2008 Second International Conference on Future Generation Communication and Networking

Representation and Comparison of Fuseki Patterns of Go-games

Woo-Jun PARK
Dept. of Comp. Eng., Hannam University
woojunpark@gmail.com

Abstract

We propose a different method to define patterns on the Go board, which has a tengen pattern and tengen centered 9 square-patterns as well as necessary conditions for two patterns to be equivalent. Using this method, we show that our methodology can easily locate a specific or a desired pattern out of multiple games.

1. Introduction

Final goal of this study is to construct a database for patterns on the go board at n-th move in a go game of a professional player. The constructed database can be used to select next move to win the game by computer go program or by human player.

Patterns in a go game study have two meanings. First, it expresses the configuration of the moves on the board[1][2]. Second, it describes the relation between existing moves and next moves[3][4]. The site [2] provides good services for go-news, contents on go, and pattern searches. Currently we have no information on the representation and comparison techniques for patterns in the site.

There exist various patterns in a go game. For example, there are pattern on the go board at the time of n-th move, fuseki pattern, joseki pattern, and pattern for end-game.

Fuseki pattern is a critical strategy in go games. However, with the currently used recorded files of go game, it is not easy to confirm whether two fuseki patterns from different games are the same pattern. It is because the go board has the top/ bottom and the right/left symmetricity and because the pattern is not affected by the order of the moves.

In this study, we show why current go-game record with up to n-th move cannot represent the pattern of the board at the time of the move. We propose a new coordinate system that enables easy computation of rotated patterns in a go game, and we define the pattern of a go-board which has a tengen pattern and tengen centered 9 square-patterns using the coordinate system.

In order to build a pattern on the board from the go-game record, sometimes the stone-group that has no liberty should be taken out from the board. Fuseki pattern is a pattern without the point which originally occupied with a stone but later emptied.

Using the pattern built from go-game record, we define the relation "same" between two patterns. After implementing the Boolean method "samePattern(p1, p2)" by Java, we show how many times two well known fuseki patterns appear among 250 go-games.

2. Existing Format for Recording Go-games

If we use Cartesian coordinate system to express the location of a point on a go board, a pattern on the board can be defined as a set of three different (empty, black, white) states at any point on 19x19 go board.

Up to first 60 moves is often called fuseki stage. Books on fuseki usually do not name or introduce the patterns which include more than 15 moves because those patterns have too much variability.

Smaller patterns with the first five or six moves are named and classified. For example, there are fuseki of diagonal-star- points , fuseki of star-points and night's move , and one space jump fuseki[10].

Fuseki books do not fix the order of the moves in a pattern and do not show the rotated patterns.

MultiGo 4.0[5], a software which shows the go game record, supports SmartGo Format (*.sgf, *.mgt), BDX Format (*.bdx), NGF Format (*.ngf), GOS Format (*.gos), GIB Format (*.gib), UGF Format (*.ugf, *.ugi), GO Format (*.go). SmartGo Format is widely used[6]. The Korean go server, from which we obtained game files for this study, uses GIB Format[7].

Current formats for a go-game cannot accurately represent patterns in the game because patterns should not be influenced by the order of the moves or the rotation of the go board by 90, 180 or 270 degrees[8][9].

3. Method of Go-board Pattern Representation

In the following, we show a fuseki pattern of first 14 moves of a game file apr2912.sgf and corresponding text data at the sgf game file.

978-0-7695-3431-2/08 $25.00 © 2008 IEEE
DOI 10.1109/FGCN.2008.65

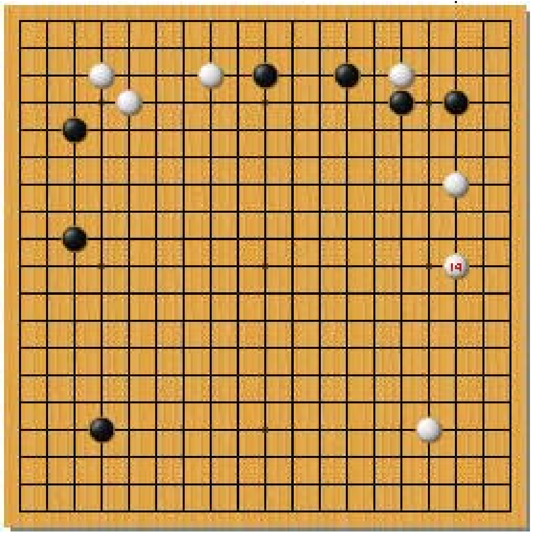
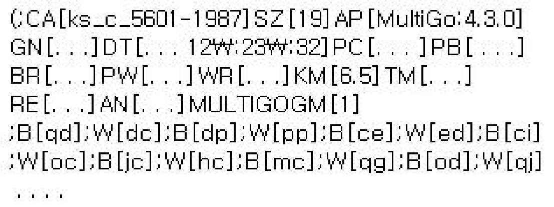


Figure 1. Go-board with 14 moves



```
(;CA[ks_c_5601-1987]SZ[19]AP[MultiGo:4.3.0]
GN[. . .]DT[. . . 12₩:23₩:32]PC[. . . ]PB[. . . ]
BR[. . .]PW[. . .]WR[. . .]KM[6.5]TM[. . .]
RE[. . .]AN[. . .]MULTIGOGM[1]
;B[qd];W[dc];B[dp];W[pp];B[ce];W[ed];B[ci]
;W[oc];B[jc];W[hc];B[mc];W[qg];B[od];W[qj]
 . . . .
```

The order of stone moves does not have to be expressed in patterns. 19x19 board can be expressed as a set of 19 strings which each string of length 19 expresses the states of each point in a row with '0' (empty), '1' (black), or '2' (white). Note that character is used for the state of point instead of the enumerated type constant. Character representation makes it easier to compare patterns in a line of points because the patterns of the points in a line are able to be transformed to a ternary number.

However, above method requires the calculation of ternary number string on each different rotation (90, 180, 270 degree). Each move in fuseki stage is mostly scattered on 3rd, 4th, 8th, 9th, 10th, 16th, 17th rows. In order to solve above problems, we came up with a different representation method. We divided go board into tengen and 9 tengen-centered squares. The area of all the points on a side of each square was named as 'North', 'South', 'East', and 'West'. Level of each point on a square is defined to be same with the length from tengen to the square. There exist nine squares with level 1, 2, …, 9. Offset of each point was expressed with length starting from a vertex moving clock-wise to the point.

Then we can represent all the points on the go-board with unique value (area, level, offset) respectively. The following figure shows the new coordinate system and the definitions for area, level, and offset well as we discussed.

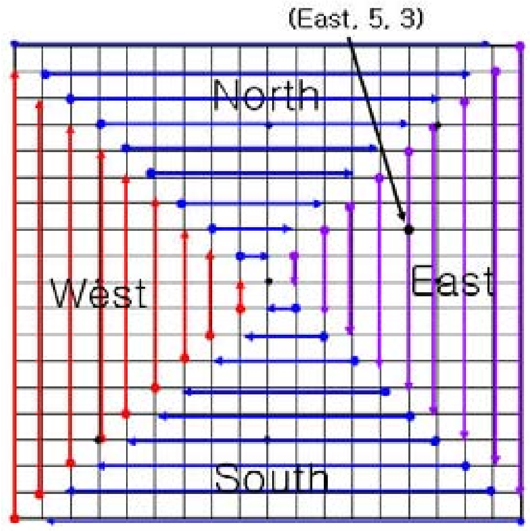


Figure 2. (area, level, offset) Coordinate System

In the following we show data structures for pattern representation in Java[12][13]:

```
enum   Area {Center, West, North, East, South}
enum   MoveColor {B, W}
public class PatternBoard {
      public class RectPat {
      public int no_moves_on;
      public char [] ws;
      public char [] nr ;
      public char [] es ;
      public char [] st;
      RectPat(int lev){ . . . }
      } // end class RectPat
public int NineNine;
// 0: empty,   1: black,   2: white
public RectPat [] patlevel;
PatternBoard(){. . .   }
PatternBoard add(MoveColor moveC, int [] alp)
{   . . .   } // end of add( )
static void xy2ALP( int X, int Y, int [] alp){ . . .   }
static boolean samePattern( PatternBoard p1,
PatternBoard p2) {   . . .   }
PatternBoard addSgfMoves(FileInputStream sgf,
int numMoves )   throws IOException {   . . .   }
PatternBoardS   rotatesClockwise(PatternBoardS
org ){ NineNine = org.NineNine;
for (int k=1; k<=9; k++){
      if (org.patlevel[k].no_moves_on >0) {
      patlevel[k].no_moves_on =
            org.patlevel[k].no_moves_on;
      patlevel[k].nr = org.patlevel[k].ws;
      patlevel[k].es = org.patlevel[k].nr;
```

```
        patlevel[k].st = org.patlevel[k].es;
        patlevel[k].ws = org.patlevel[k].st;
        }
    }
    return(this);
    }
    PatternBoard diagSymmBd(PatternBoard fj, int
angle){   . . .      //fj is a pattern board for a fuseki}
    public void printPatternBoard(OutputStream
patbd) throws IOException{. . .   }
    public static void main(String[] args)throws
IOException{   . . .   }
    }
```

In the above, we showed how easily the method **rotatesClockwise** computes and returns the rotated pattern.

To study patterns in existing go games, number of game file information (BorW, X, Y) can easily be transformed to BoardPattern information (BorW, area, level, locaton). For example, a pattern on the first 14 moves of apr2912 file is printed as the following:

```
=== Pattern for apr2912.sgf ===
lev= 6  num moves = 4
W:100000000000:::177147
N:020000000001:::118099
E:000000000000:::0
S:200000000000:::354294
----------------------------
lev= 7  num moves = 10
W:00000000100010:::246
N:02000201001020:::1076766
E:01002002000000:::572265
S:00000000000000:::0
----------------------------
```

4. Comparison of Multiple Patterns

Patterns p1 and p2 from two different games are the same if at least one of two conditions is true.

Condition 1:

Tengen data in the two patterns are equivalent.
Square patterns are equivalent for each level.

Condition 2:

The pattern $p1^r$•• • •••• •••• •••• ••••••rotating p1 clockwise 90 or 180 or 270 degrees, is equivalent to p2.

Condition 3:

The pattern $p1^s$• • •••••••••••• ••• •• ••••••••••• •••••
• • • • • • •••• ••••••••• • ••••• ••• p1, is equivalent to p2.

Method **samePattern**( ) computes the value of the condition 1.

In the following we show outlines of the method:

```
    static boolean samePattern( PatternBoardS p1,
PatternBoardS p2) {
        boolean   same = true;
        for (int k=1; k<=9; k++){ same =
p1.patlevel[k].no_moves_on ==
p2.patlevel[k].no_moves_on;
        if (!same)
{System.out.println("no_moves_on?"); return
(same);};
        int edge1ws =
Integer.parseInt(String.copyValueOf(p1.patlevel[k].ws,
0, 2*k), 3);
      . . .
        int edge2st =
Integer.parseInt(String.copyValueOf(p2.patlevel[k].st,
0, 2*k), 3);
       same = (edge1ws == edge2ws) &&
(edge1nr == edge2nr) && (edge1es == edge2es) &&
(edge1st == edge2st);
            if (!same) {System.out.println("diff
patlevel?"); return (same);}
        }
        same = p1.NineNine == p2.NineNine;
        if (!same) {System.out.println("NineNine?");
return (same);};
        return (same);
    }
```

To check condition 2 and 3, method **rotatesClockwise**( ) and **diagSymmBd**( ) are necessary.

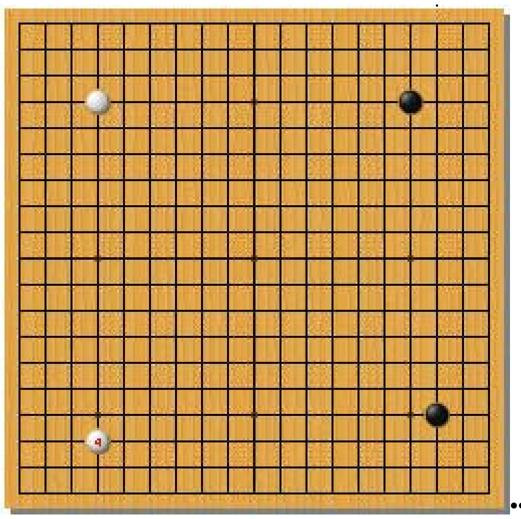


Figure 3. fuseki1.sgf

The player could have played above 14 games with either black or white. However, on the actual recording of the games, the player mostly played as white.

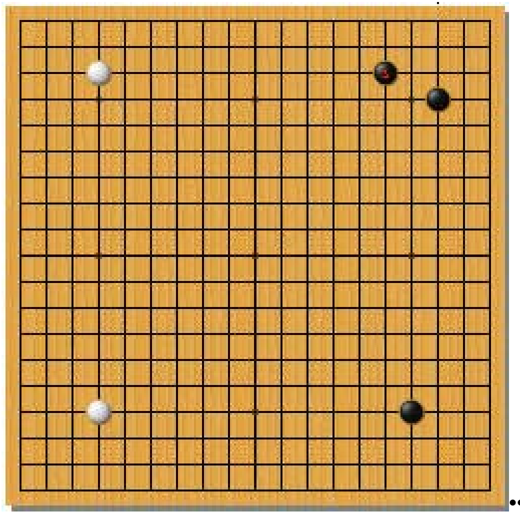


Figure 4. fuseki2.sgf

Table 1 Extracting games with same fuseki pattern

	122 games by player A			128 games by player B			250 games		
	rotated	symm-etrical	same pattern	rotated	symm-etrical	same pattern	rotated	symm-etrical	same pattern
fuseki1	14	10	24	17	12	29	31	22	53
fuseki2	6	1	7	5	1	6	11	2	13

Table 1 shows that fuseki1 was adopted for opening in more than 20 % of go-games.

In order to express go-board pattern at n-th move, **addSgfMoves** should be extended to adjust the pattern after removing the connected stone group with no liberty[11].

5. Conclusion

We have proposed a new 3-dimensional coordinate system for a plane figure whose point location (area, level, offset) is easily computed from a point location in Cartesian coordinate system. We have defined the pattern of the go-board using the point state of the point in our new coordinate system.

In addition, we have defined necessary conditions for two patterns to be equivalent, and we also implemented several methods for comparing patterns in Java to confirm our methodology can easily locate a specific or a desired pattern out of multiple games.

Current method of adding a move to the go-board should be extended in order to express go-board pattern at n-th move.

••

References

[1] M. Muller, Computer Go, Artificial Intelligence, Vol. 134, Issues 1-2, 2002, pages 145-179.

[2] Jan van der Steen, http://gobase.org/.

[3] T. Cazenave, Generation of Patterns With External Conditions for the Game of Go, Advances in Computer Games 9, 2001.

[4] R. Coulom, Computing Elo Ratings of Move Patterns in the Game of Go, ICGA Computer Games Workshop, Amsterdam, The Netherlands, 2007.

[5] http://www.ruijiang.com/multigo/.

[6] http://www.red-bean.com/sgf/go.html.

[7] http://www.tygem.com.

[8] R.O. Duda, P.E. Hart, and D.G. Stork, Pattern Classification, John Wiley & Sons Inc., 2001

[9] F.S. Hill, Jr., Computer Graphics Using OpenGL, 2nd Ed., Prentice Hall, 2001.

[10] http://senseis.xmp.net/?GoTerms.

[11] R. Vila, T. Cazenave, "When one eye is sufficient", Advances in Computer Games 10, pp. 109-124, Kluwer 2003.

[12] W. Savitch, JAVA - An Intro. to Computer Science & Programming, Pearson Prentice Hall, 2004.

[13] K. Arnold, J. Gosling, and D. Holmes, The Java Programming Language Third Edition, Addison Weley, 2000.