UCT 探索における局面評価関数の使用方法と性能評価

松本　渉[†1]　小林康幸[†1]

新たなゲーム木探索法としてモンテカルロ木探索，特に UCT が成功を収め広く研究されている．しかし，モンテカルロ木探索の主なターゲットが囲碁であるため，局面評価関数を使用した UCT の研究は少ない．本研究では UCB 値に局面評価関数を使用する際の新たな手法，モンテカルロシミュレーションに局面評価関数を使用する手法，プレイアウトを早く打ち切り，その際に局面評価関数を使用する手法の 3 つに着目し，その有効性を示す．実験ではすでに優れた局面評価関数が存在するオセロにおいて行い，局面評価関数には世界最強クラスのプログラムである Zebra のものを使用した．その結果，単純に UCB 値に局面評価関数を用いた UCT よりも優れた性能を発揮できることが実証された．

Performance Evaluation and How To Use Position Evaluation Function in UCT Search Method

Wataru Matsumoto[†1] and Yasuyuki Kobayashi[†1]

The Monte Carlo tree search, particularly UCT, gains a great success and is being widely studied as a new game tree search method. However UCT using position evaluation function has hardly been studied because the main target of UCT search is the game of GO. We focus on a new method when the UCB value uses position evaluation function, a method that the Monte Carlo simulation uses position evaluation function, and a method that use position evaluation function when stopping the Monte Carlo simulation, and show effectiveness of these method. Experiments are performed by using the game of Othello, that already has strong position evaluation function. Evaluation function of the Zebra, the strong othello program, was used for the experiments. The results showed that these method outperformed UCT using simply position evaluation function in the UCB value.

1. はじめに

2 人零和完全情報ゲームはミニマックス探索と評価関数を組み合わせるのが一般的である．オセロのトッププログラムもパターンの学習による評価関数を用いているが，探索手法はミニマックス法を基本としたものである．パターンの学習を用いたプログラムはすでに人間では勝てない強さ[⋆1]ではあるが，まだオセロは解明されていない．近年，コンピュータ囲碁ではモンテカルロ木探索[1] が注目を集めている．モンテカルロ法と呼ばれる乱数を用いたプレイヤー同士で終局までゲームを行ない，その結果を局面の評価ととしてゲーム木探索と組み合わせたものがモンテカルロ木探索である．代表的なものとして UCT[2] がある．UCT は UCB[3] という値が高いノードに多くの探索を割り当て，多く探索されたノードだけを展開する手法である．

†1 島根大学大学院総合理工学研究科
Graduate School of Science and Engineering, University of Shimane

⋆1 M.Buro の Logistello が 1997 年に当時の世界チャンピオンを破る

モンテカルロ法は囲碁やオセロのようにプレイヤーがお互いにランダムに手を選択しても終局を迎えることができるゲームで適用しやすい手法である．いつ終わるかわからずランダムだと収束しにくい将棋ではモンテカルロ法の使用は難しいことが知られている[4][5]．またモンテカルロ法はルール以外のゲームに関する知識が不要な手法である．そのため，ヒューリスティックによる評価関数が作りにくいコンピュータ囲碁において有効な手法として注目され，研究が盛んに行なわれている[1][2][6]．近年ではパターンマッチングなどにより手の評価値を計算し，純粋なランダムではなく強いプレイヤーに近いモンテカルロシミュレーションを実現する事で強化を図ることが盛んに行われている[6][12][13]．一方，囲碁以外のゲームで UCT を使った研究も現在は盛んで，Amazons[7][8] や LOA[9] など 2 人零和完全情報ゲームをはじめ，Nested Monte-Carlo 探索を使ったパズル Morpion Solitaire の研究[10][11] など多岐に渡っている．

本稿では局面評価関数を使った UCT の性能向上について論じる．UCT に局面評価関数を用いることで UCT の性能を向上させる手法を提案し，コンピュー

タオセロで実験を行う．

2. UCB 値とモンテカルロ木探索

モンテカルロ法は全ての手を乱数によって選択していく．しかし，ゲームにおいて明らかに悪い手は結局選ばれないので，できるだけ探索しないほうがよい．計算量を小さく保ちつつ，良さそうな手だけに探索を割り当てる方法として UCB（Upper Confidence Bound）が考案された[3]．UCB 値が最も高いノードを選び続けることで，良さそうなノードにより多くの探索が割り当てられる．

UCB 値は，ある局面で i 番目の手に n_i 回のプレイアウトが行なわれた時の勝率を $\overline{X_i}$，n をある局面で行なわれたプレイアウト全ての回数，c をゲームごとに調整を行なう何らかの係数としたとき，以下のように表される．

$$\overline{X_i} + c\sqrt{\frac{2\log n}{n_i}} \tag{1}$$

UCB 値にはいくつかの種類があり，上記の値は正確には UCB1 値[3] だが，本稿では式 (1) を UCB 値として進めていく．

モンテカルロ木探索とは，ある一定数の探索回数を超えたノードは展開され，その子ノードが記憶され，そこからプレイアウトが行なわれる探索法である．

モンテカルロ木探索に UCB 値を用いてノードを選んでいく手法が UCT(UCB applied to Trees)[2] である．

UCT 探索では UCB 値の高いノードが多く選ばれ，ある閾値を超えるとそのノードが展開される．

3. 関 連 研 究

UCT に局面評価関数を用いる手法として，UCT+がある[14]．この UCT+は局面評価関数を UCB 値に直接用いる手法で，オセロにおいて実装・実験され，有効性も実証されている．

また、モンテカルロ木探索に局面評価関数を用いる研究も存在する[7][9]．そこでは純粋なランダムではなく，局面評価関数によって強いプレイヤーに近いモンテカルロシミュレーションを実現することで強化を図ったり，終局までモンテカルロシミュレーションを行わず，一定の深さに達したら評価関数を呼び出し，その評価が相手より高いプレイヤーを勝ちとみなすことで強化を図ったりしている．

4. 提 案 手 法

UCT に新たな評価関数を用いる方法はいくつかあるが，今回取り上げるのは UCB 値に直接用いる方法，モンテカルロシミュレーション部分に用いる方法，プレイアウトを途中で打ち切る際に用いる方法の 3 つである．

4.1 UCB 値に用いる評価関数の改良

$$\overline{X_i} + E_i + c\sqrt{\frac{2\log n}{n_i}} \tag{2}$$

この手法では式 (2) のように E_i を合法手 i に対する新たな評価関数として UCB に組み込む．文献[14] では i の評価を行う際には i に対する局面評価のみを用いていたが，本研究ではプレイアウトの最中に出現した先手番・後手番の評価関数による評価値をそれぞれ逐次加算していき，それらから適当な評価値を導き，それを E_i として UCB 値に加える．さらに，プレイアウト中に加算する評価値を深さ別に分け，探索局面の深さに応じてそれらに重みを加えることでより良い評価が可能となると予測できる．

4.2 モンテカルロシミュレーションの改良

勝率項を求める際の大部分を担っているモンテカルロシミュレーションであるが，その精度は非常に低い．そこで乱数同士による対戦というシミュレーション方法からランダム要素を省き，局面評価関数を用いてほぼ同等の強さをもった強者同士の対戦によるシミュレーションへ置き換える手法を実装する．既にモンテカルロシミュレーションにおいて評価値に基づいて指し手を選択する手法は有るが[9]、本研究ではパターン学習などを用いた静的な評価関数を用い，これの実装を目指す．

静的な評価関数は同一の局面に対しては常に同一の評価を与える．よって，バイアス項によるバイアスがある程度かかるとはいえ，ノードが展開するまでは特定のノードへプレイアウトが集中するという状況が多発する．そのため単純に静的な評価関数を用いたモンテカルロシミュレーションへと変更したとしても，勝敗判定をメモリできる分実行時間は短縮できるが，正解率は悪化する．評価関数自体の正解率が 100 %でない以上それは良い状況とは言えないため，ここではそれを閾値を極端に低くすることで対応する．また，静的な評価関数同士の対戦であるため，ノードが展開するまでは同一の結果が返ってくる．そのため，初回プレイアウト時の結果をメモリしておき，以降のプレイアウトではその結果を呼び出して再度のプレイアウトはノードが展開されるまでは省略することができる．

4.3 プレイアウトの打ち切り

プレイアウトが一定の深さまで達したら勝ち負け判定の代わりに局面評価関数を呼び出すという手法がある[7]．本研究では純粋なモンテカルロシミュレーションにこれを実装したものと，評価関数を用いて強いプレイヤー同士の対戦へと改良したモンテカルロシミュレーションにこれを実装したものにて実験を行い，オセロにおいても有効であるかを実証する．

5. オセロによる実装

5.1 UCB 値に用いる評価関数の改良

まず UCB 値に評価関数を用いる手法では，Zebra という世界トップクラスのオセロプログラムに使用されている評価関数を利用する．この評価関数はパターン学習を用いたもので，静的な評価を行う関数である．

本研究では最も浅い局面が 20 手目まで終了した局面であるため，21〜30 手目，31〜40 手目，41〜50 手目，51〜60 手目と 4 分割して先手番・後手番の評価値を逐次加算していく．

以降この手法を用いた UCT を UCT+dZ のように文字列中に「+dZ」を加えて呼ぶことにする．

5.2 モンテカルロシミュレーションの改良

UCT では乱数同士による対戦によって勝率項を求めている．繰り返しによりある程度は精度が向上するが，結局は勝利葉ノード数の割合の近似と大差ない．これを乱数同士ではなく Zebra 同士の対戦へと変更することでより精度の高い勝率項を求めることが可能となる．

また，プレイアウトの一極集中を避けるため，この手法を用いる際はノードを展開するプレイアウト回数の閾値を 5 に変更する．

以降この手法を用いた UCT を UCT_PlayZebra のように文字列中に「_PlayZebra」を加えて呼ぶことにする．

5.3 プレイアウトの打ち切り

プレイアウトを打ち切る条件はモンテカルロシミュレーションの開始から 10 手進むこととしている．これは判別分析の結果から探索開始局面から 10 手進んだところまでの評価値が重視されていると判断したためである．プレイアウトを打ち切る際，勝敗判定の代わりに呼び出す局面評価関数にもまた Zebra のものを用いた．

以降この手法を用いた UCT を UCT-break のように文字列中に「-break」を加えて呼ぶことにする．

6. 実　　験

6.1 実 験 方 法

予め完全探索によって最善手とその値が分かっている棋譜ファイル 300 件を用いる．これはオセロの棋譜から，調べたい探索開始局面において必ず完全探索の値が勝ちであるノードと負けとなるノードが 1 つずつは含まれている局面だけを集めている．完全探索はオセロの終盤ソルバーである scrzebra[⋆1]によって行なった．このプログラムはミニマックス法に枝刈りを加えた，α β 法を基本としたアルゴリズムを用いて完全探索を行なっている．

S は探索開始局面を表すこととする．S=40 と書いている場合，初手から 40 手目が終了した局面から探索を開始しているということである．

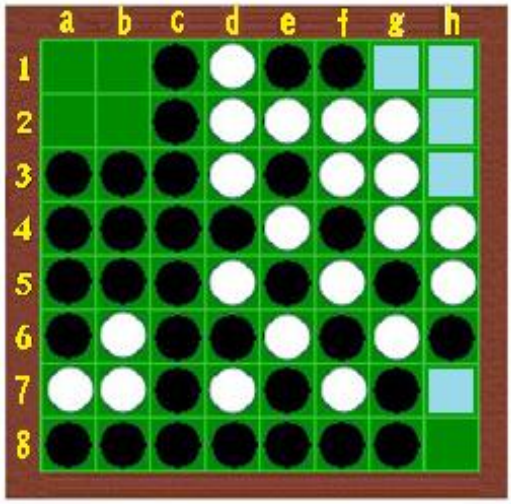

図 1　棋譜ファイルの例

例として，S=50 は以下のように書き込まれている．

(g,1),-10, (h,1),14, (h,2),-10, (h,3),0 (h,7),2,

括弧の中が着手できる手の場所，値が終局の石の差（先手ー後手）である．

図 2 は上記の例の実際のオセロの盤面である．この例では先手は黒であり，着手できる場所は 5 か所ある．ファイルにはその着手できる場所においた場合から終局までお互いに最善手を打った場合の先手にとっての最終値が記載されている．つまり，先手が (h,1) か (h,7) を打った後，お互いに最善手を打てば先手が勝てることになる．

実験で選んだ手の，完全探索結果のファイルに記載されている値が正なら正解の手，負なら不正解の手，0 なら引き分けの手として以下の式により正解率を求める．

正解率＝ (正解数＋引き分け数× 0.5) ÷ 棋譜数

なお，S=20 では完全探索を行うのにあまりにも時間がかかりすぎるため，24 手読みの設定をした Wzebra[15] を用いて各合法手を評価し，その値が正なら勝ち，負なら負けとして実験に用いている．

以下は実験環境．

- CPU Intel Core i7 2700K 3.5GHz
- メモリ 16GB
- 言語 C

⋆1 http://radagast.se/othello/download2.html

6.2 実験結果

それぞれの探索手法の実験結果を表 2，表 3，表 5，表 6 に，また，比較のために UCT+による結果を表 1 に載せる．これらの正解率は 10 回の実験の平均値を載せている．

表 2 はプレイアウト中に深さ別に Zebra の評価値を収集し，判別分析を用いてそれらを適当な値にスケーリング，合計したものを評価値として用いた場合の結果である．UCT+と比べ正解率が向上していることがわかる．このとき用いた判別係数を表 7 に載せる．このとき，score は勝率項を，zeb1_S は S 手目終了局面から S+10 手目終了局面までの先手番の Zebra 評価値を収集した評価値を，zeb2_S は S 手目終了局面から S+10 手目終了局面までの後手番の Zebra 評価値を収集した評価値を表す．

表 3 は UCT+dZ のモンテカルロシミュレーション部分を Zebra 同士の対戦に変更したものの結果である．UCT+dZ からさらに正解率が向上した．しかし，ノードを展開するプレイアウト回数の閾値を大幅に下げたためか，実行時間は延長してしまっている．このとき用いた判別係数を表 8 に載せる．

表 4 は UCT+dZ に対して閾値をそのままにモンテカルロシミュレーション部分を Zebra 同士の対戦に変更したものの結果である．UCT+dZ と比べ，正解率が悪化している．モンテカルロシミュレーションを静的な評価関数同士の対戦へと変更する場合は，今回のように何らかの対策を講じなければ逆効果であるとわかる．

表 5 は UCT+dZ のプレイアウトを途中で打ち切り，Zebra の評価値によってその勝敗を予測させたものの結果である．プレイアウトを途中で打ち切っているため実行時間は短縮されているが，正解率は下がってしまった．

表 6 は UCT+dZ_PlayZebra のプレイアウトを途中で打ち切り，Zebra の評価値によってその勝敗を予測させたものの結果である．UCT+dZ_PlayZebra と比べ，S=20 では正解率，実行時間ともに向上したが，S=30 では正解率は若干下がっている．このことから 3 つを組み合わせたこの手法は中盤が得意，あるいは終盤が苦手なのではないかと予測できる．

結果として，プレイアウト中に評価値を収集する手法の優位性，モンテカルロシミュレーション部分を乱数でなく，優れた評価関数に置き換える手法の優位性が示された．しかし，プレイアウトを打ち切る手法については実行時間のみに優位性があるものと，両者に優位性を持つものとの 2 種の結果が出た．

6.3 考察

表 2 より，プレイアウト中に評価値を収集する手法が有効であることがわかる．これは，疑似的にではあるが，より大局的な評価を行えるようになったことが要因であると考える．

表 1 UCT+による正解率と実行時間
Table 1 Accuracy rate by UCT+.

Playout	S=20	time	S=30	time
10000	78.43%	109s	87.30%	80s
30000	76.33%	338s	86.80%	250s
50000	75.50%	573s	86.88%	432s

表 2 UCT+dZ による正解率と実行時間
Table 2 Accuracy rate by UCT+dZ.

Playout	S=20	time	S=30	time
10000	85.63%	133s	92.85%	84s
30000	86.63%	370s	92.98%	228s
50000	85.53%	593s	92.60%	360s

表 3 UCT+dZ_PlayZebra による正解率と実行時間
Table 3 Accuracy rate by UCT+dZ_PlayZebra.

Playout	S=20	time	S=30	time
10000	86.67%	749s	93.83%	448s
30000	88.67%	2104s	94.83%	1149s
50000	89.33%	3490s	94.83%	1750s

表 4 UCT+dZ_PlayZebra(閾値 100) による正解率と実行時間
Table 4 Accuracy rate by UCT+dZ_PlayZebra(Threshold100).

Playout	S=20	time	S=30	time
10000	77.00%	50s	86.67%	29s
30000	81.00%	134s	90.50%	81s
50000	81.00%	219s	91.67%	133s

表 5 UCT+dZ-break による正解率と実行時間
Table 5 Accuracy rate by UCT+dZ-break.

Playout	S=20	time	S=30	time
10000	56.50%	49s	86.50%	41s
30000	79.40%	159s	89.50%	129s
50000	83.30%	266s	91.67%	221s

表 6 UCT+dZ_PlayZebra-break による正解率と実行時間
Table 6 Accuracy rate by UCT+dZ_PlayZebra-break.

Playout	S=20	time	S=30	time
10000	90.67%	302s	92.17%	227s
30000	90.67%	794s	93.50%	649s
50000	91.00%	1283s	93.50%	1130s

表 7 UCT+dZ に用いた判別係数
Table 7 Discriminant coefficient by UCT+dZ.

評価値	S=20	S=30	S=40	S=50
score	0	0	-0.66918	0
zeb1_20	-0.00036	-	-	-
zeb2_20	0.00013	-	-	-
zeb1_30	0.00009	-0.00037	-	-
zeb2_30	0	0.00009	-	-
zeb1_40	0	0.00013	-0.00018	-
zeb2_40	0	0	0.00028	-
zeb1_50	0	-0.00008	0	0.00011
zeb2_50	0	0	-0.00015	0.00042

表 8　UCT+dZ_PlayZebra に用いた判別係数
Table 8　Discriminant coefficient by UCT+dZ_PlayZebra.

評価値	S=20	S=30	S=40	S=50
score	0	-0.28279	-0.55436	0
zeb1_20	-0.00024	-	-	-
zeb2_20	0.00032	-	-	-
zeb1_30	0	-0.00028	-	-
zeb2_30	0	0.00035	-	-
zeb1_40	0	0	-0.00009	-
zeb2_40	0	0	0.00009	-
zeb1_50	0	0	0	-0.00005
zeb2_50	0	0	-0.00015	0

また，モンテカルロシミュレーションの改変による改善は当然の結果であるとように思えるが，実際は表 4 に示されるように落とし穴も存在する．しかし，表 3 に示されるように対策もあるため，この手法も今後さらなる改善の余地があるのではないかと思われる．

一方，プレイアウトの打ち切りを行った表 5 では正解率が下がってしまっているが，表 6 では正解率も向上していることがわかる．また，手法や実験対象は異なるが，2 人零和完全情報ゲームにおいてプレイアウトの打ち切りが効果的との報告もある[7]．つまり条件次第では正解率にも向上が見られるということであり，今後もプレイアウトの打ち切りの研究は有効であると思われる．

7. おわりに

本稿ではプレイアウト中に評価値を収集する手法の有意性，モンテカルロシミュレーションの改善案，プレイアウト打ち切りの有意性を示した．

ほとんどの場面で UCT+を上回る正解率が出ていることや，それらを組み合わせた手法も非常に良い結果となったことから，提案手法は有効であることがわかった．

今後の課題としては，プレイアウトの打ち切りを行った際に正解率が低下する場合の条件等を調べる必要がある．また，今回はモンテカルロシミュレーション部分を完全に Zebra 対 Zebra としたが，乱数などと併用し，探索に偏りを生ませる手法とどちらのほうが有効であるかを調べる必要がある．

参考文献

1) Coulom, R.: Efficient selectivity and backup operators in monte-carlo tree search, *Proceedings of the 5th International Conference on Computers and Games*, Turin, Italy (2006).
2) Kocsis, L. and Szepesvari, C.: Bandit based Monte-Carlo Planning, *Proceedings of the 15th European Conference on Machine Learning*, pp.282–293 (2006).
3) Auer, P., Cesa-Bianchi, N. and Fischer, P.: Finite time Analysis of the Multi-armed Bandit Problem, *Machine Learning*, Vol. 47, pp. 235–256 (2002).
4) 橋本隼一, 橋本剛, 長嶋淳: コンピュータ将棋におけるモンテカルロ法の可能性, Proceedings of The 11th Game Programming Workshop pp. 195-198 (2006).
5) 佐藤佳州, 高橋大介: モンテカルロ木探索によるコンピュータ将棋, 第 13 回ゲーム・プログラミングワークショップ，pp.1–8 (2008).
6) Gelly, S., Wang, Y., Munos, R. and Teytaud, O.: Modifications of UCT with Patterns in Monte-Carlo Go, Technical Report RR-6062, INRIA (2006).
7) Lorentz, R.: Amazons Discover Monte Carlo, Computers and Games, Lecture Notes in Computer Science, Vol. 5131, pp.13–24, (2008).
8) Julien Kloetzer, Hiroyuki Iida and Bruno Bouzy: Playing Amazons Endgames, ICGA Journal, To be appear,
9) Winands, M.H.M. and Bjornsson, Y. (2010): Evaluation Function Based Monte-Carlo LOA, In Advances in Computer Games (ACG 2009), Lecture Notes in Computer Science (LNCS 6048), pp.33–44. c Springer, Berlin Heidelberg.
10) Tristan Cazenave: Nested Monte-Carlo Search, IJCAI2009, pp.456-461(2009).
11) 秋山晴彦，小谷善行： Nested Monte-Carlo 探索の AMAF を用いた探索数調整による改良, 情報処理学会ゲーム情報学研究会報告，Vol. 2010, No.7, 2009-GI-23, pp.1–7 (2010).
12) Coulom, R.: Computing Elo Ratings of Move Patterns in the Game of Go, In Computer Game Workshop, Amsterdam, The Netherlands (2007).
13) 松井利樹，野口陽来，土井佑紀，橋本剛: 囲碁における勾配法を用いた確率関数の学習, 情報処理学会ゲーム情報学研究会報告，Vol.2009, No.27, 2009-GI-21, pp.33–40 (2009).
14) 橋本剛，前原彰太，川島哲哉，小林康幸: 局面評価関数を使う新たな UCT 探索法の提案とオセロによる評価, 情報処理学会論文誌, Vol.54, pp. 1930-1936 (2013).
15) Andersson, G.: Gunnar's Othello page, http://radagast.se/othello/