A library of eyes in Go
II: Monolithic eyes

Thomas Wolf
Department of Mathematics, Brock University
500 Glenridge Avenue, St.Catharines,
Ontario, Canada L2S 3A1
email: twolf@brocku.ca

Matthew Pratola
Department of Statistics and Actuarial Science,
Simon Fraser University
8888 University Drive, Burnaby,
British Columbia, Canada V5A 1S6
email: mpratola@gmail.com

February 14, 2009

Abstract

We describe the generation of a library of eyes surrounded by only one chain which we call monolithic eyes. Apart from applying the library in the life & death program GoTools it also can be used as a source for the study of unusual positions in Go as done in the second half of the paper.

1 Introduction

In using principles of combinatorial game theory it has been discussed in the literature how in the game of Go one can assign values to eyes in order to decide whether a position lives unconditionally, simply by adding these values and checking whether their sum reaches the value of two or not (see the paper of Howard Landman [1]). These concepts are applied in computer Go programs (e.g. in [3]) and a computer generated library of eye shapes is available from Dave Dyer (see [2]).

In this contribution we describe the generation of a library of eyes surrounded by only one chain which we call monolithic eyes. Compared to the library of Dave Dyer ([2]) our database has a number of extensions: an evaluation of the number of ko threats needed to live or to kill the eye, the consideration of a larger number of external liberties and of an extra attached eye, larger eye sizes, the determination of all winning moves and others.

In the following section we describe a procedure to bring any set of empty or occupied points into a unique position by using shifts and symmetries of the board. The purpose is to avoid the generation of equivalent eyes as described in section 3. The evaluation of eyes is done with the program GoTools (see [4]) as outlined in section 4. Computational aspects including a listing of optimizations, comments about performed consistency tests, usefulness and availability follow in section 5.

The database of monolithic eyes is a rich source of strange positions. In section 6 we give two examples of how to inspect it: 1) by looking for boolean-calm eyes, i.e. eyes which are not settled but where passing belongs to the boolean-best moves[1] of both sides, and 2) by checking for eyes where the boolean-best attacking move depends in a non-monotonic way on the number of external liberties.

A summary concludes the paper.

2 Unique representations

If positions in the game of Go are to be stored then there is the option either to store all 8 versions obtained from rotation and reflection of a single position, or to move the position into a unique location and to store it only once. The first version is probably faster, whereas the second is definitely memory saving. Our choice follows partially from the intended use of the database of monolithic eyes which is to be applied in the program GoTools. Much of the execution time of this program is spent on tasks other than evaluating eyes. The speed of the eye database is therefore not crucial which is a reason to use the memory-saving approach. Another reason is that eyes which are not located in the corner have to be shifted to a norm position anyway, and performing a rotation and reflection in the same step does not take much more time. We therefore took the second option and bring each eye into a norm position through shift, rotation and reflection before storing or looking up the position in a database.

We have three cases: eyes in the corner, on the edge and in the centre of the board. For eyes in the corner we have no shift, the rotation is apparent as we rotate each eye into the lower left corner and only the reflection has to be decided by comparing successively two points, one on either side of the board diagonal. Eyes on the edge are rotated to the lower edge of the board. A horizontal centre of the eye is determined as the average of the extreme x-values of inner points of the eye. The middle of the eye can have an integer or a half integer value. For both cases we have a procedure to decide whether a reflection is needed. Finally, for mid-board problems we have the three cases that the centre of the problem falls onto a point of the board, between two points on a line or on the centre between 4 points. For each of these cases we have a procedure to start with points close to the centre of the position and to move gradually away in comparing points. Through the computation of dipole, quadrupole and octupole moments (in analogy to the distribution of electrostatic charges in physics) we continue with the computation until the symmetry is completely broken, i.e. until a rotation around the centre of the position and a reflection are completely determined or until all points of the positions (in the application all inner points of the eye) have been considered and, as a by-product, the exact remaining symmetries of the position are determined. The 7 possible remaining symmetries can be visualized with the examples in figure 1 where `+` marks the centre of rotation, `*` a single stone, `ref` stands for a reflection symmetry on the x, y diagonal, `rot1` for a 90 degree rotation counterclockwise around `+` and `rot2` for a 180 degree rotation counterclockwise around `+`.

The procedure to *norm-locate* a position (i.e. to put it into a unique place through shift, rotation and reflection) is used repeatedly when creating all essentially different positions

[1] In [4] we define 'boolean-best' moves as all the moves reaching the best possible result in a boolean search with only two outcomes: life/seki or death. In other words, moves achieving seki are regarded as good as moves achieving life.

```
       *            *                           *
       + *      * + *           + *               +
                *                                   *

      ref        rot2         rot1+ref      ref, rot2


                         *
                            *
     * + *                +                    +
                        *
                           *

 rot1+ref, rot2      rot1, rot2      ref, rot1, rot2
```

Figure 1: Examples for the possible symmetry types of positions

(to have them evaluated later). It is also used when applying the database during a life and death computation to norm-locate any encountered monolithic eye in order to look it up in the database. The norm-locating procedure does not only output the new position but also the transformation leading to it. The purpose is to use the inverse transformation on the moves stored in the database to obtain the moves that should be done in the encountered position.

3 The generation of positions

To identify a position in the database we generate a hash code according to Zobrist (see [7]) which is the standard technique in computer Go. The hash codes have to be large enough to avoid clashes for over $2 \cdot 10^8$ eyes of size up to 11 but should be as short as possible to save memory. It turns out that a 32 bit code is too short but we confirmed that a randomly chosen 64 bit code was sufficiently large.

In this article the monolithic eye is made from white stones and Black is the attacker. Monolithic eyes are represented completely through their interior which includes empty, black and white points, not the eye enclosing stones, like ○ on a3 in diagram 1 below.

The computation starts with three eyes of size 1: one in the lower left corner, one in the middle of the lower edge and one in the centre of the board. Eyes of increasing size are successively generated. The generation of all eyes of size s proceeds in four steps.

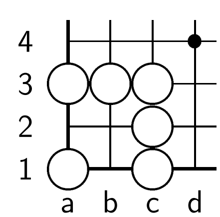


Diagram 1.

At first all internally empty eyes, like the eye of size 11 in diagram 2, are generated by extending all internally empty eyes of size $s-1$ in all possible directions. Duplicate eyes are avoided by storing a hash code for each empty eye in a database. At this stage all eyes with internal isolated stones are excluded. For example, an eye like in diagram 1 is excluded when generating eyes of size three but it would come up when generating all eyes of size four.

In the second step all possible legal combinations of white stones within each eye are generated as, for example, the eye in diagram 3, with the restriction that extra white stones may not be attached to the white eye enclosure as the inner size would be reduced. Hence, white stones could not be put on points like c2 in diagram 2.

On the other hand, white stones on a1 and b2 (for example) are allowed, even if they split the monolithic eye into two or more non-monolithic eyes. For each resulting position all possible legal combinations of additional internal black stones are generated, such as the eye in diagram 4.

Finally, for all eyes which include one or more ko-fights the position is duplicated once for each empty potential ko point which in the duplicate position is marked as forbidden. The position in diagram 4 would therefore appear twice, once without the forbidden ko point (diagram 4) and once with the forbidden ko point, like the point marked as K in diagram 5. If the point a2 in diagram 4 would be occupied by Black or White then b1 could not be a ko-forbidden point as the position could not have resulted from White making a move on c1 and capturing a single black stone on b1.

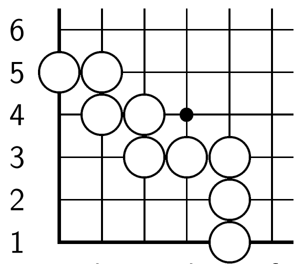


Diagram 2.

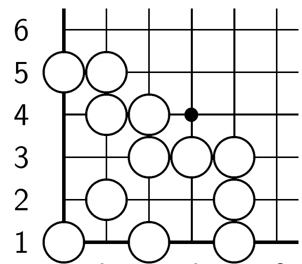


Diagram 3.

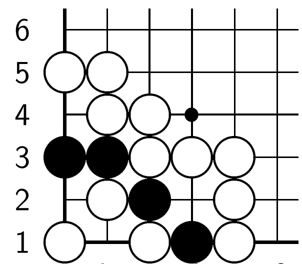


Diagram 4.

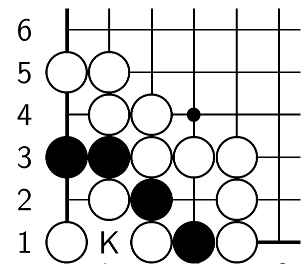


Diagram 5.

With nearly $2 \cdot 10^8$ eyes of size 11 a hash database holding all of them to exclude duplicates would be too large. Fortunately, it only needs to be big enough to hold all empty eyes of one size to exclude duplicate empty eyes of same shape. If two empty eyes are not symmetric to each other (by reflection, rotation or shift) then they cannot become symmetric by putting extra stones in. But if an empty eye has a symmetry, for example a reflection symmetry, then filling it with stones will produce positions which are pairwise equivalent with respect to this symmetry. The hash database therefore has also to be big enough to hold all eyes which are generated from a single empty eye through input stones.

Three diagrams in figure 2 illustrate the growth of the numbers of positions in dependence on the size of eyes for corner, edge and centre eyes.

Comments

- Because of $\log(0) = -\infty$ we replaced 0 on the vertical axis each time by 1 to have $\log(1) = 0$.
- Growth factors are remarkably constant and allow to estimate the number of eyes of larger sizes.
- The number of positions grows only little by adding white throw-in stones. It grows most for corner positions as for edge and centre eyes most inner points are on the edge of the eye which cannot be occupied by further white stones without decreasing the inner size.
- The number of extra positions due to an initial ko grows faster than other numbers and is highest for corner eyes, again because for corner eyes the surface of the eye (the number of inner points neighbour to the enclosing chain) is smallest.

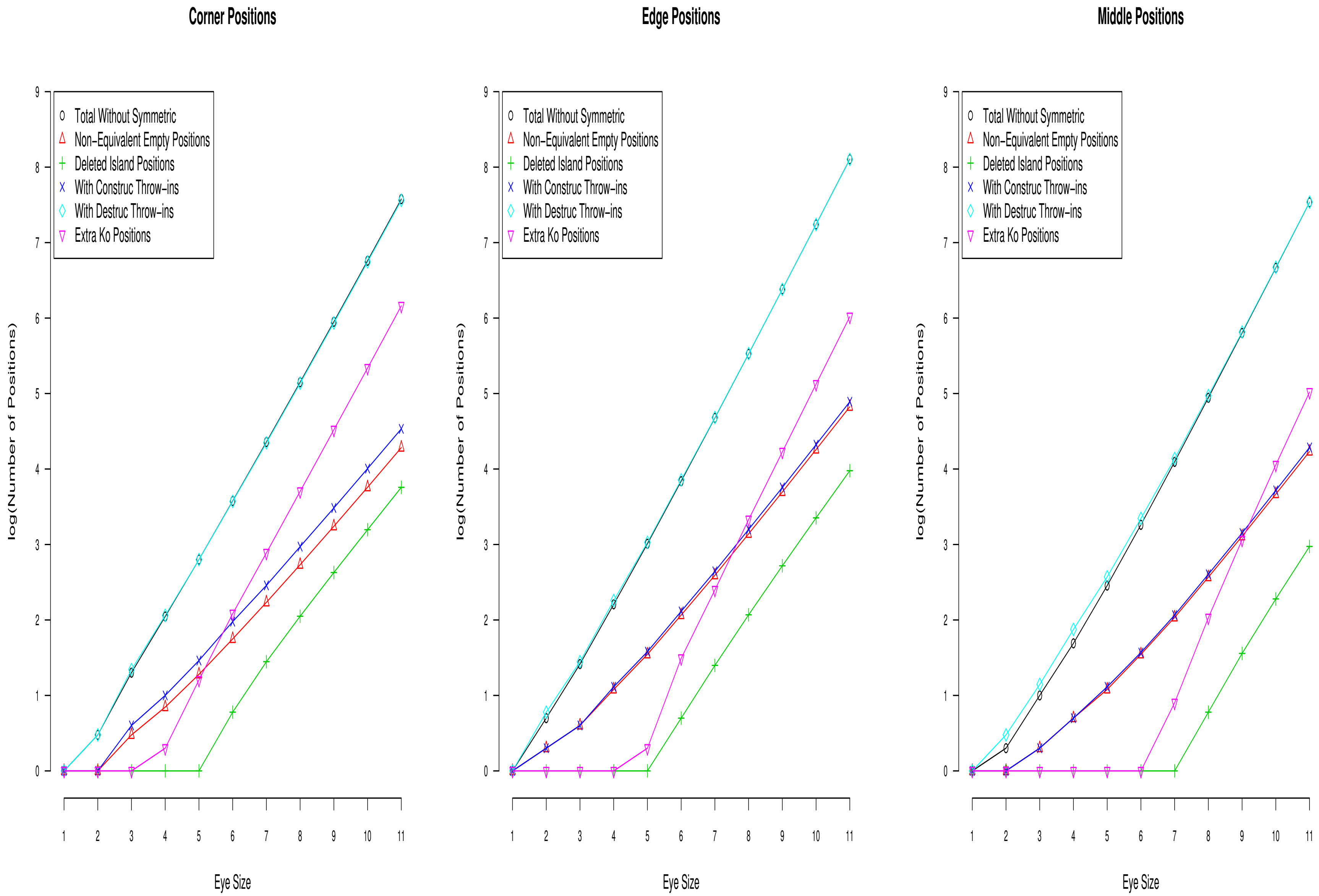


Figure 2: Numbers of generated positions

4 The evaluation of positions

To evaluate the eyes we use the program GoTools. It performs a boolean search which has, for example, the consequence that seki is treated as life. Other consequences and limitations are described in the appendix of [6] in this volume. A strength of GoTools is that it is able to determine how many ko threats are needed for the weaker side to win. When evaluating eyes, beginning with size 1 and increasing the size successively, GoTools already makes use of the hash database filled with data for smaller eyes.

Because GoTools is a life & death program it cannot evaluate whether the position is at least 'worth one eye' so that it could live if it had another 'eye'.[2] We compensate this weakness by investigating the position twice, first on its own in order to find out whether it is 'worth two eyes' and if not then we attach externally a secure 1-point-eye and investigate it again to find out whether it is 'worth at least one eye'. The information whether a position is 'worth $\frac{1}{2}$ eye' (i.e. 'worth one eye' if White moves first and 'worth no eye' if Black moves first) or similarly 'worth 1 $\frac{1}{2}$ eyes' follows from evaluating it for both sides moving first.[3]

The status of the eye and/or the boolean-best moves often depend on the number of external liberties. Therefore computations are done for 0 to 31 external liberties for two cases: 1) the eye on its own and 2) one extra external 1-point eye being attached. Only in the case that the eye has no internal liberty, at least one external eye or liberty has to be present.

Table 1 gives an overview of the number of positions, the percentage of those which are not unconditionally alive, the average number of records to be stored for each not unconditionally alive position and times needed to evaluate the most expensive single eye and all eyes of each size. All times are measured on a Pentium IV, 3GHz. Programs are written in FreePascal (`http://www.freepascal.org`).

5 Computational aspects

5.1 Optimizations

The following are observations and ideas that allow to make the computer programs more efficient.

- Because the database is generated only once, it can afterwards be converted to a read-only database saving one pointer for each record and thus memory.
- From table 1 it follows that a large proportion of all eyes are unconditionally alive. We do not need to store them. If we do not find an eye with a size that is covered by the database then we simply conclude that the eye is unconditionally alive. This works because we generated and evaluated all eyes up to the maximal considered size (currently 11).

[2] In this article the word *eye* normally denotes a whole position consisting of a white chain with stones inside and a surrounding black enclosure. Only in this paragraph the phrase 'worth one eye' is a short form of saying 'one protected liberty which has to be the last liberty taken for capture' with the understanding that two such liberties ensure life.

[3] with the exception that if an initial ko forbidden point is present then only the side for which this point is forbidden may move first

s	a	b	c	d	e	f	g
1	3	3	100.0	1.000	76	< 0.1	< 0.1s
2	10	10	100.0	1.000	220	< 0.1	< 0.1s
3	56	55	98.2	1.036	868	< 0.1	0.4s
4	321	259	80,7	1.062	4,180	< 0.1	2.5s
5	1,938	809	41.7	1.116	15,004	< 0.1	7.7s
6	12,477	2,917	23.4	1.243	58,540	0.1	27.4s
7	82,808	9,955	12.0	1.464	233,572	2.5	2m 31.9s
8	565,104	41,831	7.4	1.608	1,040,548	2.5	13m 51.4s
9	3,931,849	196,402	5.0	1.659	6,601,924	5.2	1h 32m 26s
10	27,800,486	965,405	3.5	1.716	33,118,212	18.4	11h 6m 1.2s
11	199,169,127	4,990,259	2.5	1.739	215,042,444	45.1	3d 11h 19m

s : size of eye
a : all positions of this size
b : positions not unconditionally alive
c : percentage of positions not unconditionally alive
d : average number of records for each not unconditionally alive eye
e : size of final read-only hash database in bytes
f : maximal time in sec for the evaluation of one eye
g : total time for all positions of this size

Table 1: Numbers of positions, sizes of final databases, times for evaluation.

- If we store only a small fraction of all eyes, we do not have to guarantee that there are no clashes between hash codes of eyes which are not stored. To ensure that there is no clash between any two stored eyes and not between a stored eye and an eye that is not stored, shorter hash codes are sufficient in the final read-only version.

- Column d in table 1 shows the average number of records that each (not unconditionally alive) eye contributes to the database. The larger the eye, the more likely the status and/or the boolean-best moves depend on the number of external liberties and the more records are needed for the eye. The pointers between these records have a 32 bit size but if the database has to hold eyes only of size ≤ 10 then the number of stored records is much smaller than 2^{32}, so we can use some of the bits to encode part of the hash code. Column e in table 1 shows the memory requirement for a hash database including only eyes up to this size.

- In the process of evaluating an eye of inner size n the eye either keeps its size, or its inner size shrinks. Any smaller eye that appears has already been investigated before because we evaluate eyes of successively increasing size. Therefore, either it is found in the database, then the information can already be used, or it is not found and then it is clear that it must be unconditionally alive.

  The situation is different if throw-in stones are placed which change the eye (i.e. its content and thus the eye) but in a way that its inner size is still n. Because not all eyes of size n have already been investigated, one cannot conclude that it is unconditionally alive if it is not found in the database. Nevertheless, one can look it up and use the information if the eye *is* found.

- When studying an empty eye of size n then other eyes of size n with more throw-in stones appear, but when studying an eye of size n with many throw-in stones then it is rare that eyes of same size with fewer throw-in stones turn up in this evaluation. To increase the chance that appearing eyes of that size have already been investigated, one can evaluate eyes with many interior stones first and empty ones later.

- In the worst case scenario each of the $2 \cdot 10^8$ eyes would be evaluated 128 times (= $2 \cdot 2 \cdot 32$ for both sides moving first, with/without extra eye and each time 0-31 external liberties) but this can always be sped up. If, for example, Black moves first and the eye has no external liberty and no extra eye is attached and Black loses then Black will always lose and White will always win.

- In each of the at most 128 computations of an eye, *all* possible first moves are checked to determine all moves giving the best status. After the first computation determining the status of the problem and one of the (boolean-)best moves is completed, any computation checking another first move can be stopped early (i.e. one can avoid ko re-runs with more ko threats as described in the section on the ko status in [6]) as soon as it is clear that the status will be suboptimal.

5.2 Safety

The program GoTools which evaluated the positions operates apart from the permanent monolithic eye database also a temporary hash database which is cleared before a new eye is considered. For any new entry to be stored, both databases check whether this entry is consistent with the content so far. For example, if for a position it is already known that White moving next will lose despite having one external ko threat, then a new entry saying that White even loses with two ko threats is consistent, but a new entry saying that White wins in this position without ko threat would create an inconsistency. For the eye database, further consistency checks result from the requirement that Black must not do worse when White has fewer external liberties and/or no extra external eyes compared to White having an external eye. Equivalently, White must not do worse when having more external liberties or extra eyes. These consistency checks, having been done thousands of billions of times during the generation of the database, led to the fixing of a number of bugs (mainly linked to the proper handling of ko in connection with the temporary database) and to the development of 'bent-4 compatibility' reported in [6].

5.3 Efficiency and availability

The monolithic eye database is a single permanent read-only file with a size depending on the size of eyes stored (see column e in table 1). To use the database, a module within the program GoTools norm-transforms any monolithic eye encountered during a computation by performing a rotation, reflection and shift as necessary, generates a hash code, finds the relevant entry in the database (depending on the number of external liberties and on who moves next) and decodes the information (status and moves). The size of the database grows exponentially with the size of eyes to be stored and the frequency to encounter monolithic eyes falls quickly with increasing eye size. Being able to read the status and first moves for a big eye from a database is especially timesaving but on the other hand, loading a large database takes considerable time when starting the program and occupying much memory with a large database makes cache memory less effective and slows down the whole program.

Therefore in the online version of GoTools on [5] only a database including eyes of size up to 8 is used.

6 Unusual positions in the database

The database of monolithic eyes is a rich resource for simple but also special positions. Because of the large number of eyes a human visual inspection is out of the question. To filter out interesting positions automatically, one needs to come up with criteria that make a position interesting. These criteria depend on how positions are evaluated and which data about the positions are available. The positions in our database are computed with the program GoTools. As described in more detail in [6], the possible status are unconditional life, death or characterized by the number of external ko threats needed for the weaker side to win. The status and all first moves that result in this status are determined for both sides moving first, for 0-31 external liberties and for the two cases that an additional 1-point eye is attached or not. Given these data for all monolithic eyes in the corner, on the edge and in the centre of the board, what are criteria that make life and death positions interesting?

The filters applied in this section are not unique, they are examples and one may come up with other criteria people are interested in. Also, limitations like seki=life have their consequences on what one can find. For example, *mannen-ko* as shown in [6], section 4.2, diagrams 10-13, is a seki type position but both sides can start an unfavourable ko about unconditional life or death. Because the ko is unfavourable to whoever starts it, the seki is in some sense stable. In this paper we treat seki=life with the consequence that these positions are simply unfavourable ko's for Black, so passing is the boolean-best move for White.

6.1 Calm positions

A first class of positions we want to explore has the property that passing is one of the best moves for both sides. To be more precise, passing is one of the boolean-best moves, i.e. one of the moves that achieve the best possible result in a boolean search where seki is treated as life. If such a position is not settled, i.e. if it is not unconditionally dead or alive/seki then we called it *boolean-calm* in [6] and gave an example there. Treating seki=life means that the 'double-ko seki' in diagram 6 is regarded as settled and thus not listed as a boolean-calm position, although both sides have passing as one of their boolean-best moves.

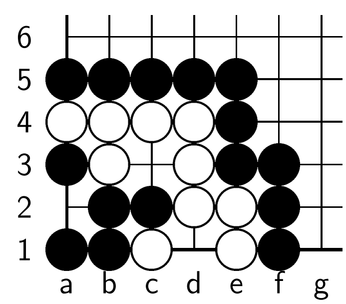


Diagram 6.

A convention: *To make the text better readable and appear less clumsy we will drop 'boolean-' from now on and write 'calm' instead of 'boolean-calm' and 'best' instead of 'boolean-best'.* We have to keep this in mind as otherwise statements in this section become wrong. Passing is often not one of the truly best moves, for example, in diagram 7 where ① pass, ❷ on h1 is boolean-best for ○ , achieving a seki whereas ① on a1 achieves life.

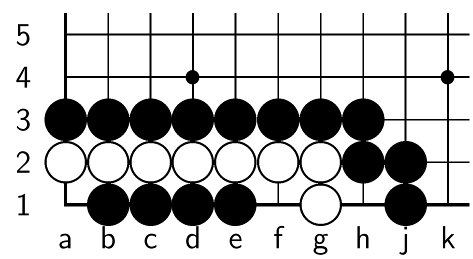


Diagram 7.

If passing is one of the best moves of both sides, then the status does not depend on who moves first, i.e. such positions have only one status. Ko-banned positions (i.e. positions with

size	corner	edge	center	total
≤ 4	0	0	0	0
5	4	0	0	4
6	6	0	0	6
7	69	34	0	103
8	232	109	0	341
9	405	197	0	602
10	746	98	2	846
11	2817	470	11	3298

Table 2: Numbers of calm eyes in the corner, on the edge and in the centre

a ko-forbidden point) can have only one side moving next and hence cannot be calm.

Table 2 lists the number of calm position for different eye sizes. For each size only a very small percentage is calm, each position only for one or very few specific numbers of external liberties. In this sense these positions are special, although not as exotic as bent-4. The numbers in table 2 serve merely as an illustration, as many positions are equivalent, not by rotation or reflection of the whole position but by deforming the eye, deforming the black throw-in chain(s) and re-arranging internal liberties without changing anything substantially.

To construct a strange position with a large area that involves many chains, nesting eyes and multiple ko's does not seem to be too difficult. It is more of a challenge to embody the spirit of a strange position in as few stones as possible. By generating eyes beginning with minimal size, such minimal positions will be found. An example is discussed in the subsection about size 6 eyes below.

6.1.1 Bent-4-type positions

One way for a position to be strange is when its status depends on the global situation on the board. The simplest are straight ko's where the side with more threats wins. The next level of strangeness is reached when the status does not depend on the number of threats on the board, but on the type of threats, whether they are removable (like cuts) or unremovable (seki). This is the case when the local position is a ko, and somewhere in any optimal sequence of moves, the side for which the ko is unfavourable has the passing move as the only best/feasible move. Such positions have been called bent-4-type positions in [6], section 3. From all the investigated eyes up to size 11, the 4 bent-4 positions in [6], section 3, were the only ones with this property. They are commonly regarded as dead, because Black (the side for whom the ko is favourable) can wait arbitrarily long before making moves at no cost that eliminate removable ko-threats of White and then starting the favourable ko. Therefore, the program GoTools had been modified as described in [6] so that bent-4-type positions are evaluated as an unconditional loss for the side with the passing move as single best move. It is with this version of GoTools that the monolithic eye database has been computed, therefore bent-4 is evaluated as unconditionally dead and will not be found below in our search for calm positions.

6.1.2 Eyes of size 5

Starting with eyes of small size we find that eyes need at least 5 inner points to have a status that is not unconditionally dead or alive and to allow both sides to pass without penalty. Up to rotation and reflection there are 4 such positions. One of them is shown in diagram 8. The other three have the same black throw-in stones but the liberty at b2 is located at c1,b3 or a4. In all four positions the eye has exactly one external liberty and the status is that White needs one external ko-threat to live.[4] If a position/eye can only be calm if it has one or more external liberties then in 'real Go' (with seki $\neq$ life) they cannot be a seki because it does not harm Black to occupy these liberties. Therefore these four positions are no seki's.

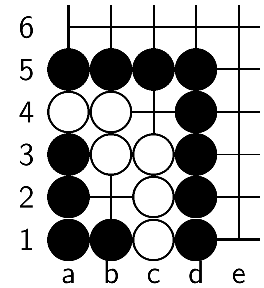


Diagram 8.

The interested reader can find solution sequences for all positions in this paper by solving the problems online on [5].

6.1.3 Eyes of size 6

There are two types of calm positions of size 6. One is the position in diagram 9 where White needs one ko-threat to live.

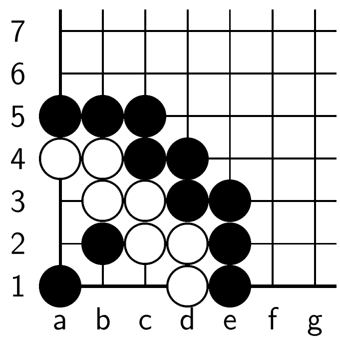


Diagram 9.

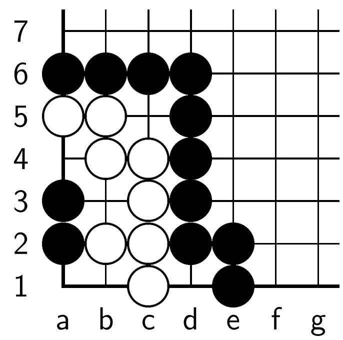


Diagram 10.

In the other 5 positions Black needs one ko-threat to kill. These are:

1. diagram 10 with 0, 1 or 2 external liberties for the eye, and
2. the same diagram but with ● added on b1 and 0, 1 or 2 external liberties, or
3. ○ added on a1 and 0 external liberties, or
4. ● added on a4 or b3 and 0 external liberties.

In cases 3 and 4 the only good move for White is to pass which does not make the position to be of bent-4-type because the ko is favourable for White. To be of bent-4-type the ko would have to be unfavourable for the side with passing as the single best move.

From these 5 positions we chose the one in diagram 10 as the representative position - the 'mother' position - because all other ones can be reached from it. Positions in diagram 10, cases 2 and 3 are versions of a '10,000 year ko' or, also called 'mannen ko'. Compared to the usually known and published form (see [6], section 4.2, diagrams 10-13) the form in diagram 10 needs a smaller eye and fewer stones, supporting the claim that from all strange positions the database provides the more interesting minimal versions.

[4]If there would be no external liberty of White then Black moving first can catch and if there are two external liberties, White can live moving first, so in both cases not a calm position. The difference to bent-4 is that White can start the Ko at b2 and does not have to pass.

6.1.4 Eyes of size 7

For all calm eyes of size 7 Black needs exactly one external ko-threat to kill.

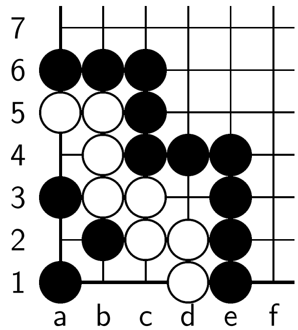


Diagram 11.

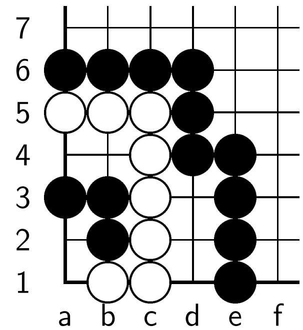


Diagram 12.

Most of the positions are related to the eyes in diagrams 11 and 12. The position in diagram 12 is a 'mother position' for a number of calm positions, including mannen ko and is itself calm for 1-3 external liberties.

6.1.5 Eyes of size 8

Like for size 7, in all calm eyes of size 8 Black needs exactly one external ko-threat to kill. Due to the increasing number of calm eyes we can only give examples. Many eyes are enlarged variations of mannen ko. Here are two other examples. The position in diagram 14 is calm for 0 - 2 external liberties.

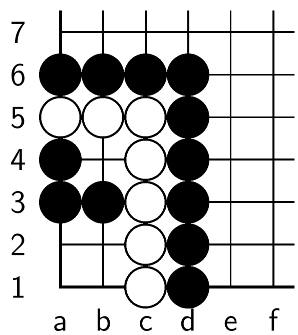


Diagram 13.

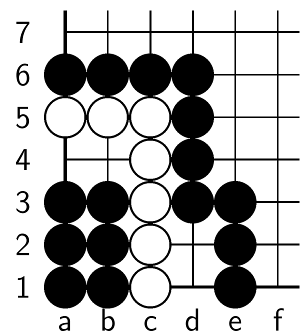


Diagram 14.

6.1.6 Eyes of size 9

Calm eyes of size 9 are large enough that for some of them Black now needs 2 ko-threats to kill (e.g. diagram 15 with 0 or 1 external liberty), or, that the black throw-in stones are strong enough that White needs an extra eye and one ko-threat to live (e.g. diagram 16)[5]. The position in diagram 17 is strange in that it is calm with 2 external liberties (as shown) with Black needing 2 ko-threats, it is also calm with zero external liberties and the different status that Black needs 1 ko-threat to kill, but it is not calm for one external liberty.

[5]Already an eye of size 8 is large enough to house a black living position so that even an extra white eye and many external liberties are not enough, but we are talking here only about *calm* eyes.

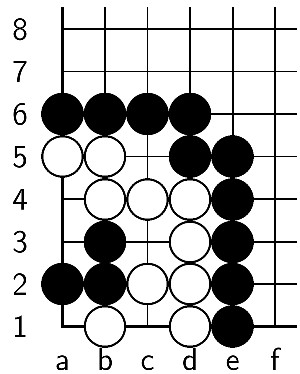


Diagram 15.

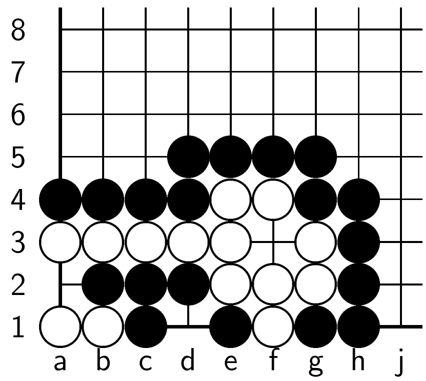


Diagram 16.

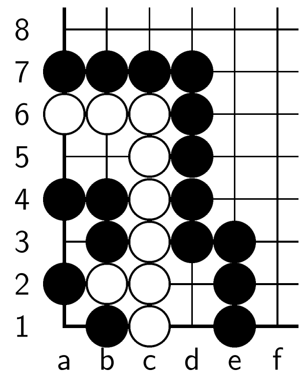


Diagram 17.

6.1.7 Eyes of size 10

Calm eyes of size 10 can, similarly to size 9, have the status that Black needs 2 ko-threats (diagrams 18, 19), Black needs one ko-threat (diagram 20 for 0 - 2 external liberties), or, White needs one ko-threat (diagram 21).

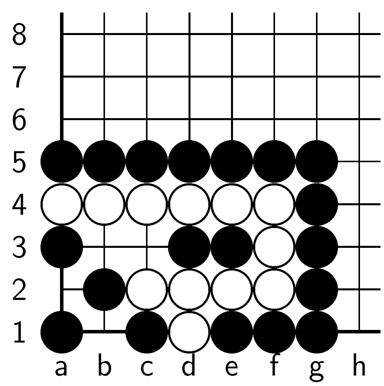


Diagram 18.

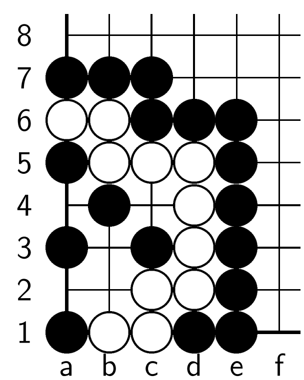


Diagram 19.

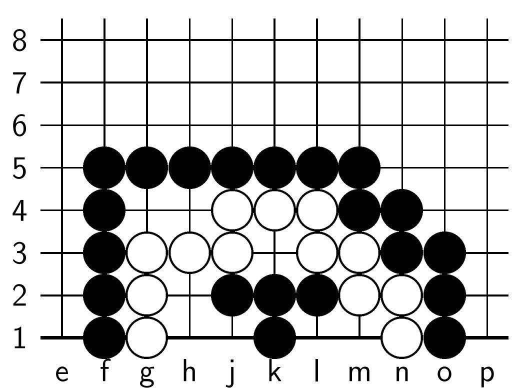


Diagram 20.

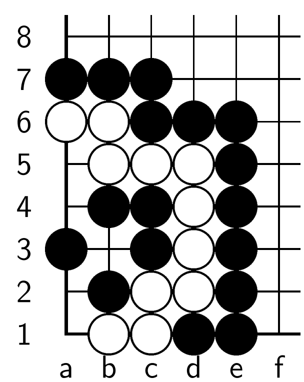


Diagram 21.

6.1.8 Eyes of size 11

A novelty of size 11 calm eyes is that Black may need 3 ko-threats to kill as in diagram 22. The same diagram with 1 external liberty does not give a calm position but with zero external liberties the position is again calm with Black needing only 2 ko-threats to kill. In diagram 23 Black needs also 2 ko-threats, in diagram 24 with 0 - 2 external liberties Black needs 1 ko-threat and in diagram 25 White needs 1 ko-threat.

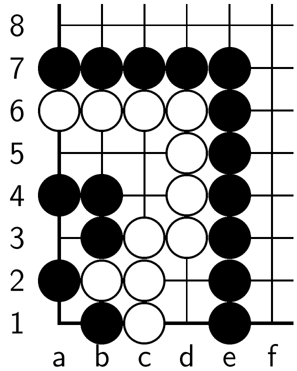


Diagram 22.

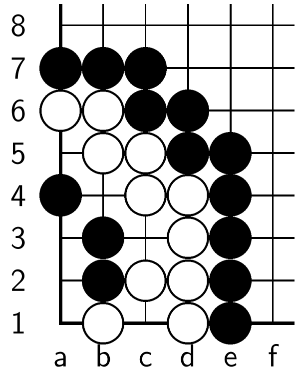


Diagram 23.

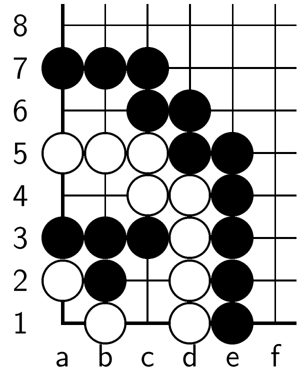


Diagram 24.

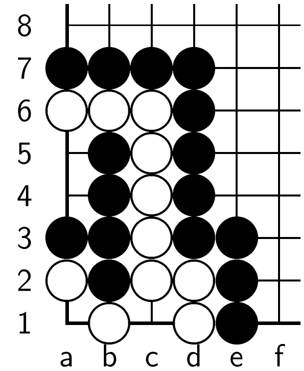


Diagram 25.

7 The influence of external liberties

In this section we search the database for eyes which change their character and/or best first moves with the number of external liberties of the eye.

7.1 Positions with a solution strategy depending on the number of external liberties

The smallest eye where a different number of external liberties requires different first moves has size 4. In diagram 26 with ko-forbidden point a1 White moving first has to fight the ko and after playing a ko threat White has to play on a1 otherwise Black goes there. If White had one more external liberty then White could afford to squeeze Black by playing on a3 and live.

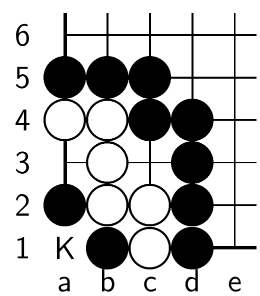


Diagram 26.

For size 5 there are 10 such positions. They are of the type above when White plays first. If Black plays first then in all of them Black can catch White if there are no outside liberties. If there is at least one then in these positions Black must start a ko either by playing inside the eye or taking the last outside liberty.

In the above and most other examples a higher number of external liberties allowed White to go for a more ambitious aim, i.e. a complete win instead of a ko. The different aims required different first moves. The following example is more exotic in that the result is the same but the different nature of external liberties dictates different approaches which require different first moves.

In the diagram 27 White has to create 2 eyes and this can be prevented only by ❶ a2. Other attempts fail: ❶ a1, ② b1 or ❶ b1, ② a2.

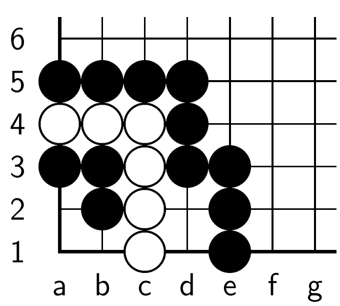


Diagram 27.

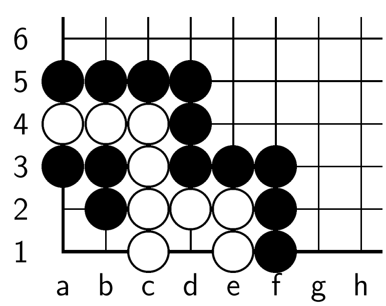
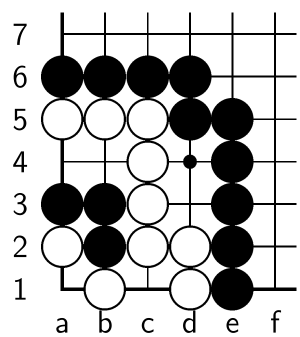


Diagram 28.

In diagram 28 two external liberties are exchanged for a single safe one. Thus, White does not need to split the left eye. Here ❶ a2 is useless. Instead Black can catch directly with ❶ b1 which was useless in diagram 27. If there are no or few external liberties, Black may be able to catch the monolithic eye by building up strength inside, for example, by having a long and thin shape that is able to enclose an own eye inside. The situation is different if the eye has many external liberties where the only strategy for Black may be to prevent White from splitting the eye into two eyes by going for a compact and thick throw-in shape.

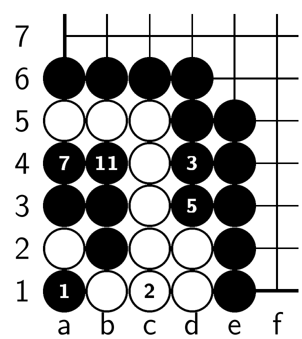
7.2 Non-monotonic lists of best moves

If the number of external liberties of a monolithic eye is decreased then the situation (i.e. the status value) should definitely not get better for the monolithic eye (i.e. for White), no matter what the shape and the internals are. Can one make a similar 'monotony' statement for the list of best moves? Let us look at the eye in the following diagrams with 2,3 and 4 external liberties. To minimize comments underneath the diagrams, ko-threats and ko-answers are not explicitly listed as they become clear from the move sequences in the diagrams.



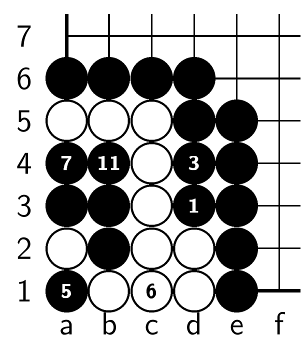
● to move first

Diagram 29.



④ pass ⑥ pass ⑧ @ a2
❾ @ ❶ ⑩ pass

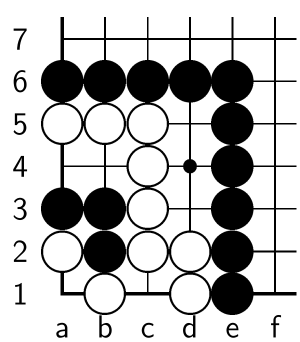
Diagram 30.



② pass ④ pass ⑧ @ a2
❾ @ ❺ ⑩ pass

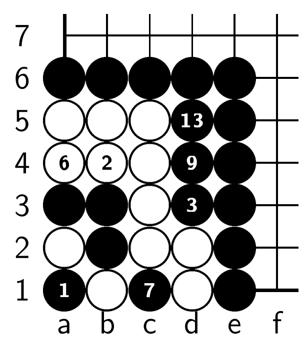
Diagram 31.

For one and two external liberties (diagram 29) the two moves ❶ a1 (diagram 30) and ❶ d3 (diagram 31) are fully equivalent. In both cases Black needs one external liberty (and White can pass or play elsewhere three times).



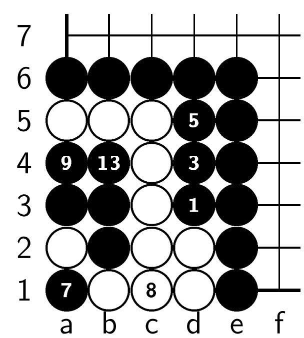
● to move first

Diagram 32.



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

Diagram 33.



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

Diagram 34.

For three external liberties (diagram 32) the move ❶ d3 (diagram 34) is the better one if we only compare the number of ko-threats, as it needs only one ko-threat compared to ❶ a1 (diagram 33) which needs two ko-threats.

Diagrams 33 and 34 are also useful as an illustration that counting the number of possible passes for both sides does matter but will also complicate the characterization of positions. Depending on what the value of the passing move is in a game, either ❶ a1 as in diagram 33 is the best move, costing Black two ko-threats and *two* passing moves of White, or ❶ d3 as in diagram 34 which costs Black only one ko-threat but *four* passing moves of White. Passing moves could be used by White either to protect removable ko-threats or to gain points somewhere else on the board.

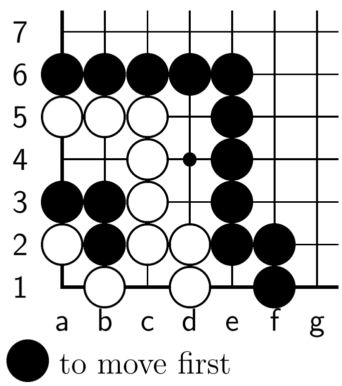


Diagram 35.

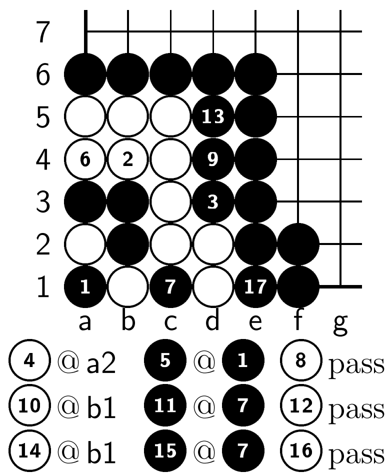


Diagram 36.

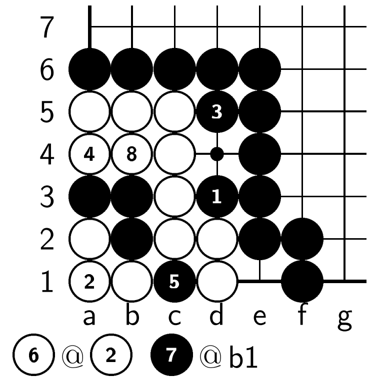


Diagram 37.

At last, for four external liberties (diagram 35) ❶ d3 gives Black an unconditional loss (diagram 37) whereas ❶ a1 in diagram 36 at least leads to a ko in which Black needs three ko-threats.

To summarize, what we call ‘non-monotonicity’ is that for one and two external liberties, the moves a1 and d3 are equivalent, for three liberties, d3 is better (at least at the end of game when passing has little value) and for four and more liberties again a1 is better.

Finally, the eye we looked at in previous diagrams, but now without external liberty provides us with yet another somewhat unusual situation. In diagram 38 the point a1 is ko-forbidden. In this position the best first move for Black is to pass and the best answer of White is to pass too after which Black should play on a1 as shown in diagram 39. This is a nice counterproof to the often met believe that two consecutive passes end a Go game.

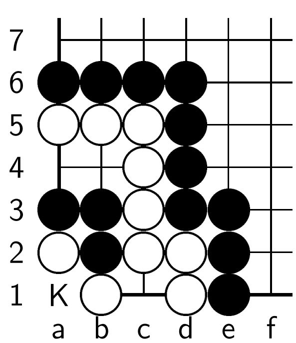


Diagram 38.

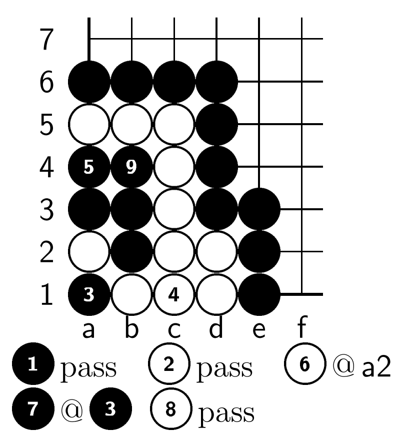


Diagram 39.

8 Summary

The database of monolithic eyes described in this contribution offers a number of features that other libraries of monolithic eyes do not have.

- We cover eyes of size up to 11 compared to size 7 plus incomplete groups of size 8 provided in [2].
- All possible internal configurations with stones of both colours inside are considered.
- We include eye positions that have an initial ko forbidden point.
- Each eye is evaluated for both sides moving first, with 0-31 external liberties and again with 1 external eye and in addition 0-31 external liberties.
- In each such evaluation *all* moves giving the best status are determined including the passing move if it is one of them.
- In all evaluations of eyes that result in ko, the number of external ko threats that are necessary in order to win are determined. Weaknesses of the computation are addressed in the appendix of [6]. The two main ones are 1) the number of passes is not maximized, which is correct if passing has no value, and 2) seki is treated as life.

Despite limitations in evaluating the eyes, the database is still a good source of unusual positions. We gave examples of eyes which are not settled and where nevertheless passing belongs to the best moves of both sides. In the last subsection we show an eye where the best attacking move depends on the number of external liberties of the eye in a non-monotonic way, i.e. the best move alternates with an increasing number of external liberties.

It would be interesting to see whether positions shown in section 6 are also special from the point of view of a thermographic analysis.

Acknowledgements

The authors thank Bill Spight for comments to the manuscript. This text and also [6] were typeset with GoLaTeX from Volkmar Liebscher.

References

[1] H. Landman: Eyespace Values in Go, in Richard J. Nowakowski (ed.), Games of No Chance : Combinatorial Games at MSRI, 1994 (Mathematical Sciences Research Institute Publications, No 29) and
`http://www.msri.org/publications/books/Book29/files/landman.ps.gz`

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

[3] The GnuGo project: Eyes and Half Eyes (2005), in
`http://hamete.org/static/gnugo36/gnugo_8.html`

[4] T. Wolf: Forward pruning and other heuristic search techniques in tsume go, Special issue of *Information Sciences* **122** (2000), no 1, pp. 59-76.

[5] J. P. Vesinet and T. Wolf: GoTools Online:
`http://lie.math.brocku.ca/GoTools/applet.html`

[6] T. Wolf: A library of eyes in Go I: Life & death definition consistent with 'bent4', in this volume.

[7] A. L. Zobrist: A New Hashing Method with Application for Game Playing. Technical report 88, Univ. of Wisconsin, April 1970.