卒業研究報告書

題目

モンテカルロ法を用いたオセロプログラム

指導教員

石水　隆　講師

報告者

11-1-037-0170

上野　早也香

近畿大学理工学部情報学科

平成27年1月31日提出

概要

オセロは二人零和有限確定完全情報ゲームである。８×８のマスのある盤面に、交互に石を打ち、相手の石を挟んで自分の石の色に変える。最終的に石の多いプレイヤーが勝利するというルールである。最大手順は60しかないが、局面の組み合わせは莫大であるため完全解析はされていない。一方、オセロ AI の研究は進んでおり、現在トップクラスのオセロ AI は人間を上回る。オセロ AI の一般的な手法として、定石データベースの利用や盤面の評価関数、終盤での完全読み等が挙げられており、研究が進められている。本研究では、近年、囲碁の主力アルゴリズムになりつつあるモンテカルロ法を用いたオセロ AI を作成する。

モンテカルロ法とは、乱数を用いてシミュレーションを実行する計算手法の総称である。ある1つの局面を評価するときに、その局面からランダムに終局までシミュレーションを行い、結果を得る。この操作を何度も繰り返すことで勝率を求め、勝率が一番高い局面の手を優れた手として判断するという手法である。

目次

１　序論

1.1二人零和有限確定完全情報ゲーム

オセロ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。[2]

- 二人または２チームでゲームを行う。
- 勝ちをプラス１、負けをマイナス１、引き分けを０としたとき、最終的に全プレイヤーの数値の和が零になる。
- 全プレイヤーの着手可能手が有限である。
- プレイヤーの着手によって、ゲームの進行が確定する。
- 局面の情報が完全に全プレイヤーに見えている。

二人零和有限確定完全情報ゲームは双方最善手を打った場合、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。例を挙げれば、可能な局面数はオセロが 10^{28} 通り、チェスが 10^{50} 通り、将棋が 10^{69} 通り、囲碁が 10^{170} 通り程度あるとされており、現在の計算機の性能を越えている。一方、可能な局面数が少ないゲームでは完全解析されているものもある。連珠は双方最善手を打った場合、47 手で先手が勝つ[10]。チェッカーは双方最善手を指すと引き分けとなる[11]。

局面数が大きいゲームについては、ゲーム盤をより小さいサイズに限定した場合の解析も行われている。囲碁については、サイズ 4x4 の囲碁は双方最善手を打つと持碁(引き分け)[12]、5x5 の囲碁は黒の 25 目勝ちとなる[13]。オセロについては、J.Feinstein がサイズ 6x6 のミニオセロの完全解析を行い、双方最善手を打つと 16 対 20 で後手勝ちとなることを示した[14]。

1.2オセロのルール

オセロはボードゲームの１つで、８×８のマスのある盤面で行う。初めに中心の２×２のマスに交差させるように白石と黒石を並べた状態で、黒石側のプレイヤーを先手としてゲームを開始する。縦・横・斜めの合計８方向のどこかで、自分の色の石で相手の色の石を挟めるマスに石を置くことができ、できない場合のみパスをすることができる。石を置いた時、自分の色の石で挟んだ相手の色の石を裏返して、自分の色の石にすることができる。交互に着手し、双方のプレイヤーとも石を挟めなくなった時点でゲームが終了する。ゲームが終了した時点で，石の数が多いプレイヤーが勝ちとなり、白石と黒石が同数の場合は引き分けとなる。

1.3 ゲーム AI の手法

可能な局面数が多いゲームに対して完全解析を行うことは困難である。そのようなゲームに対しては確実な最適手を得ることはできないが、局面の評価値計算、定跡・定石データベース、一定手数の先読み、終盤での必勝読みと完全読み、モンテカルロ法[4]などを用いてより有利だと思われる手を選択することができる。

1.3.1 局面の評価値計算

局面の評価値計算とは、盤上にある石・駒の種類やその位置,着手可能手の数、各駒の稼働範囲 等から計算される評価関数を作成し、その値により優劣を判定する手法である。AIの強さは評価関数 の作り方でに応じて決まるため、評価関数はできるだけ戦局を適切に評価できるように工夫して作る 必要がある。

1.3.2 定跡・定石データベース

定跡・定石データベースとは、ゲームの定跡・定石をデータベース化し、各局面で有効な定跡・定石があればそれに従って指す、または打つという手法であり、特にゲーム序盤においては有効である。定跡・定石データベースを使用することで、強い手を選択することができる。しかし、相手があえて定跡・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。

1.3.3 先読み

先読みとは、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。一般に先読みする手数が多いほど強い AI となるが、先読み手数の増加に伴い探索時間が指数的に増えるため、適度に枝刈りをして探索範囲を減らす工夫をする必要がある。

1.3.4 必勝読みと完全読み

ゲーム終盤になるとし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。終盤での読みは、必勝読みと完全読みがある。必勝読みとはゲーム終盤で勝敗のみを読み切り、必ず勝てる手を指すことを言う。完全読みとは、そこ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ことを言う。必勝読みの方が計算時間が少なくてすむため、一般にまず必勝読みで勝ちを確定させた上で、残り手数が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。

1.3.5 モンテカルロ法

モンテカルロ法とは、各着手可能手に対し、その手から先終局までをランダムに指し勝敗判定を 行うという作業を数千~数万回繰り返し、最も勝率の高い着手可能手を採用するというものである。この手法は将棋ではあまり使われないが、局面数が極めて多い囲碁プログラムでは最近主流になっている[4]。

以上の手法を用いることにより、完全解析を行わなくてもある程度の強さのプログラムを作ることが可能であり、ゲームによってはプロに勝つこともできる。

1.4既存の AI

オセロプログラムは、定石データベースの利用、序盤の先読み、中盤の評価関数の利用、終盤での完全読みの手法で作成されることが多く、それらの手法の組み合わせや値の調整で性能に変化を与えている。手法として定石データベースのみを用いたオセロプログラムとして、Logistello[16]がある。Logistello は1997年には村上健八段(当時の世界チャンピオン)に勝利しており[22]、定石データベースを使うことでそれなりの強さのプログラムとなることが示された。しかし、Logistello は定石データベースのみを使用するため、定石以外の手を打たれた場合には対処できない。また、その他に Zebra[17]や Edax[18]などもある。これらについても book と呼ばれる序盤・中盤の定石データや対戦データベースを使用しているが、中盤の先読みと終盤の完全読みを行っている。特に Edax はマルチコア対応で高速処理が可能となっているが、 book 非使用の時は勝率が下がるというのが一般的な見方である[19]。2015年現在の既存のトップクラスのオセロ AI として、WZebra[5]、MasterReversi[6]等が挙げられる。

1.5本研究の目的

囲碁の主力アルゴリズムになりつつあるモンテカルロ法であるが、オセロでもモンテカルロ法は有効な戦略と成り得る可能性がある。しかし、既存のオセロプログラムで実装しているものは多くない。よって、本研究ではモンテカルロ法を用いたオセロAIを作成し、その性能の変化を調査することを目的とする。

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章で、作成したオセロ AI の手法について述べる。第３章では、オセロ AI を用いた実験結果を報告し、考察を述べる。第４章では、研究の結論と、今後の課題を示す。

２　研究内容

本章では、研究で作成した、モンテカルロ法を用いたオセロプログラムについて述べる。

2.1着手可能手の探索

ある局面での着手可能手の探索は以下の手順で行われる。
盤上の各相手色石に対して以下の処理を行う。まずその周囲8マスを調べる。空きマス、自色石、 盤端のいずれかであればその方向への探索を終了する。相手色石であれば、その方向へ相手色石が続 く限り探索する。空きマスまたは盤端にぶつかればその方向への探索を終了する。自色石にぶつかれば、今度は反対方向へ相手色石が続く限り探索する。自色石または盤端にぶつかればその方向への探索を終了する。空きマスにぶつかれば、その空きマスは着手可能手である。この操作を8方向全てに行う。

全ての相手色石に対して上記の操作を行った後、着手可能手の条件を満たさなかった空きマスは着手可能手ではない。

2.2着手可能手の選択

本節では、前節で探索した着手可能手のうちどの手を採用するのか、その戦略について述べる。 本研究では、オセロでも有効な戦略と成り得る可能性があるモンテカルロ法を戦略として採用する。

モンテカルロ法を用いたオセロAIの手順を説明する。まず、１つの局面において、着手可能手の中からひとつを選び、双方がランダムに終局まで打ち続けるシミュレーションを試行回数行い、結果を得る。結果を用いて評価値を計算する。これを全ての着手可能手で行い、各手の評価値を求める。着手可能手のうち、もっとも評価値が高い手を次の一手とする。

本研究での評価値はシミレーション毎に黒石の数(モンテカルロ法を用いたオセロAI)から白石の数(対戦したオセロAI)を引いた数値の合計とする。

2.3オセロ AI プログラム

本節では、オセロ AI プログラムについて述べる。付録１に本研究で作成したクラスである MntAI のソースプログラムを示す。

2.3.1 MntAI クラス

MntAI クラスは、オセロ AI の戦略を決定するクラスである。

表 1 に MntAI クラスのメソッドを示す。

表 1．MntAI クラスのメソッド

メソッド	処理内容
void move(ConsoleBoard board)	着手可能手の選択を行う
int vertualGame(ConsoleBoard vB,Point MntPoint)	シミュレーションを行う
int besteval(**int**[] valList)	次の一手(評価値が一番高い手)を選ぶ
AI makeRanAI()	シミュレーションでの仮想ボードを動かす

void move(ConsoleBoard board)は着手可能手の選択を行うメソッドである。board は現在の盤面を表す引き数であり、メソッド実行後、着手可能手は kakuteiP に格納される。

int vertualGame(ConsoleBoard vB,Point MntPoint)は決定した着手可能手を打った後、双方がランダムに終局まで打ち続けるシミュレーションを行うメソッドである。仮想ボードを作成し、ランダムに打つ AI(RanAI クラス)を利用して終局まで打つ。vB にシミュレーション用の仮想ボードの盤面を格納する。

int besteval(int[] valList)は評価値が一番高い手を選ぶメソッドである。着手可能手のそれぞれの評価値を格納している valList を用いて、一番高い評価値を持つ手を採用する。メソッド実行後、一番高い評価値となったマスを masu に格納して返す。

AI makeRanAI()はシミュレーションでの仮想ボードを動かすメソッドである。 vertualGame メソッドでランダムに打つ AI を使うため、RanAI()を返す。

３　結果・考察

本研究で作成した mnt AI の強さを評価するため、試行回数を変化させた場合の勝率と石の数の変化を調べた。試行回数は５回、１０回、１００回の３種類で行う。

対戦させる AI はランダムに打つ AI(以下 ran AI とする) と、盤面のマスの位置から評価値を得て、打つ手を決定する AI (以下 bp AI とする) の 2 種類とし、それぞれ100 回の対戦を行った。２つのAIとの対戦において、先手（黒石）はすべてmnt AIに設定する。mnt AIの性能の変化を調査するため、終局時の勝利数と「黒石の個数−白石の個数」の平均を求める。mnt Aiの評価の取り方は、黒石の個数−白石の個数を評価値として用いる。

実験中、全てのマスが埋まっていない状態で白石が 0 になった場合、黒石側の完全勝利として、終局時の黒石の個数を６４個全て獲得できたものとする。

bpAIにおいて、評価値の計算方法は、６４マス全てに評価値を置き、自分の色の石が置かれたならばその値を加算し、相手の色の石が置かれたならばその値を減算し、その合計を盤面の評価値とする。本研究での盤面の評価値は図１に示す[1]。

mnt AI と ran AI の対戦結果を表1、 mnt AI と bp AI の対戦結果を表2に示す。

100	−50	10	0	0	10	−50	100
−50	−70	−5	−10	−10	−5	−70	−50
10	−5	10	−5	−5	10	−5	10
0	−10	−5	0	0	−5	−10	0

0	−10	−5	0	0	−5	−10	0
10	−5	10	−5	−5	10	−5	10
−50	−70	–5	−10	−10	−5	−70	−50
100	−50	10	0	0	10	−50	100

図１．盤面の評価値

表２．mntAIとranAIとの対戦結果

試行回数	勝ち	負け	黒石−白石
５回	68	32	26.4
１０回	79	21	39.2
１００回	93	7	50.5

表３．mntAI と bpAI との対戦結果

試行回数	勝ち	負け	黒石−白石
５回	59	41	19.5
１０回	70	30	35.8
１００回	87	13	47.6

表２と表３より、試行回数が大きくなるほど、勝率が上がることが示される。また、 黒石の個数-白石の個数の平均が大きくなることから、mnt AI の強さが変化して性能が良くなっていることが示された。

表２から、ランダムで打つ AI と比べると試行回数が少なくてもはっきりと差がついていることが分かる。試行回数が１００回になると、完全勝利する試合も多くあった。

表３から、試行回数が極端に少ない場合、勝利数にほとんど差はなかったが、黒石の個数-白石の個数の平均からも分かるように、mntAI が勝利した場合は大差を持って勝利することが多い。この原因は、図１に示されるように bpAI は隅のマスを高く評価するが、すでに隅のマスが埋まっている場合には評価の 高い手を選択することができず、そのため多くの石が獲得できなかったと考える。

表２と表３の違いから、AI の勝率は ran AI、bp AI、mnt AI の順番で良くなり、3つの AI の中で mnt AI の性能が優れていることが分かった。

４　結論・今後の課題

本研究ではモンテカルロ法を用いたオセロの AI(mntAI) を作成し、その性能について調査した。mnt AI の評価の取り方は、黒石の個数−白石の個数を評価値として付けた。その結果、AI との対戦では試行回数が大きくなるほど、勝率が上がることが分かった。

今後の課題として、２つの課題が挙げられる。ひとつめは、評価値の取り方を変えることで、どのように mnt AI の性能に 影響するのか検証することが挙げられる。本研究で用いたモンテカルロ法の評価値は「黒石-白石」 の数のみであるが、黒石の個数で評価するなど、研究が必要な評価値の取り

方は多数ある。検証結果によっ て、より良い評価値の取り方への改善が必要である。

次に、既存のトップクラスのオセロ AI にモンテカルロ法を組み込むことでより強いオセロ AI が作成されるのか実験を行うことである。本研究では、オセロにおけるモンテカルロ法の性能を調査したが、その性能がトップクラスのオセロAIでも生かすことができるのか、どのように変化するのか、実験を行うことが課題である。

謝辞

本研究書を作成するにあたり、担当して下さった石水隆先生には大変お世話になりました。ご迷惑をおかけしたことのお詫び、及び丁寧なご指導に大変感謝している気持ちをこの場を借りて心より申し上げます。

参考文献

[1] Seal software, リバーシのアルゴリズム C++ & Java 対応, 工学社 (2003).
[2] 小谷善行 編著, ゲーム計算メカニズム, コロナ社 (2010).
[3] 大筆豊, オセロプログラムの評価関数の改善について, 研究報告ゲーム情報学 (GI), Vol.2003-GI-011, pp.15-20, 情報処理学会, (2004).
[4] 美添一樹, 山下宏, 松原仁, コンピュータ囲碁―モンテカルロ法の理論と実践―, 共立出版 (2012).
[5] 最強オセロ「WZebra」,http://homepage3.nifty.com/akky-han/100529.html .
[6] MasterReversi Home Page, http://homepage2.nifty.com/t_ishii/mr/index.html.
[7] 久保田悠司, 佐藤佳州, 高橋大介, マルチコアプロセッサと SIMD 演算によるモンテカルロ木探索を用いたオセロの実装, 研究報告ゲーム情報学(GI), Vol.2009-GI-22, No.7, pp.1-8, 情報処理学会, (2009), http://id.nii.ac.jp/1001/00062419/.
[8] 谷田邦彦, 図解早わかりオセロ これが必勝のコツだ!! , 日東書院, (2003).
[9] 村山要司,楽しく学べるJavaゲーム・アプレット,工学社(2002).
[10] Janos Wagner and Istvan Virag, Solving renju, ICGA Journal, Vol.24, No.1, pp.30-35 (2001), http://www.sze.hu/~gtakacs/download/wagnervirag_2001.pdf
[11] Jonathan Schaeffer, Neil Burch, Yngvi Bjorsson, Akihiro Kishimoto, Martin Muller, Robert Lake, Paul Lu, and Steve Suphen, Checkers is solved, Science Vol.317, No,5844, pp.1518-1522 (2007). http://www.sciencemag.org/content/317/5844/1518.full.pdf
[12] 清慎一, 川嶋俊:探索プログラムによる四路盤囲碁の解, 情報処理学会研究報告, GI 2000(98), pp.69- -76 (2000), http://id.nii.ac.jp/1001/00058633/
[13] Eric C.D. van der Welf, H.Jaap van den Herik, and Jos W.H.M.Uiterwijk, Solving Go on Small Boards, ICGA Journal, Vol.26, No.2, pp.92-107 (2003).
[14] Joel Feinstein, Amenor Wins World 6x6 Championships!, Forty billion noted under the tree (July 1993), pp.6-8, British Othello Federation's newsletter., (1993), http://www.britishothello.org.uk/fbnall.pdf
[15] Michael Buro,Logistello,https://skatgame.net/mburo/log.html,1997
[16] Gunnar Andersson, WZebra, 2006,http://radagast.se/othello/
[17] Richard Delrme, Ohello programing, 2012, http://abulmo.perso.neuf.fr/index.htm
[18] T.Ishii,MasterReversi,他アプリとの対局結果http://homepage2.nifty.com/t_ishii/mr/gameresult.html,200

付録について

以下に、本研究で作成したモンテカルロ法を用いた MntAI クラスのソースを示す。

```
package osero;

import java.util.ArrayList;
import java.util.Random;
import java.util.Vector;

//モンテカルロ法 AI

public class MntAI extends AI{
	class Move extends Point
	{
		public int eval = 0;
		public Move()
		{
			super(0, 0);
		}

		public Move(int x, int y, int e)
		{
			super(x, y);

			eval = e;
		}
	};

	//private class モンテカルロは仮想が必要
	class RanAIPlayer implements Player
	{

		private AI RanAi = null;

		public RanAIPlayer()
		{
			RanAi = new RanAI();
		}

		public void onTurn(ConsoleBoard board) throws GameOverException
		{

			RanAi.move(board);
			if(board.isGameOver()) throw new GameOverException();
		}
	};
```

```
//試行回数（ここでは５回に設定）
private int sikou_kaisu=5;
private Random rnd = new Random();
private Board vertualBoard;


/**着手可能手の選択を行う**/
public void move(ConsoleBoard board)
{
	BookManager book = new BookManager();
	Vector<Point> movables = book.find(board);

	if(movables.isEmpty())
	{
		// 打てる箇所がなければパスする
		board.pass();
		return;
	}

	if(movables.size() == 1)
	{
		// 打てる箇所が一カ所だけなら探索は行わず、即座に打って返る
		board.move((Point) movables.get(0));
		return;
	}

	int limit;

	//打てる場所を格納する ArrayList
	ArrayList<Point> pointsarray = new ArrayList<Point>();
	ArrayList<Integer> evals = new ArrayList<Integer>();
	//最終的な評価値
	ArrayList<Integer> finalEvals = new ArrayList<Integer>();

	//着手可能手を抽出。
	for(int i=0; i<movables.size(); i++)
	{
		//動ける場所の座標を習得し board.move へ
		board.move((Point) movables.get(i));
		//打てる手をすべてリスト格納
		pointsarray.add((Point)movables.get(i));
	}



	for(int i=0; i<movables.size(); i++){
		//試行する一つを決める。
		Point syori = pointsarray.get(i);
```

```
                    for(int j=0;j<sikou_kaisu;j++){

                            evals.add(vertualGame(board,syori));
                            System.out.println("vertual"+j+"　　回　　終　　了
**************************");
                            board.print();

                    }
                    int plus=0;
                    for(int p=0; p < evals.size(); p++){
                            plus+=evals.get(p);
                    }
                    //その手の最終的な評価値を代入
                    finalEvals.add(plus);
                    System.exit(0);

            }

    }



    /**シミュレーションを行う**/
    private int vertualGame(ConsoleBoard vB,Point MntPoint){

            ConsoleBoard vBoard = new ConsoleBoard();
            System.out.println("%%%%%%%仮想ボード%%%%%%%");
            vBoard.print();

            vBoard.setRawBoard(vB.getRawBoard());


            System.out.println("%%%%%%%仮想ボード2%%%%%%%");
            vBoard.print();

            //評価値
            int evaluation=0;

            //シミュレーション開始
            Player[] player = new Player[2];


            //今どちらのターンかを知る。
            int current_player;
            int fast_turn = vBoard.getCurrentColor();

            fast_turn*=-1;
```

```
System.out.println("モンテカルロ AI の色（確認）: "+fast_turn);
boolean fastProcess = true;

player[0] = new RanAIPlayer();//黒 PL
player[1] = new RanAIPlayer();//白 PL

//farst_turn の値によって、current_player の値を0,1で決定。
if(fast_turn==1)
{
	current_player=0;
}
else
{
	current_player=1;
}

//最初の一回のみ、引数で受け取った座標を打つ(着手可能手)。以降はランダム。

//動かす手を確定し送る
vBoard.move(MntPoint);
// プレイヤーの交代
current_player = ++current_player % 2;

System.out.println("current_player2"+current_player);
System.out.println("color:"+vBoard.getCurrentColor());

//終局までループ
while(true)
{
	try{
		player[current_player].onTurn(vBoard);
	}
	catch(UndoException e)
	{
		do
		{
			System.out.println("undo!");
			vBoard.undo(); vBoard.undo();
		} while(vBoard.getMovablePos().isEmpty());
		continue;
	}
	catch(ExitException e)
	{
		e.printStackTrace();
	}
	catch(GameOverException e)
	{
```

```
                                        System.out.println("仮想ゲーム終了");
                                        System.out.print("黒石" + vBoard.countDisc(Disc.BLACK)
+ " ");
                                        System.out.println("　　白　　石　　"　　+
vBoard.countDisc(Disc.WHITE));
                                        if(fast_turn==1){
                                                evaluation = vBoard.countDisc(Disc.WHITE) -
vBoard.countDisc(Disc.BLACK);
                                        }else{
                                                evaluation = vBoard.countDisc(Disc.BLACK) -
vBoard.countDisc(Disc.WHITE);
                                        }

                                        break;
                                }
                                catch(Exception e)
                                {
                                        // 例外
                                        System.out.println("Unexpected exception: " + e);
                                        System.exit(0);
                                }
                                // プレイヤーの交代
                                current_player = ++current_player % 2;
                        }

                        System.out.println("*******仮想シミュレーション終了");
                        vBoard.print();

                        return evaluation;

        }


        //一番評価値の高い着手可能手で採用
        //一番高い評価関数を返す。
        public int besteval(int[] valList){
                int masu=0;
                for(int i=1;i<valList.length;i++){
                        if(valList[i]>valList[masu]){
                                masu = i;
                                }
                        }
        return masu;
        }

        /**シミュレーションでの仮想ボード専用メソッド**/
        public AI makeRanAI(){
                return new RanAI();
```

```
            }

}
```