A small Go board Study of metric and dimensional Evaluation Functions

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. The difficulty to write successful 19x19 go programs lies not only in the combinatorial complexity of go but also in the complexity of designing a good evaluation function containing a lot of knowledge. Leaving these obstacles aside, this paper defines very-little-knowledge evaluation functions used by programs playing on very small boards. The evaluation functions are based on two mathematical tools, distance and dimension, and not on domain-dependent knowledge. After a qualitative assessment of each evaluation function, we built several programs playing on 4x4 boards by using tree search associated with these evaluation functions. We set up an experiment to select the best programs and identify the relevant features of these evaluation functions. Thanks to the results obtained by these very-little-knowledge-based programs, we can foresee the usefulness of each evaluation function.

1 Introduction

19x19 computer go is difficult because of tree search explosion, but also because of the complexity of the evaluation function [Chen 2001]. For several years, we have been developing a go playing program, and we have accumulated experience on two aspects of 19x19 computer go, the go model and the programming techniques.

With regard to the programming techniques, a go playing program may contain tactical look-ahead, pattern-matching, evaluation function and highly selective global search. In this paper we simplify this technical aspect by reducing the size of the board. Thus, we leave tactical look-ahead, pattern-matching and selectivity and we keep the evaluation function and tree search without selectivity.

Besides, a go model may contain knowledge about “strings”, “liberties”, “groups”, “eyes”, “connections”, “territories”, “life” and “death” and other useful concepts embedded in an evaluation function. But, in this paper we chose not to take this large domain-dependent knowledge into account and so as to explore the dimensional and metric features of evaluation functions only. We call “dimensional”, a model in which the dimension of objects play an important role and we call “metric”, a model in which the distance between objects plays an important role.

Thus, the aim of this paper is to study *dimensional and metric go evaluation functions* with a system which uses *tree search* on *small boards*. We call GGG the

system that we developed to test the “metric” and “dimensional” ideas. GGG is short for “Gic-Gac-Goe”, to emphasize that the game is as simple as Tic-Tac-Toe and is played on small boards.

Section 2 of this paper highlights the motivation behind the “dimensional” evaluation functions and their definitions. Section 3 defines evaluation functions based on the distance notion. Section 4 shows examples and contains a qualitative assessment of the evaluation functions. Section 5 describes the practical experiment that we set up to assess the evaluation functions. Before the conclusion, section 6 underlines the results of the experiment. To shorten this presentation, EF stands for Evaluation Function.

2 “Dimensional” evaluation functions

This section shows the motivation of the dimensional feature and defines dimensional EF.

2.1 the motivation

The classical Go evaluation function E corresponds to the sum of the abstract color of each intersection i of the go board I:

$$E = \sum_{i \in I} \text{abstractColor}(i) \quad (1)$$

The function abstractColor(i) returns +1 (respectively -1) if the intersection i is *controlled* by Black (respectively White) and returns 0 if the intersection is not controlled at all. The *control* notion can be more or less complex. [Bouzy 2001] used an EF with a simplified control. [Bouzy & Cazenave 2001] described EFs with complex control including life and death knowledge and morphological operators. In our study, we want to keep the abstractColor function as simple as possible. We do not insert life and death knowledge or morphological knowledge into the abstractColor function. The abstractColor(i) function returned +1 (respectively -1) if the intersection i was either occupied by Black (respectively White) or empty but surrounded by Black (respectively White) intersections only, otherwise returns 0. Other ways of writing the (1) formula are possible. The notion of “group” in go being fundamental, we may render this notion by writing:

$$E = \sum_{g \in G} \sum_{i \in g} \text{abstractColor}(i) \quad (2)$$

G is the set of the groups situated on the board, whatever the definition of a group might be. Furthermore, the abstractColor function returns the same value for each intersection belonging to the same group. Thus we can define the abstractColor(g) of a group g as the constant value of the abstractColor(i) in which i is an intersection of g.

$$E = \sum_{g \in G} size(g).abstractColor(g) \quad (3)$$

Furthermore, we can define G_B as the subset of G whose groups g_b are black, in other words $abstractColor(g_b) = +1$ and we can define G_W as the subset of G whose groups g_w are white, in other words $abstractColor(g_w) = -1$. Then we can write formula (3) as follows:

$$E = E_B - E_W \quad (4)$$

$$E_B = \sum_{g \in GB} size(g) \quad (5)$$

$$E_W = \sum_{g \in GW} size(g) \quad (6)$$

The size of objects being a basic feature when studying dimensionality of objects, we should bear in mind that the dimensionality of a set S is defined on the basis of measures $M_{d, r}(S)$ such as:

$$M_{d, r}(S) = \sum_{g \in G} size(g)^d \quad (7)$$

d is a dimensional parameter and G is a set of balls g of radius r whose union covers S.

$$S \subseteq \bigcup_{g \in G} g \quad (8)$$

These formulas are applied to sets of continuous space to determine their fractal dimension [Mandelbrot 1982]. When the radius r of the ball falls to zero, $M_{d, r}(S)$ reaches a value $M_d(S)$. It is proved that $M_d(S)$ equals either 0 or $+\infty$. The fractal dimension δ is defined as the unique value for which

$$M_d(S) = 0 \text{ for } d>\delta \text{ and } M_d(S) = +\infty \text{ for } d<\delta \quad (9)$$

Of course, a go board is not continuous but discrete and finite. Thus, decreasing the radius of ball to zero is nonsense. Nevertheless, we are not aiming at finding any fractal dimension of any object but we may wonder whether the measures defined by (7) provide useful information to a Go program or not. This is the “dimensional” motivation of our paper.

2.2 the dimensional EF

With d integer, we can now define the EF E_d by the following formula:

$$E_d = \sum_{g \in Gb} size(g)^d - \sum_{g \in Gw} size(g)^d \quad (10)$$

In our study, we assume that $d \in [0, 2]$. E_1 is the classical EF useful for the endgame. E_0 is the count of black groups minus white groups. E_2 measures the ability of one color to get large groups of this color and small groups of the other color.

3 "Metric" evaluation functions

This section defines a "metric" EF. As the connection and the distance notions are important in go, we may define simple evaluation functions by using simple distance functions like in [Van Rijswijk 2000] or in [Enzenberger 1996].

Formula 11 is a simple way to define an evaluation function for color c. When combined with formula 4, formula 11 leads to the definition of the "metric" evaluation function. The minus sign stands for increasing the evaluation function of color c when the distance of two intersections is low. The player of color c wants to minimize the distance of color c between the intersections.

$$E_c = -\sum_{i,j\in I} d(c, i, j) \quad (11)$$

d(c, i, j) is a distance function between two intersections i and j depending on color c. It is defined by formula 12 with a usual distance d, c can be equal to 'Black', 'White' or 'Empty', otherC(Black) equals 'White', otherC(White) equals 'Black' and otherColor(Empty) equals 'Empty', c(i) is short for abstractColor(i).

$$d(c,i,j) = +\infty \text{ if } c(i)=\text{otherC}(c) \text{ or } c(j)=\text{otherC}(c) \quad (12)$$

$$\text{otherwise, } d(c,i,j) = 0 \text{ if } i, j \in S \text{ connex set and } c(S)=c$$

$$\text{otherwise, } d(c,i,j) = 1 \text{ if } d(i,j)=1$$

$$\text{otherwise, } d(c,i,j) = \text{Min}_{c(k)!=\text{otherC}(c)} \{d(c,i,k)+d(c,k,j)\}$$

The first line of formula 12 means that two intersections are situated at an infinite distance for color c if one of them is of the opposite color of c. Otherwise, the second line shows that two intersections belonging to the same connected set S of color c are situated at distance 0 for color c. Otherwise, the third line indicates that two distance one intersections for the classical distance are also at distance one for the colored distance. Finally, colored distance d is defined by the fourth line for the other cases.

Formula 12 provides an almost correct definition of a distance. First, when applied on intersections x and y, d of color c is not a distance because $d(c,x,y)=0 \Rightarrow x=y$ is false. But when aggregating the elements of a connected set into one element, then d is respectful of $d(c,x,y)=0 \Rightarrow x=y$. Second, $d(c,x,y)=d(c,y,x)$ is true. Third, $d(c,x,y)\leq d(c,x,z)+d(c,z,y)$ is true if we mention that $+\infty+\infty=+\infty$.

4 Qualitative assessment

This section provides a set of examples of evaluations and several remarks showing that each of these evaluations corresponds to some meaningful concept of go and recalling the possible downsides of each one.

4.1 Example evaluations

This subsection gives several examples of position evaluations. But first, we need to distinguish between “open” from “closed” boards. Figure 1 gives examples of such boards.

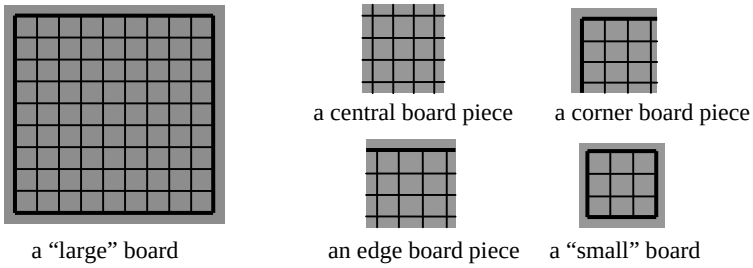


Figure 1. The “open” boards and the “closed” boards. Go is always played on “closed” board, for example the 10x10 board on the left or the 4x4 board on the right. But, when studying a local position of a large board it is easier to define board pieces. A board piece contains edges that are either “open” or “closed”. A closed edge of a board piece corresponds to an actual edge of the initial board. It is drawn with a thick line. An open edge of a board piece is “open” toward other parts of the initial board. It is drawn as if the initial board was cut along this edge. An intersection of an open edge has an unknown number of liberties that depends on the hidden part of the initial board. A board that contains at least one open edge is defined as open, and closed otherwise.

Secondly, figures 2, 3, 4 and tables 1, 2, 3 show evaluations E_1, E_2, E_{d4} and E_{d8} of some example positions.

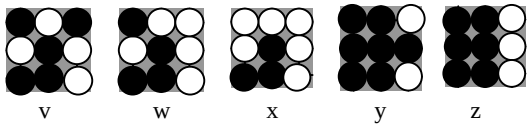
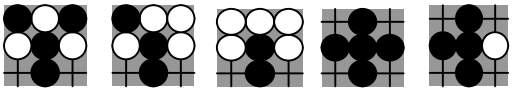


Figure 2. Five terminal positions. The boards are open.

Table 1. The evaluation of the terminal positions of figure 2. +∞ value is set to 1024.

	v	w	x	y	z
E_0	0	0	0	-1	0
E_1	+1	-1	-3	+5	+3
E_2	+5	-7	-27	+47	+27
E_{d4}	+5120	-7168	-27648	+48128	+27648
E_{d8}	+9216	-9216	-27648	+48128	+27648

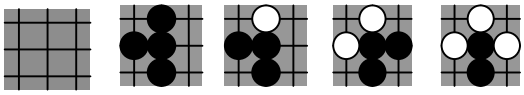


a b c d e

Figure 3. non terminal positions. The boards are open.

Table 2. The evaluation of the non terminal positions of figure 3. +∞ value is set to 1024.

	a	b	c	d	e
E_0	0	0	0	+1	0
E_1	+1	-1	-3	+5	+3
E_2	+3	-5	-21	+25	+15
E_{d4}	+9208	-3076	-33780	+78784	+54224
E_{d8}	+11260	-11260	-33780	+78784	+54224



f g h i j

Figure 4. Other positions. The boards remain open.

Table 3. The evaluation of the terminal positions of figure 2. +∞ value is still set to 1024.

	f	g	h	i	j
E_0	0	+1	0	-1	-2
E_1	0	+4	+2	+1	-1
E_2	0	+16	+8	+7	+1
E_{d4}	0	+71612	+38876	+11274	-31640
E_{d8}	0	+71612	+38878	+23528	-13304

Finally, table 4 sums up the final evaluations of perfect play on nxn closed boards with $n \in \{2,3,4\}$.

Table 4. The evaluation of the terminal positions of perfect play on small boards.

size	2x2	3x3	4x4
E_0	0	+1	0
E_1	+1	+9	+2
E_2	+3	+81	+28
E_{d4}	+5118	+82944	+36856
E_{d8}	+5118	+82944	+36856

4.2 Remarks

This subsection contains a list of qualitative remarks about the EFs.

Remark 1. E_0 alone is adapted to the opening of games on large boards. The small upper board of figure 5 shows the perfect sequence played by using E_0 without using the capture rule. The large upper board of figure 5 contains the sequence mapped from the small board into the large one by a scaling operator. To some extent, this sequence contains adequate moves of an opening on a large board. The moves are played to occupy big empty points far from friendly stones, which is one of the most important strategies in the opening.

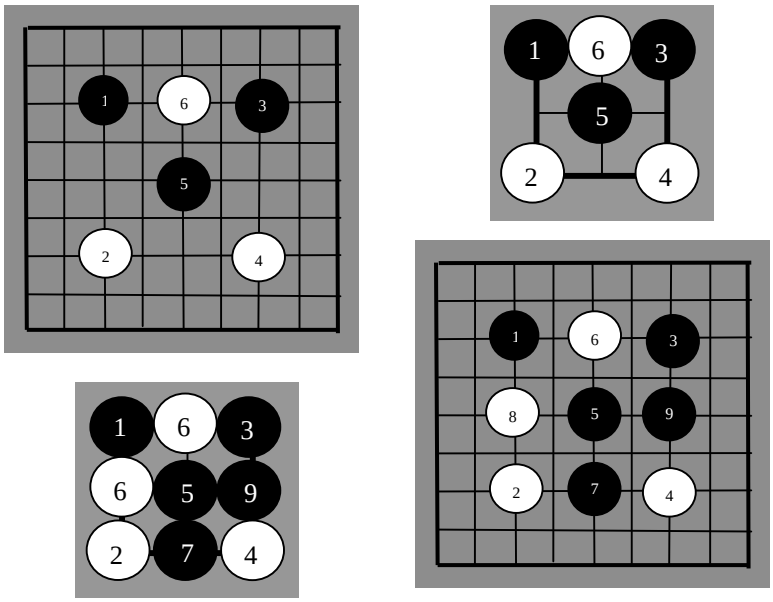


Figure 5. two openings on a large board obtained by a mapping from the perfect play on a small board by using either Eo or λE_1 + Eo (and not the capture rule).

Remark 2. Associated with E_1, E_0 is also adapted to the openings of games on large boards. The small lower board of figure 5 shows the perfect game played on a 3x3 board by using a $\lambda E_1 + E_0$ evaluation function without the capture rule. The large lower board of figure 5 contains the sequence mapped from the small lower board by a scaling operator. This sequence completes the previous one by adding the moves to occupy normal empty points, which is another important feature in the opening.

Remark 3. E_1 is adapted to the endgame. Associated with the abstractColor function, this is the classical EF in Go.

Remark 4. E_2, E_{d4} and E_{d8} are well suited for middlegame. E_2 leads the program to grow its own large groups and to reduce the opponent's ones. Figure 6 shows four open position in which it may be worthwhile to connect or disconnect the stones. Table 5 gives the evaluations of positions of figure 6. In the context of middle game, human go players will agree that position B is the best option for Black and that C is the best one for White.

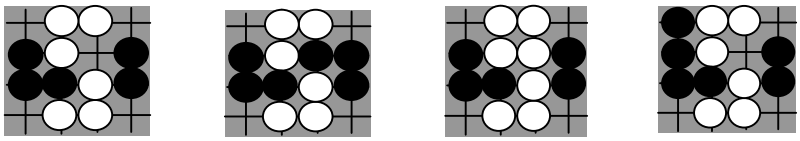


Figure 6. Four open positions. Position A is the starting position in which B is a "good" option for Black, C is a "good" option for White and D is a "bad" option for Black.

Table 5. Evaluations of the positions of figure 6.

	A	B	C	D
E_0	0	0	+1	0
E_1	-1	0	-2	0
E_2	-5	0	-36	+2
E_{d4}	-72544	0	-81868	-51080
E_{d8}	-21564	0	-81868	-70

E_1 is a dull evaluation situating every move of position A on the same level, with an incentive +1. E_2 is more suited to middle game because, when playing white, option C is far ahead. But unfortunately, when playing black, depth one search using E_2 cannot clearly discriminate between the set of moves. Depth one tree search using E_{d4} enables the system to select option B for black because connecting two 4-connected sets of color c into one slightly increases $E_{d4}(c)$. But option C is not far ahead when playing white because adding one element to a connected set does not increase E_{d4}. Moreover, 8-connected sets correspond either to the boundary of territories recognized at the end of the game or to the dividers in a fighting position. Therefore, E_{d8} cannot be of no use. Depth one tree search using E_{d8} enables the system to select option C for White. Option B is ahead when playing Black, because connecting two 8-connected sets of color c into one increases the $E_{d8}(c)$. Therefore, E_2, E_{d4} and E_{d8} seem to be relevant evaluation functions for middle game. This will be confirmed by the experiments.

Remark 5. E_{d4} is a linear combination of E_2 on terminal positions. We demonstrate that E_2 and E_{d4} are linked by formula 13 on terminal positions.

$$E_{d4}=\infty E_2 \qquad (\mathbf{13})$$

Let us assume that the position is terminal. Given the importance of the connected sets, ordering the sum of formula 11 according to the connected sets S and S' of the position is appropriate. This yields formula 14.

$$E_c = -\sum_{S,\,S'} \sum_{i \in S, j \in S'} d(c, i, j) \quad (\mathbf{14})$$

Now, an intersection is either Black or White. Furthermore, d(c, i, j) either equals 0 or +∞. If i and j are of color c and belong to the same connected set, then d(c, i, j) equals 0, otherwise equals +∞. This gives formula 15.

$$E_c = -\left(\sum_{S,\,S'} \sum_{i \in S, i' \in S'} \infty - \sum_{S\ c(S)=c} \sum_{i,j \in S} \infty\right) \quad (\mathbf{15})$$

Then, we can count the number of elements of these two sums. If T is the number of intersections of the terminal position, the first sum contains T^2 elements and the second one contains $E_{2,c}$ elements (by definition of $E_{2,c}$). Thus, we simply obtain formula 16.

$$E_c = -\infty(T^2 - E_{2,c}) \quad (\mathbf{16})$$

Finally, the use of formula 4 and formula 16 demonstrates formula 13. We could of course get a similar formula linking E_{d8} and E_2 by changing the connection from 4-connection to 8-connection.

Remark 6. E_{d4} and E_{d8} are more reliable than E_1 or E_2 on non-terminal positions. On 19x19 middle-game positions, E_1 cannot be used appropriately. Moreover, E_2 has the downside of being insensitive to some good moves (see remark 4). On the positions of figure 7, depth-one tree search based on E_{d4} or E_{d8} selects the right moves for Black and White.

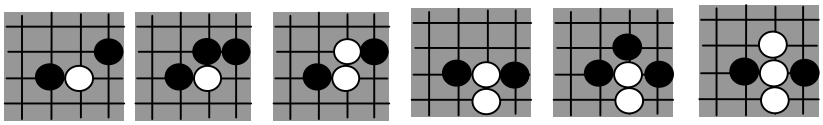


Figure 7. Six open positions corresponding to larger middle-game positions. Position G and J are the starting positions in which H and K are "good" options for Black, I and L are "good" options for White.

Table 6. The evaluations of the positions of figure 7.

	G	H	I	J	K	L
E_1	+1	+2	0	0	+1	-1
E_2	+1	+4	-2	-2	-1	-7
E_{d4}	+29630	+57438	-100	-80	+27718	-27810
E_{d8}	+29642	+57342	-66	-30	+27684	-27732

5 The practical experiment

This section describes the two main experiments we carried out: the 4x4 go resolution speed-up with E_2 instead of E_1, and the automatic weight adjustments of a combination of E_1, E_2, E_{d4}, E_{d8} and other parameters by means of an evolving population of 4x4 go programs. First, we briefly go over the state of the art of

programs playing on small boards to define the test set of the first experiment. Then, we describe the main properties of the tree search algorithm that we used to perform the experiments. Finally, we point out the main features of the evolving population of 4x4 go programs which aims at finding the adapted combination of parameters.

5.1 small Go board resolution

[Thorpe & Walden 1972] and [Lorentz 1997] focused on 2xN boards while [Sei & Kawashima 2000] and [Bouzy 2001] focused on NxN boards (N<=4). [Sei & Kawashima 2000] provided a solution of 2x2, 3x3 and 4x4 go by using Japanese rules, little go knowledge, and alpha-beta with transposition table. [Bouzy 2001] highlighted the retrograde analysis of go patterns of size 3x3 or 4x4 with Chinese rules. Table 7 points out the results of NxN go using either Japanese rules or Chinese rules and Figure 8 shows the optimal sequence for each size of board from 2x2 up to 4x4. The sequences apply to both Chinese and Japanese rule sets.

Table 7. The results of NxN go in Japanese and Chinese rules (0<N<5)

Size	1x1	2x2	3x3	4x4
Japanese	draw	draw	win	draw
Chinese	0	{+1|-1}	{+9|-9}	{+2|-2}

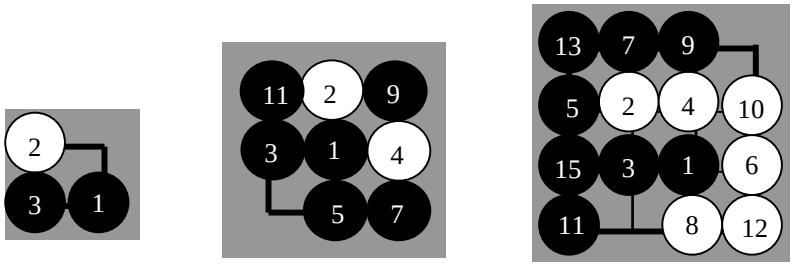


Figure 8. The perfect play on 2x2, 3x3 and 4x4 boards.

5.1 Tree search algorithm

Our reference algorithm is alpha-beta with iterative deepening [Slate & Atkin 1977] [Korf 1985]. The time limit, at which the last iteration was triggered, was 900 seconds (on a Pentium 450Mhz with 128Mo). Iterative deepening enabled the search algorithm to find the correct move without exploring a lot of nodes. What's more, the first three positions of the optimal 4x4 game could not be played without iterative deepening because of a lack of memory. We used transposition table [Greenblatt & al. 1967], [Marsland 1986] with 2^{19} entries. For each entry, we stored the zobrist key of the position, the next player to move, whether the last move was a pass or not, the set of moves forbidden by repetition, the depth, the alpha beta bounds and the move found by the previous iteration.

We used the history heuristic [Schaeffer 1989]: when a move is found to be "sufficient" to create a cut-off somewhere in the tree, the history move value is increased by 2^{depth} in the history table. We observed a 22% reduction of visited nodes. Thus, the history heuristic offers a very positive enhancement. Of course, it is not indispensable but it is so easy to implement without any downside that we inserted it into our reference algorithm. We did not use null-move pruning reduction [Donninger 1993] because, in go, null-move is a normal move. We did not use MTD(f) [Plaat & al, 1996] either because the reduction was too small.

Apart from the rules of go, and the evaluation function that uses a simple abstractColor function, we insert as little go knowledge as possible into the move-ordering algorithm. A move has a domain dependent priority that is low near the corners and high in the centre of the board. A move has a very low priority when the rules of the game forbid it to the opponent, illustrating the proverb that "my good moves are also my opponent's good moves".

5.3 Population of 4x4 programs

When starting the experiment, we were looking for a good combination of E_1, E_2, E_{d4} and E_{d8}. Therefore, we used evaluation E defined by formula 17.

$$E = a_1.E_1+a_2.E_2+b_4.E_{d4}+b_8.E_{d8} \quad (\mathbf{17})$$

We set up a population of eight 4x4 go programs, each of them using an instance of the evaluation of formula 17. In the first stage, $+\infty$ was set to the 1024 value. We decided that a_1, a_2, b_4 and $b_8 \in [0,16]$. The first eight programs were picked up at random. One tournament consisted in 56 games in which every program matched all other programs twice (one game with black and one with white). One win gave 2 points to the winner, one draw gave one point to both players and one loss gave no point. After the 56 games, the programs were ranked according to their point number.When the point numbers were identical, the programs were ranked according their score average. The timeout limit of iterative deepening was set to one second to make the programs play quickly to shorten the time of the experiment.

When a tournament was finished, the first sixth programs plus two new programs attended the next tournament. The first new program was the copy of the first program of the tournament with a random mutation $a_i=a_i\pm$. The second new program was created at random. The tournaments were performed over a "period". After each tournament, a population had an average value that was the average value of the first six programs. We measured the convergence of the population average value with the sum of the square of the error of each weight. When the convergence was situated below a threshold, the period ended.

When a period ended, the space in which the population evolved was adjusted to the average value for the next period. This adjustment was performed for several reasons. First, to avoid too slow a convergence. For example, if one parameter, say a_2,

converged to a fixed value, say 4, and the population was in the interval [2,6], then generating a program with one parameter set at random in [0,16] was not appropriate. Therefore, a new set of values for this parameter such as [2, 6] with 17 values was chosen. Secondl, when one parameter reached one frontier of the interval, the size of the interval was doubled. For example, if one parameter, say b_4, reaches the max frontier, say 16, then the new interval was [0, 32]. When a parameter did not converge significantly during the period, this parameter was considered as "noisy" and its interval remained the same for the next period.

We supervised all this process "by hand" and we stopped it when we considered that either some parameters were sufficiently adjusted or some parameters remained "noisy". This was the end of an "era" - one supervision iteration. At the end of each era, a convergent parameter was fixed to its value, and disappeared from the list of features for the next era, and a "noisy" parameter may give rise to the birth of new features for the next era.

The first era was the life of a_1, a_2, b_4 and b_8 at the end of which, we observed that the linear combination of formula 17 could greatly benefit from the tuning of other parameters. The second era marked the adjustment of ∞, used by E_{d4} and E_{d8}. At the end of this era, no more convergence was foreseeable. But the dimensional and the metric evaluation functions being very different by nature, we wondered whether they could be applied two different stages of the game. Thus, the third era witnessed the introduction and the adjustment of "temporal" parameters reflecting the split of a 4x4 game into an opening phase, a middle-game phase and an endgame phases. Our method was closely supervised "by hand".

6 Results of the experiments

This section provides numerical results from the two experiments assessing the dimensional and metric EF. First, we highlight the node number reduction provided by E_2 on 4x4 resolution. Then, we underline the weights of a linear combination of E_1, E_2, E_{d4} and E_{d8} obtained by an evolving population of programs playing go on 4x4 boards.

6.1. E_2 "enhancement" assessment

Table 8 illustrates the node number reduction resulting from the classical alpha-beta enhancements such as transposition table, iterative deepening, history heuristic, null move pruning and MTD(f). The measurements were performed on the test set made up of 17 positions of the optimal sequence of 4x4 go.

Table 8. The results of the enhancements

TT	ID	HH	null-move	MTD(f)
+∞	+∞	22%	10%	6%

Without well-formulated go knowledge about move ordering, we observed that transposition table and iterative deepening were mandatory. The history heuristic is efficient (22%). Null move reductions are possible (10%). Like in chess, null move pruning on 4x4 go actually reduces the number of visited nodes. MTD(f) is a positive but small enhancement (6%).

Instead of E_1, we tried to use E_2. First, we noticed that the sequences found by the search algorithm using E_2 were exactly the same as the sequences found by the normal algorithm. Hence, changing the evaluation function does not change the external behavior of the program and E_2 is still correct over the perfect play on 4x4 go. Furthermore, as shown in Figure 9, the advantage is that, from position number 2 up to the last position of the optimal game, the number of nodes visited by E_2 algorithm is situated 21% below the number of nodes visited by E_1 algorithm.

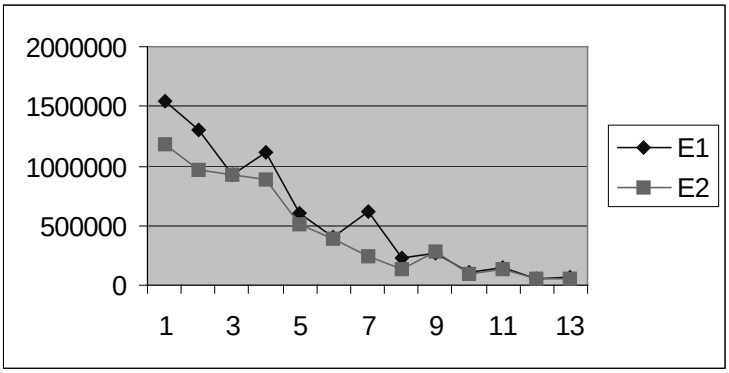


Figure 9. Number of visited nodes by the reference algorithm with E_1 or E_2

We try to explain this visited node number reduction. E_2 increases the position evaluations in which the friendly sets are connected. Therefore, the algorithm "first explores the moves that increase the size of the connected sets" and "give a bad evaluation to positions in which friendly stones are split into parts". This explanation is closely linked with the fact that, unfortunately, E_2 based tree search visits slightly more nodes than the normal one on positions 0 and 1. In these two positions, one color is not present. Consequently, E_2 cannot benefit from increasing a connected set of this colour because this set does not exist. Then, the question is to know how the node number evolves with the "power" of the evaluation function but we have not carried out this experiment yet.

6.2. The "dimensional" versus "metric" evaluation function assessment

This subsection deals with results obtained by the evolving population. First, we describe the results of the weight adjustment of formula 17. This adjustment corresponds to the first era. The first era contained 8 periods and about 400 tournaments at the end of which we observed the results of table 9:

Table 9. Results at the end of the first era

A_1	A_2	B_4	B_8
[0,8]	[0,8]	[32, 96]	[16, 48]

These results showed the superiority of the metric evaluations over the dimensional ones and also the slowness of the convergence, all parameters remaining "noisy" in their convergence interval.

Because the "best" move of each iteration of iterative deepening was unstable, the decision to set the timeout with a low value lead to almost random play for the first few moves in the openings played in the first tournaments. But the advantage lay in the possibility to explore much space and make the evaluation function weight adapt more quickly.

Given the importance of the metric evaluations and the important weight of $+\infty$ within these evaluations, it was an urgency to detect the relative importance of the $+\infty$ value. Therefore, we added the new feature $+\infty$ to the population of programs and started the second era with $+\infty \in [0, 2048]$. This era lasted 200 tournaments and 6 periods. Table 10 shows the results.

Table 10. Results at the end of the second era

A_1	A_2	B_4	B_8	$+\infty$
[0,8]	[0,1]	[60,68]	[28, 36]	[384, 640]

Beyond the value of $+\infty$, 512, we observed a better convergence in this era than in the previous era: a_2 decreased a lot and b_4 and b_8 converged toward 64 and 32 to some extent. But still, a_1 remained noisy. At the end of this era, several observations could be made.

First, the games were not played until their end, as defined by the rules, but stopped before. This was a positive consequence of the weight adjustment. The downside was that the good programs stopped early without physically capturing the virtually captured stones. When two programs disagreed, the referee decided who the winner was. Unfortunately, the referee used E_1 and did not count the correct evaluation[1] but the physical one, and the programs playing well had a penalty.

Second, we saw that the first iterations of iterative deepening produced a very fluctuant "best" move. Because we wanted the experiment to be finished early enough, we were obliged to set up a short timeout to iterative deepening. Consequently, the first moves of the game (about the first four ones), produced by elapsed timeout iterative deepening were almost random, whereas the middle and endgame moves, produced by enough depth iterative deepening, were correct. For the opening, we also noticed that the move produced by the first iteration of iterative deepening was suprisingly the good one, and that the next iterations gave worse moves.

Therefore, for the next era, we required a population of programs whose parameters depended on the stage of the game and also included the maximal depth of iterative deepening as a parameter.

[1] the referee did not perform mini-max search as the players did!

In the third era, we fixed the old features of the playing programs with $a_1 = 1$, $a_2 = 8$, $+\infty = 512$ and we correlated b_4 and b_6 with the formula $b_8 = b_4/2$. We defined five new parameters: ‘depthOpening’, ‘bOpening’, ‘bEndgame’, ‘openEndGame’ and ‘endOpening’. ‘endOpening’ was the last move number of the opening phase of a game. ‘openEndGame’ was the number of the first move of the endgame. ‘depthOpening’ was the maximal depth of iterative deepening during the opening. ‘bOpening’ was the value of b_4 during the opening and the middle game and ‘bEndgame’ was the value of b_4 during the endgame. We started the third era with ‘depthOpening’ $\in$[1, 6], ‘endOpening’ $\in$[1, 16], ‘openEndGame’ $\in$ [‘endOpening’, 20], ‘bOpening’ $\in$[48, 80], ‘bEndgame’ $\in$[0, 80]. After 200 tournaments these parameters converged toward the values shown by table 11.

Table 11. The results at the end of the third era

depthOpening	endOpening	openEndGame	bOpening	bEndGame
1	4	10	[64, 80]	[8, 16]

One result is very surprising: the iterative deepening depth that produces the best result is depth one! We are not able to provide an adequate explanation for it. The other results had no unexpected element. We expected the value of ‘bEndgame’ to decrease to give more importance to the basic E_1 evaluation function, relevant to the endgame. This was observed because ‘bEndGame’ converged toward 8, which is the E_1 weight. We expected ‘bOpening’ to keep the same value and we actually observed this result. Finally, the two boundaries fixing the opening and endgame phases reached satisfying values: the value 4 marks the end of the opening and the value 12 corresponds the beginning of the endgame.

7 conclusion

Here are the following contributions of this paper:

- definition of three ”dimensional” go evaluation functions E_0, E_1 and E_2,
- definition of two ”metric” go evaluation functions E_{d4} and E_{d8},
- E_0 is qualitatively adapted to the opening on large boards,
- E_1 is the classical go evaluation function,
- E_2 is an experimental speed-up on the resolution of 4x4 go to be compared to the classical alpha-beta enhancements,
- E_{d4} and E_{d8} are qualitatively relevant to middle-game on large boards and experimentally adapted to small board go associated with depth-one search. This constitutes the main contribution of this study.

This paper opens up various perspectives. First, extending the experiment to larger boards until the life and death module becomes necessary, then introducing life and death knowledge into the abstractColor function. Furthermore, it seems worthwhile to integrate the metric evaluation functions E_{d4} and E_{d8} into our 19x19 program to improve its middle-game play, and to integrate the E_0 evaluation into the opening evaluation of our 19x19 playing program.

References

1. Bouzy B., Go generated patterns by retrograde analysis. In: J.W.H.M. Uiterwijk (ed.): the 6th Computer Olympiad Computer-Games Workshop Proceedings, Report CS 01-04, Universiteit Maastricht, (2001)
2. Bouzy B., Cazenave T., Computer Go : an AI oriented Survey, Artificial Intelligence, Vol. 132 n°1 (2001), 39-103
3. Chen K., Computer Go: Knowledge, Search, and Move Decision, ICCA Journal, Vol. 24 n°4 (2001), 203-215
4. Donninger C., Null move and deep search: selective-search heuristics for obtuse chess programs, ICCA Journal, Vol. 16 n°3 (1993), 137-143
5. Enzenberger M., the integration of a priori knowledge into a Go playing neural network, http://www.markus-enzenberger.de/neurogo.html, 1996.
6. Greenblatt R.D., Eastlake III D.E., Crocker S.D.: The Greenblatt chess program, fall joint computing conference proceedings 31, (New-York ACM), San Francisco, (1967), 801-810
7. Lorentz R., 2xN Go, Proceedings of the 4th Game Programming Workshop in Japan'97, Hakone, pp. 65-74, 1997
8. Mandelbrot B., The fractal geometry of nature, W.H. Freeman and Company, San Francisco (1982)
9. Marsland T.A., A Review of Game-Tree Pruning, ICCA Journal, Vol. 9. n°1 (1986) 3-19
10. Plaat A., Schaeffer J., Pijls W., de Bruin A.: Best-first fixed-depth minimax algorithms, Artificial Intelligence, Vol. 87 n°1&2 (1996), 255-293
11. van Rijswijk J., Computer hex: are bees better than fruitflies?, M.Sc. Thesis, University of Alberta, Edmonton, AB, 2000
12. Schaeffer J.: The history heuristic and alpha-beta search enhancements in practice, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 11 n°1 (1989), 1203-1212
13. Sei S., Kawashima T.: A solution of Go on 4x4 board by game tree search program, Fujitsu Social Science Laboratory, (2000), manuscript
14. Slate D.J., Atkin L.R.: Chess 4.5 – the north-western university chess program, in Chess Skill in Man and Machine, P. Frey (ed.), Springer-Verlag, (1977), 82-118
15. Thorpe E.O., Walden W.E.: A computer assisted study of Go on MxN boards, Information sciences, vol 4 (1972) 1-33