The Move Decision Strategy of Indigo

Bruno Bouzy[1]

[1] C.R.I.P.5, UFR de mathématiques et d'informatique, Université Paris 5,
45, rue des Saints-Pères 75270 Paris Cedex 06 France
bouzy@math-info.univ-paris5.fr
http://www. math-info.univ-paris5.fr/~bouzy

Abstract:

This paper describes the move decision strategy of Indigo. By using the example of Indigo, the paper shows that the move decision process of a Go program can be very different from the processes used in other games with lower complexity than the complexity of Go, even if the basic modules are conventional (move generator, evaluation function and tree search). Indigo uses them in a specific way, adapted to computer Go, which may be of interest for researchers on other mind games as complex as Go. The evaluation function can be "quick", "slow" or "strategic". It may or may not include local tree search. The move generation brings about different kinds of moves : "urgent" moves, "life and death" moves and "calm" moves. Urgent moves are statically qualified with a global urgency. A two-player quiescence search verifies that the urgent move does not decrease the position evaluation. Calm moves are used within two-player selective global search at a very low depth. Besides, Indigo also uses single-agent search to refine the strategic importance of goals. Lastly, Indigo chooses either the calm move, the life and death move or the urgent move to be the global move.

keywords:
computer go, quick & slow evaluation, strategic evaluation, urgent move selection, single-agent & two-agent search.

1. Introduction

Recently, [Chen 2001a, 2001b] have discussed move decision strategies used by current Go programs, on the basis of answers to questions sent to their authors. It showed that they could be roughly divided into four categories: "static analysis", "try and evaluate", "global selective search", and "incentive/temperature approximation". It mainly showed that move decision strategies used by current Go programs are very different from those used in other games with lower complexity than Go. Figure 1 shows the classical model used in computer games [Campbell & al. 2002], [Buro 2002], [Schaeffer 2001], mainly a move generator and an evaluation function used by a tree search.

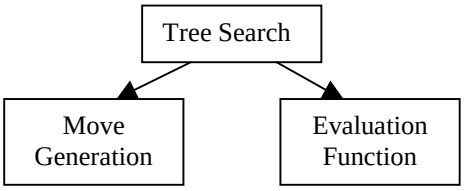


Figure 1. The classical model

Following the general trend provided by [Chen 1990, 2000] or [Mueller 2002], our paper provides the example of the move decision strategy of Indigo [Bouzy 1995a, 1995b, 1999, 2002a]. Although [Chen 2001a, 2001b] classifies Indigo into the "global selective search" category, (which was in keeping with our answer), this paper will show that Indigo could also have been classified into the "static analysis" category or into the "incentive/temperature approximation" category as well. Figure 2 provides the synthetic view of the different

modules of Indigo presented in this paper. Modules correspond to rectangles and the arrows indicate the utility relationships between two modules.

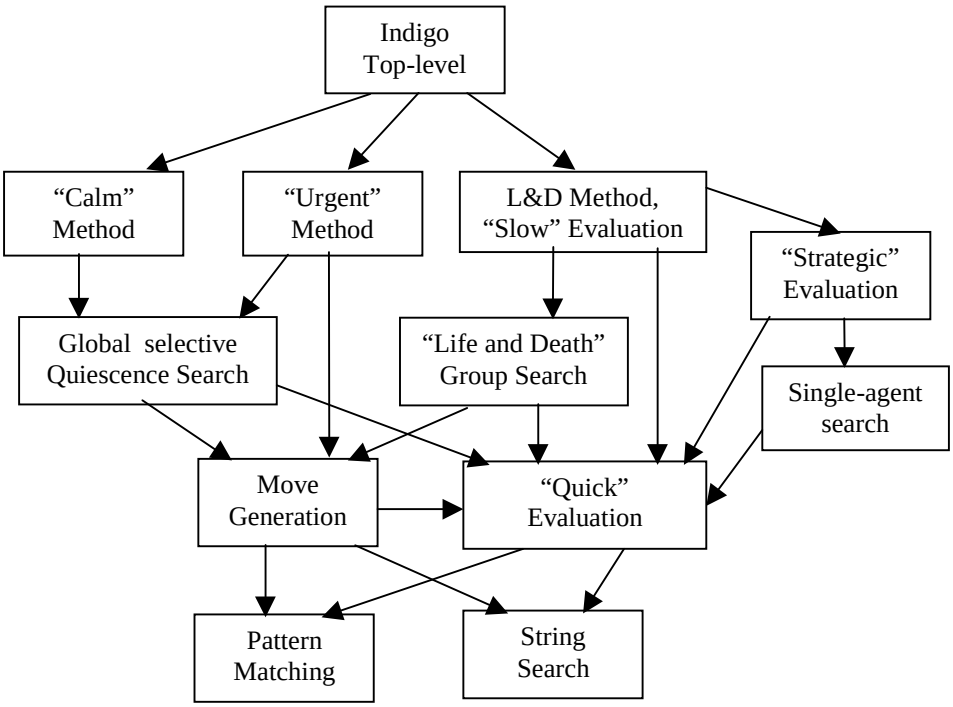


Figure 2. Indigo main modules and their utility relationships

Of course, Figure 2 remains much more complex than Figure 1. Let us link the two figures, starting with the 3 basic modules of Figure 1. First, the Evaluation Function (EF) of Figure 1 corresponds to the "quick" EF of Figure 2. Second, the move generation module (MG) remains the same. Third, the tree search (TS) of Figure 1 is split into three search modules of Figure 2: the quiescence search, the Life and Death (L&D) search and the string search modules. Section 2 provides a Figure-1-oriented view of the basic modules used by Indigo. They are necessary to understand the next sections.

Conversely, several new modules appear on Figure 2. The main new module is the Indigo top-level. It uses different move selectors: the "urgent" method, the "calm" method and the "life and death" method. Each move selector chooses one move with an "incentive" and the top-level then decides which move to play by assessing the different "incentives" and keeping the best one. The pattern matching and the string search modules are drawn in Figure 2 because they are basic modules. However, they will not be described in more details because they do not directly interact with the move decision strategy. The strategic evaluation and the single-agent-search modules are tools included in the strategic method, discussed in section 3. The "slow" evaluation function, that uses tree search on life and death is highlighted in section 4. The "urgent" method of Figure 2 selects urgent moves and checks their validity through a selective quiescence search. The urgent method is the most important element of Indigo architecture, and will be described in section 5. The "calm" method and the selective quiescence search are presented in section 6. Section 7 describes the "life and death" method of Indigo. Before conclusion, section 8 provides experimental results showing the relative strength of each module within the global architecture of Indigo shown by Figure 2.

2. Basic modules

This section recalls the classical aspect of Indigo: an evaluation function and several move generators that hide most of the domain-dependent knowledge and an alpha-beta tree search engine.

2.1 "Quick" Evaluation Function

There are two evaluation functions in Indigo: the "quick" one and the "slow" one. The quick/slow alternative results from a difficult optimization task of the whole program over the past years. In the earliest versions of

Indigo, there was one evaluation function. Once optimizations were performed, this evaluation function became the quick evaluation function. Besides, there was a lot of time left for other processing tasks within the whole program. Consequently, we designed the "slow" evaluation function (described in section 4).

The quick evaluation function provides *an approximation of the set of groups and territories*. There are many ways to model groups and territories. The aim of our paper is not to discuss the way to do it. For information, Indigo mainly uses mathematical morphology [Serra 1982] to model groups and territories, but it also uses domain-dependent heuristics and patterns. The quick evaluation function is like a black box hiding the domain-dependent heuristics and providing the groups, with their *status ("alive", "undetermined" or "dead")*, the territories, and *the score* of a given position. In this paper, we shall also use the ("won", "undetermined" or "lost") terminology when the context is game-oriented. This evaluation corresponds to the cognitive model of our thesis [Bouzy 1995a, 1995b] and to the evaluation function description of [Bouzy & Cazenave 2001] which gives details of the model used by the quick evaluation function. To give an order of magnitude, the quick evaluation function of a 19x19 board is processed in 50 milliseconds on a 450Mhz computer. This time remains quite stable. It depends on the size of the board and on the number of dead groups detected (one pass per dead group detected). It gives correct results on positions that does not require accurate reading.

2.2 Move Generation

In Go, the move generation is closely linked to the evaluation of a position. Unlike in other games in which move generation only uses the rules of the game, move generation in most Go programs selects plausible moves only. It is performed simultaneously with the quick evaluation of the position. The plausible moves are selected according to many domain-dependent considerations: string capture tree search, local shapes, dividers, connectors, eyes, simple life and death shapes, territory growing/reducing. In Indigo, most of this knowledge is represented with three hundred patterns. In the following sections, move generation is considered as a basic module. For a clear understanding of this paper, it is necessary to point out two important properties of a move generated by Indigo: the *urgency* property and the *ordering/disordering* property. The urgency of a move generated by a pattern is computed by looking up the urgency information contained in a pattern and multiply it by the size of "undetermined" groups that intervene in the pattern. The urgency contained in patterns will be discussed in section 5. The urgency of a move generated by string tactics depends on the size of the string and other information discussed in section 5. The ordering/disordering property of a move is a true/false value that is used by the quiescence search of Indigo, described in the calm move selection method in section 6.

2.3 Tree Search

There are different kinds of tree searches in Indigo: the *string capture tree search*, the *global tree search*, and the *life and death tree search*. The simplest one is the string capture tree search. It can read if a string with 1 or 2 liberties can be captured. The algorithm is alpha-beta with transposition table. The global tree search is a very selective alpha-beta quiescence search using the quick evaluation function at leaf nodes. The quiescence is defined in a very restrictive way by using the urgency and the ordering/disordering property of moves. It is used by the calm and urgent method. It will be dealt with in section 6. The life and death tree search is an alpha-beta search using a three-value evaluation function ("alive", "undetermined", "dead") based on the quick evaluation at internal and leaf nodes. This tree search will be discussed in section 4.

3. Strategic method

The strategic method in Indigo consists in a strategic evaluation and a single-agent search. Indigo spends most of its time examining the life and death status of groups (see section 4) with intensive use of tree search. To reduce this time, the aim of the strategic method is to assign an *importance* to each group in order to concentrate its life and death computations on important groups and not on unimportant ones.

3.1 Strategic evaluation

The strategic evaluation is performed to guess the strategic importance of goals such as killing or making life of "undetermined" groups. For each "undetermined" group, two quick evaluations are performed, the first one,

'E_{dead}', assumes that the group is dead and the second one, 'E_{alive}', assumes that the group is alive. Thus, each "undetermined" group is endowed with two values, 'E_{alive}' and 'E_{dead}', noted $\{E_{alive}|E_{dead}\}$. The value $|\ E_{alive} - E_{dead}\ |$ is a good domain-dependent heuristic to approximate the importance of a group. Moreover, the strategic evaluation costs little time because it is a simple loop performing two quick evaluations for each "undetermined" group which are scarce.

3.2 Single-agent search

[Schaeffer & al. 2001] offered a unifying view of single-agent search and two-player search. More specifically, we believe that single-agent search is worth being used in computer Go within a strategic background. In [Bouzy 1999] we presented some heuristics to label each group with two numbers, K and L. K was the number of moves which are played by the same player and are necessary to kill the group. Furthermore, L was the number of moves which are played by the same player and are necessary to make a group live. Our paper drew a comparison with other terms which arose practically in the computer Go community, the possible omission number [Tajima & Sanechika 1998] and Cazenave's taxonomy [Cazenave 1996], and with the combinatorial game theory [Conway & al. 1982]. K and L help to determine the strategic importance of an "undetermined" group. K and L are useful because the quick evaluation function is far from being perfect. When K=1, the opponent of the group can kill it in one move and the game is 'IP' in Cazenave's taxonomy. When L=1, the group can live in one move and the game is 'GI'. In these two cases, it is relevant to start a look-ahead to determine whether the group is already dead or alive. When K=2, the opponent has a threat to kill the group. When L=2, the group has a threat to live. Therefore, when K or L=2, a look-ahead can also be relevant. The more K and L grow, the less relevant the two-agent look-ahead becomes. In conclusion, two-agent life and death search is not performed when K and L are strictly greater than 2. Giving an accurate status to groups when K and L are high remains an open question. Very recently, [Cazenave 2002] outlined the relevance of searches in which one player may play several moves in a row in order to find admissible moves, which constitutes an appealing perspective to enhance our current strategic method.

4. "Slow" evaluation function

This section explains the aim of the slow evaluation function, the nine states of a group and their reliability.

4.1 Aim

When a group is detected to be dead by one pass of the quick EF, the opponent groups which kill the dead group aggregate around it, to give a new big group [Bouzy 1995], [Bouzy & Cazenave 2002], and the data structure of the current position completely changes. Then the next pass of the quick EF starts with these new data as input. Consequently, the position evaluation changes completely when a death occurs. To avoid totally wrong position evaluations, the "quick" evaluation function should *never* conclude on the death of a group that is not actually dead. A similar remark can be made about life of group. However, when a life is detected, no aggregation happens. The data structure of the position does not change, and its effect on the final evaluation is less dramatic than a death. Therefore, the quick EF is not accurate on all non-terminal positions. The downside is that a lot of groups stay "undetermined" after its processing. L&D tree searches performed on each group with "undetermined" status enable the program *to give a more precise evaluation of the position*, the "slow" evaluation. As far as we know, the use of local tree searches to build the evaluation itself is specific to computer go. At least, it is very different from other mind games in which the tree search simply uses the evaluation function like a black box containing the domain-dependent heuristics.

4.2 Nine states

For each "undetermined" group, two two-player look-aheads are performed along the L&D of the group. The first look-ahead is carried out with the friendly color playing first and the second one with the opponent color playing first. This idea is inherited from [Conway & al. 1982]. The status of a group computed by the quick evaluation function being "won" (W), "undetermined" (?) or "lost" (L), the outcome of these two tree-searches can theoretically be either {W|W}, {W|?}, {W|L}, {?|W}, {?|?}, {?|L}, {L|W}, {L|?} or {L|L}. Due to limited

computing power, '?' means that the look-ahead cannot terminate with a L or W conclusion. This idea was first expressed by [Cazenave 1996].

4.3 Reliable values

The {L|L} value is the most useful and reliable result in practice. When the two tree searches conclude on the L value, the group is dead whoever plays first. The points occupied by the group are assigned to the opponent and not to the friendly player. Like the "quick" evaluation for dead groups [Bouzy 1995], the "slow" evaluation builds a new group by aggregating the opponent groups of the dead group into one big group. There is no certainty that the new group is alive after such aggregation. In most cases, the new group is alive but in some cases, it can be still undetermined or even dead.

The {W|W} value is another reliable result which indicates that the group is alive whoever played first. The points occupied by the group are counted +1 by the "slow" evaluation function for the friendly player (they were counted 0 by the "quick" evaluation because the group was "undetermined").

The {W|L} value is a reliable value which indicates that the first player will kill or make the group live. In such a case, the contribution of the group to the "slow" evaluation function is zero (the average value of the contribution of the group if it lives and the contribution of the group if it dies). Thus, the "slow" evaluation contains an uncertainty $|E_{alive}-E_{dead}|$. This is analogous to the uncertainty of the "quick" evaluation [Bouzy 1996].

The {L|W} value is one feature of Go, the seki: the first player obtains the converse of what he wants, therefore he does not play at all, and the group can be considered as alive. Its contribution to the evaluation corresponds to its size.

4.4 Unreliable values

When considering computing power as a precious resource, {?|?} is the bad case: the program has spent a lot of its thinking time to conclude that the group is undetermined, which was already known through the quick evaluation function. The contribution of the group to the "slow" evaluation is zero and no move is stored for the L&D method (described in section 7). The evaluation contains uncertainty $|E_{alive}-E_{dead}|$. {W|?} and {?|L} are complicated cases. The left search of {W|?} and the right search of {?|L} obtain certain results but the other two searches cannot reach their end. It is possible to consider these cases either as similar to the {?|?} case or similar to the {W|L} case. The first question is to specify the contribution of such groups to the slow EF: either zero or a strictly positive fraction of the size of the group. The second one is the possible storage of the moves leading to certain conclusions for a possible use by the L&D method (see section 7). To simplify, we decided to fix the contributions of such groups to zero. And we decided to store the moves corresponding to computations containing no uncertainty (the left part of {W|?} and the right part of {?|L}) for the L&D method. {?|W} and {L|?} are not considered because no player wants to move in such games.

4.5 Performing life and death tree search

GoTools solves L&D problems on completely closed areas [Wolf 1996, 2000]. But in actual games, the tree search must be performed on open areas. Therefore, the group can grow and the number of possible moves increases quickly, which make the search explode as mentioned by [Wolf 2000]. Another difficulty lies in the fact that the group can be split into several parts during the search. Each part can be dead, alive or undetermined and these cases are complicated. In the current version, Indigo simplifies this problem by considering the two clear goals only: the whole group is either alive or dead. The other cases are considered as "undetermined" by our program. In order to circumvent this difficulty, it could possible to use the "quick" evaluation function score instead of the three-value (alive, undetermined, dead) evaluation function but this approach misses the initial aim: knowing the status of a group. We have not explored this research direction yet. In conclusion, the "slow" evaluation is much more accurate than the "quick" evaluation but the price to pay for such accuracy is very high in terms of computing resource.

5. “Urgent” method

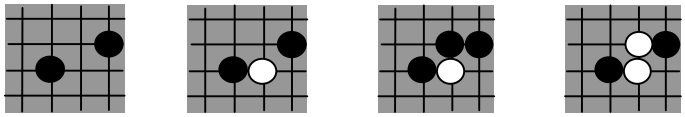
This section deals with the urgency of moves. The urgent method is the most relevant and important method in Indigo. Subsection 5.1 presents the founding principle of the urgent method. Subsection 5.2 describes the limits to this principle. Subsection 5.3 highlights the domain-dependent considerations: the splitting notion and 8-connection. Subsection 5.4 underlines the importance of the stability of the goal focus during middle-game. Then, subsection 5.5 provides the hints used by Indigo to assess urgency of moves, particularly the “doubling” strategy. Subsection 5.6 deals with minor remarks and subsection 5.7 concludes on the formula used to compute move urgency.

5.1 Founding principle

The aim of the “urgent” method is to select the move with the highest urgency. Let us assume we get an urgency function with static domain dependent heuristics, for example pattern-based heuristics. Moreover, we assume that the urgency function plays well on most positions but not optimally. The weakness of static heuristics lies in the lack of verification of the move effect [Bouzy 1996]. Therefore, to avoid playing bad moves with high urgency, a verification must be done. Indigo takes the move with the highest urgency and performs a very selective quiescence search (see section 6). This search uses the quick EF to check that the selected move does not decrease the position evaluation. This verification avoids a lot of blunders. The move with the highest urgency that is verified is called the best “urgent” move. It is important to notice that the move which has the best mini-max score for the “quick” EF is called the “calm” move” (see section 6), and that the “urgent” move is not equal to the “calm” move in most cases.

5.2 Limits

The reason for this difference lies in the fact that, for some classes of urgent moves, it is easier to set the move urgencies with simple domain-dependent heuristics than model the corresponding EF. For example, it is easy to teach a program that playing moves such as B->C or B->D in Figure 3 are worth playing. But it is hard to find the adequate corresponding EF which could be used by a tree search to verify the validity of moves: such is the problem we are faced with in developing Indigo. [Bouzy 2002b] addressed this problem.



pattern:	A	B={C|D}	C	D
state:	won	{won|lost}	won	lost
urgency:	0	high	0	0

Figure 3. 8-connection goal: examples of patterns and urgencies.

In summary, for some classes of moves, there are no corresponding EF, and Indigo has to use another method than TS to select the best urgent move. The following subsections specify the move urgency.

5.3 Middle-game, splitting and 8-connection

Indigo is more group-oriented than territory-oriented and more interested in middle-game than in any other phases of the game. Consequently, we focus on the group aspect of Indigo during the middle-game. As a Go player, we believe that strategy in the middle-game is mainly based on the dividing/splitting notion. For the time being, this observation is entirely personal and to be taken as an assumption within our computer Go project.

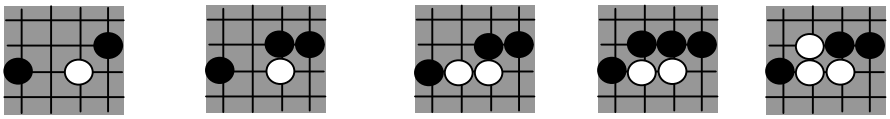
In such a background, Indigo contains patterns advising the program how to split or divide the opponent into parts and to link its own stones. We suppose that splitting the opponent into parts is adequately modeled with 8-

connection[1]. Figure 3 shows four patterns A, B, C and D, with regard to the 8-connection. In all these patterns, the black 8-connection is considered and the 8-connection enables the black player to split the upper part of the pattern from the lower part of the pattern. Pattern C shows three black stones that are 8-connected. Pattern A is a pattern in which the opponent cannot avoid the 8-linkage between the friendly stones. This is proved by pattern B showing a plausible threat. Pattern D shows the lost 8-connection if the opponent plays on pattern B.

With regard to 8-connection, pattern C is "won" and has an urgency which equals 0 by definition of 8-connection. Pattern A is won as well because the opponent cannot avoid the 8-connection. Pattern D is "lost" and urgency is 0 because the 8-connection is no longer possible. But, on the contrary, pattern B has a non-zero urgency. Moreover, the following paragraph explains why this urgency is "high".

5.4 Goal focus stability

The pattern urgency follows the principle that if the friendly player has played once to divide/split the opponent, he will play again later with greater probability, when the opponent threatens the division. Therefore, the pattern urgency of a descendant of another pattern is higher than the other pattern. Figure 4 shows the urgency of pattern E and G which are "quite high" and "very high".



pattern:	E	F	G={H|I}	H	I
state	{won|lost}	won	{won|lost}	won	lost
urgency:	quite high	0	very high	0	0

Figure 4. Examples of two depending urgencies.

Pattern H is won by definition of 8-connection. Pattern I is lost. Pattern F is won by a proof using G and H. Patterns E and G are unstable, their state is {won|lost}. They have a high urgency. But pattern G has an urgency higher than pattern E, which we are going to demonstrate.

5.5 The doubling strategy

Let us assume that pattern E arises on the board. Black plays and reaches pattern F. This fact uncovers the interest of Black in reaching the goal of 8-linking its stones. Later, when White plays using pattern G, Black may not answer. Black takes the risk of seeing White play using pattern I to make the 8-linkage lost. Black having previously shown his interest in the 8-linkage, it is almost mandatory for him to play using pattern H. Having an urgency of pattern G significantly higher than the urgency of pattern E increases the probability of playing G->H, when E->F has been played before. In practice, we set the pattern urgency of G at the double of the pattern urgency of E.

More generally, Indigo also applies this principle to other kinds of goals. If Indigo has reached a goal by playing a move with a given urgency, and if the opponent tries to retrieve the goal, the urgency of the move enabling the program to keep the goal reached has an urgency which is doubled. This way, Indigo has a behavior which mimic human players. When no other important goal arises in the position, it tries to maintain the goal reached. We call this strategy, the "doubling" strategy.

In actual games, Indigo may want either to keep some reached goals or to reach new ones. Without the use of the doubling strategy, Indigo has an erratic behavior, reaching some goals, then leaving them and reaching new ones with a great probability. Thanks to the doubling strategy, Indigo preferably chooses to keep the goals reached before trying to reach new ones. Of course, if a new, sufficiently important goal, occurs when the program is playing a sequence of moves to keep a given goal reached, then it may choose the move to reach the new important goal. For example, on pattern E, if the white moves which try to disconnect also threaten another important connection, then the program has either to defend this other connection or the current one. The choice

[1] 8-connection means that an intersection has eight neighbors.

between the two connections will be made on the basis of the move urgency (see formula 1 in subsection 5.7). The doubling strategy does not forbid any change of goal focus during a game but it makes the current focus more stable.

5.6 Additional remarks

When a goal is completely reached, the opponent has no threat move left to avoid the achievement of the goal. For example, a completely reached goal is the actual capture of string, when the stones are removed from the board. The goal of dividing the opponent is completely reached when the friendly stones are completely alive and 8-connected. Moreover, the goal of connecting the friendly stones is completely reached when the friendly stones are completely alive and 4-connected.

The doubling strategy could not be uncovered while we were modeling the urgency of the string capture goal. Actually, on the string capture goal, as long as the threat-to-escape/kill pair of moves is played, the size of the string keeps increasing. Consequently, the urgency defined with the size of strings increases in the same way as the string capture goal urgency and the urgent method does not need the doubling strategy. While we were considering non-increasing-with-size urgency goal like the 8-connection goal, we uncovered the "doubling" mechanism.

5.7 Move urgency

Now that pattern urgency is explained, Indigo defines the urgency of a move m generated by a pattern p as the pattern-urgency multiplied by the size of set of intersections i belonging to an undetermined group g intervening in the pattern p, as explained by formula 1.

$$\text{urgency}(m, p) = \text{patternUrgency}(p).\text{size}(\{i | i \in g \text{ and } g \cap p \neq \emptyset\}) \qquad (\mathbf{1})$$

The doubling strategy is powerful enough to hide most of the noise resulting from group size variations between positions.

6. "Calm" method and restrictive quiescence search

The "calm" method plays every "calm" move generated by the move generation module and performs a very selective quiescence search [Beal 1989] at a low depth. The evaluation function that is called at leaf nodes is the quick EF. The move with the best incentive is called the best "calm" move. When compared to the other methods, the calm method is the easy part of the program.

The quiescence search is very selective. First, the moves are generated locally. Second, the moves are only advised by the string capture games and the connection games, and not by L&D games. Third, only very urgent moves are generated, that is moves with a pattern urgency greater than a given value. Moreover, the quiescence itself is defined in a very restrictive way. *Ordering* moves are moves enabling transition {W|?} -> W, {?|L} -> L and *disordering* ones: {W|?} -> ?, {?|L} -> ?. After a given depth p, the restrictive quiescence definition *forbids disordering moves*. Of course, on the one hand, the downside is that the search is not fair because a player may play more moves than the opponent if it has more ordering moves. But, on the other hand, the advantage cannot be overseen as the restrictive quiescence search terminates the search, by finding positions containing less uncertainty, more quickly. We tried to use a quiescence search in a less restrictive way by allowing disordering moves. This approach gave very bad results (see section 8): the search may not terminate, the evaluated positions contain great uncertainty, and are not significant. The maximal depth of the quiescence search is set to 15, but the average depth of the quiescence search is about 3. The average branching factor is situated at a low level, between 2 and 3. The restrictive quiescence used by the "urgent" method is identical.

7. “Life and death” method

This method is part of the move decision process and uses the “slow” evaluation.

7.1 Description

{L|L} and {W|W} states are reliable results which notify that the group is stable whoever plays first. In these cases, the move decision process deletes the L&D moves of the group. {W|L} is a reliable value which signals that the first player will kill or make the group live. In the case of {W|L}, the moves obtained by these two tree searches are very crucial and they must be used with a high priority by the top-level process. Because, the opponent cannot kill, the {L|W} state is considered as alive and the moves are also deleted. {?|?} is the bad case: no reliable conclusion has been reached. The move decision process uses the “urgent method” instead (see section 5). When considering the reliability of the obtained result and the consequent action to perform, {W|?} and {?|L} are very bad cases. {?|L} for example: if you are the friendly player, you cannot use the result because the selected move leads to uncertainty, but you know that doing nothing leads your opponent to kill you, therefore, you have to choose the move by the “urgent” method instead. If you are the opponent player, you know that playing will kill the group but you may waste a move in doing so, because the group can also be dead if the friendly player plays first. {?|W} and {L|?} correspond to seki positions which are too complex for one of the two L&D searches: one of them concludes on a negative result with certainty whereas the other one does not terminate at all. For these two reasons, these cases are not taken into account.

7.2 Limitations

During actual games, the program behavior using the slow evaluation may be erratic. First, a friendly “alive” group, determined by the quick evaluation function, may go directly to the {L|L} status in one move without passing through the {W|L} status, because of the lack of precision of the quick evaluation function. In fact, splitting the global game into several life and death sub-games a priori deletes the interaction between sub-games. In Go, where interactions between groups are of huge importance, the splitting of the whole game into several group sub-games may be awkward. When considering life and death test positions only, Indigo using the slow evaluation function is better than Indigo without it (see section 8). When matching the two programs in complete games, Indigo without the slow evaluation plays paradoxically better than Indigo with it. When a group is {L|L}, there are still many interaction moves to play, for instance the moves to retrieve an advantage from the death of the group (“aji” in Japanese). These moves have no effect on the {L|L} status but have some effect on the quick evaluation. The same can be said for {W|W} groups: many opponent moves may decrease the evaluation function without changing the group status. Besides, the L&D method consumes a lot of CPU time when compared to other methods.

8. Experimental results

We now need experimental evidence to assess the relative importance of each module, or strategy, presented so far. The most important question is to evaluate the urgent and calm methods (subsection 8.1). The experiment of subsection 8.2 shows whether the doubling strategy is worthwhile or not. The next two experiments highlight the relevance of the quiescence search, and its restrictive derivation (subsection 8.3). Then, subsection 8.4 yields the results of the L&D method, with its speed enhancements brought by the single-agent search and the strategic evaluation. Subsection 8.5 sums up all the experiment results.

IndigoBase is constituted of the quick EF, the move generation, the pattern-matching and the string search modules. For each experiment, we set up a test of N games between two programs derived from IndigoBase. For example, ICalm is the program which includes IBase and the calm method, and IUrgent is the program which includes IBase and the urgent method.

During the test, one given program plays Black for one half of the games, and White for the other half. Each program uses a random generator to modify the games. The standard deviation of 19x19 self-play score equals sixty points. One hundred games at least were necessary to decrease the standard deviation of the average score

down to five points. On a Pentium 450 Mhz, one game lasts about five minutes on average. We set N=100 and the result of the test amounts to a percentage of games won. Because 2 or 3% are not significant for this test, the results are rounded at 5%.

8.1 Urgent and calm methods

This subsection describes the assessment of the urgent method and the calm method. In addition to the ICalm and IUrgent programs defined above, ICalmUrgent is the program which includes both methods with a top level.

	result
ICalm versus IUrgent	55-45
ICalmUrgent versus IUrgent	95-5
ICalmUrgent versus ICalm	90-10

Table 1. Urgent and calm method results

Table 1 shows that IUrgent and ICalm levels are almost equal. The qualitative features of ICalm and IUrgent plays are worth commenting. On the one hand, ICalm plays "good" calm moves, but it does not keep the initiative. The calm move incentive smoothly follows the global incentive of the game. There is no stage in the game at which ICalm plays very bad moves. It always plays average moves. On the other hand, IUrgent plays "good" urgent moves on fighting positions. But, it plays very bad moves on positions with low urgency because low urgency patterns are not well-tuned. When no urgent move can be selected, it plays at random on uncontrolled intersections. Because ICalm and IUrgent have very different qualitative property, it is interesting to consider the result of a program integrating the two methods.

Table 1 shows that IUrgentCalm is significantly stronger than ICalm and IUrgent. IUrgentCalm *still plays* good calm moves on quiet positions, and high urgency moves on fighting positions. Furthermore, IUrgentCalm *does not* play bad urgent moves on positions with low urgency, or random moves on uncontrolled intersections, because these moves are always hidden by the calm moves of the calm method. Moreover, the urgent moves are often moves which keep the initiative. Therefore, IUrgentCalm does not have the main weakness of ICalm. In summary, IUrgentCalm keeps the upsides, but not the downsides, of both IUrgent and ICalm. ICalmUrgent is, by far, the best of the three programs. The handicap stone difference between ICalmUrgent and the other two programs is four stones.

8.2 Pattern urgency: constant versus doubling strategy

This subsection assesses the importance of the doubling strategy (see section 5). Therefore, IConstant is the program ICalmUrgent with all pattern urgencies set to a constant but tuned value. I2Strategy equals ICalmUrgent of the previous subsection, for each class of patterns, all pattern urgencies respecting the doubling strategy.

	result
I2Strategy versus IConstant	90-10

Table 2. Whether or not to use the doubling strategy?

Table 2 yields the comparative results of the two programs. IConstant and I2Strategy behave similarly on calm positions. But on fighting positions, IConstant switches from one goal to the other too frequently while I2Strategy follows its goal until achievement, which is rewarding on long games such as 19x19 games. I2Strategy outperforms IConstant on the fighting positions and generally wins the game. Being too secure, I2Strategy may add unnecessary moves.

8.3 Quiescence Search

To evaluate the quiescence search of section 6, we set up a test between one program, called IQuiesce, using the quiescence search of section 6 and another one, called IProf1, using the quick EF directly. IQuiesce corresponds

to ICalmUrgent of subsection 8.1. The name IProf1 was chosen because the urgent and calm methods of IProf1 carry out depth-one search.

	result
IQuiesce versus IProf1	65-35

Table 3. Whether or not to use the quiescence search?

Table 3 shows the superiority of IQuiesce over IProf1. IProf1 and IQuiesce have the same qualitative behavior, and the difference is only quantitative. IProf1 plays twice as quickly than IQuiesce. IQuiesce is one stone stronger than IProf1.

The next interesting point is to consider whether the restrictive notion of quiescence defined in section 6 is valuable or not. IRestrictiveQuiesce(p) is IQuiesce in which the quiescence search enables the program to play disordering moves until depth p and not after. IQuiesce = IRestrictiveQuiesce(0). p=0, 2, 4, 8.

p	0	time per game
0	50-50	5’
2	45-55	7’
4	60-40	12’
8	65-35	35’

Table 4. IQuiesce against IRestrictiveQuiesce(p)

Table 4 shows that performance of IRestrictiveQuiesce(p) decrease with p. In addition, the time spent increases with p. The bad performance can be explained a posteriori by the fact that the disordering moves make the position contain greater uncertainty than the root position. Consequently, we kept p=0.

8.4 Life and death method

To assess the L&D method, ISlow(d) is the ICalmUrgent program associated to the “L&D method and slow EF” and “L&D Group Search” modules (see Figure 2), in which d is the maximal depth of L&Dsearch (d=0,2, 4, 6, 8). We stored L&D problems selected from actual Indigo games into a test set of about 50 positions. Table 5 shows the number of problems solved by ISlow(d). ISlow(0) corresponds to ICalmUrgent.

d	L&D problems solved
0	35/50
2	37/50
4	38/50
6	41/50
8	45/50

Table 5. Life and death problems solved by ISlow(d)

Thus, ISlow(8) solves ten more problems than ICalmUrgent. We also assess ISlow against ICalmUrgent on actual games. For each depth d, table 6 shows the result of the one-hundred-game match between ISlow(d) and ICalmUrgent. The duration of one game with, or without, the strategic method is also provided.

d	result	without Strat	with Strat
0	50-50	5’	5’
2	50-50	20’	10’
4	50-50	50’	20’
6	55-45	120’	40’

Table 6. ISlow(d) against ICalmUrgent

In fact, the quick EF requires fifty milliseconds on a 450Mhz computer and most searches reach the maximal depth without finding certain conclusions. Thus, ISlow(d) plays approximately at the same level as ICalmUrgent

and spends much more time than ICalmUrgent. Breaking down the global problem into L&D sub-problems creates a bias which reduces the benefit of solving each sub-problem separately. When performing L&D searches on open positions, the sub-problem breakdown made before the searches may become invalid after some moves, and the L&D searches become irrelevant. A group may be proven to be alive by the L&D search, but this result may bring about the death of another group. The lesson to be drawn might be to maintain all the computations as global as possible.

Table 6 shows that the strategic method greatly increases the speed of the L&D method. For each group, the strategic evaluation helps to bound the global evaluation. Therefore, knowing the calm incentive and the urgent incentive of the top-level, the L&D method may not study groups which would surely yield an inferior incentive, thus enabling Islow to save irrelevant computations. Finally, the fact that one should wait for a maximal depth d = 6 to observe a small improvement in terms of won games, combined with the slowness of Islow(6), leads us to select ICalmUrgent as our "best" release.

8.5 Summary

The previous experiments provide evidence of the three following positive points.

- The association of the urgent and calm methods gives good results, each method hiding the downsides of the other without decreasing its upsides. This enhancement allowed us to observe a four handicap stone improvement.

- With no adequate EF, the doubling strategy can be viewed as a very good hint. This hint allowed us to observe a four handicap stone improvement.

- When compared to depth-one search, quiescence search enhances the level of play while the forbidding disordering move heuristic is powerful to reach positions containing few uncertainty more quickly. The corresponding improvement is more than one handicap stone.

Although it solves more L&D problems encountered in actual games, the program using the slow EF and the L&D method does not enable Indigo to improve on full games. In Figure 2, the most useful path is the one used by the urgent method which produces 70% of moves. The calm method provides 25% of moves played by Indigo and the L&D method supplies the remaining 5%.

In its ICalmUrgent version, Indigo took part in the 21st Century Cup [Intelligent Go 2002] in Edmonton, last July. It finished 10th out of 14 participants. Besides, it has regularly participated to the computer go ladder [Pettersen 2002] (Indigo is ranked 6th out of 16 programs on the 9x9 ladder and 8th out of 16 programs on the 19x19 ladder).

9. Conclusion

This paper has described the move decision strategy of Indigo and has shown that the move decision process of a Go program can be very different from the processes used in other games with lower complexity than Go. Although Indigo uses three classical modules to perform its global move choice, it uses them in a specific way, adapted to computer Go, which is of interest for researchers on other mind games as complex as Go.

When removing the L&D method from the Indigo top-level, the program play at the same level in actual games but it solves fewer L&D test problems. As far as we know, the use of local tree searches to build the step-by-step evaluation of a position is specific to computer Go. Furthermore, it is very different from other mind games in which tree search simply uses the evaluation function like a black box containing the domain-dependent heuristics. The quick evaluation module forms the basic module of Indigo, whose strong and weak points are reflected in the program behavior.

The urgent move selection process is very characteristic of computer Go, or more generally, of games with high complexity. First, the complexity lies in the difficulty in modeling a correct evaluation function. For some classes of urgent moves like splitting the position into parts, it is easier to evaluate the moves with simple

domain-dependent heuristics like 8-connection rather than model the corresponding evaluation function used by a tree search. Second, the complexity of Go is obviously combinatorial, which forbids some tree searches on current computers. Therefore, we can offer three explanations for the relevance of the "urgent" move selection mechanism for some aspects of the game: impossibility of building a sound position evaluation function, simplicity of the static move evaluation model and avoidance of tree search.

Our urgent method description contains two hints that are of importance to make Indigo efficient:

- the *doubling strategy* to assign an urgency to patterns of the same family,
- and the *ordering/disordering* property of moves to drive the quiescence search within a multi-game approach. To our knowledge, these hints are new ones within the computer Go community. At least, no publication has, so far, highlighted such hints, although some commercial programs might have been using them for a long time.

Moreover, calm moves are generated to perform another selective global search at a very low depth. The move with the best incentive of the evaluation function is called the best calm move. This bears a resemblance with tree search classically used in computer games.

Furthermore, we believe that the strategic evaluation and the single-agent search are also features inherent in complex games. In Indigo, the groups are qualified with strategic properties such as what-is-the-evaluation-if-I-am-alive (respectively dead) and number-of-moves-to-play-in-a-row-to-be-alive (respectively dead). This information helps the L&D method to focus on strategic goals and to save computations.

To refine the discussion initiated in [Chen 2001a, 2001b], we are now able to classify the "urgent" method into the "static analysis" category, the "calm" method into the "global selective search" category while the L&D method and the top-level of Indigo falls into the "incentive/temperature approximation" category.

10. References

[Beal 1989] D. Beal, "A generalised quiescence search algorithm", Artificial Intelligence, 43 (1), (1989), pp. 85-98.

[Bouzy 1995a], B. Bouzy, "Modélisation cognitive du joueur de Go", (in French), Ph.D. thesis, University Paris 6, 1995.

[Bouzy 1995b], B. Bouzy, "The Indigo program", 2th Game Programming Workshop in Japan, GPW'95, pp. 197-206, Hakone, September 1995.

[Bouzy 1996], B. Bouzy, "There are no winning moves except the last", Proceedings IPMU 96, pp. 197-202, Grenade, July 1996.

[Bouzy 1999], B. Bouzy, "Complex Games in Practice", 5th Game Programming Workshop in Japan, GPW'99, pp. 53-60, Hakone, October 1999.

[Bouzy 2002a], B. Bouzy, http://www.math-info.univ-paris5.fr/~bouzy/INDIGO.html, August 2002.

[Bouzy 2002b], B. Bouzy, "A small Go board Study of metric and dimensional Evaluation Functions", Computer and Games 2002, Edmonton, to be published.

[Bouzy & Cazenave 2001], B. Bouzy, T. Cazenave, "Computer Go : an AI oriented Survey", Artificial Intelligence Journal, Vol 132/1, pp. 39-103, 2001.

[Buro 2002], M. Buro, "Improving mini-max search by supervised learning", Artificial Intelligence Journal, Vol 134, pp. 85-99, 2002.

[Campbell & al. 2002], M. Campbell, A.Joseph Hoane Jr, Feng-hsiung Hsu, "Deep Blue", Artificial Intelligence Journal, Vol 134, pp. 57-83, 2002.

[Cazenave 1996], T. Cazenave, “Système d’Apprentissage par Auto-Observation. Application au Jeu de Go”, (in French), Ph.D. thesis, University Paris 6, 1996, http://www.ai.univ-paris8.fr/~cazenave

[Cazenave 2002], T. Cazenave, “Admissible moves in two-player games”, in: Proceedings of SARA 2002, Edmonton, Canada.

[Chen 1990], K. Chen, “The move decision process of Go Intellect”, Computer Go, 14, (1990), pp. 9-17.

[Chen 2000], K. Chen, “Some practical techniques for global search in Go”, ICGA Journal, 23 (2) , (2000), pp.67-74.

[Chen 2001a], K. Chen, “Computer Go: knowledge, search, and move decision”, ICGA Journal, 24 (4) , (2001), pp.203-215.

[Chen 2001b], K. Chen, “Search and Knowledge”, ”, 6th Game Programming Workshop in Japan, GPW’2001, Hakone, October 2001.

[Conway & al. 82], J. H. Conway, E. R. Berlekamp, R. K. Guy, “Winning Ways”, Academic Press, 1982.

[Intelligent Go 2002], Intelligent Go Foundation, http://www.intelligentgo.org/, November 2002.

[Mueller 2002], M. Mueller, “Computer Go”, Artificial Intelligence Journal, Vol 134, pp. 145-179, 2002.

[Pettersen 2002], E. Pettersen, http://www.cgl.ucsf.edu/go/ladder.html, November 2002.

[Schaeffer 2001] J. Schaeffer, “Technology transfer from one high-performance search engine to another”, ICGA Journal, 24 (3) , (2001), pp. 131-146.

[Schaeffer & al. 2001] J. Schaeffer, A. Plaat, A. Junghanns, “Unifying single-agent and two-player search”, Information Sciences, 135 , (2001), pp. 151-175.

[Serra 1982], J. Serra, “Image analysis and mathematical morphology”, Academic Press, London, 1982.

[Tajima & Sanechika 1998], M. Tajima, N. Sanechika: “Estimating the Possible Omission Number for Groups in Go by the Number of n-th Dame”, Computers and Games, Lecture Notes in Computing Science, 1558, pp. 265-281, Springer (1999).

[Wolf 1996] T. Wolf, “About problems in generalizing a tsumego program to open positions”, in: Proceedings of the third Game Programming Workshop in Japan, Hakone, 1996, pp. 20-26.

[Wolf 2000] Thomas Wolf, “Forward pruning and other heuristic search techniques in tsume go”, Information Sciences, 122, (2000), pp. 59-76.