卒業研究報告書

題目

4 人版リバーシ Yonin の解析

指導教員

石水 隆　講師

報告者

11–1–037–0080

藤本 侑花

近畿大学理工学部情報学科

平成 28 年 1 月 29 日提出

概要

本論文は通常 2 人でするリバーシを 4 人版に拡張した Yonin[1] について述べる。2 人版リバーシとは違う Yonin ならではの性質があるのか研究しする。通常のリバーシは二人零和有限確定完全情報ゲームであり,Yonin はその内プレイヤ人数以外の要素を引き継いでいる。二人零和有限確定完全情報ゲームにおいて強い AI を作るためには先読み手数を大きくすればよい。これはプレイヤ人数以外の要素を引き継いでいる Yonin でも同様である。では、通常のリバーシと Yonin の性質に違いはあるのか、いろいろな戦略を使った AI を用いて Yonin の解析を行い、Yonin にゲームとしてどのような性質があるのか追求する。

目次

1 序論

1.1 本研究の背景

Yonin リバーシは 2 人版リバーシを拡張したものである。2 人版リバーシと違い、陣地があり、ディスクを置けるところが狭くなっている。このようにゲームを拡張することにより、新たなゲームとして生まれ変わることができる。拡張することによって生まれた変化や性質はどのようなものがあるのか解析していく。

1.2 リバーシのバリエーション

リバーシにはいろいろなバリエーションがある。

盤面のサイズが 6 × 6 の『ミニオセロ』や 10 × 10 の『グランドオセロ』などがある。グランドオセロの各角から 6 マス取り除いた八角形状の盤を用いる『88 オセロ』や、盤面を円状にすることによって角をなくし終盤でも逆転できるようにした『ニップ』[2] というゲームもある。

多人数で対戦できるリバーシとしては、四面体の石を使用する『みんなでオセロ』[3] などがある。同様に多人数で対戦できるリバーシとして Yonin[1] がある。Yonin は通常のリバーシで使う 8x8 盤面と石をそのまま使用して 4 人でプレイできるように拡張さたリバーシである。

1.3 リバーシの解析

リバーシは二人零和有限確定完全情報ゲームに分類される。二人零和有限確定完全情報ゲームとは、以下の条件を満たすゲームである。

- 零和:プレイヤーの利得の合計が 0 になるゲームである。
- 有限:プレイヤーの指し手の組み合わせが有限であるゲームである。
- 確定:ランダム性が無いゲームである。
- 完全情報:各プレイヤーが自分の手番に相手の手も含めて、過去、現在の情報を全て知ることができるゲームである。

二人零和有限確定完全情報ゲームは双方最善手を指した場合、先手勝ち、後手勝ち、引き分けのどれになるかはゲーム開始時点で決定しており、理論上、全ての可能な局面を解析することができれば最善の手を打つことができる。しかし多くのボードゲームでは、可能な局面の総数が極めて大きいため，完全解析を行うことは不可能である。

リバーシも可能な局面数が非常に大きいため、完全解析はできていない。また、リバーシのバリエーションのうち，ミニオセロは完全解析されており。最善手を打ち合った場合，16 対 20 で後手が勝利する [5]。グランドオセロは盤面が広く、着手できる箇所が多いため解析されいない。88 オセロ，ニップ，みんなでオセロなども解析は進んでいない。

Yonin の場合は、二人ゲームではないために完全解析は不可能である。これは自分以外の 3 人のプレイヤーの着手が不確定要素として働くためである。二人ゲームであれば、相手プレイヤーが最善手以外の着手をした場合、即座に自分にとっては有利となる。しかし 3 人以上のプレイヤーがいる場合、あるプレイヤーが不利な手を打った場合に、それがどのプレイヤーの利得となるかはわからない。このため、一般に 3 人以上でプレイ

するゲームは完全解析はできない。

1.4 リバーシについての着手選択の手法

前節で述べた通り、リバーシは完全解析はされていない。しかし、最善ではないがより良い手を求める手法として、局面の評価、一定手数の先読み、定石データベース、対戦データベースの利用、終盤での完全読みなどを用いることにより、人間では勝てない非常に強いオセロ AI を作ることができる。

局面の評価とはその局面での石の数は位置、石同士の繋がり具合等を数値化し、有利不利を点数化する方法である。このとき、評価値が最も高くなる手を選択する。局面の評価法の代表的なものに評価マップがある。これは、盤面の各マスに価値を設定し各マスに自石があればそのマスの価値を足し、敵石があればそのマスの価値を引くことで局面を評価する手法である。

一定手数の先読みは可能な範囲で一定数の先の手を読み、その手から作られる局面の評価値を求め、最も評価値が高い手を採用することである。

定石データベースはリバーシの定石をデータベース化し、各局面で有効な定石があればそれに従って打つという手法である。定石データベースを使用することで強いリバーシプログラムとなる。しかし、相手があえて定石以外の手を打つなどして、データベースに無い局面が出てきたときにはこの手法は使えない。

対戦データベースの利用は過去の対戦において、その手が有効であったかどうかを対戦結果から判定し、データベースに蓄える。数多く対戦することで、その手が有効かどうかより精度が高い判定をすることができる。対戦データベースを用いることで、対戦経験が増えるにつれて強くなる人工知能型のリバーシプログラムになる。対戦データベースを使うためには、事前に繰り返し対戦して学習しておく必要がある。しかし、学習が足りなかったり、事前の対戦でなかった手を打たれたりした場合には使えないという欠点がある。

ゲーム終盤になるとそこから勝負が付くまでの手数が少なくなり、また指せる手が限定されてくるため、勝負が付くまで読み切ることが可能となる。終盤での読みの手法として必勝先読みと完全読みがある。ゲームの終盤で勝敗のみを読み切ることで必勝の手を打つことを目指すのが必勝読み、終盤から得られる全ての局面を読み、最も点数の高くなる手を指すのが完全読みである。必勝先読みの方が計算時間が少なくて済む為、一般にまず必勝読みで勝ちを確定させた上で、残り少なくなると完全読みに切り替えてより点数の高い勝ちを目指すことが多い。

1.5 本報告書の目的

リバーシのバリエーションがたくさんあるなかで、拡張の仕方によって性質が変化すると考えている。1.3 節で述べた通り、Yonin は二人ゲームではないため、本質的に解析は不可能である。そのために、従って、Yonin の性質は実験的に評価するしかない。そこで本研究ではリバーシのバリエーションの 1 つである Yonin に対して、限られた先読み手数の中で探索を行う AI を作成し、通常のリバーシを拡張することによってどのような性質の変化があるか考察する。

1.6 本報告書の構成

本報告書の構成は以下の通りである。

まず第 2 章において、本研究の対象となる Yonin について説明する。続いて第 3 章では本研究で作成した Yonin の AI が打つ手を決定する手法について述べる。第 4 章では作成した AI の性能を見るためランダムに候補手を打つ AI との対戦結果を記す。第 5 章では第 4 章の結果について考察し、続いて結論を述べる。

2 Yonin について

まず Yonin について簡単に説明する。先に述べたように Yonin はリバーシを 4 人でプレイできるように拡張させたゲームである。使う盤面と石は通常のリバーシと同じものを使用する。

図 1　4 人版リバーシの 4 領域分割

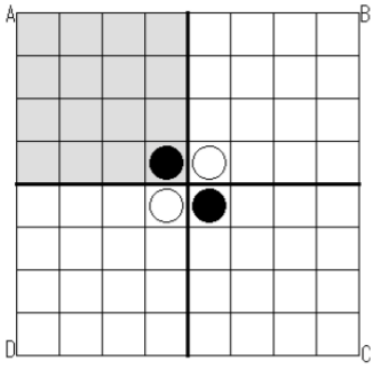
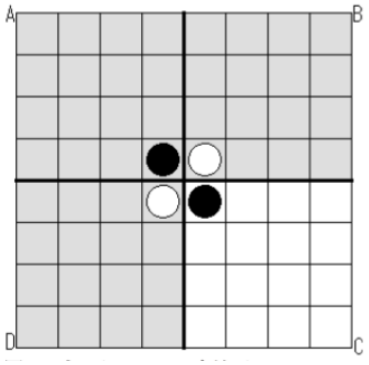


図 2　プレイヤ A の着手可能マス



2.1 盤・陣地

図 1 のように 8 × 8 の盤を 4 × 4 のサイズに 4 分割する。4 人のプレイヤを図 1 のように配置し、それぞれの陣地とする。

2.2 着手

リバーシと同様に自分の色の石で相手の色の石を挟み、自分の色の石に変える。ただし、図 2 のように自分の陣地の対面の陣地には着手できない。

2.3 パス

通常のリバーシ同様必ず自分の色の石で違う色の石を挟み、裏返さなければならないが、このような差し手が存在しないときはパスとなる。

2.4 配置・順番

何らかの方法でプレイヤの位置と最初に着手するプレイヤを決める。手番は順に隣のプレイヤへと移行する。すなわち順番は、時計回りか反時計回りて試合に途中に手番はジャンプしない。

配置は対面するプレイヤと石の色は同じになる。よって自分の左右のプレイヤの石の色は自分と違う色になる。

2.5 ゲーム終了条件と勝敗の判定

2 人版リバーシと同様にすべてのプレイヤが石を置けなくなるとゲームは終了する。勝敗の判定は、ゲーム終了時の自分の陣地にある自分の色の石の数で決まる。

3 研究内容

3.1 Yonin の AI

本研究では、Yonin の性質を実験的に評価するために、異なる戦略に従う複数の YoninAI を作成した。付録に本研究で作成した YoninAI のソースプログラムを示す。

3.2 評価関数

本研究で作成した AI は打てる手が複数ある場合、その手をうった場合に得られる局面を先読みし、評価値を用いて手を決定する。

局面を評価する代表的な手法として評価マップの使用がある。評価マップとは盤面の各マスに価値 を設定し、マスに自分の石が置かれている場合は評価に価値を足し、相手の石がおかれている場合は評価から価値を引くというものである。通常のリバーシでは角のマスの価値を高く、角に隣接するマスの価値を低く設定するのが良いとされている。本研究では、通常のリバーシの評価値マップを元に、Yonin 特有の要素である陣地を利用して変更を加えた評価値マップを使った AI を作成する.

また、通常のリバーシでは自駒と相手駒が明確に分ることができるため、評価値は、各マスに付与した値に自駒ならプラスを、相手駒ならマイナスをかけることで求めることができる。しかし、Yonin では自分と同じ色の石を使う相手プレイヤーが存在している。そのため自分の打った石を相手に利用されることや、逆に相手の打った石を自分が利用することができ、盤面上の同色の石を自駒と相手駒に明確に分けることは困難となっている。このことから本研究で使う AI の評価関数では各マスに付与した値に自色と同じ色の石が置かれていればプラスを、自色と異なる色の石が置かれていればマイナスをかけて評価値を求める。

3.3 AI 戦略

3.1 節で述べたように、本研究では異なる戦略を持つ複数の YoninAI を作成する。本研究で作成する AI は評価値マップを使ったものであるが、この評価値の割り当て方により戦略が変わってくる。

自分の陣地の左右や対面の評価値を高くすることによって勝敗にどのような変化がでるのか解析する。本研究で用いた戦略は、自分の陣地の評価値を高くする Self、自分の陣地と右のプレイヤの陣地の評価 値を高く

する Right、対面のプレイヤの陣地の評価値を高くする Opposite、自分の陣地と左のプレイヤの陣地の評価値を高くする Left、陣地による補正をかけない Normal の 5 つである。

表 1　Self の評価値マップ

100	-30	0	-1	-51	-50	-80	50
-30	-50	-3	-3	-53	-53	-100	-80
0	-3	0	-1	-51	-50	-53	-50
-1	-3	-1	0	-50	-51	-53	-51
-51	-53	-51	-50	-50	-51	-53	-51
-50	-53	-50	-51	-51	-50	-53	-50
-80	-100	-53	-53	-53	-53	-100	-80
50	-80	-50	-51	-51	-50	-80	50

表 2　Right の評価値マップ

100	-30	0	-1	-51	-50	-80	50
-30	-50	-3	-3	-53	-53	-100	-80
0	-3	0	-1	-51	-50	-53	-50
-1	-3	-1	0	-50	-51	-53	-51
-21	-23	-21	-20	-50	-51	-53	51
-20	-23	-20	-21	-51	-50	-53	-50
-50	-70	-23	-23	-53	-53	-100	-80
80	-50	-20	-21	-51	-50	-80	50

表 3　Opposite の評価値マップ

100	-30	0	-1	-51	-50	-80	50
-30	-50	-3	-3	-53	-53	-100	-80
0	-3	0	-1	-51	-50	-53	-50
-1	-3	-1	0	-50	-51	-53	-51
-51	-53	-51	-50	-20	-21	-23	-21
50	-53	-50	-51	-21	-20	-23	-20
-80	-100	-53	-53	-23	-23	-70	-50
50	-80	-50	-51	-21	-20	-50	80

表 4　Left の評価値マップ

100	-30	0	-1	-21	-20	-50	80
-30	-50	-3	-3	-23	-23	-70	-50
0	-3	0	-1	-21	-20	-23	-20
-1	-3	-1	0	-20	-21	-23	-21
-51	-53	-51	-50	-50	-51	-53	-51
-50	-53	-50	-51	-51	-50	-53	-50
-80	-100	-53	-53	-53	-53	-100	-80
50	-80	-50	-51	-51	-50	-80	50

表 5　Normal の評価値マップ

100	-30	0	-1	-1	0	-30	100
-30	-50	-3	-3	-3	-3	-50	-30
0	-3	0	-1	-1	0	-3	0
-1	-3	-1	0	0	-1	-3	-1
-1	-3	-1	0	0	-1	-3	-1
0	-3	0	-1	-1	0	-3	0
-30	-50	-3	-3	-3	-3	-50	-30
100	-30	0	-1	-1	0	-30	100

4　実験結果

本研究では, 先読み手数を 4 として先に挙げた 4 つ種類の戦略のプログラムとランダムに手を打つプログラムを総当 たりでそれぞれ 100 回対戦させた。表 6 にその対戦結果を示す。

表 6　対戦結果

	Self	Right	Opposite	Lest	Normal
1位回数	32	19	22	27	30
最下位回数	16	25	14	15	18
平均順位	2.2	2.8	2.4	2.6	2.5

表 6 より、戦略により、順位が変化することが示される。自分の陣地の評価値を高くした Self は 1 位回数、最下位回数、平均順位がすべて、Normal よりも良い結果がでた。そして、右側の陣地の評価値を高くした Right は Normal に比べて 悪い結果になった。

表 7　順位を変えた場合の対戦結果

プレイ順	1番目	2番目	3番目	4番目
1位回数	32	35	36	31
最下位回数	18	19	20	19
平均順位	2.4	2.1	2.1	2.5

表 7 はプレイ順を変えて、ランダムと対戦させた結果で ある。結果は順位は多少の変化があるものの、どの手順が 有利ということはないと思われる。2 番目と 3 番目の順位 がよくなっている程度で目立った有利不利はない。

5　結果・考察

第 4 章で示した結果から、通常のリバーシと違い陣地があることを利用し評価値マップを作ることで、補正をかけていない AI よりも良い結果を出すことができる。しかし、AI よりも悪い結果になる場合もあると推測される。

Self と Opposite が他の戦略より順位が良かったのは同じ色の石を使用していたからではないかと考えられる。自らの利益を最大化するためには、対面のプレイヤと自然と協調関係が構築されることを考慮していかなければならない。そしてこれが 4 人版リバーシ Yonin の最大の特徴だと言えるだろう。

そして人数が増えれば順番で順位に大 きな変化が出ると考えていたが、簡単なプログラムでは目立った変化が出ないということもわかった。

本研究では、Yonin の最適な戦略を検証するたに、各マス に付加する重みを変化させてその優劣を検証した。今回の 研究は簡単なプログラムだったので、先読みの深さが小さ く、正確な結果が出ていない。従って、先読みの深さを変化 できるプログラムをつくるのが今後の課題であると考える。

6　謝辞

本論文を作成するにあたり、指導教官の石水隆講師から、丁寧かつ熱心なご指導を賜りました。ここに感謝の意を表します。

参考文献

[1] 藤井昌典, 北隼人, 村田朋紀, 橋本隼一, 飯田弘之:4 人版リバーシ Yonin, 情報処理学会研究報告 2006-GI-015,pp.73-80 (2006).
http://id.nii.ac.jp/1001/00058524/

[2] Nipp - アブストラクトゲーム博物館, http://www.nakajim.net/index.php?Nipp

[3] みんなでオセロ, メガハウスのおもちゃ情報サイト,(2013), http://www.megahouse.co.jp/megatoy/products/item/1139/

[4] Seal software：リバーシのアルゴリズム　C++ & Java 対応、工学社 (2003).

[5] Joel Feinstein, Amenor Wins World 6x6 Championships!, Forty billion noted under the tree (July 1993), pp.6–8, British Othello Federation's news,
http://www.britishothello.org.uk/fbnall.pdf

[6] 垰田基成, 4 人版リバーシ Yonin の解析, 近畿大学 理工学部 情報学科 平成 23 年度卒業研究報告書, (2014)
http://www.info.kindai.ac.jp/~takasi-i/thesis/2013_10-1-037-0040_M_Taoda_thesis.pdf

付録 A　付録

以下に本研究で作成した Yonin の AI である Self のプログラムを示す。Self 以外の AI に関しては評価値の設定以外同一のプログラムとしているので省略する。[4]

```
public class SelfStrategy extends ValueStrategy{

        public SelfStrategy(int player_no){
                setValue(player_no);
        }

        /**
         * プレーヤーが自身の手を指定する
         * @param player_no
         */

        @Override
        public Point decide(Player player) {
                ArrayList<Point> possibles = player.getPossibles();
                Board board = player.board;
                int leng;
                if(possibles.size() > 5){
                        int[] vals = new int[possibles.size()];
                        for(int i=0; i<possibles.size(); i++){
                                Point point = possibles.get(i);
                                board.putDisc(point.x, point.y, player.getColor());
                                vals[i] = calculateValue(player,board);
                                board.undo();
                        }
                        possibles = sort(possibles, vals, player);
                        leng = 5;
                }else{
                        leng = possibles.size();
                }
                int[] vals = new int[leng];
                for(int i=0; i<leng; i++){
                        vals[i] = getValue(player, this.depth);
                }
                int max = 0;
                for(int i=1; i<leng; i++){
                        if(vals[max] == vals[i]){
                                Random r = new Random();
                                if(r.nextInt(2)==0){
                                        max = i;
                                }
                        }else if(player.getColor() == Board.WHITE){
                                if(vals[max] > vals[i]){
                                        max = i;
                                }
                        }else{
```

```
                        if(vals[max] < vals[i]){
                                max = i;
                        }
                }
        }
        return possibles.get(max);
}
/**
 * 評価値の設定
 * @param player_no
 */

@Override
public void setValue(int player_no){
        for(int i=0; i<Board.Y_SIZE/2; i++){
                for(int j=0; j<Board.X_SIZE/2; j++){
                        if(player_no ==1){
                                values[i][j] +=  0;
                        }else if(player_no ==2){
                                values[i][j] -= 50;
                        }else if(player_no ==3){
                                values[i][j] -= 50;
                        }else if(player_no ==4){
                                values[i][j] -= 50;
                        }
                }
        }
        for(int i=0; i<Board.Y_SIZE/2; i++){
                for(int j=4; j<Board.X_SIZE; j++){
                        if(player_no ==1){
                                values[i][j] -= 50;
                        }else if(player_no ==2){
                                values[i][j] +=  0;
                        }else if(player_no ==3){
                                values[i][j] -= 50;
                        }else if(player_no ==4){
                                values[i][j] -= 50;
                        }
                }
        }
        for(int i=4; i<Board.Y_SIZE; i++){
                for(int j=4; j<Board.X_SIZE; j++){
                        if(player_no ==1){
                                values[i][j] -= 50;
                        }else if(player_no ==2){
                                values[i][j] -= 50;
                        }else if(player_no ==3){
                                values[i][j] +=  0;
                        }else if(player_no ==4){
                                values[i][j] -= 50;
                        }
                }
        }
```

```
        for(int i=4; i<Board.Y_SIZE; i++){
                for(int j=0; j<Board.X_SIZE/2; j++){
                        if(player_no ==1){
                                values[i][j] -= 50;
                        }else if(player_no ==2){
                                values[i][j] -= 50;
                        }else if(player_no ==3){
                                values[i][j] -= 50;
                        }else if(player_no ==4){
                                values[i][j] +=  0;
                        }
                }
        }
}



/**
 * 評価値の計算
 * @param player
 * @param depth
 * @return
 */
@Override
public int getValue(Player player, int depth) {
        Board board = player.board;
        if(depth == 0 || board.isgameOver()){
                int pn;
                if(player.getPlayerNumber()==1){
                        pn = 4;
                }else{
                        pn = player.getPlayerNumber()-1;
                }
                return calculateValue(board.getPlayer(pn-1), board);
        }else{
                player.setPossibles();
                ArrayList<Point> possibles = player.getPossibles();
                int size = possibles.size();
                if(size == 0){
                        board.pass();
                        Player pl = board.getPlayer(board.currentPlayer);
                        int val = getValue(pl, depth-1);
                        board.undo();
                        return val;
                }else{
                        int[] vals = new int[size];
                        for(int i=0; i<size; i++){
                                Point p = possibles.get(i);
                                board.putDisc(p.x, p.y, player.getColor());
                                Player pl = board.getPlayer(board.currentPlayer);
                                int v = getValue(pl, depth-1);
                                vals[i] = v;
                                board.undo();
```

```
                        }
                        int val = vals[0];
                        for(int i=1; i<size; i++){
                                if(player.getColor() == Board.WHITE){
                                        if(val > vals[i]){
                                                val = vals[i];
                                        }
                                }else{
                                        if(val < vals[i]){
                                                val = vals[i];
                                        }
                                }
                        }
                        return val;
                }
        }
    }

    @Override
    public int calculateValue(Player player, Board board) {
        int val=0;
        setValue(player.getPlayerNumber());
        for(int i=0; i<Board.Y_SIZE; i++){
                for(int j=0; j<Board.X_SIZE; j++){
                        int v = board.getPoint(j, i) * values[i][j];
                        val += v;
                }
        }
        return val;
    }

/*  @Override
    public Point decide(Player player) {
        ArrayList<Point> possibles = player.getPossibles();
        Board board = player.board;
        int leng;
        if(possibles.size() > 4){
                int[] vals = new int[possibles.size()];
                for(int i=0; i<possibles.size(); i++){
                        Point point = possibles.get(i);
                        board.putDisc(point.x, point.y, player.getColor());
                        vals[i] = calculateValue(player,board);
                        board.undo();
                }
                possibles = sort(possibles, vals, player);
                leng = 4;
        }else{
                leng = possibles.size();
        }
        int[] vals = new int[leng];
        for(int i=0; i<leng; i++){
                vals[i] = getValue(player, this.depth);
        }
```

```
                int max = 0;
                for(int i=1; i<leng; i++){
                        if(vals[max] == vals[i]){
                                Random r = new Random();
                                if(r.nextInt(2)==0){
                                        max = i;
                                }
                        }else if(player.getColor() == Board.WHITE){
                                if(vals[max] > vals[i]){
                                        max = i;
                                }
                        }else{
                                if(vals[max] < vals[i]){
                                        max = i;
                                }
                        }
                }
                return possibles.get(max);
        }


        @Override
        public int getValue(Player player, int depth) {

                return 0;
        }

        @Override
        public int calculateValue(Player player, Board board) {

                return 0;
        }*/
}
```