1. Introduction

Ideas about chance in Nature have evolved a lot since the beginning of modern Physics. Laplace thought the Universe was deterministic and *completely predictible*. Experiments were not perfect and oblige to use probabilities [Laplace 1776]. Poincaré was closer to modern Physics. He thought that a small uncertainty on the initial state of some dynamic system was amplified as time goes on, it was impossible to predict evolution of such systems after a long time [Poincaré 1902], even with tools as precise as possible. Theory of deterministic chaos confirms Poincaré's intuition [Ruelle 1995]. If every neighborhood of the initial state of a system is dense in the space of the final states after a given time, then the system is said to be *chaotic*. Chaotic systems are everywhere in Nature : a drop of milk in a cup of coffee, traffic jam, the weather, population dynamics, seisms, fire propagation, etc.

Our work dealt with the Go[1] game that has many chaotic features. During our thesis [Bouzy 1995] on Cognitive Modelling of the human Go player, we developped, in 1992 and 1993, a Go playing program : INDIGO[2] . It is ranked on the international Computer Go Ladder [Petersen 1994]. We have already shown that fuzzy sets are useful to model "territories" and "influences" in Go [Bouzy 1995c]. In this paper, we want to shed light on links between Computer Go and Uncertainty Management with fuzzy methods.

The nature of this paper is prospective. It suggests an Uncertainty Management in INDIGO with numeric methods and fuzzy functions. It will screen the chaotic behaviour of Go positions and could then increase the INDIGO's level in the next release.

To do so, part 2 shows related work in Computer Go and introduces the INDIGO program. Part 3 explains main weaknesses of INDIGO and states the problems to be solved. Part 4 shows that some fuzzy methods can be applied to quiet Go positions and that chaotic positions are very difficult to handle without lookahead. Before conclusion, part 5 explains that fuzzy functions screen chaos of unstable positions and therefore are useful to the global move decision process.

---

[1]Go is a two-person, zero-sum, complete information game that is famous in China, Korea and Japan.
[2]Is Now Designed In Good Objects.

2. Related work in Computer Go

After recalling the theoretic complexity of the game of Go (§2.1), this part shows that Go is too complex to be computed as Chess and Othello are (§2.2). Then, it evaluates the complexity of current Go algorithms (§2.3). At the end, it shows what a sub-game is (§2.4).

2.1. Theoretic complexity of the whole game

Papers have shown that most common board games Chess [Fraenkel & Lichtenstein 1981], Checkers [Robson 1982] and Go [Robson 1982b] generalized to NxN boards are theoretically of equal complexity. What has been shown is that, in each case, the problem of deciding whether Black (say) can force a win from a given position is *exponential in time*.

2.2. A priori complexity of the whole game

[Allis 1994] defines combinatorial complexities of games as follows : E is the "space state complexity" defined as the number of possible positions. And A is the "whole game tree complexity". Considering the average length of actual games L and average branching factor B, we get $A = B^L$. Literature on games and Human (H) versus Computer (C) results give the following table :

	Othello	Chess	Go[1]
E	$3^{64} \approx 10^{30}$	10^{43}	$3^{361} \approx 10^{172}$
A	10^{58}	$35^{80} \approx 10^{67}$	$200^{250} \approx 10^{575}$
Hum. v.Comput.	H<C	H≈C	H>>C

These games get increasing complexity and seem to be correlated with Human versus Computer results. They show that "brute force" [Knuth 1975] succeeds in Othello and Chess but not in Go. Therefore, Go is one of the most exciting challenge for AI [Bradley 1979].

Of course, this is an approximation because one could argue that 7x7 Go is as complex as Othello. This is wrong : 7x7 Computer Go results are still very weak on the human scale. It seems that *the complexity of Go is less due to the size of the board but more to some intrinsic properties of Go positions*.

2.3. Reducing the complexity with sub-games approach

As the human player is much stronger than the computer in Go, programmers get many ideas to program Go by observing Go players. Go players do not embrace all moves in

[1] In average, a 19x19 game lasts 250 moves and there are 200 possible moves. Of course, this is wrong in the beginning and at the end of the game but true in average.

each position. They cut the positions in sub-positions and they treat them independently. Criterias to cut the position into sub-positions are based on matching of objects. In Go, "groups" and "territories" are the main types of objects.

Most of Go programs use this approach. Let us evaluate their actual strength. The best Go program is Goliath [Boon 1991] but the recent FOST Cup has been won by Handtalk. Many faces of Go [Fotland 1992] is on the top of the Computer Go Ladder [Petersen 1994]. Other good programs are Go Intellect [Chen 1990] and Star Of Poland [Kraszek 1988]. Our Go playing program is INDIGO [Bouzy 1995b]. As of November 1st 1995, it is ranked 4th upon 7 on the 19x19 Computer Go Ladder and 8th upon 11 on the 9x9 one.

2.4. The static states of sub-games

The whole game is a composition of sub-games. A sub-game is a composition of two opposite goals that are bound to an object. Capturing/saving a string of stones (the string sub-game), connecting/cutting a group of stones (the group sub-game) , expanding/reducing a territory (the territory subgame) are typical sub-games of the whole game.

A game gets three static states : "won", "other" and "lost". Players make moves until the "won" state or the "lost" state is reached. Figure *static* represents transitions between static states of a game.

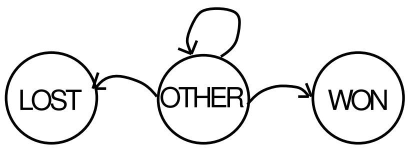


Figure static

Transitions are not possible from "lost" and "won" states because a game which is "lost" or "won" is finished.

3. The problems in INDIGO

INDIGO has many sub-games to manage. Some of them are simple and they are managed using lookahead. This way, it is possible to know their issue completely, according to who moves first [Conway 1982]. Other ones are less simple and they cannot be managed using complete lookahead. This part shows the problems that arise in these problematic sub-games (§3.1). Then, it shows the problem that arises at the global level (§3.2) because of the incompleteness of the results.

3.1. Problematic sub-games

Intention

Let us call the "perception time" of a sub-game, the time the computer uses to update data in order to select a move. The move is then played at "action time". Selection of a move c is implicitly made in order to leave the state "other" and reach the state "won". But at perception time, INDIGO does not effectively play c. Therefore, at perception time INDIGO does not know if the selected move will reach the state it was selected for. Using an anthropomorphic style, INDIGO has the "intention" to reach the state "won" but it is not sure it will actually reach it. Figure *intention* shows the "intention" of the system with the point arrow.

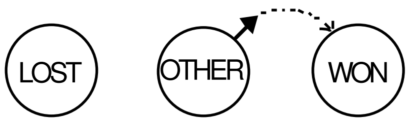


figure intention

Effectiveness of a move

After action time, when the selected move has been played effectively on the board, it is then possible to see if the state "won" is reached, as figure *effectiveness* shows it.

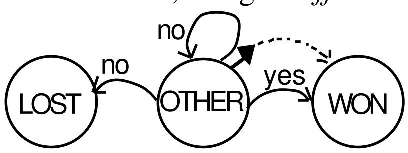


figure effectiveness

So, in problematic sub-games, the problem is *reducing the gap between intention and effectiveness.*

3.2. Problems of the global level

Instead of the three symbolic states ("won", "lost" and "other"), the global game has numeric states giving the score. Figure *global* shows this fact using the same notation as figure *static*.

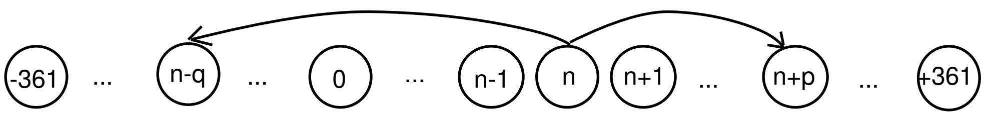


figure global

One player can make the score jump from n to n+p and the other one from n to n-q. If a move increases the score by p points, its value is p.

At the global level, there are two problems. One problem is *reducing the gap between intention and effectiveness* again. For example, the intention of a selected move can be a p points jump, going from state n toward state n+p, but the effective move can make the state n-q to be reached instead... The nature of this problem is similar to the nature of the problematic sub-games and we suppose that solutions will be also similar. The other problem is *managing the uncertainty that is created by problematic sub-games*.

4. Solutions to the problematic sub-games

This part shows a partial solution in closing the "perception-action loop"(§4.1) and in splitting the "other" state into numeric sub-states (§4.2). Then it shows how chaotic behaviour in Go still remains a problem in fighting positions (§4.3).

4.1. Closing the perception-action loop

The first idea to reduce the gap is to close the following loop : build static description, select the move, play the move, build new static description, etc. In Robotics or Distributed Artificial Intelligence terminology, this loop is called the "perception-action" loop. Main idea in Robotics is to close the "perception-action" loop [Dudek & Jenkin 1995]. The idea is to make the system know the results of its own actions according to its own perception (or representation). The previous figure *effectiveness* already showed this fact.

The problem that arises with such an approach is computing power. If the new description shows that the move does not reach the goal it was selected for, what must INDIGO do ? It cannot play this bad move. So, it must select and play another move. This way, it makes lookahead at depth one. So, this approach is possible if lookahead does not consume too much time.

4.2. Decomposition of the "other" state into numeric sub-states

When closing the perception-action loop, another problem arises. Many moves correspond to other→other transition. They are eliminated because they are ineffective according to INDIGO. With such a method, INDIGO passes[1] in many positions. It is still sound but very passive.

To be more active, one must split the state "other" into many numeric states as shown on figure *split*.

[1]A player passes when he does not make a move on the board.

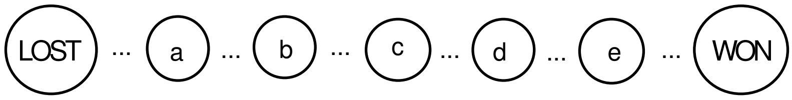


figure split

Many transitions are then mpossible. Figure *many* shows an example of a set of transitions, assuming the sub-game is in the sub-state c.

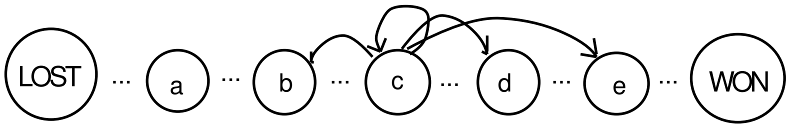
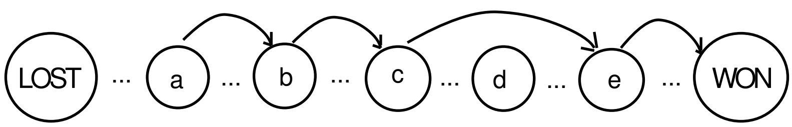


figure many

Many of these transitions are negative, such as the c→b transition, another one is ineffective such as c→c, and other ones are positive such as c→d and c→e. Without such decomposition of the "other" state, INDIGO would have considered that all the moves would be ineffective and would have eliminated all of them.

Conversely, with such decomposition INDIGO can choose the "best" move. On the previous figure, it corresponds to the c→e transition. Then, INDIGO can reach the "won" state from any sub-state, using several moves if it is necessary. Figure *sequence* shows an example of an effective sequence of moves from the sub-state a toward the "won" state.



sequence

Then, the problem is to get a *measure* in order to split the "other" state in a sound way. [Bouzy 1995c] shows how fuzzy sets model "territories" and "influence". Each group has a numeric value that depends on the size of the influence and on the territories around it. This value is computed with the multiplication of the territory value with the influence value of the group. Coefficients are adjusted so that the value of the state is "lost" (resp. "won") if both territory and influence values are below (resp. above) a threshold. On figure *smooth*, a player secures one group in four moves. The progression toward the goal is effective.

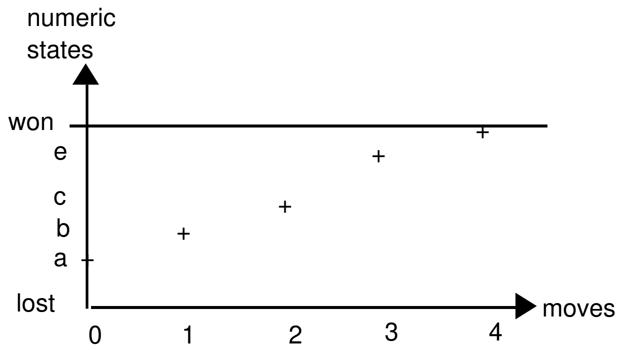


figure smooth

To conclude this paragraph, splitting the "other" state into numeric sub-states leads to the presence of many effective numeric moves instead of a few "symbolic" ones. This refinement will make INDIGO play moves that it discarded until now.

4.3. Chaotic behaviour arises from spatial interactions and from efficiency of moves

Spatial interactions and chaotic behaviour

In fact, the previous method does not apply to fighting positions where states of groups depend more on (group→group) interactions than on (group→empty-space) interactions. In this case, groups are very near to each other and do not have much empty space around them. Figure *theoretic* shows two examples of what would happen theoretically if spatial interaction between groups did not exist.

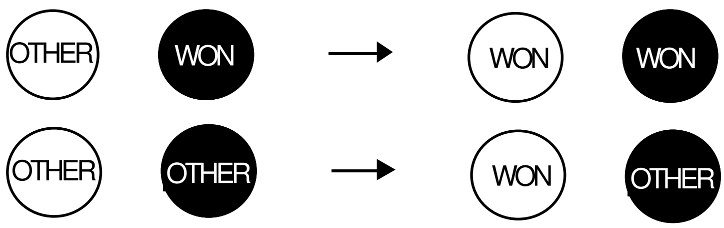


figure theoretic

When White makes a move in order to apply an other → won transition to its white group, the black group is not affected by the transition of its white neighbour.

But actually, spatial interactions exist [Bouzy 1995d]. Figure *actual* shows that the state of a group can change when one of its neighbours undergoes a transition.

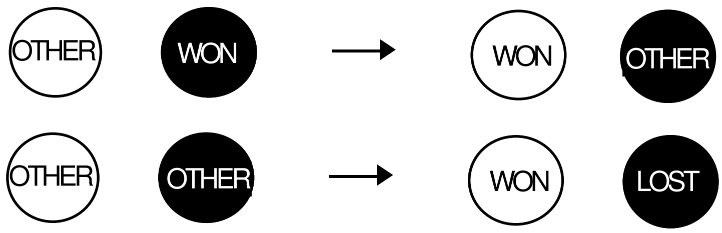


figure actual

On this example, the state of black group is affected by a move that first affects the white group.

This simple rule can propagate spatially through groups that are, in state "other" (say) before applying the rule, and in state "won" (say) after it. These spatial interactions give place to unpredictible behavior that is a feature of chaos [Ruelle 1995].

Rentability of moves, risk and chaotic behaviour

In order to make its own moves more efficient, the Go player focuses his mind on killing opponent groups (making them reach the "lost" state) and saving his own ones (making them reach the "won" state). But when you want to kill a group (say the prey), the opponent groups (say the predators) must take some risk. You must weaken the prey (the numeric state of the prey decreases towards the "lost" state) and while doing it, the opponent can weaken some of your predators. Except if your opponent is much weaker than you, he can weaken one of your predators at least. Then, when predators weaken a prey, one of the predator is weak also. As the prey's numeric state decreases towards the "lost" state, the predator's numeric state decreases towards the "lost" state also. When the "lost" state is almost reached for the prey and the predator, one of them kills the other one and eats it (then its state is "won"). But here is the point : the predator can kill the prey, but the converse case is possible. So, as the state of a group is nearer and nearer from the "lost" state, the state of the group can become "won" suddenly.

This sudden jump is very annoying and there is no other solution than making lookahead as far as possible. If the system does not get computing time, lookahead must stop and the results that are given on the global level include some uncertainty.

5. Solutions to the global level

This part shows a static description that screens the chaotic behavior of the previous level (§5.1). Then it shows that the second problem can be solved with a similar solution to the iterative level problem (§5.2).

5.1. Static description of the global level

We saw that spatial interactions and risk both give chaotic behavior. The results from the problematic sub-games include uncertainty.

We define the fuzzy state [Zadeh 1992] of a numeric state n as the set of the numeric states that can follow n. We define the function H from the set N of the numeric states to a set F of fuzzy states with H(n) = [n-p(n), n+q(n)].

Figure *hide* shows graphically a definition of H.

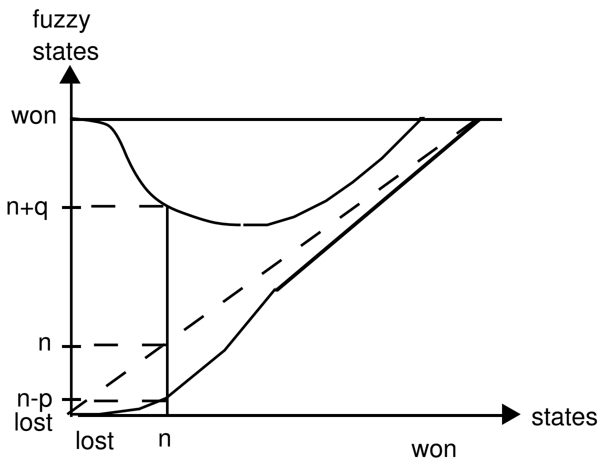


figure hide

When the numeric state is nearer and nearer to "lost" state, its fuzzy state is an interval that becomes larger and larger. Fuzzy state of the "lost" state is the whole ["lost", "won"] interval. Every neighborhood of points of N that are very near from the "lost" point, gets an image that is almost dense in H(N). That is the feature of chaos [Ruelle 1995].

The H function assumes that when a sub-game is in a state n, the state of the sub-game after the next move cannot be out of the interval H(n). This way, the system can manage uncertainty that is generated by a sub-game.

Let us suppose that a small sub-game s has the "small" value v(s) (the value of a sub-game is the size of the group that is involved in the sub-game) and that another sub-game b has the "big" value v(b). We suppose that v(s) << v(b). Let us suppose that the sub-game s (resp. b) is in state n(s) (resp. n(b)) with H(n(s)) ~ ["lost", "won"] (resp. H(n(b)) ~ [n(b), "won"]). We mean that the sub-game s (resp. b) is in the very left (resp. right) of the figure *hide*.

Without the use of H, the system chose to play the big value b in any positions. On some position, it would be right but on other ones, it would not.

With the use of H, the system can gain :

g(b) ~ v(b).("won"-n(b)) if it plays in b

and it can gain :

g(s) ~ v(s).("won"-"lost") if it plays in s.

Therefore, the system will not play b each time. That is correct. The system will play where uncertainty exists. It is a good behaviour unless uncertainty persists for several moves.

One could think that the fuzzy states with a low and a high value can be linked with the pessimistic and an optimisitic value of the B* algorithm [Berliner 1979] that is used in Chess. But our method with fuzzy states does not make uncertainty decreasing as B* does.

Quiescence Search [Beal 1989] can also be linked with our method : the system plays on sub-games where uncertainty is big, i.e. on unstable local positions as Quiescence Search does. In Go, lookahead with quiescence search does not work as well as in Chess because the computing cost of playing one move is higher in Go than in Chess.

5.2. Uncertainty of the global level itself

As it is done in the problematic sub-games, one can close the perception-action loop in order to correlate intention and effectiveness of moves. This point has been discussed in the previous paragraph and everything applies also for the global level.

Then, the value of a global move is fuzzy even if the moves are effectively played.

6. Conclusion

We saw that INDIGO gets uncertainty in "problematic" sub-games and at the global level also. We suggested solutions to manage this uncertainty.

First, we suggest to reduce uncertainty in closing the perception action loop of the problematic sub-games. But, in order to keep most of the other→other transitions, we split the state "other" in many numeric sub-states. This technique is sound on all the quiet Go positions where increasing influence or territory means something. On these positions, it is possible to give moves a precise numeric value.

But this technique is useless in fighting positions where interactions between groups are strong and where risk is present. If interactions between groups are not managed, chaotic behavior arises when the state of a group is nearer and nearer to state "lost". To manage this chaotic behavior, the system has not other solution than lookahead. When lookahead fails because of lack of computing power, the value of a move is then encapsulated into a fuzzy number [Zadeh 1992].

This way the system will play in sub-games where uncertainty is big. It will be better than the present solution. But this is only an assumption. We must now verify our theory by implementing it on the computer. We cannot say if it will be true yet because Go programming is... chaotic also. Going from an idea to its implementation and practical results follows an unpredictible path!

7. Bibliography

[Allis 1994] - Allis L.V. - Searching for Solutions in Games and Artificial Intelligence - Ph.Thesis - Vrije Universitat Amsterdam - Maastricht - Septembre 1994

[Beal 1989] - Beal D.F. - A Generalized Quiescence Search Algorithm - Artificial Intelligence, vol. 43, n°1, pp. 85-98 - 1989

[Berliner 1979] - Berliner H.J. - The B* Tree search algorithm : the best-first proof procedure - Artificial Intelligence, vol. 12, pp. 23-40 - 1979

[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 du joueur de Go - Thèse de doctorat d'informatique de l'université Paris 6 - Janvier 1995

[Bouzy 1995b] - Bouzy B. - The INDIGO program - Proceedings of the 2nd Game Programming Workshop in Japan - pp. 191-200 - Kanagawa 1995

[Bouzy 1995c] - B. Bouzy - Les Ensembles Flous au jeu de Go - Rencontres francophones sur la Logique Floue et ses Applications - LFA95 - Paris - 27-28 Novembre 1995

[Bouzy 1995d] - B. Bouzy - Un modèle basé sur les interactions au jeu de Go - Rapport interne du LAFORIA n°95/27 - Novembre 1995

[Bradley 1979] - Bradley M.B. - The game of Go, the ultimate programming challenge ? - Creative computing, 5, 3, pp. 89-99 - March 1979

[Chen 1990] - Chen K. - Move decision process of Go intellect - Computer Go 14, pp. 9-18 - Spring 1990

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

[Dudek & Jenkin 1995] - Principles of Perception and Action in Mobile Robotics - Tutorial of the 14th IJCAI - Montréal - 1995

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

[Fraenkel & Lichtenstein 1981] - Fraenkel A.S., Lichtenstein S. - Computing a perfect strategy for n by n Chess requires time exponential - Journal of Combinatorial Theory, Serie A, Vol. 31, N°2, September 1981, pp. 199-214

[Knuth 1975] - Knuth D.E., Moore R.W. - An analysis of alpha-beta pruning - Artificial Intelligence, vol. 6, n° 4, pp. 293-326 - 1975

[Kraszek 1988] - Kraszek J. - Heuristics in the life and death algorithm - Computer Go n°9, pp. 13-24 - Winter 1988

[Laplace 1776] - Essai philosophique sur les probabilités - Paris - 1776

[Petersen 1994] - Petersen E. - The Computer Go Ladder - http://cgl.ucsf.edu/go/ladder.html

[Poincaré 1903] - Science et méthode - Paris - 1903

[Robson 1982] - Robson J.M. - N by N Checkers is exptime complete, TR-CS-82-12, Australian National Unviversity Department of Computer Science - 1982

[Robson 1982b] - Robson J.M. - The Complexity of Go, TR-CS-82-14, Australian National Unviversity Department of Computer Science - 1982

[Ruelle 1995] - Ruelle D. - Le Chaos - Paris - 1995

[Zadeh 1992] - Zadeh L. - Fuzzy Logic for the Management of Uncertainty - New York - Wiley - 1992