The INDIGO program

Bruno Bouzy
LAFORIA-IBP, Paris 6 University
Tour 46 2ème étage, 4, place Jussieu 75252 PARIS
Tél.: 33 (1) 44 27 70 10 Fax: 33 (1) 44 27 70 00
e-mail : bouzy@laforia.ibp.fr

Why a Computer Go program ?

Between 1991 and 1994, we built a cognitive model of the human Go player. This work is related in [Bouzy 1995]. We tested the validity of the cognitive model on a computer. In this meaning, our model is computational. We developed a Go playing software named INDIGO. This program aims to play full Go games as well as possible. We already gave a one hundred pages description of INDIGO in our french thesis dissertation. The aim of this paper is now to give a ten pages description of this program.

In the first part we describe INDIGO. In particular, we show how an interaction model between groups is useful to static evaluation of life and death of not fully cercled groups. Also, we show how fuzzy mathematical morphology is useful to static evaluation of territories. In the second part, before conclusion, we show the results and we discuss about strong and weak points of our approach.

How INDIGO is built ?

Main paradigms

INDIGO follows three paradigms : object, game and level paradigms.

Object paradigm

To play Go, human players recognize objects on the board such as strings, eyes, connections, dividers, groups, territories, zones, empty spaces and so on. These objects are very useful to elaborate strategy. On the contrary of Chess or Shogi where "objects" such as Kings, Queens, Rooks, Horses or Pawns are already clearly defined by the rules of the game, Go objects are constructed by the player, not by the rules. Most of the time their construction is vague or fuzzy. More, this construction is often evident and implicit for human players. Therefore, it is very difficult to extract knowledge from Go players. In front of this ontologic difficulty, that is also present in real world complex domains, object-oriented paradigm is very adapted.

Object-oriented paradigm provides conception tools or langages useful to classify objects. We choose an object-oriented langage that was quick enough at run-time and carriable from system to system : C++.

Full name of INDIGO is MOOD INDIGO. This acronym is significant from the object paradigm. It means : My Object Oriented Design Is Now Designed In Good Objects!

Game paradigm

Go players act on objects they recognized. Black and White make opposite actions on objects such as save/capture a string, kill/save a group or expand/reduce a territory. These actions are sub-actions of the main action that is to win the whole game. To each couple of opposite actions we bind a "game". In such approach, the whole game is decomposed in sub-games. Other Computer Go programs use this kind of approach [Boon 1991] [Müller 1995] [Cazenave 1995]. Each game get specifications of its rules : legal moves, winning and losing static conditions with two terminal states : "won" and "lost". For each game, INDIGO computes what happen if Black or White move first. This idea is inspired from [Conway 1982] and is adapted to sub-games of the whole Go game because you do not know a priori who will play first in a sub-game. Depending of the outcome ("won" or "lost") there are four possible dynamic states :

> the game is won whoever plays first,
< the game is lost whoever plays first,
* who plays first wins,
0 who plays first loses.

INDIGO uses this game definition and we will show an example later.

Level paradigm

We define different levels. Each level uses the levels below it. A level gathers similar classes of objects. The levels are :

- the zero level,
- the elementary level,
- the iterative level,
- the global level.

Figure 1 shows the global architecture of INDIGO including the four levels.

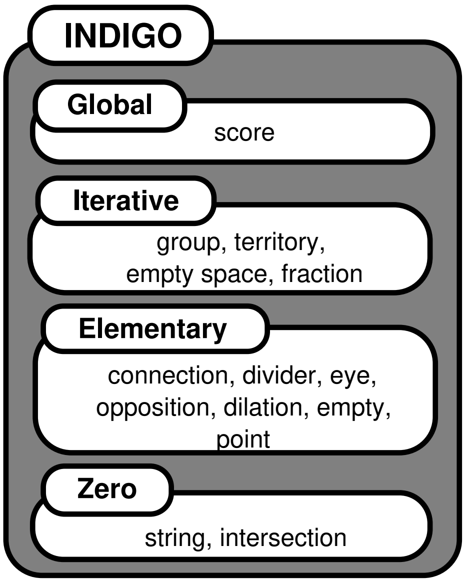

figure 1

These abstraction levels will be the guideline of the following description. For each level, we ask a question about INDIGO and give an answer.

Can INDIGO read ladders ?

The proverb *"If you cannot read ladders, do not play Go."* has been adapted to Computer Go by Mark Boon [Boon 1991] : *"If you cannot program ladders, do not program Go."* Faced to our ambition to program Go, we were obliged to program ladders. We defined the partisan "string game" that is won if the string gets 3 liberties or more and that is lost if the string is removed from the board. Legal moves are the liberties of the string and liberties of adjacent strings. Figure 2 shows an illustration of the "string game" with the four possible results : >, *, < and 0.

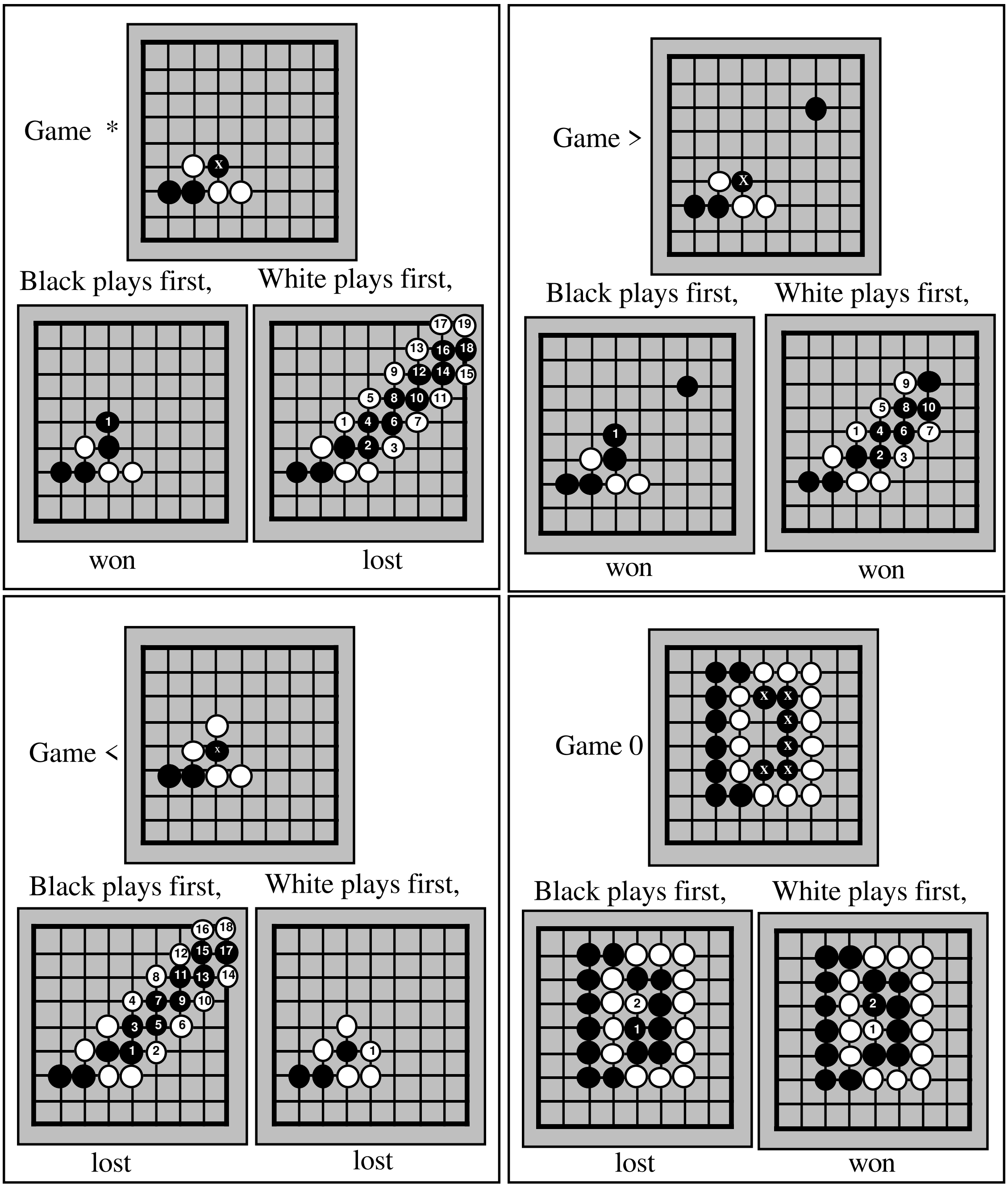

figure 2

The goal of this level is to know the state of each string and empty intersection of the board using as few concepts as possible, liberties of strings at least. This level contains two sub-games : the "string game" presented above and the "intersection game" that reads what happen if the players try to occupy an empty intersection. This level is largely used so it must be fast. Other programs are using a "string game" with four liberties at least for the string to be stable, but we think that catching a three or four liberties string uses more complex concepts that we prefer to put in higher levels.

Which elementary concepts does INDIGO use ?

Go board contains a priori confusing informations. The aim of this level is to classify this information following elementary concepts such as connection, divider, eye, 2-eyes, opposition, dilation, empty space. For each elementary concept, it corresponds a sub-game and a set of rules. We describe the rules and then we give examples of sub-games.

The rules

Figure 3 gives an example of a rule.

figure 3

It means that playing A will connect black stones and divide white stones in two.

Obvious particularity of Computer Go rules is the visual pattern that is present in the left part of the rules. So, obvious activity of a Computer Go program is pattern-matching. We reuse principles described in [Boon 1989] to implement pattern-matching.

The elementary sub-games

For each sub-game of the elementary level, figure 4 gives one example of one rule used by one elementary sub-game .

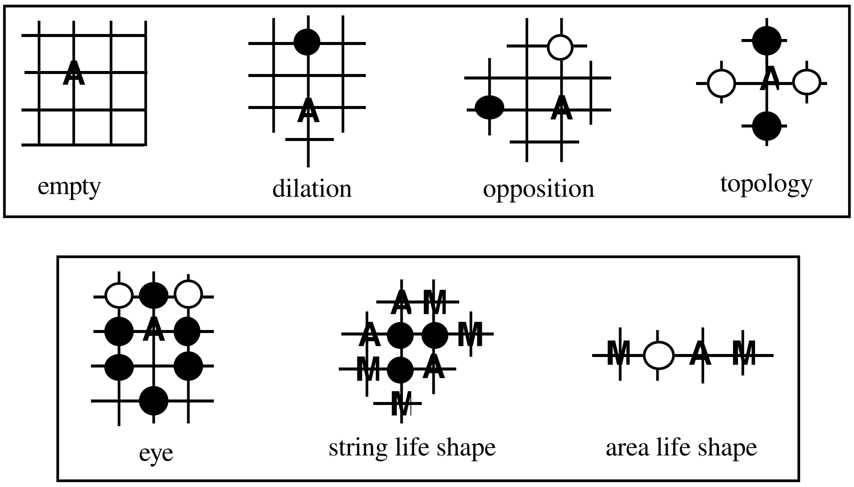

figure 4

"empty game" is an obvious symetric game where each player wants to occupy an empty space. First player that plays wins. The result of this game is always *. The "empty" rules' base contains about 10 rules.

"dilation game" is an obvious partisan game where one player wants to expand its group while the other one wants to reduce it. The result of this game is always *. The "dilation" rules's base contains about 20 rules.

"opposition game" is an obvious symetric game where each player wants to expand its group while reducing opponent group. First player that plays wins. The result of this game is always *. The "opposition" rules' base contains about 80 rules.

"connection game" is a partisan game where one player wants to connect two groups while the other one wants to cut them. "division game" is also a partisan game where one player wants to divide a space in two zones while the other one wants to keep it in one piece. "connection game and division game" are symetric from each other. They use the same "topologic" rules' base (about 200 rules). These two games are very important as much moves in Go games are connect or cut moves.

"eye game" is a partisan game where one player wants to keep a real eye while the other one wants to destroy it. This game use the "eye" rules' base (about 40 rules).

"two-eyes game" is a partisan game where one player wants to split a small territory in two parts. This game use the "string life" and "area life" rules' bases (about 60 rules). On this example, M means "not Black".

Elementary level contains about 400 rules and six classes of sub-games.

What are the specificities of INDIGO ?

The iterative level contains two classes of objects : groups and territories that are constructed iteratively and simutaneously.

How does INDIGO recognize life and death of groups ?

Construction of groups

Groups are the most important objects in the game of Go. They are constructed iteratively with stones linked with > "connection games". They get several attributes : lifebase, friendship, emptyship, opponentship and health. These attributes are very useful to qualify life and death of groups even when they are not completely surrounded.

Friendship attribute

For each friend neighbour of a group, INDIGO creates one "friend interaction game" that gathers information about all "connection games" between the group and the neighbour group. "friend interaction game" adds results of "connection games". "friend interaction game" can be >, < or *.

If a "friend interaction game" is >, the two groups are connected in one group and the cycle runs back again from start.

Friendship attribute is a game that gathers the results of all "friend interaction games" into a result that is >, < or *. When "friendship game" is <, that means that the group cannot connect to an other group.

Lifebase attribute

This attribute is a game that gathers information about "eye games" and "two-eyes game". For each adjacent territory to the group, INDIGO put the "number" of eyes of the territory in the lifebase attribute. Then it "adds" these "numbers" to find the lifebase of the group. "Lifebase game" can be >, < or *. When "lifebase game" is <, that means that the group cannot get two eyes.

Emptyship attribute

This attribute is a game that gathers information about "dilation games" and "divide games" to know if the group can run away toward empty spaces. "Emptyship game" can be >, < or *. When "emptyship game" is <, that means that the group cannot runs away into an empty space.

Opponentship attribute

When a group gets all its previous attributes <, the problem is to know if the group is dead or alive. For each opponent neighbour of a group, INDIGO creates one "opponent interaction game" that gathers information about all relative results of previous attributes between the two groups. INDIGO compares "friend interaction game" of the group to the "friend interaction game" of the opponent group, and so on for "lifebase games" and "emptyship games" of the both groups. Result of an "opponent interaction game" can be >, <, *, (if the comparaison is too difficult, INDIGO tells that the result is fuzzy) or 0 (the two groups are in a seki situation).

Opponentship attribute is a game that gathers the results of all "opponent interaction games" of the group into a result that is >, <, *, or 0. When "opponentship attribute" is <, that means that the group cannot win any battle against one of its opponents.

Health attribute

Health attribute gathers information of previous attributes. Result of "health game" is >, <, *, or 0. Main point is that when all previous attributes are <, then "health game" is also <. In this case, the group is dead. It cannot get two eyes, nor connect to another group, nor run away, nor win a battle against an opponent.

In this case a territory is created and opponent groups are clustering around it to create a new big group.

Interpretation of life and death of groups stops when no dead groups are detected.

How moves are propagated through sub-games to parent game ?

When a parent game and sub-games are *, the moves computed by sub-games to attack or defend are given to the parent game. This way, each group, with an "health game" whose result is *, knows the moves it must play to make its "health game" become > or <. This method gives to the program a good abductive behaviour that is satisfactory in order to simulate human Go players.

Conclusion

This model is strongly based on interaction between groups. That means that the state of a group cannot be found independantly from the state of the neighbour groups. This model is one of the strong point of INDIGO. With this model INDIGO can kill groups, that is essential to Go play. Unfortunately, we do not get place enough in this paper to give examples and diagrams. Interested reader can read our dissertation thesis where many drawings and examples can be found.

How does INDIGO recognize territories ?

Construction

Territories are very important objects in the game of Go. They are constructed with fuzzy dilations and contractions, two well-known mathematical morphology operators [Serra 1982]. Territories are also constructed when dead groups are found (see before).

Initialization

Our algorithm starts with +128 (resp. -128) value on black (resp. white) intersections and 0 value on empty intersections. It makes 4 dilations and 13 contractions or 3 dilations and 7 contractions according to the scale of territories to recognize.

Dilation

For each intersection of the board, if it is positive (resp. negative) or null and not neighbour of a negative (resp. positive) intersection, the algorithm adds (resp. substracts) the number of positive (resp. negative) neighbours to the value of the intersection.

Contraction

For each intersection of the board, if it is positive (resp. negative), the algorithm substracts (resp. adds) the number of negative (resp. positive) or null neighbours to the value of the intersection, checking it stays positive (resp. negative) or null.

Examples

A didactic example

In this initial situation (let us suppose the line above is the edge of the board) :

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	128	0	0	128	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

After three dilations, the algorithm gives :

0	0	0	1	0	0	1	0	0	0
0	0	2	2	2	2	2	2	0	0
0	2	4	6	5	5	6	4	2	0
1	2	6	136	7	7	136	6	2	1
0	2	4	6	5	5	6	4	2	0
0	0	2	2	2	2	2	2	0	0

Then after seven contractions, it gives a small "territory" :

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	128	6	6	128	0	0	0
0	0	0	0	4	4	0	0	0	0
0	0	0	0	0	0	0	0	0	0

A real example

Then, figure 5 is a real example (a Shusaku's position). Black (resp. white) squares represent black (resp. white) territories. They are recognized with 4 dilations and 13 contractions.

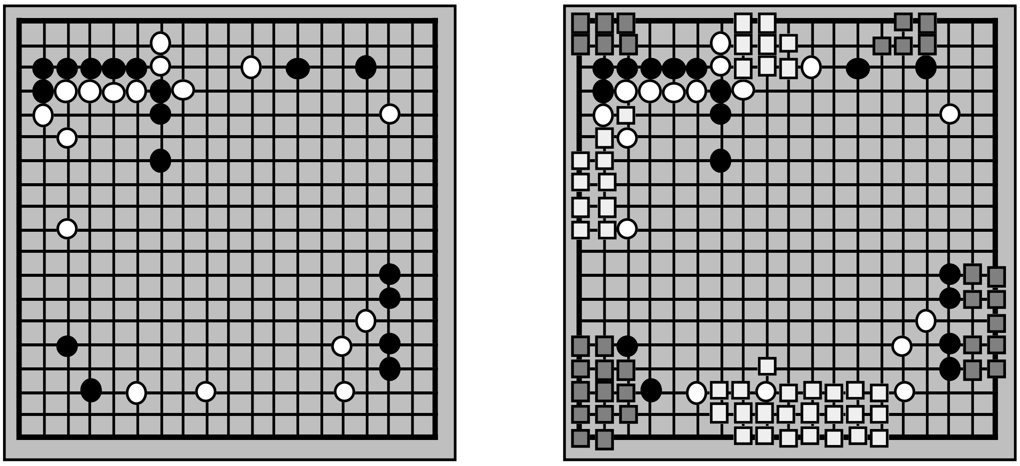

figure 5

This territory evaluation is very good and corresponds almost to what human players perceive.

How INDIGO chooses the global move ?

The aim of the global level is to compute the score of the game and to choose one move.

The score is calculated in chinese rules. INDIGO adds one point per alive stone or territorial intersection.

In its state, INDIGO does the evaluation of a 19x19 board described before in about 10" on a Sun sparcstation10. Then, it is not possible to make lookahead on the global evaluation in a reasonable time. So, INDIGO chooses the global move using static heuristics. Principle followed is to use all objects whose state is *. If an object contains information to play on an intersection, it contributes proportionnally to its size to the notation of this intersection. The notation of an intersection is then the sum of all the contributions of objects in state *.

This heuristic is more or less exact for strings whose game is *. It is fuzzy for groups whose health game is * because sometimes a move does not kill or save the group definitely but only make it stronger or weaker. This heuristic is in part false for territories because one move on a territory does not destroy or expand it in proportion of its size.

How strong is INDIGO ?

Precautions

It is very difficult to evaluate the strength of a Computer Go program for many reasons :

- Two strong players (let us say 1 Dan) get comparable knowledge. But two beginners (let us say 20th) does not necessary. One of them can get some knowledge about territory and almost nothing about group and the other one can get the contrary. Their knowledge are incomparable. But chance can make outcomes of their games almost equal and by the way, their levels can seem equal. So, as the levels decrease, they reflect less truely the knowledge used by the player and they get less meaning. Computer Go programs are 10th Kyu and below and they enter in this category.

- Human players and Computer Go programs get very different skills. Human players are very good to judge a position, knowing what to do instantly. Computer Go programs can be good on fixed goals, as local life and death [Wolf 1992] or endgames situations

[Berlekamp & Wolfe 1994], where they can use their computing power. But they are unable to judge a position as an human player does. Giving a Dan-Kyu level to Computer Go programs should let people thinking that Computer Go programs play as human people do and they do not.

- Dan-Kyu levels are impregned of historical human utilisation. Some Go people think a beginner is 20th Kyu and cannot imagine that a 30th or 40th Kyu level could never exists. Telling them your 3 man-years Go program is 25th Kyu can surprise them!

Those precautions being said and loosely speaking, we can announce that INDIGO is about 19th Kyu. Now, we can see some results, strong points and weak points of INDIGO.

Results

A Computer Go Ladder [Pettersen 1994] has been created on the International Go Server (IGS) last year. It is a very good way to start confrontations between Go programs and contacts between programmers. Programs are ordered on two ladders : a 19x19 ladder and a 9x9 ladder. A new program can enter at the bottom of a ladder challenging the "weakest" program. If it wins against it, it jumps before it. Then, it can challenge the next program, etc. On the 1srt of July, there are seven programs on the 19x19 ladder; Many faces of Go is at the top. INDIGO is 4th between Explorer and Gogol. There are eleven programs on the 9x9 ladder; Many faces of Go is also at the top; INDIGO is 8th between gottaGo and Explorer. Places on the ladder must not be taken ipso facto but must be interpreted with the precautions above. Therefore, one will attach to them the importance he wants.

Strong points and weak points

Strong points

- Reads low level concepts : ladders, eyes, connections, dividers, etc.

- Recognizes territories as well as humans players do, with mathematical morphology tools.

- Evaluates statically a position with an interaction model of groups. INDIGO can recognize big dead groups, still with many liberties, or sekis.

- Plays with incremental data updating. A 19x19 game with 300 moves against itself lasts about 1h40 on a Sun sparcstation 10. That means 20" per move in average. INDIGO can answer instantly when datas do not change much after the last move.

Weak points

- Dislikes growing or reducing territory. INDIGO finishes its games without big territories. Its points are almost dead opponent groups. In the fuseki, this lack of knowledge can be dramatic.

- Iterative and global levels are static. At those levels, INDIGO plays using static heuristics without checking if it reaches its goal at those levels. Moves generated by these levels are sometimes as useful as a pass... These levels are static because of lack of computing power.

- Size of the program makes it difficult to be modified. We spent a lot of time to study the games that INDIGO played, to detect and fix bugs. Time necessary to do such things is an obstacle to increase the program's level.

What will happen to INDIGO next ?

Of course, we want to make the weak points disappear letting the strong points in place. But it is not an easy work. We are happy of the interaction model between groups that allows static evalution of Go positions. We like the way INDIGO recognizes territories using fuzzy mathematical morphology. But we must include expand/reduce territories heuristics in the global level in order to reduce the gap with other programs. That would not be so difficult. More difficult will be to find a method, or at least some tricks, to make the global and/or the iterative levels becoming dynamic. That would increase significantly INDIGO's level. We also wants to clarify the code lines that have been growing in an historical manner for 3 years. To do so we might rewrite INDIGO from scratch. Such objectives are very time-consuming and do not increase our financial situation...

But are you a fanatic Computer Go hacker or not ?

Bibliography

[Berlekamp & Wolfe 1994] - E.R. Berlekamp, D. Wolfe - Mathematical Go Endgames - Nightmares for the Professional Go Player - Ishi Press International - San Jose, London, Tokyo - 1994

[Boon 1989] - M. Boon - Pattern matcher of Goliath - Computer Go 13, winter 89-90

[Boon 1991] - M. Boon - Overzicht van de ontwikkeling van een Go spelend programma - Afstudeer scriptie informatica onder begeleiding van prof. J. Bergstra - 1991

[Bouzy 1995] - B. Bouzy - Modélisation cognitive du joueur de Go - Thèse de doctorat de l'Université Paris 6 - Janvier 1995

[Cazenave 1995] - T. Cazenave - Learning and Problem Solving in Gogol a Go playing program - LAFORIA Internal Rapport - Paris 6 University - March 1995

[Conway 1982] - J. Conway, E.R. Berlekamp, R. Guy - Winnings ways - tome 1 & 2 - Academic press - 1982

[Fotland 1992] - D. Fotland - Many Faces of Go, documentation and playing algorithm - 1992

[Müller 1995] - M. Müller - Computer Go as a Sum of Local Games : An Application of Combinatorial Game Theory - A dissertation submitted to the Swiss Federal Institute of Technology for the degree of Doctor of Technical Sciences - Zürich 1995

[Pettersen 1994] - E. Pettersen - The Computer Go Ladder - World Wide Web page, http://cgl.ucsf.edu/go/ladder.html - 1994

[Serra 1982] - J. Serra - Image Analysis and Mathematical Morphology - Academic Press - London - 1982

[Wolf 1992] - T. Wolf - Tsumego with Risiko - School of Math. Sciences, Queen mary & Westfield College, Mile end road, London E1 4NS, England - 1992