囲碁と五目並べにおける SHOTアルゴリズムの有効性の実験的評価

本上 雅央[1,a)]　鶴岡 慶雅[2]

概要：ゲームアルゴリズムにおいてプレイアウトを用いる探索手法としてはモンテカルロ木探索、中でもUCT が主流であるが、最近 SHOT という木探索手法が提案され一部のゲームで UCT との比較がなされた。本研究では、それに加え囲碁と五目並べを用いた対戦実験を行った。その結果 SHOT はプレイアウト数に対して着手可能点が多い場面では UCT より優れた探索をする一方、プレイアウト数を増やした時は UCT に及ばないことが分かった。また、詰碁による探索の性能評価も行い、SHOT が UCT に比べ、正解手が限定され、深い読みが必要となる場面での探索が苦手であることも分かった。

An Empirical Evaluation of the Effectiveness of the SHOT algorithm in Go and Gobang

Masahiro Honjo[1,a)]　Yoshimasa Tsuruoka[2]

Abstract: Today, UCT is the most widely used Monte-Carlo Tree Search (MCTS) method in games like Go. Recently, a new MCTS method called SHOT has been proposed and compared to UCT in some games. In this paper, we compare the performance of SHOT and UCT in the game of Go and Gobang. Experimental results suggest that SHOT is inferior to UCT when many playouts are performed, and that SHOT is superior to UCT when there are many legal moves in comparison with the number of playouts. We have also conducted experiments with composed Go problems and found that SHOT can perform poorly at situations that require a deep search.

1. はじめに

囲碁は将棋やチェスなどのゲームに比べて局面状態数が膨大であることや、使用される 1 つ 1 つの石が無個性であることや、局所的な形勢が全体の形勢とは必ずしも一致しないなど、評価関数が作りにくく近年までアマチュア級位者のレベルにとどまっていた。しかしコンピュータ囲碁の世界大会において、モンテカルロ法と木探索を組み合わせた MCTS (Monte-Carlo Tree Search) を採用した CrazyStone [5] が優勝したことによって、MCTS の囲碁への適用の研究が進められ棋力が飛躍的に向上し、2008 年には MCTS の 1 手法である UCT (Upper Confidence Bound applied to Trees) を用いたプログラムである Mogo [6] が 9 路盤でプロ棋士相手に互先で勝利するという快挙を達成した。これらを経て現在では、UCT をベースに囲碁に適合するようにアルゴリズムの改良が重ねられて、19 路盤でもおおよそアマチュア六段レベルの力をつけたと言われるまでに成長している。

一方最近、UCT に対して新たな MCTS である SHOT (Sequential Halving applied to Trees) という手法が Cazenave によって提案された [4]。UCT が Multi-Armed Bandit 問題の近似解法の 1 つである UCB1 値に基づくノード選択を基本としているのに対して、SHOT は Sequential Halving [7]

[1] 東京大学工学部電子情報工学科
Department of Information and Communication Engineering, The University of Tokyo

[2] 東京大学大学院工学系研究科電気系工学専攻
Department of Electrical Engineering and Information Systems, Graduate School of Engineering, The University of Tokyo

[a)] honjo@logos.t.u-tokyo.ac.jp

アルゴリズムを基本としてノード展開を行う。Sequential Halving は UCB 値と同じく、多腕バンディット問題における探索制御手法の 1 つであるが、UCB 値が試行を通して得られる報酬の期待値を最大化する手法であるのに対して Sequential Halving は投入できる資源が固定されているとき、報酬期待値が最大である Arm を見つける確率を最大化する手法である。

文献 [4] では SHOT は UCT に対して探索時間や必要メモリの点で優れていると示されている。また囲碁のルールを改変した特殊なゲームである Nogo において特定の条件下で SHOT と UCT の対戦実験を行った結果、UCT に対して有意に勝ち越すということも示されている。しかし Nogo はモンテカルロ木探索が有効なゲームであるかどうか明確に知られてはいない上、囲碁を始めとする実際にコンピュータプログラムでモンテカルロ木探索が用いられているゲームではその有効性が示されていない。

そこで本研究では先行研究で試されていないゲーム、特にモンテカルロ木探索が有効であることが知られているゲームにおいて UCT と SHOT の性質の違いを比較する。これにより、UCT よりも SHOT が有効である場面を明確に示すことができれば、現在 UCT が用いられているコンピュータプログラムの強化が期待される。

本稿では、第 2 章で関連研究の紹介、第 3 章で本研究で行った実験を示す。第 4 章で本研究のまとめについて述べる。

2. 関連研究

2.1 多腕バンディット問題 と Regret

2.1.1 多腕バンディット問題問題

多腕バンディット問題とは機械学習などの分野で頻出する問題である。今、K 本のスロットのアームがある。それぞれ報酬期待値が設定されており、それぞれ X_j とするが、この確率分布を事前に知ることはできず、実際にコインを投入することでしか推定できないとする。この時、T 枚のコインを最も報酬期待値が高くなるように投入する方法はどうすればよいか、という問題である。

2.1.2 Cumulative Regret

Cumulative Regret [2], [8] は、T 枚のコインを投入し終わった時点で、仮に報酬期待値最大のアームを選択をし続けた場合に比べてどれだけ報酬期待値に差が出るかと定義され、以下の式で表される。

$$R_n = \sum_{t=0}^{n} (X^* - X^t) \tag{1}$$

ただし R_n は n 枚投入時点での Cumulative Regret、X^* は K 本のアームの中で最も高い報酬期待値を持つアームの報酬期待値、X^t は t 番目に選択したアームの報酬期待値とする。

2.1.3 Simple Regret

ここで多腕バンディット問題を別の方向から考えてみる。今 K 本のスロットのアームがある。K 本のアームにはそれぞれ報酬期待値が設定されておりそれぞれ X_j とするが、この確率分布を事前に知ることはできず、実際にコインを投入することでしか推定できないとする。この時 T 枚のコインを使って K 本のアームの中で最大の報酬期待値を持つアームを見つけるにはどのようにコインを投入すればよいか、という問題である。

この問題は 2.1.1 項で定義した多腕バンディット問題と違って、T 枚のコインを投入した際の報酬は関知せず、T 枚でどれだけ正確に最大報酬期待値を持つアームを割り出せるかが重要になる。ここで Simple Regret [3] を次の式 (2) で定義する。

$$r_n = X^* - X^{S(n)} \tag{2}$$

ただし、r_n は n 枚投入時点での Simple Regret、X^* は、K 本のアームの中で最も高い報酬期待値を持つアームの報酬期待値、$X^{S(n)}$ は、n 枚投入時点で最も高い報酬期待値を持つとアルゴリズムによって判断されたアームの報酬期待値とする。

2.1.2 項で述べた Cumulative Regret と 2.1.3 項で定義した Simple Regret は片方の Regret を小さくしようとすると、もう片方の Regret が大きくなってしまうことが知られている [9]。

2.2 Regret 最小化と木探索

多腕バンディット問題において Regret を最小化する考え方をゲームに適用することを考える。ゲームに勝利することを報酬を得ることだとすると、Regret を最小化することによって最善の選択肢を得られることができると考えられる。しかし、ゲームは多腕バンディット問題問題と違い、相手も最善の選択肢をとろうとするので、単に多腕バンディット問題問題の手法を適用しても上手くいかない。そこでこれらの手法を木構造に落とし込むと、結果的に数手先の Regret が最小になるように手が選択できるはずだと考えられる。前述のように Cumulative Regret と Simple Regret はトレードオフの関係にあるので両方の期待値を最小化することはできず、どちらを最小化した方がよいかはゲームの目的や性質、状況により異なると考えられる。

2.2.1 Sequential Halving

Sequential Halving は多腕バンディット問題の探索制御手法の一つであるが、UCB1 値と違って 2.1.3 項で述べた Simple Regret を最小化しようとするコインの投入の仕方の 1 つである。Sequential Halving のアルゴリズムを Algorithm2.1 に示す。Algorithm2.1 は文献 [7] に記述されているアルゴリズムと基本的に同一であるが、投入されるコインを計算した結果が 0 になってしまった時に、1 枚は

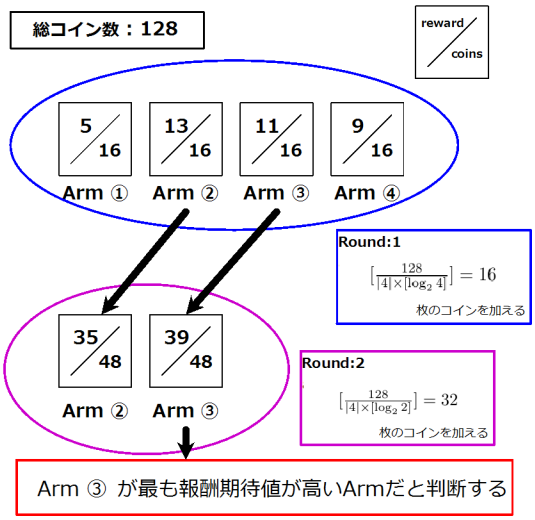

図 1　Sequential Halving の適用例

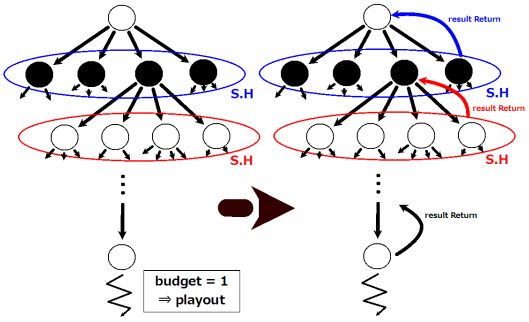

図 2　SHOT アルゴリズム

投入することにした点と、その結果途中でコインがなくなってしまった場合処理を中断してその時点で最良のアームを選択するようにした点が異なっている。

また図 1 に Sequential Halving の適用例を示す。Sequential Halving のコインの投入はラウンド制で行われる。初めは、K 本全てのアームに 10 行目において計算される枚数だけコインの投入を行う。そして K 本のアームについてそれまでの平均報酬を記憶しておく。コインの投入が終わったら、K 本のアームを平均報酬が大きい順から並べ替える。その内 $K/2$ 本のアームだけが次のラウンドに進めて、また同じ操作を繰り返す。これをアームが 1 本になるか、コインが 0 枚になるまで続け最後に残ったアーム (もしくは生き残ったアームの内最も平均報酬の大きいもの) を「K 本のアームのうち最も報酬期待値が高いアーム」と推定する。

Algorithm 2.1 Sequential Halving

IMPUT :
$T \leftarrow$ *total budget*,
K *arms* $\{Arm\ 1,\ Arm\ 2,\ Arm\ 3,\ \ldots,\ Arm\ K\}$
OUTPUT:
Sequential Halving が最も報酬期待値が高いと判断した Arm

$S_0 \leftarrow \{Arm\ 1,\ Arm\ 2,\ Arm3,\ \ldots,\ Arm\ K\}$
$k \leftarrow 0$
$B \leftarrow T$
while $|S_k| \neq 1$ **do**
  $n_k \leftarrow max(1, [\frac{T}{|S_k|[\log_2 K]}])$
  for $i = 0$ to $|S_k|$ **do**
    for $j = 0$ to n_k **do**
      S_k の中の Arm i にコインを 1 枚入れる
      Arm i の報酬平均値を更新
      $B \leftarrow B - 1$
      if $B = 0$ **then**
        break WHILE loop
      end if
    end for　▷ end i loop
  end for　▷ end j loop
  $S_{k+1} \leftarrow S_k$ の中で報酬平均値の大きい順から $[\frac{S_k}{2}]$ 個の Arm
  $k \leftarrow k + 1$　▷ $Max\ k = [\log_2 K] - 1$
end while

return　S_k の中で最大の報酬平均値を持つ Arm

2.2.2　SHOT

SHOT では Sequential Halving を MCTS に拡張する。SHOT の疑似コードを Algorithm2.2 に示す。文献 [4] に掲載されていた疑似コードではノード情報の保持にトランスポジションテーブルを用いていたが、ここでは用いずに木構造でノード情報を保持する方法に書き直している。

また図 2 に SHOT アルゴリズムの動作の概要を示す。図のように SHOT は MCTS の木の各階層で Sequential Halving を実行する。Sequential Halving ではラウンド毎に各ノードに一定数のコイン (ゲーム木の場合はプレイアウト数) が割り当てられるが、SHOT の場合、割り当てられたプレイアウト数が 1 でない限り 1 つ下の階層においてその割り当てられたプレイアウト数をまたそのノードでの Sequential Halving の数式に従ってその階層のノードに割り当てる。これを繰り返し、割り当てられたプレイアウト数が 1 になった局面で実際にプレイアウトを実行し、結果を親ノードに返し、ノードの情報を更新する。

こうして Sequential Halving によるノードの絞り込みを続けていき、ルートノードの 1 つ下の階層においてノードが 1 つまで絞り込まれたら、それをルートノードにおける最善ノードだと決定する。

3.　実験

3.1　Sequentail Halving と UCB の比較実験

Multi-Armed Bandit 問題に対して、Sequential Halving の UCB に対する性能の違いをより理解するために実験を行う。具体的には、多腕バンディット問題において、次の

Algorithm 2.2 SHOT

```
struct NODE{
 INT possible_moves_num
 CHILD child[possible_moves_num]
 INT budgetUsed
 MOVE move
}

struct CHILD{
 INT num_games
 FLOAT rate
 NODE my_node
}

Shot(INT budget, NODE node){
budgetUsed ← 0
if board is terminal then
    result ←CheckWin()
    return result and update rate and games
end if
if budget = 1 then
    result ←Playout()
    return result and update rate and games
end if
if possible_moves_num = 1 then
    Play(child.move)
    Shot(budget, child.my_move)
    UnPlay()
    return result and update rate and games
end if
if the c.move with 0 playout exist then
    for all the c.move with 0 playout do
        result ←Playout()
        update child.games, child.rate, and budgetUsed
        if budgetUsed ≥ budgetUsed then
            return result and update rate and games
        end if
    end for
end if
S ← child[ ]
sort child in S according to their chile.rate
b ← 0
while |S| > 1 do
    b ← b + max(1, [budgetUsed+budget / |S|[log2 possible_moves_num]])
    for child in S by decreasing child.rate do
        if child.games < b then
            b1 ← b − child.games
            if at root ∧ |S| = 2 ∧ child is the first in S then
                b1 ← budget − budgetUsed−
                 (b − NEXT_child.games)
            end if
            b1 ← min(b1, budget − budgetUsed)
            Play(child.move)
            Shot(b1, child.my_node)
            UnPlay()
            update budgetUsed, child.rate and child.games
        end if
        break if budgetUsed ≥ budget
    end for
    S ← the [|S/2|] child from S with best child.rate
    break if budgetUsed ≥ budget
end while
update budgetUset, child.rate and child.games
return The child with best child.rate of S
```

２つの目的を UCB と Sequential Halving がどのくらい達成できるかを調査する。

- 決められた枚数のコインを使い、より多くの報酬を受け取れるようスロットにコインを投入する
- 決められた枚数のコインを使い、報酬期待値が最大であるスロットを推定する

スロットの数を変化させていくとき、UCB と Sequential Halving のこれらの目的に対する達成率がどのように変化するかを調べる。

シミュレーションは次の条件で行う。

- スロットを $n(n = 2, 4, 8, 16, ..., 1024)$ 台用意する。以降これらを x_j (j = 1, 2, 3, ... , n) とする。
- x_j が報酬を返す確率を $P(x_j)$ とする。$P(x_j) = 0.1 + 0.8 * \frac{j-1}{n-1}$ と設定する。返す報酬は常に 1 とする。
- 投入する資源 (コイン) は 20,000 とする。
- UCB, Sequential Halving のそれぞれのアルゴリズムに従って資源をスロットに投入する。
- これを各スロット数において 1,000 回ずつ行い、一回当たりの報酬平均値と最大報酬期待値を持つスロットの的中率を算出し、UCB 値と SHOT で比較する。
- また、Sequential Halving の数式において、ラウンドに投入される資源が 0 と算出された場合、1 にする。またこれによって資源の投入中に資源が足りなくなった場合途中で中断する。

、

実験結果を図 3、図 4 に示す。図 4 を見ると、コインの数に対してスロットの数が少ないときは、最大報酬期待値を持つスロットの的中率は UCB、Sequential Halving ともに 1 となっている、また図 3 を見るとその間は UCB の報酬平均値が Sequential Halving のそれを上回っている。しかし、スロットが 128 台をこえると、UCB, Sequential Halving の両者とも最大報酬期待値を持つスロットの的中率が徐々に落ち始めている、また、常に Sequential Halving の的中率が UCB のそれを上回っていることが分かる。また、128 台を越えてから報酬平均値も Sequential Halving が UCB を上回っていることが分かる。

3.2 五目並べにおける SHOT と UCT の対戦実験

次に五目並べを使った対戦実験を行った。五目並べは囲碁と同様に、ある程度ランダムに交互に着手を行ってもいずれ終局するので MCTS や SHOT をなどのプレイアウトを用いた探索が可能であり、かつ有効である可能性が高いと考えられる。まず、以下のような条件で SHOT と UCT の対戦実験を行う。

- SHOT と UCT のプレイアウト数をそれぞれ 10,000、20,000、30,000、40,000、50,000 とする。
- 盤面の広さは 10×10 と 20×20 の 2 種類で行う。

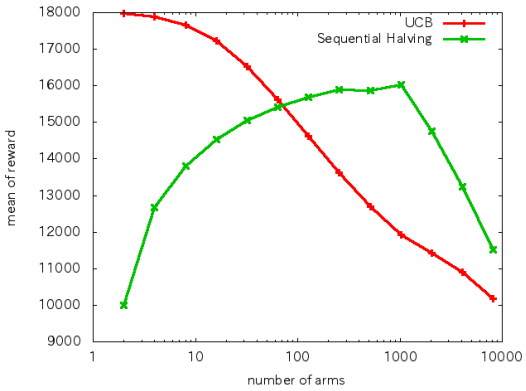

図 3 1 回あたりの平均報酬値

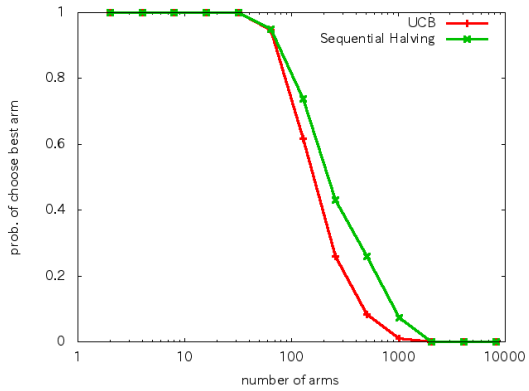

図 4 最大報酬期待値を持つスロットの的中率

- 1 つの条件に付き対戦を 1,000 回行い (500 回ごとに先手後手を入れ替える)、それぞれ SHOT の勝率を算出する。

次に以下のような条件で SHOT と UCT の対戦実験を行う。

- SHOT のプレイアウト数は 10,000、20,000、50,000、100,000 の 4 種類とする。
- これに応じて思考時間がほぼ等しくなるよう UCT のプレイアウト数を調整する。
- 盤面の広さは 10×10 と 20×20 の 2 種類で行う。
- 1 つの条件につき対戦を 1,000 回行い (500 回ごとに先手後手を入れ替える)、それぞれ SHOT の勝率を算出する。

また上記 2 つの実験ともに、五目並べのルールは以下であるとする。

- 網目上のマス目に先手から交互に石を置いていき、先に自分の石を縦・横・斜めのいずれかに 5 つ連続で並べば勝利となる。
- 先手の三三 [*1]、四四 [*2]、長連 [*3] は禁じ手としていない
- 先手後手ともに、長連は勝利条件とはしていない

さらに、SHOT、UCT に用いるプレイアウトは完全なランダムではなく以下の改良を行っている。

- プレイアウト中 (または開始直後) 相手の最後の手によって相手の石が 4 つ並びあと 1 つ並べば相手の勝利になってしまう状態では、必ずそれを阻止するような手を選択することとする。

また UCT の式のパラメタ C の値は、事前に検討してもっとも有力だった 0.41 に全て統一している。

まずプレイアウト数を同一にした場合の結果を表 3.2 に示す。プレイアウトを少ない場合は SHOT が勝ち越しているが、増やしていくにつれ UCT に逆転されてしまっていることが分かる。

次に探索時間を同一にした対戦の結果を表 2 に示す。まず、思考時間当たりのプレイアウト数が 2 倍以上違っていることが分かる。探索時間自体の差は [4] における Nogo の実験の時より小さくなっているが、これに関する考察は囲碁の探索時間と合わせて 3.3 節で考察する。この探索時間の差により 3.2 節の、プレイアウト数を同一にした場合の結果と違い、10×10 の盤面で最も思考時間を長くした対戦以外は、SHOT が UCT に有意に勝ち越していることがわかる。また、10×10 の盤面に比べて、20×20 の盤面の対戦結果の方が UCT に勝ち越している割合が大きいことがわかる。しかし、10×10 の盤面では思考時間を長くするにしたがって、SHOT の勝率が減少している様子が見て取れる。これは [4] の Nogo による対戦実験では見られなかった傾向である。

この結果から、SHOT は UCT に対して五目並べでは

表 1 10 × 10 五目並べにおける同プレイアウトの UCT と SHOT の対戦結果

プレイアウト数	SHOT の勝率 [%]
10,000	60.9
20,000	54.2
30,000	49.9
40,000	45.9
50,000	42.9

表 2 五目並べにおける同思考時間の UCT と SHOT の対戦結果

	SHOT		UCT		
盤のサイズ	プレイアウト数	思考時間 [秒/一手]	プレイアウト数	思考時間 [秒/一手]	SHOT の勝率 [%]
10 × 10	10,000	0.072	4,900	0.071	85.6
10 × 10	20,000	0.142	9,200	0.142	78.4
10 × 10	50,000	0.343	21,500	0.345	67.4
10 × 10	100,000	0.686	41,000	0.682	45.3
20 × 20	10,000	0.216	4,900	0.232	92.1
20 × 20	20,000	0.434	9,200	0.440	93.7
20 × 20	50,000	1.078	21,500	1.048	95.8
20 × 20	100,000	2.146	41,000	2.080	95.4

*1 一手で、両端が相手の石によって塞がれていない 3 つの石による “三” という状態を、同時に 2 つ以上つくること

*2 一手で、少なくとも 1 つの端が相手の石によって塞がれていない 4 つのつながった石による “四” という状態を、同時に 2 つ以上つくる

*3 六つ以上の石を連続で並べること

- 盤面が広いなど選択肢が多い時
- 盤面が広くなくても、それに対するプレイアウトが少ない時

によい探索をするのではないかと考えられる。逆に

- 盤面が狭く、選択肢に対して十分なプレイアウト数が与えられている時

の時には UCT の探索には及ばないと考えられる。

この結果の原因はプレイアウト数が増えたときの UCT と SHOT のプレイアウトの振り分け方の違いによるものだと考えられる。UCT はプレイアウト数が増えていくにつれ、プレイアウトが有望な選択肢に偏り、木が成長をしていき深い探索が可能となる。一方 SHOT はプレイアウト数が多くなっても Sequential Halving の性質から、有望な選択肢以外に対してのプレイアウトも同様に増えていき有望な選択肢に対する木の成長が UCT に比べて深くなりにくいと考えられる。

また、この SHOT 探索の性質は、プレイアウト数を増やしても UCT に比べて強くなりにくいことにつながるのではないかと考えられる。これを裏付けるために以下の条件で実験を行った。

- 一方のプレイヤをプレイアウト数 41,000 回の UCT に固定する。
- まずこのプレイヤとプレイアウト数を 100,000、150,000、200,000、300,000 と変化させた SHOT と 1,000 回対戦させて、勝率の変化を調べる。
- 次に、プレイアウト数を 60,000、80,000、120,000 と変化させた UCT と 1,000 回対戦させて、勝率の変化を調べる。

表 3 にこの対戦実験の結果を示す。これを見ると UCT がプレイアウト数を 41,000 回から約 1.5 倍、2.0 倍、3.0 倍と増やしていくにつれ勝率が上昇していっているのに対して、SHOT は同様にプレイアウト数を増やしても勝率がほとんど変化していない。この後 SHOT のプレイアウト数を 400,000、500,000 と増やしていってもそれぞれ 49.9、49.3% と、ほとんど勝率に上昇が見込めなかった。この実験により SHOT は UCT に比べプレイアウト数を増やしても探索が強化されにくいという我々の仮説の裏付けが一つとることができたと言える。

表 3 10 × 10 五目並べにおける UCT41,000 回プレイヤに対する対戦成績

探索手法	プレイアウト数	勝率 [%]
SHOT	100,000	47.7
	150,000	44.2
	200,000	47.9
	300,000	47.6
UCT	60,000	63.4
	80,000	68.9
	120,000	70.7

3.3 囲碁における SHOT と UCT の対戦実験

次に囲碁における UCT と SHOT の対戦実験を行った。囲碁はゲームプログラムにおいて MCTS、とりわけ UCT が広く使われているゲームであり MCTS の有効性が実証されているゲームの 1 つである。これに SHOT が適合するかどうかを調べる。

以下の条件で実験を行う

- 囲碁の基本処理及び UCT のコード及びプレイアウトの関数は囲碁のフリープログラムである「彩」[10], [11] のサンプルコードを利用する。
- UCT と SHOT のプレイアウト数は 500、1,000、2,000、5,000、10,000 とする。
- 盤面の広さは 9×9 とする
- コミ [*4] は 6 目半とする
- プレイアウトは改良を行っておらず、着手禁止点へ打たない、連続コウを打たない、自分の眼を潰さないこと以外は完全ランダムにプレイアウトを行う。
- 各プレイアウトごとに 100 回の対戦を行い (50 回で先後を交代する) それぞれ SHOT の勝率を算出する。

囲碁における SHOT と UCT の対戦実験の結果を表 4 に示す。まず探索時間に注目する。探索時間は、3.2 節での実験の時と違いプレイアウト数が同一にでも SHOT と UCT の思考時間はほとんど変わらないことが分かる。[4] における Nogo の探索時間、3.2 節における五目並べの探索時間、そして本節における探索時間の差の大きな違いとして、実装の仕方が原因だとも考えられるが、そのほかにも 1 プレイアウトにかかる処理時間の差が原因だと考えられる。初期局面から五目並べと、囲碁それぞれにおいて 10,000 回プレイアウトを行い、その平均としてプレイアウト 1 回当たりの処理時間を計測したところ、五目並べは 2.3×10^{-5} [秒/1 回] であるのに対し、囲碁は 3.2×10^{-4} [秒/1 回] と 10 倍以上の差が見られた。また、[4] 中では Nogo 一回当たりのプレイアウト数は掲載されていないが、UCT、SHOT 探索ともに五目並べに比べ探索時間が短くなっているので (実装の仕方の違いを考慮しなければ) Nogo のプレイアウト 1 回あたりの処理時間は五目並べのそれより小さくなると考えられる。もちろん探索中のプレイアウトはいつも初期局

表 4 9 × 9 囲碁における SHOT と UCT の対戦実験結果

プレイアウト数	平均思考時間 (SHOT) [秒/一手]	平均思考時間 (UCT) [秒/一手]	SHOT の勝率 [%]		
			黒番	白番	計
5,00	0.144	0.145	14	44	29
1,000	0.296	0.298	24	68	46
2,000	0.596	0.600	38	62	50
5,000	1.486	1.511	26	54	40
10,000	3.002	3.034	26	36	31

[*4] 囲碁は互いの色の陣地のマス目の大きさを競うゲームであるが、一般的に先手が有利となるため後手番は終局時に自分の陣地を設定されたコミだけ増やすことができる。

面で行われるわけではないので探索時間が単純にこのプレイアウト数と探索時間の合計になるわけではないが、この差が探索時間の差の違いに寄与している可能性は高いと考えられる。

対戦結果を見てみると、どのプレイアウト数においても SHOT は UCT に勝ち越すことができずこの実験では囲碁における SHOT の有効性を示すことはできなかった。また、プレイアウト数に関係なく黒番に比べて白番の方が勝率が高いことが分かる。この結果は以下の 2 つの原因が考えられる。

- プレイアウトや終局判定が単純であり、コミの寄与が大きくなりすぎたこと
- 石の生き死にに対して SHOT が UCT に及ばず、それが直接勝敗にかかわりやすい事

まず、プレイアウトについてであるが、実験の条件設定でも述べたようにプレイアウトは非常に単純でありルールで禁止されている点と、自らの眼を潰す以外の全ての手をランダムで選択するようになっている。よって眼はつぶさなくとも自分の手を潰すような手は選択されることになる。よってプレイアウト中、仮にコミを入れて黒と白がほぼ互角の進行をしていたとしても打てる場所がなくなった際黒が自分の地を潰す手を選択したりするのでプレイアウトにおいて白が勝利しやすくなっていると考えられる。このようなプレイアウトの偏りによりどの条件でも白番の勝率が高くなってしまっているのではないかと考えられる。

また、石の生き死にについての UCT と SHOT の得手不得手であるが、これは UCT と SHOT の対戦を見ていると、SHOT のプレイヤが正しい手を打てば大きく石を取られずに済む場面で間違えてしまい UCT のプレイヤに大きく石を取られてしまい負けてしまう光景が散見されて感じたことである。囲碁における石の生き死には正しい手が限定される上に、深い読みが必要とされる場面が多く 3.2 節の実験考察で述べたように SHOT は深い読みを苦手としているのでこの点で UCT に劣るのではないかという仮説が立てられる。

この仮説をさらに検証するために次のような実験を行う。プレイアウト数をそれぞれ 5,000、10,000、50,000 回の UCT と SHOT に対して詰碁 [*5] を与える。この時各探索手法、プレイアウト数において詰碁を解くことができるかどうかを調べる。解かせる詰碁は Web サイト「詰碁を楽しむ会」内の「楽々詰碁」コーナー [1] より、15 題選んだ [*6]。詰碁を解かせる際には図 5 のように 9 × 9 路盤の周りを詰碁の結果に影響しない程度に石を埋める。また、コミは 0 に変更しておく。これは仮に図 5 の左上の石をなくした状態で思考させると右下の詰碁を探索するよりも左上の地を取ることを優先してしまうなどの可能性をなくすためである。

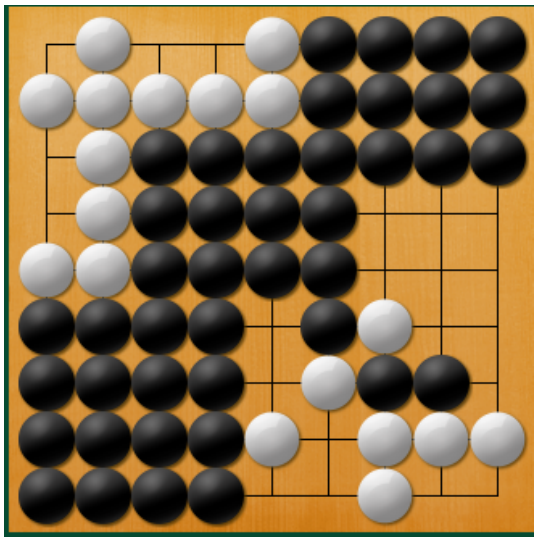

図 5 詰碁番号 (1) (黒先白死 黒の手番で、右下の白が殺せる特定の手順を考える。) 左上に白の陣地を少し作っておくことによって、右下の白を殺して黒の陣地にする他は黒が勝利できないようにしておく。

表 3.3 に詰碁の結果を示す。ただし表 3.3 における○は「5 回連続で正解手を選ぶことができた」ことを意味し△は「5 回の中で正解手を 1〜4 回選ぶことができた」ことを意味し、× は「5 回の中で一回も正解手を選べなかった」ことを意味している。これをみてみると、全ての条件で解くことのできた問題もあればどの条件でも解けなかった問題がある一方で、(1)、(9)、(12)、(13)、(15) 番など UCT のみが解けた問題も存在する。この結果から SHOT に比べ UCT は詰碁に見られるような、「正解手が限定され、深い読みが必要とされる場面」でよりよい選択肢を選ぶことができるという我々の仮説の裏付けを一つとることができた。

4. まとめ

本研究では五目並べと囲碁において、UCT と SHOT の対戦実験を行い SHOT のゲームにおける有効性を検証した。結果として、五目並べでは 1 プレイアウト数当たりの探索時間で SHOT は UCT を大きく上回り、思考時間を同一にした対戦では盤面が広い場合、SHOT が UCT にかなり大きく勝ち越す結果が出た一方、盤面を狭くしプレイアウトを増やしていくにつれ勝率が徐々に落ちてしまうことから、UCT に比べプレイアウトを増やしても探索が深くなりにくい性質がありそうだということが分かった。囲碁ではプレイアウトの処理時間が五目並べに比べて遥かに大きいためか、探索時間に大きな差が出なかった。さらに、対

[*5] 相手の石を殺したり、自分の石を生かしたりする特定の手順を考える囲碁のパズル

[*6] 詰碁番号 (1) 創刊号第 1 問 (2) 創刊号第 4 問 (3) 第 2 号第 1 問 (4) 第 2 号第 2 問 (5) 第 2 号第 4 問 (6) 第 2 号第 5 問 (7) 第 2 号第 7 問 (8) 第 2 号第 8 問 (9) 第 3 号第 8 問 (10) 第 1 号第 2 問 (11) 第 1 号第 3 問 (12) 第 32 号第 3 問 (13) 第 32 号第 6 問 (14) 第 32 号第 1 問 (15) 第 27 号第 1 問

表 5 UCT と SHOT の詰碁に対する達成率

		詰碁番号														
探索手法	プレイアウト数	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)
UCT	5,000	○	×	○	○	×	×	×	×	○	○	○	△	△	×	○
	10,000	○	×	○	○	×	×	×	×	○	○	○	○	△	×	○
	50,000	○	×	○	○	×	×	×	×	○	○	○	○	○	×	○
SHOT	5,000	×	×	○	○	×	×	×	×	△	○	△	×	×	×	×
	10,000	×	×	○	○	×	×	×	×	△	○	○	×	△	×	×
	50,000	×	×	○	○	×	×	×	×	△	○	○	×	○	×	×

戦成績も SHOT や UCT に対してよい成績を出すことができなかった。これはプレイアウトの実装方法による問題も考えられるが、それに加え我々は負け越した原因を、SHOT が正解手が限定され、深い読みが必要な場面が UCT に対して得意でないからだと考えた。この仮説は詰碁の実験から裏付けることができた。以上から SHOT 探索は UCT に比べて深い読みが苦手であると分かった一方で、盤面が広い時など一部の条件では UCT より良い性能を示すことが分かった。

参考文献

[1] 詰碁を楽しむ会　楽々詰碁. `http://www.h-eba.com/tsumego/japan/j310.html`.

[2] Andrew G. Barto and Richard S. Sutton. *Reinforcement Learning*. Mit Press, 2012.

[3] S Budeck, Munos. R., and G. Stolts. Pure exploration in finitely-armed and continuous-armed bandits. *Theoretival Computer Science*, pp. 1832–1852, 2010.

[4] T. Cazenave. Sequential halving applied to trees. *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. PP-99, , 2014.

[5] Remi Coulom. Computing elo ratings of move patterns in the game of go. *IGGA Journal*, Vol. 30, pp. 198–208, 2007.

[6] Syvain Gelly, Y. Wang, R. Munos, and O. Teytaud. Modification of uct with patterns in monte-carlo go,technical report pr-6062. *INRIA*, 2006.

[7] Z. Karnin, T. Koren, and O. Somekh. Almost optimal exploration in multi-armed bandits. *Proc of the int. Conf. on Math. Learn*, pp. 1238–1246, 2013.

[8] T.L. Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied Mathmatics*, Vol. 6, pp. 4–22, 1985.

[9] D. Tolpin and S. Shimony. Mcts based on simple regret. *Proc.*, pp. 570–576, 2012.

[10] 山下宏. コンピュータ囲碁 〜 モンテカルロ法の理論と実践 〜実践編のサンプル一覧. `http://www.yss-aya.com/book2011/`.

[11] 美添一樹, 山下宏. コンピュータ囲碁ーモンテカルロ法の理論と実践ー. 共立出版, 2012.