Mathematical morphology applied to computer go

Bruno Bouzy

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

ABSTRACT

This paper shows how mathematical morphological operators can be applied to computer go. On one hand, mathematical morphology is a very powerful tool within image processing community. On the other hand, the Zobrist's model is well-known within the computer go community for its "influence" recognition. We present a model, derived from the closing operator of mathematical morphology and from the Zobrist's model, which yields very good results for "territory" recognition. Moreover, we give efficient implementations of the dilation operator and territory recognition for computer go. This model was found when developing Indigo, our go playing program, and is now used with success in GnuGo, the go playing program of the Free Software Foundation.

KEY WORDS

Computer Go, Mathematical Morphology, Territory, Dilation, Erosion, Closing.

1. INTRODUCTION

This paper shows how mathematical morphology [4] can be applied to computer go. Mathematical morphology is a very useful technique in the field of image processing. For example, some operators enable systems to delete the details whose size ranks below a given scale.

Besides, the complexity of Go is much greater than the complexities of other two-person, zero-sum, complete information games such as Chess, Shogi, Checkers and Othello. The results achieved by the best go programs are average on the human scale: Wulu, Go4++ and Goemate have reached an intermediate level, between a beginner and strong players [2]. The first difficulty to program the game of go lies in the combinatorial complexity which forbids brute-force tree search. The second difficulty is to correctly evaluate a position. We have worked on computer go for several years and one of our contributions to computer go was to find the link between the so called Zobrist's model [5], mathematical morphology and "territory" recognition. Zobrist model, without using mathematical morphology explicitly, already contained ideas not far from it. Zobrist model was composed of iterative dilations. It enabled the programs to recognize "influence" as human players do, but not "territory". ("territory" and "influence" are domain-dependent concepts in the game of go.)

The aim of this paper is to describe how mathematical morphology is successfully applied to computer go for the territory recognition.

Section 2 shows a first application of mathematical morphology on basic concepts of go. Moreover, section 3 underlines how an adaptation of mathematical morphology, based on the Zobrist model, can efficiently recognize "territories". Furthermore, section 4 gives the implementation details of the method, used by the go playing program Indigo. Section 5 highlights the results achieved by mathematical morphology within computer go.

2. FIRST APPLICATION OF MATHEMATICAL MORPHOLOGY TO COMPUTER GO

This section goes over the basic operators of mathematical morphology and their power on basic concepts of go. Furthermore, it presents the Zobrist model.

2.1. The Basic Operators Of Mathematical Morphology

The subsection recalls the definition of the basic operators of mathematical morphology that we use in this paper for computer go: dilation, erosion, external boundary, internal boundary, closing and opening. Let A be a set of elements, dilation D(A) is the set composed of A, plus the neighboring elements of A. Erosion E(A) is composed of A, minus the elements which are neighbors of the complement of A. External boundary ExtBound(A) equals D(A)-A. Internal boundary IntBound(A) equals A-E(A). Closing set Closing(A) equals E(D(A)). Opening set Opening(A) equals D(E(A)).

2.2. "Liberties" And "Eyes" In Go

The dilation operator enables programs to recognize "liberties" of "strings". Figure L1 shows a black string and white strings. Figure L2 shows the dilation operator applied to the black string. Finally, figure L3 gives the "liberties" of the given string (x) by intersecting L2 with the empty set of figure L1.

figure L1 figure L2 figure L3

The closing operator enables programs to recognize "small eyes". Figure E1 shows the set of four black strings which build an "eye" situated on the central intersection (e). This "eye" has to be recognized. Figure E2 shows the application of the dilation operator and figure E3 the result of the application of the closing operator minus the black set of figure E1.

figure E1 figure E2 figure E13

2.3 "Clear Territories"

"Territories" are relevant objects in go. This subsection deals with the recognition of "clear" territories. We call clear territory a territory easily recognized by go beginners. For example, figure T1 shows a set of black strings which creates a "clear territory" (t).

figure T1 figure T2

Figure T2 shows the result of the application of two dilations on the black set of figure T1. Figure T3 shows the result of the application of two dilations on the black set of figure T1 followed by two erosions.

figure T3 figure T4

Finally, figure T4 shows the territory (t) by removing the initial set. Therefore we see the relevance of defining an operator X(n) by

$$x(n) = E^n \text{ o } D^n$$

x(n) is the operator which dilates n times and erases n times.

2.4 "Fuzzy Territories"

Actual territories in go are not defined so clearly but more fuzzily than in the previous subsection. An actual territory is a territory which appears in actual games: the stones may not be clearly connected but loosely connected. Figure F1 shows a set of two friendly black stones which creates an interaction between them (i).

figure F1 figure F2 figure F3

Figure F2 shows the result of the application of two dilations on the black set of figure F1. Figure F3 shows the result of the application of two dilations on the black set of figure F1 followed by two erosions. *No territory is recognized* by this method and something else is needed. This will be the subject of section 3.

2.5. Zobrist's Model To Recognize "Influence"

The Zobrist model [5] starts by assigning values of +64 (resp. –64) to black (resp. white) intersections, and 0 elsewhere. Then, it performs the following process four times. Each point which is positive sends out a +1 to each of its neighbors, and each negative point sends out a –1 to each of its neighbors.

Our model is similar to the the Zobrist model. The difference lies in the content of the loop repeated four times. For each non zero intersection, add to the absolute value of the intersection, the number of neighboring intersections of the same sign, provided that all the neighboring intersections are empty, or of that sign. For each zero intersection which has neighboring intersections of the same sign, the process adds the number of neighboring intersections of this sign to the absolute value of the intersection.

In this process, the reader has recognized the application of four "adapted" dilations. The dilations of the Zobrist model are adapted in taking into account the two colors, black and white. The process forbids an intersection to belong to the dilation of one color if it belongs to the dilation of the other color. The Zobrist model enables systems to recognize "influence".

Moreover, the dilations of the Zobrist model also take into account numerical values. We cannot explain such a choice when considering that "influence" recognition was the goal of the method. A non-numerical model might give a same result. We think that Zobrist inserted numerical values into his model because his initial aim was to model territories, and not influence. He approximated territories with the intersections whose values were greater than a given threshold. Unfortunately, we think this method was not adequate to model territories. Nevertheless, this model with its two refinements, the two-color refinement and the numerical refinement, inspired the creation of a better application of mathematical morphology to computer go and more specifically to territories recognition.

3. A BETTER ADAPTATION OF MATHEMATICAL MORPHOLOGY TO COMPUTER GO

Given the facts that the dilation operator is useful to recognize liberties of string and that the closing operator is useful to detect potential "eyes" and "territories", we combine the idea of the Zobrist model with mathematical morphology. As we observed before, the Zobrist model corresponds to some extent to the iterative application of four "dilations" D. Therefore, we defined the "erosion" E in an analogous way.

The D_z operator consists in adding to the absolute value of an intersection of one color, the number of neighboring intersections of this color, provided that all the neighboring intersections are empty, or of that color. For an empty intersection, which has neighboring intersections of the same color, D_z also adds the number of neighboring intersections of this color to the absolute value of the intersection. D_z is a numerical refinement of the classical dilation operator mentioned above.

Similarly, the E_z operator is new, it consists in subtracting from the absolute value of an intersection of one color, the number of neighboring intersections whose value is either zero, or whose value corresponds to the opposite color of the intersection. E_z also makes sure that the sign of the value does not change - otherwise, the value becomes zero. E_z is therefore a numerical refinement of classical morphological erosion. Subsection 4.2 gives the C++ implementation of D_z and E_z operators.

Once these refinements are added, we use the operators $X_z(n) = E_z^e \text{ o } D_z^d$ where E_z is iterated 'e' times, and D_z, 'd' times. To give the same result as the identity operator on positions where no "territory" is recognized, a link between 'e' and 'd' must be established. On the trivial position, with only one stone located in the middle of the board, X_z must give the same result as the identity operator. Thus $e = d.(d-1) + 1$, in which 'd' is a scaling factor. The bigger 'd' is, the larger the scale of recognized territories is.

Now, we can try $X_z(3, 7)$ on the position of figure F1, in which x(2, 2) or x(n,n) gave no result at all.

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	64	0	0	64	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Figure X1

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	1	0	0	0	0
0	0	0	2	2	2	2	2	2	0	0	0
0	0	2	4	6	5	5	6	4	2	0	0
0	1	2	6	64	7	7	64	6	2	1	0
0	0	2	4	6	5	5	6	4	2	0	0
0	0	0	2	2	2	2	2	2	0	0	0
0	0	0	0	1	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Figure X2: after 3 dilations

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	60	7	7	60	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Figure X3: after 3 dilations and 7 erosions

Therefore this example shows that the refined operator X_z enables programs to recognize “something” (7) between the two stones (64). We call this “something” a territory and we inserted this knowledge into our playing program Indigo and it gave good results. The playing program Indigo uses operators $X_z(13,4)$ or $X_z(21,5)$ according to the phase of the game.

4. IMPLEMENTATION

This section deals with the efficient implementation of the dilation operator (subsection 4.1) and the implementation of the numerical adaptations of our model (subsection 4.2). These implementations are parts of our go playing program Indigo.

4.1. An Efficient Implementation Of The Dilation Operator

Go playing programs need fast implementations of their morphological and logical operators. The dilation operation and the logical ‘or’ operation are very often called and must be optimized. Indigo spends 20% of

its time computing these operations. This subsection gives the C++ source code of this implementation. SI is the class modeling the sets of intersections. An SI object contains 19 integers 'line[i]' (0<=i<19), whose first 19 bits correspond to the 19 intersections of a line of the board. Therefore, in this representation, the logical 'or' of two sets is quickly computed thanks to the | operator applied on the 19 integers. This representation does enable Indigo to use the shift operators, >> and <<, combined with the copy operator to model the dilation. 'mask' equals 2^{19}. The >> and << operators are clever tools to increase the speed of the dilation operator.

```
#define t 19

class SI
{
  int line[t];
  …
  void logical_or(SI *);
  void dilation(SI *);
};

void SI::logical_or(SI * SI)
{
  for(int i=0; i<t; i++) SI->line[i] |= line[i];
}

void SI::dilation(SI * q)
{
  int here, left, right, up, down;
  if (here=line[0]) {
    q->line[0] |= here;
    right = here>>1; right &= mask;
    q->line[0] |= right;
    left = here<<1; left &= mask;
    q->line[0] |= left;
  }
  if (down=line[1]) q->line[0] |= down;

  for (int J=1; J<t-1; J++) {
    if (up=line[J-1]) q->line[J] |= up;
    if (here=line[J]) {
      q->line[J] |= here;
      right = here>>1; right &= mask;
      q->line[J] |= right;
      left = here<<1; left &= mask;
      q->line[J] |= left;
    }
    if (down=line[J+1]) q->line[J] |= down;
  }
  if (up=line[t-2]) q->line[t-1] |= up;
  if (here=line[t-1]) {
    q->line[t-1] |= here;
    right = here>>1; right &= mask;
    q->line[t-1] |= right;
    left = here<<1; left &= mask;
    q->line[t-1] |= left;
  }
}
```

Other optimizations can be performed on this code, particularly using pointers instead of indexes for browsing the 'line' arrays, and deleting local variables such as 'left', 'right', 'up' and 'down', but we did not perform them here to keep the readability of the algorithm.

4.2. An Implementation Of D_z and E_z

The adaptations of dilations D_z and erosions E_z presented before are more difficult to compute quickly because they include numerical computations. Nevertheless, this subsection shows the C++ source code of Indigo which computes these "dilations" and "erosions". 'Potential' is the name given to the C++ class which contains the implementation. It is mainly made up of two arrays, 'nature' and 'intensity'. The former indicates the color (the nature) of the intersection while the latter refers to the value of the intersection. Each of these two arrays has its own copy 'buffer_nature' and 'buffer_intensity' which are copied after each dilation or erosion. 't2' corresponds the size of the arrays. For speed considerations, the arrays are one-dimensional and not two-dimensional. 'memcpy' is used to copy the 'nature' array into the

‘buffer_nature’ array and the ‘intensity’ array into the ‘buffer_intensity’ after each pass of the algorithm. IDX_VOISIN(r, idx, idxv) is an inline function which sets idxv to the rth neighbor of idx. We do not give its contents for reasons of clarity.

```
#define t2 361

class Potential { public:
  static int buffer_nature   [t2];
  static int buffer_intensity[t2];
  static int nature          [t2];
  static int intensity       [t2];

  static void copy();
  static void dilate_z();
  static void dilate_z_element(int);
  static void erase_z();
  static void erase_z_element(int);
};

int Potential::buffer_nature[t2];
int Potential::buffer_intensity[t2];
int Potential::nature[t2];
int Potential::intensity[t2];

// the algorithm with d dilations and e erosions.
{
  for (int i=1; i<=d; i++) dilate_z();
  for (int i=1; i<=e; i++) erase_z();
}

void Potential::copy()
{
  memcpy(buffer_nature, nature, 4*t2);
  memcpy(buffer_intensity, intensity, 4*t2);
}

void Potential::dilate_z()
{
  copy();
  for (int idx=0; idx<t2; idx++) dilate_z_element(idx);
}

void Potential::dilate_z_element(int idx)
{
  int ma_nature = buffer_nature[idx];
  for (int r=1; r<=4; r++) {
    int idxv;
    IDX_VOISIN(r, idx, idxv);
    if ( (idxv!=-1) &&
         (buffer_nature[idxv]) &&
         (buffer_nature[idxv] != ma_nature) ) {
      if (ma_nature) return;
      else ma_nature = buffer_nature[idxv];
    }
  }
  if (!ma_nature) return;
  nature[idx] = ma_nature;
  for (int r=1; r<=4; r++) {
    int idxv;
    IDX_VOISIN(r, idx, idxv);
    if ((idxv!=-1) && (buffer_nature[idxv] == ma_nature))
      intensity[idx] = 1 + intensity[idx];
  }
}

void Potential::erase_z()
{
  copy();
  for (int idx=0; idx<t2; idx++) erase_z_element(idx);
}

void Potential::erase_z_element(int idx)
{
  int ma_nature = buffer_nature[idx];
  if (ma_nature == 0) return;
  for (int r=1; r<=4; r++) {
```

```
    int idxv;
    IDX_VOISIN(r, idx, idxv);
    if ( (idxv == -1) ||
         (buffer_nature[idxv] == ma_nature) ) ;
    else intensity[idx] -= 1;
    if ((idx!=-1) && (intensity[idx]==0)) {
      nature[idx] = 0;
      break;
    }
  }
}
```

5. RESULTS

The results of the model can be assessed in two ways. First they can be assessed qualitatively by comparing the territories recognized by the model and those found by human players on test positions. Second, it can be assessed indirectly by observing the result of the programs in which it has been incorporated. This second assessment is more difficult to perform because territory recognition is not the unique feature of go playing programs. Nevertheless, it is an assessment proving the evidence that the model is effective in practice and not only in theory.

5.1. Qualitative Assessment Of Territory Recognition

The result of $X_z(d, d(d-1)+1)$ operators with d = 4 or 5 is very good. It recognizes territory almost correctly on all go positions. This algorithm is very robust and never yields bad results. The territories recognized by this model corresponds to the territories identified by human players. Figure Position illustrates an example position on which we applied operators X_z (4, 13) (see Figure Out).

Figure Position

Figure Out

5.2. Results In International Computer Go Competitions

The model explained in this paper belongs to the evaluation function of the Indigo program [1], and has been integrated into the GnuGo program [3], the go playing program of the Free Software Foundation. Indigo and GnuGo are ranked in the 9x9 and 19x19 computer go ladders on the Internet (http://www.cgl.ucsf.edu/go/ladder.html). Since 1998, Indigo attended the Ing cup twice (10^{th} out of 17 in 1998 and 13^{th} out of 16 in 1999), Mind Sport Olympiad once (5^{th} out of 6 in 2000) and 21^{st} Century Cup once (10^{th} out of 14 in 2002). These results confirm the practical usefulness of the territory recognition model, based on mathematical morphology, in computer go.

6. CONCLUSION

In this paper we have underlined the application of mathematical morphology in computer go. We first showed that the dilation operator is crucial to the game of go (the liberties of a string are the empty neighboring intersections of the string). Then, observing that the closing operator enables programs to detect simple “eyes”, we applied morphological operators to computer go. In the past, the Zobrist model was used in go playing programs. It modeled “influence” of stones correctly, but not territories. Our contribution is to make mathematical morphology and the Zobrist model interplay. We defined the Xz(d, e) operator which called a Zobrist adaptation of the dilation operator d times, and an adaptation of the erosion operator e times. This combination led to a model which recognized “territories” of the game of go as human players do. The recognition was accurate and robust. This model has been implemented and integrated with success into the international go playing programs Indigo and GnuGo, this proves the relevance of mathematical morphology in computer go.

7. AKNOWLEDGEMENT

I thank Christopher Smith and Erik van der Werf for proofreading the diagrams and helpful corrections.

8. REFERENCES

1. B. Bouzy, “Modélisation cognitive du joueur de Go”, (in French), PhD thesis, Paris 6 University, 1995, http://www.math-info.univ-paris5.fr/~bouzy

2. B. Bouzy, T. Cazenave, “Computer Go : an AI oriented Survey”, Artificial Intelligence, Vol. 132 n°1 (2001), pp. 39-103.

3. Gnu Go home page, http://www.gnu.org/software/gnugo/devel.html, 1999.

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

5. A. Zobrist, “A model of visual organization for the game of Go”, in: Proc. AFIPS Spring Joint Computer Conference, Boston, AFIPS Press, Montvale, NJ, 1969, pp. 103-111.