Go patterns generated by retrograde analysis

Bruno Bouzy

C.R.I.P.5,
UFR de mathématiques et d'informatique,
Université Paris 5,
45, rue des Saints-Pères 75270 Paris Cedex 06 France
tel: (33) (0)1 44 55 35 58, fax: (33) (0)1 44 55 35 35

bouzy@math-info.univ-paris5.fr
http://www.math-info.univ-paris5.fr/~bouzy/

1. Introduction

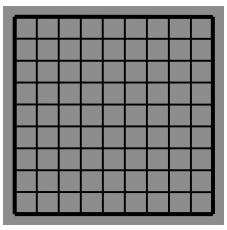
Retrograde analysis is a qualification and generation technique of positions in two-player, complete information games. In Chess [Thomson 1986], retrograde analysis enabled researchers to generate all six-piece positions [Thomson1996] and most of seven-or-eight-piece positions [Nalimov & al. 2000]. In Checkers, this technique is used by Chinook [Schaeffer & Lake 1996]. In Awari [Lincke & Marzetta 2000], Kalah [Irving & al 2000] and Nine Men's Morris [Gasser 1996], retrograde analysis was used to solve these games.

In Go, [Zobrist 1969] [Benson 1976] [Boon 1989] [Berlekamp & Wolfe 1994] [Chen & Chen 1999] [Wolf 2000] have been the most important publications over the last forty years. But the complexity of Go played on usual boards (19x19, 13x13 or 9x9) forbids direct use of retrograde analysis. [Cazenave 1996, 2000] showed how to build small patterns with retrograde analysis on specific goals such as "eye" and "connection". So far, however, no study about retrograde analysis applied to the evaluation function of Go has been done. This paper describes an experiment aiming at enabling a Go program to use small patterns automatically generated by retrograde analysis.

Section 2 of this paper contains the definitions and the notations. Section 3 shows the experiments carried out on 3x3 "open" boards, which allows to present the first feature of Go, the possibility of "passing". Section 4 describes the special management of "ko" and other loops arising on 4x4 open boards. Section 5 highlights the results obtained on 3x3 "semi-closed" or "closed" boards in a solving game perspective. Section 6 defines a "heuristic player" who plays on large boards with the small patterns described in the previous sections. Before our conclusion, section 7 discusses the results.

2. Definitions and notations

We want to get a program playing on "large" boards. As retrograde analysis is not possible on such boards, we cut the "large" board into "small" board pieces: the middle pieces, the edge pieces and the corner pieces. Figures below show examples of board pieces.



"large" board

middle piece

corner piece

edge piece

"small" board

In the following, we call "closed" board, a board with 4 edges. We call an "open" board, a board without any edge. We call "semi-open" or "semi-closed" the other kinds of boards. When a stone is put on the edge of an open board, we made the simplifying hypothesis that the stone gets enough liberties not to be captured[1]. Furthermore, we assume we get an evaluation function which the difference of "points" of each color. A black point is either an intersection occupied by black or an empty intersection whose 4 neighbors are black points.

			
Black points	5	3	9
White points	4	6	0
Evaluation	+1	-3	+9

Besides, we mainly use the combinatorial game theory [Conway 1976] notation with some adaptations. First, we assume that Left is Black and Right is White. Secondly, combinatorial game theory assumes the player who cannot play is the loser but we do not use this assumption. In our study, the moves – excluding pass - leading to a repetition on the couple (position, nextPlayer) are forbidden. A pass move is always allowed and the game ends when the two players pass consecutively. When a game G is a switch, we write G = {l|r} where l and m are numbers. We use {g|} to express that White passes. {|} is the game in which the two players pass. Furthermore, we introduce the notation G= {l|m|r}. In this case, l and r keep their meaning and m is the value of the evaluation function. This will be useful when a game is {|}. In such a case, the game is {|f|} = f. If f>0 (respectively f<0, f=0), then Black wins (respectively White wins, the game is a draw).

All the experiments described in this paper have been performed on a 450Mhz PC with 128Mo. The pattern position description is called the *left part* of the pattern. It is composed of a description of the black position and a description of the white position. Each colored description

[1] A single stone on a closed edge has 3 liberties.

is coded with a set of bits. We chose 16 bits – 2 bytes – to code patterns smaller than 4x4. Therefore the left part is included into 4 bytes. The properties of the pattern belong to the *right part* of the pattern. It is composed of the game value, denoted by {k|l}, constituted by two values coded in one byte each. The right part also contains the set of moves advised for each player (2 bytes per player). Therefore, in our study, we assume a pattern is coded into 10 bytes. For each size and kind of board in our study, the study contains the left part generation and the right part qualification by retrograde analysis.

3. 1x1, 2x2 and 3x3 open board Go: the "pass" move management

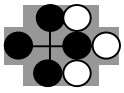
1x1 open board Go is trivial. Its value is {+1|-1}. 2x2 open board Go is also very simple. The total number of patterns 2x2 without symmetry, rotation and Black-White inversion is $3^4 = 81$. But, for boards larger than 2x2, the use of 90°, 180° or 270° rotations and symmetry reduces the total number of patterns. There are thirteen 2x2 open patterns. When the size of the board is large, the interest of managing symmetries, rotations and inversions reduces the total number of patterns by a factor 16. On closed boards (see Section 5) where the edge impact is important, this management is not advised. 2x2 open board Go is bound to be a draw.

We generated and qualified all 3x3 open patterns by retrograde analysis. The total number of patterns without capture, symmetry, rotation or inversion is $3^9 = 19\,683$. Without capture, but with symmetry, rotation and inversion this number is 1444. The only particularity of 3x3 open board is the capture of the central stone. With capture, symmetry, rotation and inversion this number is 1423. The explicit game graph is contained in 10ko. With the 3x3 open pattern base, a perfect player has been built. The game value is of course {+1|-1}. Furthermore, the values of *all* 3x3 open patterns have been calculated.

In this section, although its effect is not clear on very small open boards, we present the move "pass" management that we used in our study. In Go, a player does not want to play when he has to reach a position which is worse than the position he can reach in passing. The pass move management increases the complexity of min-max search [Mueller 2000]. On positions such as {|f|} in which nobody wants to play, the game value is f. On {a, b, c, …|f|}, min-max simplifies the game into {x|f|}. If x>f, then Black plays and the game value is x. In the other case Black passes and the game value is f. When the two players play, the game is like {a, b, c, … |f| k, l, m, … }, mini-max simplifies the game into {x|f|y}. If x>y then the game value is {x|y}. If x=y, then the game is a number. If x<y, nobody wants to play. Three cases are possible. If y>f>x, then the game value is {|f|}=f. If f>y>x, then the game value is {|f|y}=y. And symmetrically, if y>x>f, then the game value is {x|f|}=x.

4. 4x4 open board Go : the "ko" management and some combinatorial aspects

We generated and qualified all the 4x4 open patterns. The total number without capture symmetry, rotation and inversion equals $3^{16} = 43\,046\,721$. Without capture, it is 2 700 373. With capture, it is 2 611 573. It takes about one hour to generate the base. On 4x4 boards, "ko" may arise:



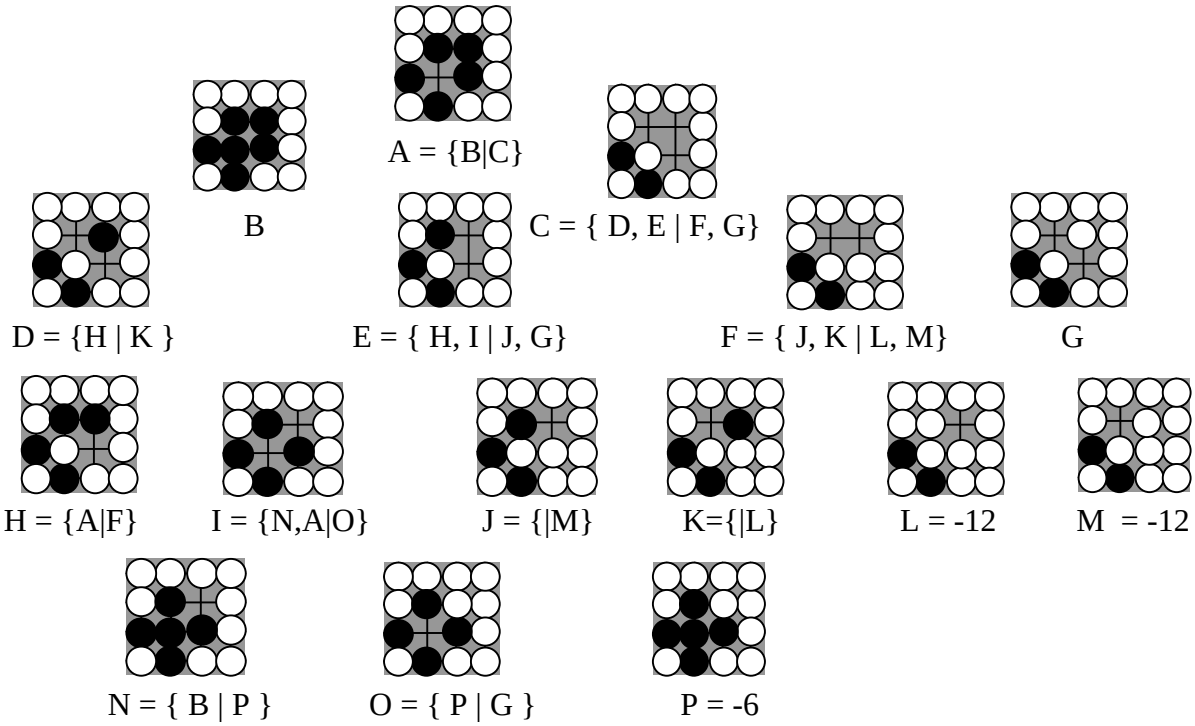
Combinatorial game theory already dealt with ko representation [Mueller & al. 1997] [Spight 1998]. What we keep from ko knowledge is the following fact. If Black plays, the game value is calculated on the set of positions reached by Black in *one* move. If White plays, the game value is calculated on the set of positions reached by White in *two* moves (the first one being the capture of the ko). This fact is of importance. When you only want to prove that a game is solved, you only have to compute the values if each player plays one move. But when you want to compute values of games containing ko to be used within a global game, you have to consider two moves in a row by the same player. It is worth noticing that a game included in a global one may violate the rule forbidding repetition.

When a game A contains a ko, we write A = {… | B, …} with B = { A, … | … }. More precisely we write A={Ag1, Ag2, …, An1, An2,…, | B, Ab1, Ab2, …} where Ag1, Ag2, … are the options in which Black wins the ko, An1, An2, … are the options in which Black does not modify the ko, B is the option in which White takes the ko and Ab1, Ab2, … are the options in which White does not modify the ko. Symmetrically, we write B = { A, Bn1, Bn2, … | Bb1, Bb2, … , Bp1, Bp2, … } where A is the option in which Black takes the ko, Bn1, Bn2, the options in which Black does not modify the ko, Bb1, Bb2, … the options in which White does not modify the ko and Bp1, Bp2, … the options in which White wins the ko.

When considering game A independently (for example if A is the whole board), the ko rule forbids A from B. Game A keeps the same expression and we replace game B by B = { Bn1, Bn2, … | Bb1, Bb2, … , Bp1, Bp2, … }.

But, if we consider A as a sub-game (a small pattern) of a whole game (the whole board), the correct management of ko is to calculate the two values by deleting the loop between A and B. The left value is the maximal value of Ag1, Ag2, … , An1, An2, … assuming these values are known. The right value is the minimal value of Ab1, Ab2, …, Bb1, Bb2, …, Bp1, Bp2, … assuming these values are known. Therefore, our ko management module replaces the A and B expressions by A = { Ag1, Ag2, … , An1, An2, … | Ab1, Ab2, …, Bb1, Bb2, …, Bp1, Bp2, … } only, which deletes the loop between A and B. By definition, An1, An2, … Ab1, Ab2, …, Bb1, Bb2, …, also contain kos. They have either been already modified by the ko management module or they will be modified later. Practically, the more the An1, An2, …, Ab1, Ab2, …, Bb1, Bb2, …, there are, the longer the execution time is.

In addition, our experiments showed other loops. Fortunately these loops were solved by min-max containing move pass.

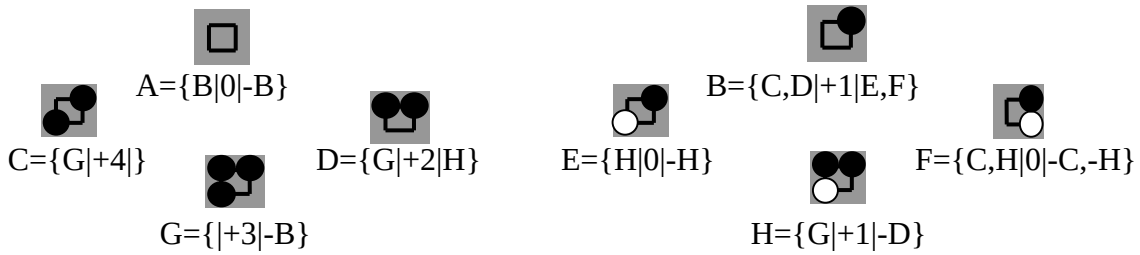


We have B = -4, G = L = M = -12 et P = -6. With a mini-max and pass move we get: A = {-4|-12}, C = D = F = G = H = J = K = -12, E = O = {-6|-12}, I = -6, N = {-4|-6}.

The 4x4 open pattern base was completely generated and qualified, and as a result, a perfect player was built. The 4x4 open Go value is 0, because no capture arise from the empty board if the two players play correctly. In our study, 26 Mbytes of memory are sufficient to store all the open patterns. Execution lasts 10 hours.

5. 1x1, 2x2, 3x3 not open boards : the edge problem

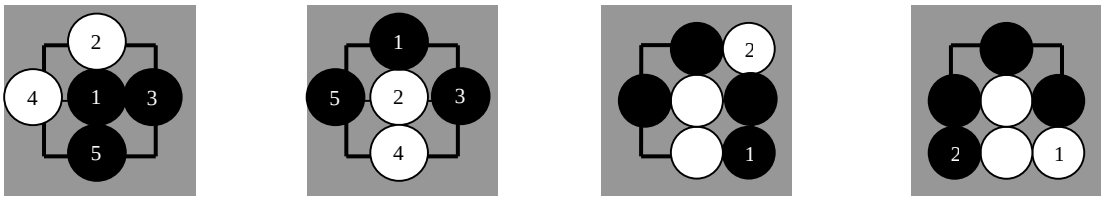
The game value of 1x1 closed board Go is {|} = 0. Due to the presence of the edge, 2x2 closed board Go is not simple. With symmetry, rotations and inversion, there are eight 2x2 closed patterns.



But, in order to find the value of the game, we did not use the rotation, symmetry and inversion reduction because the rules of Go do not forbid the repetition of positions which are identical with rotations, symmetry or inversion. The game value of each position above greatly depends on the rule forbidding repetition. Moreover, we used a mini-max including pass move with a depth

sufficient enough to contain the longest sequence of moves on such board with the forbidding repetition rule. Finally, A={+1|-1}, B={+4|+1}, C=+4, D=-1, E= {+1|-1}, F={+4|-4}, G=-1 and H=+1.

With rotations, symmetry and inversion, 3x3 one-edge board Go has 4930 patterns. Its value is {+1|-1}. 3x3 one-corner board Go has 4396 patterns. Its value is {+1|-1}. 3x3 closed board Go has 924 patterns and its value is {+9|-9}. The optimal sequence is shown on the left figure below. Some positions, like the second one starting from the left, are "seki". The third figure starting from the left gives the sequence obtained if Black plays first on the second figure. The fourth figure starting from the left shows the sequence if White plays first on the second figure.



+9 +3 -9 +9

To qualify a position without knowing all the values of the following positions, we very often use a short cut. If P = { M, N, … | O}, with M equals the maximal value of the evaluation function (+9 on 3x3), with N and other options not calculated, then a good short cut was to write P = {M|O}. The use of this rule was very efficient and pruned a lot of moves. This short cut requires to know the maximal and minimal values of the evaluation function.

6. Large boards : the "heuristic" player

To play on arbitrary size boards, we developed a "heuristic" player. To choose the move to play, the "heuristic" player chooses the board piece sub-game with the highest temperature and plays the move advised by this sub-game. For each size of large boards, the "heuristic 4x4" player, using 4x4 middle patterns and 3x3 corner and edge patterns performed twenty games against the "heuristic 4x4" player which used 3x3 patterns only. The results showed that the larger the board is, the more easily the 4x4 player wins. Although we did not build any nxn heuristic player with n>4, our results confirm the idea that the greater n is, the stronger the heuristic player is.

The heuristic nxn players sometimes play absurd moves. For example, they can put themselves into "atari" because the move is advised by a pattern whose temperature is high. This kind of error occurs often enough to enable the 3x3 heuristic player to win some games against the 4x4 heuristic player.

Besides, we also made games between the heuristic 4x4 player and our specific Go playing program Indigo [Bouzy 2000]. Of course, the heuristic player lost all the games because it did not know "ladders", "group", "life", "death" or "territory" and many other concepts useful to a basic Go program.

7. Discussion

The next stage in our study, will center on 5x5 open board Go, 4x4 closed or semi-closed board Go. The total number of 5x5 patterns without capture, symmetry, rotation or inversion is 3^{25} = 847 288 609 443. Without capture but with symmetry, rotations or inversion, this number is about 3^{25} / 16 = 52 955 538 090 = 5.10^{10}. This number can be compared to the 10^{11} states of Nine Men's Morris [Gasser 1996]. Assuming 8 bytes are needed to store the left part of each 5x5 pattern, and 10 bytes for the right part, then 18 bytes are sufficient to store one 5x5 pattern. Therefore, 10^{12} bytes = 1 terabyte would be sufficient to store all the 5x5 pattern database. By using a right hash function, the left part of the pattern is not needed and not storing the advised moves of the pattern right part is possible. Therefore, such a hash function requires 2 bytes and all the database can be contained in 100 gigabytes with the possibility of using the game value for each entry. By limiting the aim of the database to the possibility of solving 5x5 Go only, one bit is necessary for each 5x5 pattern. Thus 50 gigabits are necessary to contain all the 5x5 database without the possibility of using the game value of each pattern.

No memory space problem but only loop problems exist for 4x4 closed boards. However, the study of 4x4 closed board Go is not urgent because closed patterns are not needed on large boards. This study may simply provide a theoretical interest.

We currently think no integration of this study can be performed within a Go playing program such as Indigo [Bouzy 2001] without an important adaptation work. Actually, this approach does not take the abstract description of the position into account. The size (4x4) of the board pieces is too small to give to the heuristic player a local behavior which can be considered as correct. Some attempts have been done to insert abstract concepts into the patterns: "liberties" or "eyes" [Cazenave 1996, 2000]. Other concepts such as "group" strength and "territory" estimation should be inserted. But the combinatorial price to pay for this insertion, increases too quickly indeed with the size of the patterns, and it forbids such insertion into current computers.

Retrograde analysis has been used to solve games [Gasser 1996], [Lincke & Marzetta 2000], [Irving & al 2000] , [Schaeffer & Lake 1996]. In this context, for each size and kind of board piece, the table below gives the game value of the initial position of the associated game and the number of positions taking symmetry, rotations and inversion into account. The gray boxes have not been calculated by our study.

size	1x1		2x2		3x3		4x4	
middle pattern	{+1|-1}	2	0	13	{+1|-1}	1423	0	2 611 573
edge pattern	{+1|-1}	2	0	25	{+1|-1}	4930	0	_
corner pattern	{+1|-1}	2	0	24	{+1|-1}	4396	0	_
closed pattern	0	2	{+1|-1}	8	{+9|-9}	924	{+16|-16}	_

Instead of determining the game value of these games only, our study determined the game values of *all* the positions of these games. In performing such experiments on small board pieces we found similar results to those in former works such as [Thorpe & Walden 1972] or in recent works such as [Lorentz 1997]. Finally, the work of [Berlekamp & Wolfe 1994] in endgame being so successful, we need to compare our approach to their approach. First, Berlekamp's model only

applies on particular positions whereas our approach applies on all positions. The positions processed by Berlekamp's model include board pieces which are independent from each other and which are already processed (the game value of the board pieces is given to the model). In our approach, board pieces can be dependent on each other and the game value of the board pieces are computed by retrograde analysis. Furthermore, Berlekamp's model reaches a global decision which can be assessed as perfect, and this is not the case of our heuristic player's global decision module.

8. Conclusion

This paper answered the question of knowing how retrograde analysis, mainly studied in Chess and in other two-player complete information games, can be used in Go. After giving the definition of open, and closed boards, our approach calculated all the pattern game values up to 3x3 boards and 4x4 open boards. Beyond the values of these games, which were foreseeable, our retrograde analysis approach generated and qualified all the positions of these games so as to be used in a Go program playing on large boards.

Our approach dealt with the pass move and ko. On closed or semi-closed boards, our approach stopped on 3x3 boards. On open boards, we finished with 4x4 boards due to memory space and execution time limitations. There are 2 611 573 four by four open positions occupying 26 Mbytes of memory. They have been generated and qualified by retrograde analysis and used by a heuristic player. There are 5.10^{10} five by five open positions that we did not generate because it would need 100 Gbytes to be used by a Go program, or about 50 Gbits to be used in a solving game perspective. Besides, the automatically generated pattern database enables us to create a heuristic player playing complete games on large boards. Furthermore, the interesting point lies in the creation of a player which owns very little hand-coded knowledge: rules of the game, elementary evaluation function, ko and pass modules. This player is mainly composed of a large patterns database automatically built by the computer. Nevertheless, this approach is slightly disappointing because the size of board pieces is very small (4x4) and the local behavior of the heuristic player is still mediocre. However, this approach can be improved with 5x5 patterns in the near future or with 6x6 patterns in a distant future. This approach cannot work by itself to play a complete game with good results. An abstract description is needed. But this approach is worthwhile to help Go programs in the endgame. Our work can be adapted to other games as well and can be used with other evaluation functions relevant to other phases of the game such as middle game.

9. References

[Benson 1976] D.B. Benson, "Life in the game of go", Information Sciences, 10, pp.17-29, 1976.

[Berlekamp & Wolfe 1994] E. Berlekamp, D. Wolfe, "Mathematical Go Endgames, Nightmares for the Professional Go Player", Ishi press international, San Jose, London, Tokyo, 1994.

[Boon 1989] Mark Boon, "Pattern Matcher for Goliath", Computer go 13, winter 89-90.

[Bouzy 1995] Bruno Bouzy, "Modélisation cognitive du joueur de go", Paris 6 University, 1995.

[Bouzy 2001] Bruno Bouzy, "The go program Indigo", http://www.math-info.univ-paris5.fr/~bouzy/INDIGO.html

[Cazenave 1996] T. Cazenave, “Automatic Acquisition of Tactical Go Rules”, in: Proceedings of the 3rd Game Programming Workshop in Japan'96, Hakone, pp. 10-19, 1996.
[Cazenave 2000] Tristan Cazenave, “Generation of Patterns with External Conditions for the Game of Go”, Advance in Computer Games, 9, 2000.
[Chen & Chen 1999] Ken Chen, Zixingh Chen, “Static analysis of life and death in the game of go”, Information Sciences, 121, (1-2), pp. 113-134, 1999.
[Conway & al. 1982] John Conway, E. Berlekamp, R. Guy, “Winning ways”, Academic Press, 1982.
[Gasser 1996] R. Gasser, “Solving Nine Men’s Morris”, Games of no chance, MSRI Publications, Vol. 29, Novakowski (eds.), Cambridge University Press, pp. 101-113, 1996.
[Heinz 1999] E.A. Heinz, “Knowledgeable encoding and querying of endgame databases”, ICCA Journal, vol. 22, n°2, pp. 81-97, 1999.
[Nalimov &al. 2000] E.V. Nalimov, G.M.C. Haworth, E.A. Heinz, “Space-efficient indexing of chess endgame tables”, ICCA Journal, vol. 23, n°3, pp. 148-162, 2000.
[Irving & al 2000] Geoffrey Irving, Jeroen Donkers, Jos Uiterwijk, “Solving Kalah”, ICCA Journal, vol. 23, n°3, pp. 139-147, 2000.
[Lake & al 1994] Robert Lake, Jonathan Schaeffer, P. Lu, “Solving large retrograde-analysis problems using a network of workstations”, Advances in Computer Chess 7 (eds. H.J. van den Herik, I.S. Herschberg, and J.W.H.M. Uiterwijk), pp. 135-162, University of Limburg, Maastricht, The Netherlands, 1994.
[Lincke & Marzetta 2000] Thomas Lincke, Ambros Marzetta, “Large endgame databases with limited memory space”, ICCA Journal, vol. 23, n°3, pp. 131-138, 2000.
[Lorentz 1997] R. Lorentz, “2xN Go”, in: Proceedings of the 4th Game Programming Workshop in Japan'97, Hakone, pp. 65-74, 1997.
[Mueller & al. 1997] M. Mueller, E. Berlekamp, B. Spight, “Generalized Thermography: Algorithms, Implementation and Application to Go endgames”, TR-96-030 Berkeley, 1996.
[Mueller 2000] Martin Mueller, “Not like other games - why tree search in go is different”, in: Proceedings of the 5th Joint Conference on Information Sciences, 2000.
[Schaeffer & Lake 1996] Jonathan Schaeffer, Robert Lake, “Solving the game of checkers”, Games of no chance, MSRI Publications, Vol. 29, Novakowski (eds.), Cambridge University Press, pp. 119-133, 1996.
[Spight 1998] W. Spight, “Extended Thermography for Multiple Kos in Go”, First International Conference on Computer and Games 98, in Lecture Notes in Computer Science, n° 1558, H.J. van den Herik, Hiroyuki Iida (eds), Springer, pp. 232-251, 1998.
[Thomson 1986] Ken Thomson, “Retrograde analysis of certain endgames”, ICCA Journal, vol. 9, n°3, pp. 131-139, 1986.
[Thomson 1996] Ken Thomson, “6-piece endgames”, ICCA Journal, vol. 19, n°4, pp. 215-226, 1996.
[Thorpe & Walden 1972] E. Thorpe, W. Walden, “A computer assisted study of Go on M x N boards”, Information Sciences, 4, pp. 1-33, 1972.
[Wolf 2000] Thomas Wolf, “Forward pruning and other heuristic search techniques in tsume go”, Information Sciences, 122, pp. 59-76, 2000.
[Zobrist 1969] Albert Zobrist, “A model of visual organisation for the game of go”, Proceedings AFIPS 34, pp. 103-112, 1969.