications of a Life & Death Problem Solver in Go

Thomas Wolf

Brock University, St. Catharines, Ontario, Canada

twolf@brocku.ca

Abstract

We describe two recent applications of the computer program GoTools that solves life & death problems in the game of Go. In one of them a large database of single eyes fully surrounded by one chain is constructed and the positions are analyzed. Strange positions that appear are filtered out and some are presented in this paper.

A second application in the form of a program ScanLD allows to check thoroughly life & death problems together with given solutions. When applied to a book of 500 Korean problems, a substantial number of bugs was detected, some of which are shown in this paper.

1 Introduction

Go software, or algorithmic theory, that is of potential interest to strong players includes game databases (e.g. [1]), algorithms from combinatorial game theory that decide the very last moves in a game, computer programs specialized in the solution of life & death problems, software that runs online Go servers, and recently the program MoGo (see [2]) that at least on 9×9 boards is getting close to amateur Dan strength. In this paper we describe two applications of the program GoTools (see [3]), and show some results that may be of interest to stronger Go players.

In the following section we summarize two papers ([7] and [8]) that discuss the generation of a database of eyes surrounded by only one chain (called monolithic eyes in this paper) and give some unusual positions that came up in this work.

Afterwards we describe a program ScanLD built on GoTools which automatically checks life & death problems (together with their given solutions) for errors. ScanLD was run on the 500 problems of a Chinese book [5] of a collection of Korean problems – we found a number of errors, some we want to report on. Problems with errors are on average even more interesting than those without because at least some aspect of their solution was obviously missed by the creators of the problems and those who collected and published them.

2 A database of monolithic eyes

2.1 Introduction

It is not the first time that a database of monolithic eyes has been constructed. Compared to the database of David Dyer ([6]), we attempted to create a considerably larger, and more detailed, database by using the program GoTools for evaluating eyes. This way we were able to compute the number of ko-threats needed for the eye to live or be killed if its status is ko. Furthermore, we were able to make the status of the eye depend on the number of external liberties or on the linkage to another external eye, and also to determine all winning moves. By considering eyes in the corner, on the edge, and in the middle of the board, with up to 11 inner points we had to consider in total over 230 million eyes (after dropping all symmetry related ones), and had to evaluate each repeatedly, varying the number of external liberties, and whether or not there was an additional external eye. This computation became feasible by using the growing database for its own generation. To store a database of that size was possible because an increasing number of eyes are unconditionally alive as their size grows. From all the eyes with 11 inner points only 2.5% are not unconditionally alive. More details can be found in [8]. An algorithmic challenge was to develop an algorithm that on the one hand classifies 'bent 4 in the corner' positions as dead, but on the other hand follows the principle that a position is only dead if it can be shown how to kill it (which in the case of bent 4 would give a ko as status). A solution for this dilemma is described in [7].

The database of monolithic eyes is a rich resource of uncommon positions. It is not too difficult to combine seki and ko positions to assemble strange but *large* positions. This database is special in that it contains for enclosed strange positions their smallest and thus most elegant versions. Examples are the minimalistic 'double-ko seki' in diagram 1, the smallest fully enclosed position in diagram 3 that has an NTE (Neutral Threat Environment) temperature, t, such that $-1 < t < 0$, here $t = -1/3$ ([12]), and the position in diagram 5 with ● added on b1 or ○ added on a1 as versions of a '10,000 year ko' (also called 'mannen ko'). Compared with the well known 'mannen ko' that is given in most publications (see [7], section 4.2, diagrams 10-13) the form in diagram 5 needs a smaller eye and fewer stones.

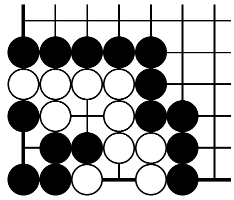


Diagram 1.

With the large number of eyes in the database any selection process of special positions had to be computerized. We wrote programs to find two types of positions. The first we call 'calm positions'. They are defined on one hand to be stable in the sense that no side has an incentive to move first – in other words, passing is among the best moves of both sides, but on the other hand not be trivial, i.e. not to be settled as fully alive/seki or dead.

The second group featured in section 2.3 shows eyes with their status and the best moves depending on the number and type of external liberties.

2.2 Calm positions

Although the 'bent 4 in the corner' positions like that in diagram 2 are special, and passing is one of the best moves for both sides, we do not regard them to be calm but settled (dead) according to common practice (see [7] for more detail).

The smallest calm eyes have 5 inner points as in diagram 3 followed by eyes with 6 inner points as in diagrams 4[1] and 5.

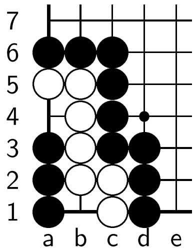


Diagram 2.

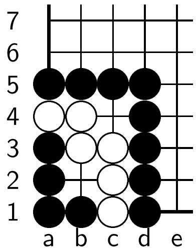


Diagram 3.

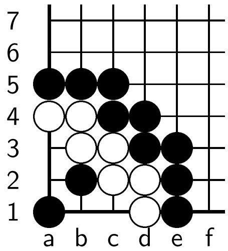


Diagram 4.

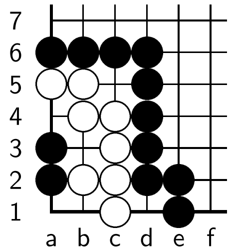


Diagram 5.

The number of calm eyes increases with their size (number of inner points) from 4 eyes of size 5, 6 eyes of size 6, 103 eyes of size 7 to 3298 eyes of size 11. Many calm eyes of bigger size are just enlarged versions of smaller ones but other types appear too as the eyes of size 8 in Diagrams 6 and 7. Eye 7 is calm for 0 - 2 external liberties.

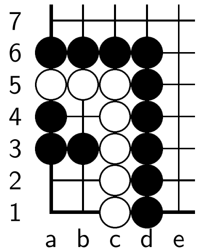


Diagram 6.

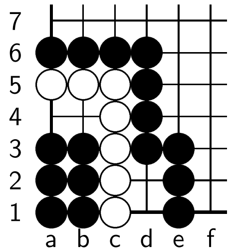


Diagram 7.

With growing size, calm eyes can have an increasingly rich status. Whereas for size 7 and 8 ● needs one external ko-threat to kill, size 9 includes cases where ● needs 2 ko-threats.

[1]According to Bill Spight this position has an NTE temperature of -1 and a mast value of 5 (for Black).

With size 9, because of the larger space inside, ● throw-in stones get stronger such that ○ may need an external eye and one ko-threat to live as in diagram 8.

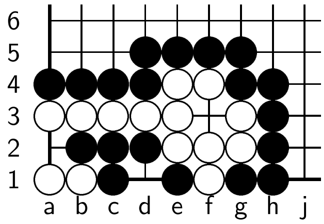


Diagram 8.

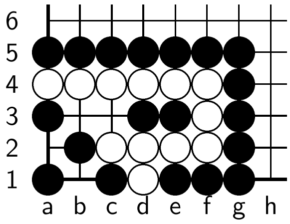


Diagram 9.

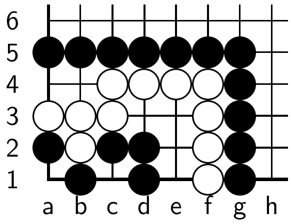


Diagram 10.

Further examples are diagram 9 showing a calm eye of size 10 with ● needing one external ko-threat, and diagram 10 of size 11 with ● needing 3 external ko-threats to kill.

2.3 Eyes sensitive to the number of external liberties

It is not unusual that the status of an eye depends on the number of its external liberties. What is less common is that the best strategy and thus the best first move, do depend on the number of external liberties or on the status of external liberties. For example, if the eye has few external liberties then the throw in stones might win a liberty race by building up strength inside, aiming for a long and slim chain that may even try to get a ● eye inside the ○ eye. In contrast, when ○ has many external liberties, then ● can only win by preventing a 2$^{\mathrm{nd}}$ eye of ○ , thus aiming at a compact clumsy throw-in shape. In the two diagrams 11, 12 the different status of external liberties has the consequence that what is the best ① move in one position (on a2 in diagram 11 and on b1 in diagram 12) is the worst or at least an useless move in the other position.

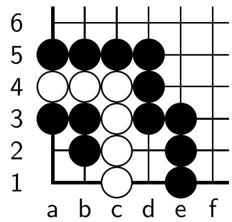


Diagram 11.

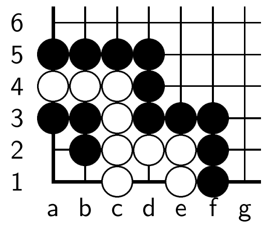


Diagram 12.

The following position in diagram 13 is even more disturbing in that the preference between two moves changes twice depending on the number of external liberties, and may also depend on the global situation, i.e. on the value of playing elsewhere on the board.

For one and two external liberties as in diagram 13 the two ❶ on a1 and d3 in diagrams 14 and 15 are equivalent as in both cases ● needs one external liberty to win and ○ can pass or play elsewhere 3 times.

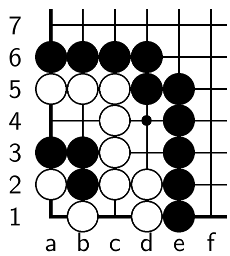

● to move first

Diagram 13.

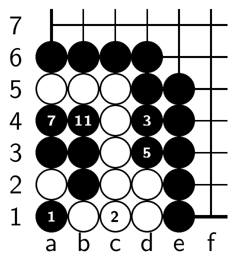

④ pass ⑥ pass ⑧ @ a2
❾ @ ❶ ⑩ pass

Diagram 14.

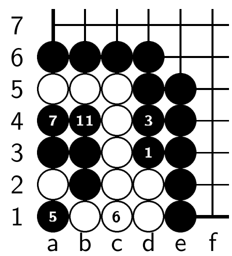
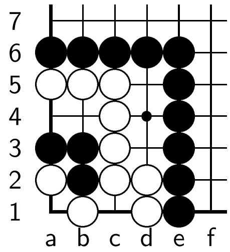

② pass ④ pass ⑧ @ a2
❾ @ ❺ ⑩ pass

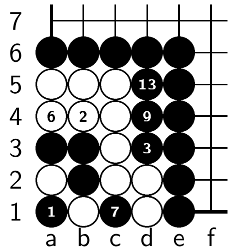
Diagram 15.

The situation changes for 3 external liberties (diagram 16) in that after ❶ on a1 ● needs two external ko-threats and ○ is allowed to pass 2 times whereas after ❶ on d3 ● needs only one external ko-threat, but ○ can pass 4 times. Thus at least towards the end of the game when playing elsewhere is usually less valuable, ❶ on d3 is better.



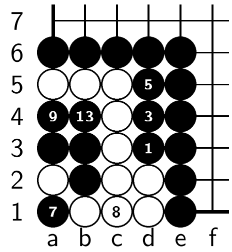
● to move first

Diagram 16.



(4) @ a2 ❺ @ ❶ (8) pass
(10) @ b1 ⓫ @ ❼ (12) pass

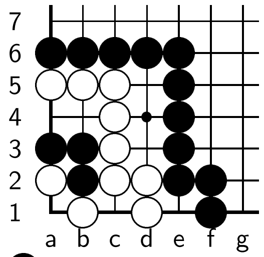
Diagram 17.



(2) pass (4) pass (6) pass
(10) @ a2 ⓫ @ ❼ (12) pass

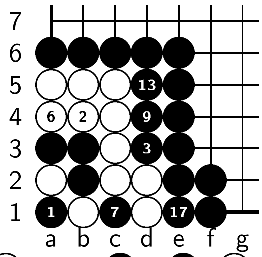
Diagram 18.

With 4 external liberties ❶ on a1 is definitely better by providing a ko (diagram 20) whereas ❶ on d3 leads to the unconditional defeat of ● .



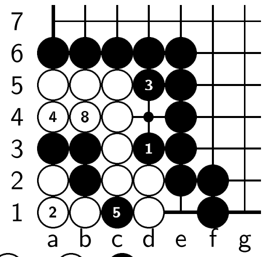
● to move first

Diagram 19.



(4) @ a2 ❺ @ ❶ (8) pass
(10) @ b1 ⓫ @ ❼ (12) pass
(14) @ b1 ⓯ @ ❼ (16) pass

Diagram 20.



(6) @ (2) ❼ @ b1

Diagram 21.

Finally the same position with zero external liberties and a1 as ko- forbidden field (diagram 22 and optimal play in diagram 23) demonstrates that common beliefs, and a lot of computer go programs, including Go servers, are wrong to end a game after two passes.

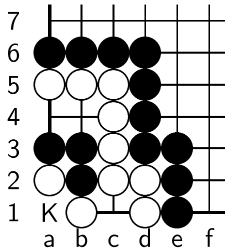
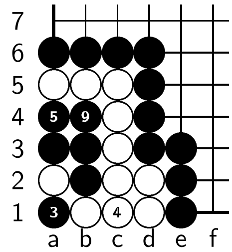


Diagram 22.



❶ pass (2) pass (6) @ a2
❼ @ ❸ (8) pass

Diagram 23.

3 Checking Life & Death Problems

3.1 The Program ScanLD

In the second half of this paper the program ScanLD is described together with its application to a book of 500 problems. Being based on GoTools the program ScanLD checks life & death problems including their solutions thoroughly. It answers the following questions:

- Does the side moving first have any success?
- Is there more than one first move that reaches the same best result?
- What is the status of all possible first moves?
- Is the first move of the supposedly best solution among the truly best first moves?
- Are all second and later moves in all sequences optimal?

- If one move is not optimal, which sequence of follow-up moves would expose this non-optimality, i.e. how can the non-optimal move be refuted?
- If one move is not optimal, which other move with follow-up sequence would be optimal, and which status do the non-optimal and the optimal moves achieve?

A detailed description of ScanLD is given in [9], – here, because of space limitations, we restrict ourselves to an loose collection of comments.

ScanLD assumes that the first solution is supposed to be correct whereas the first moves of further solutions may be incorrect but in the publication are known to be incorrect and thus are not criticised by ScanLD whereas any later non-optimal moves are criticised. A manual inspection is necessary to check whether later incorrect moves are known to be wrong or not.

Being based on GoTools the program ScanLD inherits features that are a consequence of searches only knowing about win or loss, with seki being treated as life. A detailed list of limitations is given in the appendix of [9].

To be solved effectively the position has to be fully enclosed which is done automatically but not at a very sophisticated level. If the closure is too narrow then solutions are affected, if the closure is too far then computation times are unnecessarily large. This is a problem to be worked on but it is currently not crucial because boundaries can be inspected and changed easily manually.

3.2 A first application

In a first systematic test the program ScanLD was used to check all 500 problems including their solutions of the Chinese book [5], in the following simply called 'the book'. We got groups of problems

a) that could not be closed easily (1.8%),
b) that were too hard to be solved by the program in this first attempt (33%),
c) with full agreement between the book and the program (49%),
d) with a wrong move in one solution (6%) except those
e) with a wrong move in one solution that changes the number of ko-threats or the side needing ko-threats (8.4%),
f) without bugs but with multiple first best moves (1.4%).

A complete list of problems d) and e) is given in [10]. 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 plays optimally. Later solutions typically discuss non-optimal play.

3.3 Mistakes missing a total win

The following examples are from group d) above.

Problem 296, solution 3

In Diagram 25, ❸ on a16 is wrong in that ○ can still get a ko. By playing ❸ on a13 as in Diagram 26, ● kills. A variation with a different ② in which ● kills too is shown in Diagram 27 answering to a different ②. A Diagram like the last 2 should replace solution 1 in the book because of another mistake regarding the same problem (see the comments following Diagram 36 in the next subsection).

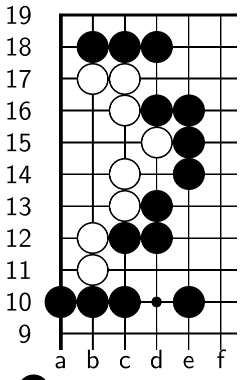

● to move.

Diagram 24.

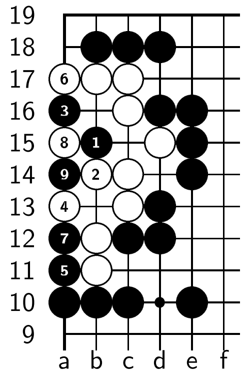

Find error after ❶.

Diagram 25.

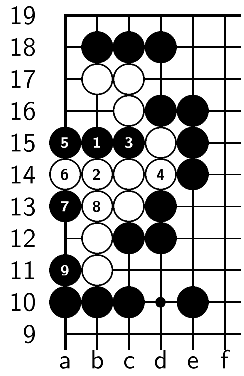


Diagram 26.

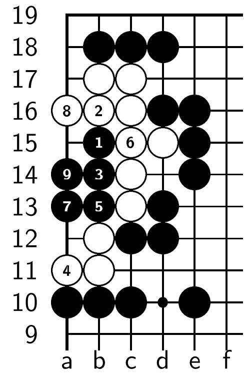


Diagram 27.

Problem 412, solution 3

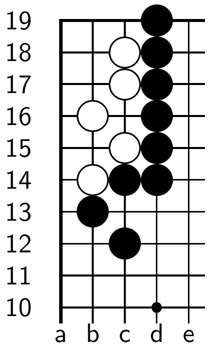

● to move.

Diagram 28.

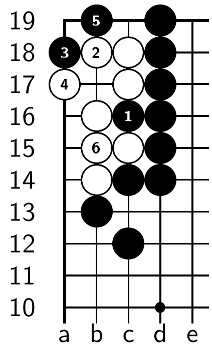

Find error after ❶.

Diagram 29.

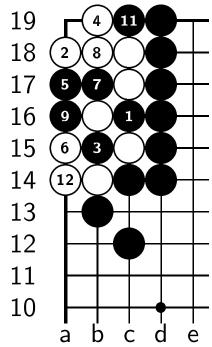

⑩ @ b16

Diagram 30.

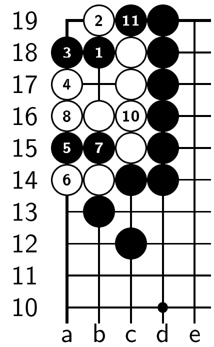

❾ @ ❼

Diagram 31.

In Diagram 29, ○ needs to win a ko to live. But ○ can live by playing ② on a18 in Diagram 30. (Diagram 31 from the book shows optimal play of ● that kills.)

Problem 429, solution 4

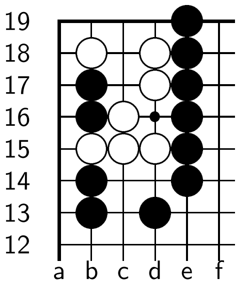

● to move.

Diagram 32.

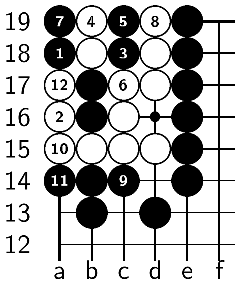

Find error after ❸.

Diagram 33.

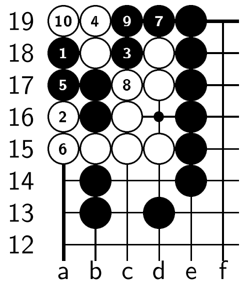

⓫ @ b17

Diagram 34.

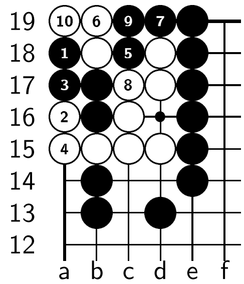

⓫ @ b17

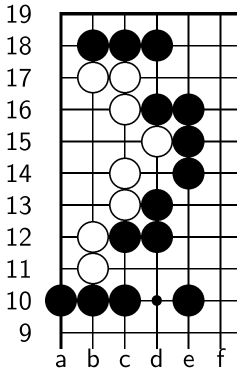
Diagram 35.

❺ on c19 in Diagram 33 is wrong in that ○ lives. By playing ❺ on a17 as in Diagram 34, ● kills. Thus, contrary to the statement in the book, ❸ on c18 in Diagram 33 is correct. (A variation of Diagram 34 is solution 1 from the book in Diagram 35 in which ● kills too.)

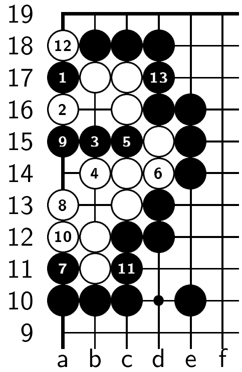
3.4 Mistakes missing a ko to avoid total loss

It is relatively easy for a human player to overlook an opportunity to get a ko. The following is such a case.

Problem 296, solution 1



● to move.
Diagram 36.



Find any error.
Diagram 37.

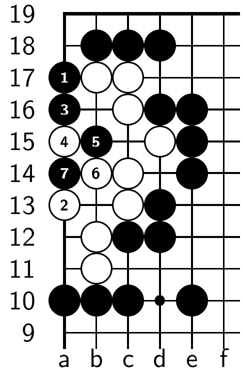


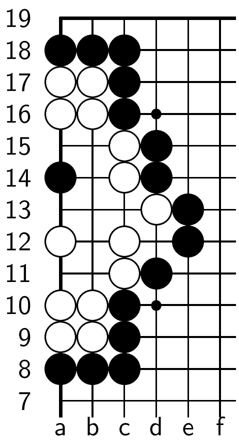
Diagram 38.

In Diagram 37, ② on a16 is wrong in that ○ dies. By playing ② on a13 as in Diagram 38, ○ could live by winning a ko. For optimal play of ● see Diagrams 26 and 27 in the previous section.

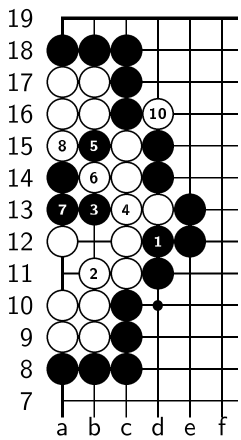
3.5 Problems with more than one best move

Although problems with multiple best fist moves are somewhat less suitable for a tsume go collection, those problems where an additional best first move is missed are very interesting because the missed move is usually non-obvious.

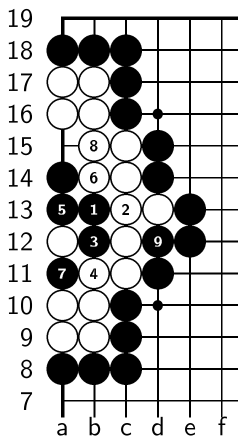
Problem 84



● to move.
Diagram 39.



❾ @ ❺ ⓫ @ a16
Diagram 40.



⑩ @ a12
Diagram 41.

In addition to ❶ on d12 in Diagram 40, ❶ on b13 as in Diagram 41 also reaches the status that ● needs one external threat to win the ko.

3.6 ScanLD - summary

The purpose of checking the content of the problem collection was not to judge the book but to demonstrate the usefulness of the software. Given the large information content of the book with 500 problem diagrams and 1990 solution diagrams including about 16000 moves the number of mistakes found is very low. In checking the error messages of ScanLD three bugs in GoTools had been identified and fixed.

To summarize, checking the book showed that before publishing life & death problems they should be run through a computer program as ScanLD, even if the authors are professional players (in this case 3P). For checking the error findings of ScanLD the program GoTools proved to be invaluable, allowing to explore any variation interactively and even to localize the 3 bugs within itself for a later fix.

References

[1] van der Steen, J: GoBase home page:
`http://gobase.org/`

[2] Gelly, S: MoGo home page:
`http://www.lri.fr/~gelly/MoGo.htm`

[3] Wolf, T: GoTools home page:
`http://lie.math.brocku.ca/gotools/`

[4] Shubert, W M and Wolf, T: Tsume Go Of The Day
`http://www.qmw.ac.uk/~ugah006/tsumego/index.html`

[5] Li, A and Li, Y: Korean Life and Death Problems (in Chinese) Beijing Sport University, ISBN 7811000601, 271 pages (2005).

[6] Dyer, D: An Eye Shape library for Computer Go,
`http://www.andromeda.com/people/ddyer/go/shape-library.html`

[7] Wolf, T: A library of eyes in Go I: A life & death definition consistent with 'bent-4', preprint, 12 pages, accepted for publication in the proceedings of the International Workshop on Combinatorial Game theory at BIRS, June 2005.
`http://lie.math.brocku.ca/twolf/papers/bent4.ps`

[8] 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, June 2005.
`http://lie.math.brocku.ca/twolf/papers/mono.ps`

[9] Wolf, T and Shen, L: Checking Life & Death Problems in Go I: The Program ScanLD.
`http://lie.math.brocku.ca/twolf/papers/bugsintro.ps`

[10] Shen, L and Wolf, T: Checking Life & Death Problems in Go II: Results.
`http://lie.math.brocku.ca/twolf/papers/bugsextra.ps`

[11] Vesinet, J P and Wolf, T: GoTools online,
`http://lie.math.brocku.ca/gotools/index.php?content=go_applet`

[12] Spight, B: private communication.