愛知工業大学研究報告
第 51 号 平成 28 年

局面タブーリストを内包したモンテカルロ木探索の 19 路盤囲碁への応用

An Application of Monte-Carlo Tree Search Including Game State Tabu Lists for Playing 19×19 Go

伊藤 雅 † 太田 雄大 ††
Masaru ITOH Takehiro OHTA

Abstract: This paper proposes a modified Monte-Carlo tree search (abbreviated as MCTS) for playing 19×19 go. Diversifying playout causes the MCTS to have behavior just like a best-first search in a more uniformly-broadened tree without extending toward depth direction. The diversification of playout could be brought about by the process of excluding the same game states obtained from successive playouts. So as to achieve the diversity, each leaf node in the searching tree includes multiple tabu lists, whose element is a game state. Here, a game state is expressed as a value of Zobrist hash. The first game state, which has a hash value, to several ones from starting a playout are sequentially enqueued into the tabu lists. If a game state is labeled as "tabu", then the move to lead the same game state is treated as a prohibited move during a preset tabu tenure. Thereby, it is possible to select a good move which may be the best move, because the varieties of candidate move in the Monte-Carlo tree have been widening. Furthermore, we also propose two update methods about game state tabu lists. One is a sequential update method, and the other is a mass update one based on winning or losing of the playout.

1. はじめに

囲碁はチェスや将棋と同様、二人零和有限確定完全情報ゲームのひとつである。チェスや将棋は盤面評価関数を使い、min-max 探索 [1] で棋力を向上させてきた。囲碁では評価関数の設定が極めて難しい。駒のように種類がなく、キングや王といった絶対的な存在がないからである。囲碁は終局での地の大きさを競うゲームである。19 路盤囲碁の局面数はチェス 10^{50} や将棋 10^{71} と比較して格段に大きく 10^{171} もある [2]。この探索空間の大きさがパターンマッチングを難しくし、盤面評価関数を作り難くしている。捨て石でさえ後になって地の確定に役立つことがある。

現在のコンピュータ囲碁の棋力向上は 2006 年に登場した囲碁プログラム "CrazyStone" に拠るところが大きい。レミ・クーロン氏が開発した CrazyStone に搭載されていた手法が今ではモンテカルロ木探索 (Monte-Carlo Tree Search: 略して MCTS)[3] と呼ばれる。乱数を使った原始モンテカルロ囲碁に UCB1 値 [4] を組み込んだ木探索手法である。UCB とは Upper Confidence Bound の略である。この木探索を UCT(UCB applied to Trees) アルゴリズム [5] という。モンテカルロ木探索の代名詞である。

原始モンテカルロ囲碁で使われるのがプレイアウト [2] である。プレイアウトとは、与えられた現局面から両者が適当に打ち合い終局させることである。終局さえすれば、囲碁は陣取りゲームなので勝敗は容易に判定できる。この勝敗に {1,0} を与え、勝率計算に利用する。

モンテカルロ木探索の改善には大きく 2 つのアプローチがある。ひとつは木探索の改善であり、もうひとつはプレイアウトの改善である。本論文は後者、プレイアウトの改善の一提案として位置付けられる。筆者らは先の論文 [6] で、タブーリストを内包したモンテカルロ木探索手法を詰碁と 9 路盤囲碁に応用した。結果は良好で、カイ二乗検定と二項検定で手法の統計的優位性も検証している。ここで、タブーリストとはタブーサーチ [7][8] で使われる短期メモリのことである。タブーサーチは組み合せ最適化問題に使われるメタヒューリスティクスのひとつである。

太田・伊藤 [6] が提案したタブーリストは詰碁や 9 路盤囲碁では有効に機能したが、19 路盤囲碁では残念ながら統計的有意差を示すには至らなかった。19 路盤囲碁の探索空間の広さを克服できなかった。本論文ではこの欠点を補うべく局面タブーリストの導入を提案する。プレイアウト中に着手した双方の手をタブーリストに順次登録するのではなく、着手した結果、生成される局面をタブーリストに登録するのである。局面の記録にはゾブリストハッシュ [9] を利用する。ゾブリストハッシュとは乱数とハッシュ値を論理式で結合し、効率的に局面と手の組み合わせを記録するハッシュ法のことである。チェス、将棋、囲碁といったボードゲームでしばしば使われる。

局面タブーリストの構成だが、これについて本論文では 2 つの構成法を提案する。ひとつは従来のタブーリストを踏襲した逐次更新法、もうひとつはプレイアウトの勝敗に

† 愛知工業大学情報科学部情報科学科 (豊田市)
†† 株式会社日本デジタル研究所 (東京都江東区)

基づく一括更新法である。

局面タブーリストをモンテカルロ木探索に内包した場合の提案法の棋力とその特性について様々な数値実験を行った。具体的には、1) 19 路盤での対局、2) 同一局面重複数の評価、3) プレイアウトで無駄になる平均候補手数、4) 思考時間である。これらについて定量的に評価した。結果だけでなく考察も与えながら詳しく報告する。

2. モンテカルロ木探索

まず、囲碁 19 路盤の表記について説明する。碁盤左下隅を A1 とし、右上隅を T19 とする。つまり碁盤の横軸方向を記号 'I' を除く ABCDEFGHJKLMNOPQRST で、縦軸方向を下から上に 1 ～ 19 で表現する。

モンテカルロ木探索を図 1 を使って要点のみ簡単に説明する。モンテカルロ木の根ノードには深さ 0 を与える。根ノードに登録する局面は探索したい現局面である。図 1 は黒番第 k 手目を探索している様子である。まず合法手を適当に生成する。図 1 ではそれらが Q4 や C5 であり、第 k 局面を生成する。第 k 局面以降は実局面ではなく、仮想局面での対局になる。実局面は進展しない。

UCB 値の大きいノードを根から順に辿り、葉ノードでプレイアウトを 1 回だけ試行する。プレイアウトの勝敗は自分が勝てば 1、負ければ 0 である。相手の場合は 0 と 1 を入れ替える。勝敗結果が今度は葉から根まで逆に伝播され、そのパス上の勝率と UCB 値をすべて更新する。葉ノードでのプレイアウト数が閾値を超えれば、その葉ノードを展開する。元の葉ノードは内部ノードになる。根ノードを含む内部ノード（図中の ×）ではプレイアウトは一切実行しない。葉ノードでのみプレイアウトを実行する。

徐々にモンテカルロ木が成長し、特定の条件が満たされた場合に木探索を終了する。特定の条件とは、例えば CPU 時間やプレイアウト数の上限を事前に設定しておけば良い。どのタイミングで探索を終了しても根ノード直下の第 k 局面で最大勝率を与える第 k 手目が決定できる。この候補手を最良手として採用する。このようにモンテカルロ木探索の木探索は最良優先探索の挙動を取る。

Auer ら [4] が解決したスロットマシンへのコイン投入問題で、彼らは 4 つの決定論的方策を提案している。UCB1, UCB2, ϵ_n-GREEDY、そして UCB1-NORMAL の 4 つである。その中の最初の UCB1 のことを本論文では UCB 値と呼んでいる。

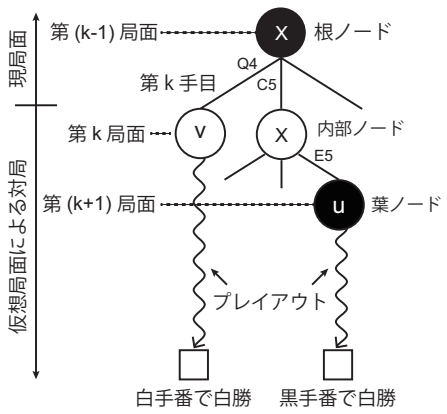


図 1 モンテカルロ木探索の概念図

複数のスロットマシンを複数の候補手に、1 枚のコインを 1 回のプレイアウトに置き換えれば、最大報酬は最大勝率に置き換えられる。UCT アルゴリズムで用いられるノード u での UCB_u の定義を式 (1) に示す。

$$UCB_u = \overline{X}_u + C\sqrt{\frac{2\ln n}{n_u}}, \quad C\text{: constant.} \tag{1}$$

ここで、$\overline{X}_u$ はノード u の勝率、n_u はノード u 以降のプレイアウト回数、n は u の親ノードでのプレイアウト総数である。図 1 にある葉ノード v や内部ノード X でもすべてのノードで UCB 値は適宜更新される。

モンテカルロ木探索の改善で顕著な功績を挙げているのが RAVE (Rapid Action Value Estimation)[10, 11] である。RAVE では、プレイアウトで自分が勝ったとき、途中で自分が打った手を全部「最初に打った」と見做して他の葉ノードの勝ち数に加算する。葉ノードでのプレイアウト数が早くに閾値に達し、葉ノードの展開が早くなる。その結果、探索木が早く深く成長する。RAVE では UCB 値と同じ構造をもった RAVE 値を使い、この両者を式 (2) のようにパラメータ $0 < \beta < 1$ で重み付けして探索木の葉ノード探索に利用する。

$$UCB\_RAVE_u = (1-\beta)\times UCB_u + \beta \times RAVE_u. \tag{2}$$

Progressive Widening[3] はプレイアウト数が増えるに従って徐々に候補手を増やしていく手法である。RAVE 同様、探索木の形状を直接的に制御する。

探索木を効率よく成長させるという観点から改善を試みたのが Virtual Loss[12] である。複数のコアをもつ同一マシン上で複数のモンテカルロ木探索を並列化するときに有効となる手法である。ある探索木に現局面を登録して複数のコア（これをスレッドという）で並列処理する状況を考える。通常、UCB 値を比較しながら木を降りると同一の葉に到達し、その葉でプレイアウトを行うことになる。これが続くと並列化した利点が失われかねない。そこで、あるスレッドで葉に到達してプレイアウトを実行して勝った場合、別のスレッドではその葉で仮想的に負けたことにする。すると、複数のスレッドで異なる UCB 値をもつことになり、スレッド毎に木の形状が異なる探索木をもつ可能性がある。部分木のロック機構が必要となるが、並列化した利点は活かせる。RAVE, Progressive Widening, Virtual Loss これらはすべてモンテカルロ木探索の木探索部分に着目した改善アプローチである。

一方、プレイアウト部分に着目した改善アプローチもある。まずプレイアウトの勝敗に直接的な影響を及ぼすのが 3×3 パターン [13] の利用である。プレイアウト中に局所的な死活を考慮できるので、ランダムな打ち手よりも明らかにプレイアウトの精度が向上する。

探索木の成長に直接的でなく間接的に影響を与える手法

のひとつに LGRF (Last Good Reply with Forgetting)[14] がある。LGRF はプレイアウト中のある局面で勝った手のみを記憶し、負けた手は忘却するという手法である。

提案する局面タブーリストの内包は、手をタブーリストに登録する手法 [6] を発展させたものである。タブーリストは同一局面を生成する手を禁じ手と見做して、プレイアウトの多様性を確保する機構として機能する。この考え方は木をより早く深さ方向に成長させる RAVE とは異なり、より均一的に木を成長させる Virtual Loss の考え方に近い。Virtual Loss は Tree 並列化を実現するために導入されたが、提案法はそれを単一の木で実現している。

本論文では局面タブーリストの更新手続き方法を 2 つ提案する。ひとつはプレイアウト中、単純に手の代わりに局面を順次タブーリストに登録していく逐次更新法である。もうひとつはプレイアウトの勝敗に基づいてタブーリストを一括で更新する一括更新法である。

3. 従来のタブーリストとその問題点

タブーリストを内包したモンテカルロ木探索 [6] は詰碁や 9 路盤囲碁といった比較的探索空間の狭い碁では有効な手法である。タブーサイズは詰碁で 5 程度，9 路盤囲碁で 12 程度が良好な結果となる。しかし、9 路盤囲碁で提案されている手法をそのまま 19 路盤囲碁に単純に適用しても統計的有意性を得ることはできなかった。つまり、プレイアウトにタブーリストを導入することの有効性を検証するに至っていない、といえる。そのため 19 路盤囲碁における探索空間の問題を解決すべく、さらなる改良と改善が必要となる。プレイアウト初手から数手先までにタブーリストを導入しても手を登録するだけでは、この探索空間の広さを克服することはできない。

4. 局面タブーリストの提案

本論文で提案するのは、タブーリストへの登録を従来の手から局面に改善することである。局面の表現にはゾブリストハッシュを利用する。タブーリストはモンテカルロ木探索の葉ノードだけに導入される。葉ノードでしかプレイアウトを試行しないからである。タブーリストの導入目的はプレイアウトの多様性確保にある。プレイアウト中の候補手の幅を広げて、結果として木探索中の好手を逃さないようにするためである。

プレイアウト中に手ではなく局面が異なるように打ち手を試行できれば、結果的に候補手の幅は広がる。

4・1 ゾブリストハッシュを用いた局面の表現

ゾブリストハッシュ [9] では、局面にハッシュ値を割り当て、手に乱数を割り当てる。そして、ある局面 r で手 x を打ったとき、次局面 s のハッシュ値を式 (3) で与える。

$$hash(s) = hash(r) \wedge rand(x) \tag{3}$$

ここで、2 項演算子 $\wedge$ はビット単位の排他的論理和である。19 路盤囲碁では通常、ハッシュ値、乱数値ともに 64 ビットを割り当てる。

図 2(a) で白◎ E5 の候補手で生成される局面 A が図 1 モンテカルロ木の葉ノード u に相当する。そこから❶初手でプレイアウトを開始する。この状況で式 (3) を簡単に説明する。紙面の都合もあるので、ここでは 4 ビットで局面のハッシュ値を与えることにする。同様に手にも適当な 4 ビットの乱数を与える。

例えば、図 2 で次のように適当な 5 値を与えてみる。

1. 局面 A のハッシュ値 $hash(A) = 1011$
2. ● C4 の乱数値 $rand(C4) = 0110$
3. ○ E3 の乱数値 $rand(E3) = 0011$
4. ● D3 の乱数値 $rand(D3) = 1010$
5. ○ E4 の乱数値 $rand(E4) = 0001$

プレイアウトを実行して局面 A から 4 手進んで局面 B になるまでには図 2(b) で❶ → ② → ❸ → ④と局面が変化する。これらの局面をそれぞれ A_1, A_2, A_3, A_4 と記す。もちろん局面 A_4 は局面 B のことである。一方、同じ局面 A から別のプレイアウトを実行して 4 手進んで局面 C になるまでには図 2(c) のようにやはり❶ → ② → ❸ → ④と局面が変化する。これらの局面を今度は A'_1, A'_2, A'_3, A'_4 で表す。先の A_1, A_2, A_3, A_4 と明確に区別するためである。やはり局面 A'_4 は局面 C と同じである。排他的論理和の $\wedge$ 演算だけで $A_4 \equiv A'_4$ となる同一ハッシュ値が得られることを確認してみる。

まず、局面 $A \rightarrow$ 局面 B ($\equiv A_4$) を確認する。

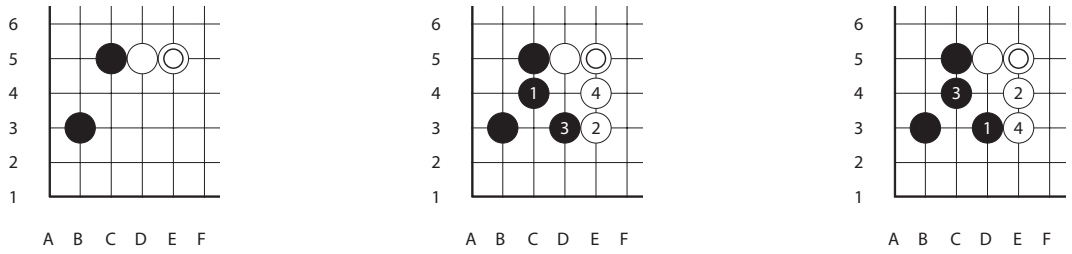
$$\begin{aligned}
hash(A_1) &= hash(A) \wedge rand(C4) \\
&= 1011 \wedge 0110 = \underline{1101} \\
hash(A_2) &= hash(A_1) \wedge rand(E3) \\
&= 1101 \wedge 0011 = \underline{1110} \\
hash(A_3) &= hash(A_2) \wedge rand(D3) \\
&= 1110 \wedge 1010 = \underline{0100} \\
hash(A_4) &= hash(A_3) \wedge rand(E4) \\
&= 0100 \wedge 0001 = \underline{0101}\ (= hash(B))
\end{aligned} \tag{4}$$

次に、局面 $A \rightarrow$ 局面 C ($\equiv A'_4$) を計算してみる。

$$\begin{aligned}
hash(A'_1) &= hash(A) \wedge rand(D3) \\
&= 1011 \wedge 1010 = \underline{0001} \\
hash(A'_2) &= hash(A'_1) \wedge rand(E4) \\
&= 0001 \wedge 0001 = \underline{0000} \\
hash(A'_3) &= hash(A'_2) \wedge rand(C4) \\
&= 0000 \wedge 0110 = \underline{0110} \\
hash(A'_4) &= hash(A'_3) \wedge rand(E3) \\
&= 0110 \wedge 0011 = \underline{0101}\ (= hash(C))
\end{aligned} \tag{5}$$

式 (4) と式 (5) のハッシュ値 $hash(B)$ と $hash(C)$ が等しくなり、着手順序が異なっても同一局面 ($B \equiv C$) を識別できることが分かる。

このゾブリストハッシュを局面タブーリストの要素として利用する。従来法では着手点の C4, E3, D3, E4 をタブーリストに登録していた。これを局面のハッシュ値に切



(a) 局面 A　(b) 局面 A から 4 手進んだ局面 B　(c) 局面 A から 4 手進んだ局面 C

図 2 プレイアウトの異なる手順で生成される同一局面の例

り替える。これだけで同一局面を避けてプレイアウトの多様性を確保できる。

因みに、連続する 2 つの局面でハッシュ値の排他的論理和を計算すれば、手の乱数値が得られる。これもゾブリストハッシュの特徴である。例えば、式 (5) 最後の $hash(A'_4)$ と $hash(A'_3)$ の排他的論理和は

$$hash(A'_4) \wedge hash(A'_3) = 0101 \wedge 0110 = 0011$$

となり、○ E3 の乱数値 $rand(E3)$ が得られる。

4・2　局面タブーリストの優位性

プレイアウトの多様性を確保し、探索木を均一的に成長させる。その結果、候補手の幅が広がり、好手が最善手となり得る可能性を残す。これがタブーリストの導入目的であった。この観点からタブーリストへの局面の登録が優位となり、手の登録が劣位となることを再度、図 2 を使って説明する。図 2 は 19 路盤でプレイアウト❶初手から④ 4 手目までを示している。

明らかに局面 B と局面 C で第 1 手目から第 4 手目まで、プレイアウト中の手はすべて異なっている。従って手をタブーリストに登録する場合、局面 B が既出であっても局面 C の第 4 手目④ E3 を受理せざるを得ない。同一手ではないのでタブー、つまり禁じ手と判定されないからである。これはプレイアウトの多様性にタブーリストが全く寄与しないことを意味する。一方、タブーリストに局面を登録していれば、プレイアウト第 4 手目で得られる局面 C は既出と判断され、図 2(c) 候補手④ E3 はタブー（禁じ手）と判定される。この判定により白は新たな 4 手目④合法手を生成し、同時に新たな局面を生成することになる。つまりタブーリストが有効に機能したことになる。

4・3　局面タブーリストの逐次更新法

再び図 1 を参照して、葉ノード v の深さを k $(= depth(v))$、葉ノード v が管理するタブーリスト (Tabu List) の集合を TL_v とする。TL_v はプレイアウト初手から第 M 手目までの M 個のキュー (queue) $Q_m,\ m = 1, 2, \cdots, M$ で実現する。つまり、$TL_v = \{Q_1, Q_2, \cdots, Q_M\}$ である。キュー Q_m の要素数 $|Q_m|$ がタブーリストのサイズ L_k である。これを以下、タブーサイズと呼ぶことにする。タブーサイズ L_k の添字 k はモンテカルロ木における葉ノード v の根からの深さである。また、提案法では、

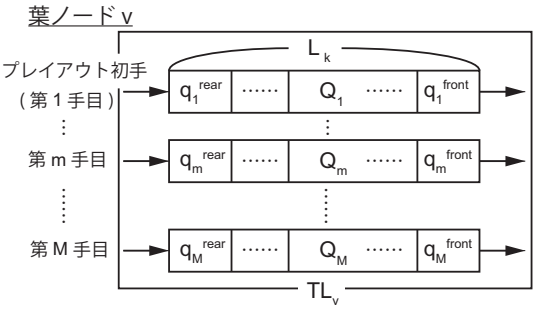


図 3 第 M 手目までに導入した局面タブーリストの構造

$|Q_1| = \cdots = |Q_M| = L_k$ を仮定する。

第 m 手目のタブーリスト Q_m の先頭要素を q_m^{front}、末尾要素を q_m^{rear} とする。図 1 の葉ノード v にプレイアウト初手から第 M 手目までに局面タブーリストを内包させたのが図 3 である。図 1 の葉ノード u にも同様の構造をもつ TL_u を内包させる。

要素 q_m^{ℓ} $(1 \le m \le M,\ 1 \le \ell \le L_k)$ に登録される値が局面のゾブリストハッシュ値となる。FIFO(First-In-First-Out) 構造をもつキューへのハッシュ値の追加と削除について詳しく説明する。

モンテカルロ木の根から UCB 値の大きい順に木を降りて、葉ノードに到達するとプレイアウトを 1 回だけ実行する。今、プレイアウト開始第 m 手目の合法手を x_m とし、局面が $r \xrightarrow{x_m} s$ に変化したとする。このとき式 (3) によって局面 s のハッシュ値は次式で求めることができる。

$$hash(s) = hash(r) \wedge rand(x_m).$$

そして次のようにしてキュー Q_m を更新する。

$$new\_q_m^{rear} = \begin{cases} hash(s) & if \quad hash(s) \notin Q_m \\ NIL & if \quad hash(s) \in Q_m, \end{cases} \tag{6}$$

$$Q_m \leftarrow Q_m \cup \{new\_q_m^{rear}\} - \{q_m^{front}\}. \tag{7}$$

ここで、演算子 $\cup$ と $-$ は要素 $\{\cdot\}$ のキューへの追加と削除であり、具体的にはキュー Q_m への Enqueue 操作と Dequeue 操作を示す。式 (6) の記号 NIL は空を意味する。

もし得られた局面 s のハッシュ値 $hash(s)$ がタブーリスト Q_m に未登録ならば $hash(s)$ を、登録済ならば代わりに NIL をキュー Q_m に追加する。$new\_q_m^{rear} = NIL$ の場合は、再び局面 r に戻って新たな合法手 x'_m を生成し、

新たな局面 s' を別途生成し直さなければならない。ただし、NIL が連続 L_k 回続けば、その次は必ず登録可能なハッシュ値が得られる。キューの大きさ $|Q_m|$ が L_k なので、"$hash(s) \in Q_m$" が連続 L_k 回続けば、キュー内のすべての要素が NIL になるからである。タブーリストに一旦登録された局面はタブーサイズ L_k 期間だけ探索できず、$L_k + 1$ 回目ではじめて再探索可能になる。これが結局はプレイアウトの多様性に繋がる。

局面タブーリストにはプレイアウト初手から第 m 手目（$1 \leq m \leq M$）まで、出現した局面ハッシュ値を図 3 縦方向に順次キューに追加していく。NIL が複数回追加されるかもしれないが、1 回のプレイアウトで交互に手が進むには、最終的には必ず次の局面ハッシュ値を追加することになる。注意すべきは、式 (6)〜式 (7) によるタブーリストへの局面追加は、プレイアウトで一手着手した直後に行われる、という点である。

さて、局面タブーリストでは、手をタブーリストに登録する方法と同様、可変長タブーリストによってタブーサイズ L_k を適宜短縮させる。序盤ではタブーサイズを大きくしてプレイアウトの多様性を高め、着手候補点が少なくなる終盤ではタブーリストが候補手生成の邪魔にならないよう小さくする。コウ・抜き・パスを考慮しなければ、N 路盤第 k 手目の局面タブーリストのタブーサイズ L_k は、式 (8) の短縮スケジュールで与えることができる。ここで、演算子 $\lfloor \cdot \rfloor$ は $floor$ 関数、L は初期タブーサイズである。

$$L_k = \begin{cases} L & if \quad 1 \leq k \leq \lfloor \frac{1}{4} N^2 \rfloor \\ \frac{2}{3} L & if \quad \lfloor \frac{1}{4} N^2 \rfloor + 1 \leq k \leq \lfloor \frac{2}{3} N^2 \rfloor \\ \frac{1}{3} L & if \quad \lfloor \frac{2}{3} N^2 \rfloor + 1 \leq k. \end{cases} \tag{8}$$

一般に、短縮スケジュールは $L_0 = L,\ L_{k-1} \geq L_k \geq L_{k+1},\ k = 1, 2, \cdots$ の条件を満たせばよい。

ここまでがプレイアウト中に手の代わりに局面をタブーリストに順次登録していく逐次更新法の提案である。

4・4　勝敗に基づく局面タブーリストの一括更新法

前節の逐次更新法では、プレイアウトで一手着手した直後にその局面のハッシュ値をタブーリストに無条件に追加した。その局面が仮に局面タブーリストに阻まれたとしても、局面をひとつ戻って別の合法手を生成し、局面ハッシュ値を計算し直し、式 (6)〜式 (7) を毎回適用した。

この部分に修正を施すのが局面タブーリストの一括更新法である。プレイアウト第 m 手目のキュー Q_m に対し、$\{Dequeue(q_m^{front})\ \&\ Enqueue(NIL)\}$ 操作と $\{Dequeue(q_m^{front})\ \&\ Enqueue(hash(s))\}$ 操作を明確に区別する。局面ハッシュ値の Dequeue 操作と Enqueue 操作において式 (6)〜式 (7) の更新手順を次のように改める。

まず、プレイアウトの第 m 手目 $(1 \leq m \leq M)$ で局面タブーリストに阻まれたケースだけを正しくキュー Q_m に反映させる。

$$\begin{cases} new\_q_m^{rear} = NIL \quad if \quad hash(s) \in Q_m \\ Q_m \leftarrow Q_m \cup \{new\_q_m^{rear}\} - \{q_m^{front}\}. \end{cases} \tag{9}$$

第 m 手目局面がタブーになることなく無事着手できた場合には、キューにその局面 s のハッシュ値 $hash(s)$ を即座に登録せず、一旦バッファ buf_m に保存する。

$$buf_m = hash(s) \quad if \quad hash(s) \notin Q_m. \tag{10}$$

1 回のプレイアウトが完全に終了した時点ですべてのバッファ buf_m $(1 \leq m \leq M)$ の値は確定している。

最後に、自分手番で始まるそのプレイアウトが負けた場合のみ

$$\begin{cases} new\_q_m^{rear} = buf_m \quad \text{for all } m \\ Q_m \leftarrow Q_m \cup \{new\_q_m^{rear}\} - \{q_m^{front}\} \end{cases} \tag{11}$$

によってバッファに溜めてあった M 個の局面ハッシュ値を M 個のキュー Q_m $(1 \leq m \leq M)$ にまとめて追加登録する。つまり、葉ノードに内包した M 個の局面タブーリストを一括更新する。結果的に更新後の局面タブーリストは前節の逐次更新法の結果と同じになる。

逆に、自分手番で始まるそのプレイアウトが勝った場合には、式 (9) の NIL の追加更新だけを行い、式 (11) の局面ハッシュの一括登録処理を省略する。このようにプレイアウトの勝敗に基づいて局面のタブーリストへの追加登録の有無を適宜決定する方式に改める。

式 (9)〜式 (11) の意味と意図を説明する。意味するところは、プレイアウト中に勝った手のみを記憶し、負けた手は忘却する、LGRF 手法 [14] と基本的に同じ発想である。

次にその意図である。「自分手番で始まるプレイアウトが負けた場合」とは、図 1 葉ノード u のプレイアウト例がこれに該当する。初手が黒手番のプレイアウトで白手番が勝利したのだから、黒手番は負けたことになる。このとき葉ノード u に導入した M 手分のタブーリスト TL_u の末尾要素をバッファに保存しておいた M 個の局面ハッシュ値ですべて一括更新する。負けたのだから、負けた局面をタブーリストに登録して、以降のプレイアウトでその局面を禁忌するよう仕向けるのである。負けた局面を避けられるので、直接的ではないが以降のプレイアウトを勝ちに誘導できる可能性がある。つまり、効果的にプレイアウトを多様化できる。

一方、「自分手番で始まるプレイアウトが勝った場合」とは、図 1 葉ノード v から始まるプレイアウト例がこれに該当する。葉ノード v では初手が白手番で始まるプレイアウトが試行される。そのプレイアウトが白勝だから、プレイアウト中はランダムながら良手を選択し、良好な局面を順次生成した、と考えて差し支えない。そうであるならば、その局面をわざわざタブーリストに登録して、その局面を避ける理由がない。よって、局面ハッシュ値の一括登録は敢えて行わず、式 (11) の処理過程を省略する。

自分手番で始まるプレイアウトが勝った場合でも、NIL の更新だけはタブーリストに正しく反映する必要がある。NIL とはその局面がタブーリストのキューにすでに登録済みであることを示す。もし NIL をキューに追加しなければ、その局面がタブーリストの要素として残留し続ける

表 1 19 路盤囲碁で使用した手法と各種パラメータの設定一覧

手法	MCTS	プレイアウト総数	閾値	タブーリスト	タブーサイズ	初手から	登録	勝敗利用
gnugo	–	–	–	–	–	–	–	–
gnugo-uct	◯	8000	1	–	–	–	–	–
gTabu0	◯	8000	30	–	–	–	–	–
gTabu18	◯	8000	30	◯	18	5 手まで	手	–
gTabu18-hash	◯	8000	30	◯	18	5 手まで	局面	–
gTabu18-hash-win	◯	8000	30	◯	18	5 手まで	局面	◯

ことになり、その局面の再探索が不可能となる。このような事態に陥らないよう NIL をタブーリストに追加して、その局面のタブー期間を 1 だけ減らすのである。

以上が勝敗に基づく局面タブーリストの一括更新法の提案である。

5. 数値実験

提案法の棋力や特性を評価するため、提案法をオープンソースの GNU Go 3.8[*1] に組み込んだ。19 路盤の対局ではオリジナルの GNU Go 3.8（思考ルーチン名: gnugo）と対戦させた。

5・1 比較する思考ルーチンとパラメータの選定

19 路盤囲碁で使用したプログラム名とパラメータの一覧を表 1 に示す。表 1 の gnugo はモンテカルロ木探索を搭載していない。GNU Go 3.8 がデフォルトでモンテカルロ木探索を 19 路盤囲碁に搭載していないからである。gnugo に 19 路盤でもモンテカルロ木探索が動作するよう修正したのが gnugo-uct である。提案手法の思考ルーチンはすべて gnugo-uct をベースに開発した。gnugo-uct の UCB 値に関する式 (1) 係数 C は、$C = 1.5$ を採用した。gnugo-uct と gnugo を対局させコミ 6 目半で先手後手の勝率がほぼ 50% になるように調整するための措置である。

gnugo-uct に文献 [6] のタブーリストを組み込んだ手法が gTabuX である。gTabuX のタブーリストには手が登録される。一方、局面をタブーリストに逐次登録する提案手法が gTabuX-hash である。さらに、gTabuX-hash に勝敗に基づくタブーリストの一括更新を採用した手法が gTabuX-hash-win である。X の部分には本来タブーサイズの L が入る。しかし、対局実験ではタブーサイズ $18(L = 18)$ のみに限定した。説明の煩雑さを避けるためである。モンテカルロ木で葉ノードを展開する閾値は 30 とした。そしてタブーリストの導入であるが、これはプレイアウト初手から 5 手 $(M = 5)$ までとした。対局では公平性を保つため、モンテカルロ木探索 (MCTS) で 1 手を決定する際のプレイアウト総数はすべて 8000 で統一している。ここまでを一覧にしたのが表 1 である。

文献 [6] を含めた提案手法の gTabuX, gTabuX-hash, gTabuX-hash-win には式 (8) のタブーサイズ短縮スケジュールを組み込んだ。19 路盤なので $N = 19$ であり、回数 $k = 1$ は互先での初手を意味する。このとき、タブーサイズ L_k は次式のように徐々に短縮されることになる。

$$L_k = \begin{cases} 18 & if \quad 1 \leq k \leq 90 \\ 12 & if \quad 91 \leq k \leq 240 \\ 6 & if \quad 241 \leq k. \end{cases}$$

5・2 19 路盤での対局結果

まず、各提案手法を先手 500 局、後手 500 局の計 1000 局で gnugo と対戦させた。対局結果を表 2 に示す。gTabu18 は gnugo に対して完全に負け越している。手のみをタブーリストに登録する gTabu18 は 19 路盤では局所的であるため有効に機能せず、勝率も良くない。その一方で局面をタブーリストに登録する gTabu18-hash は gnugo に対して勝ち越し、開発元になった gnugo-uct の勝率さえ改善している。さらに、勝敗に基づいて局面をタブーリストに一括登録する gTabu18-hash-win は、どの手法よりも gnugo に対する勝率が優っている。

対局の結果から有意水準 5%で二項検定を行った。結果を表 3 に示す。gTabu18-hash は勝率こそ改善できたものの p 値が有意水準 $\alpha = 0.05$ を越えているため、gnugo と統計上の有意な差はない。つまり棋力の明確な向上は得られなかった。一方で最も勝率が高かった gTabu18-hash-win の p 値は 0.012 となり、有意水準 $\alpha = 0.05$ を下回っている。これは統計的に gTabu18-hash-win と gnugo の棋力に有意な差があることを意味する。勝率から gTabu18-hash-win は十分な棋力向上を達成したと判断してよい。

5・3 同一局面重複数の評価

19 路盤でプレイアウトの多様性が確保されているかを検証した。プレイアウト初手から 5 手目までに出現した同一局面重複数で評価した。初期局面を図 4 として、● 43 手目を探索させた。プレイアウト初手から 5 手目までに探索した局面のハッシュ値を同一局面の評価に使った。ハッシュ値が同一ならば同一局面と判定した。使用した手法は gTabu0, gTabu18, gTabu18-hash, gTabu18-hash-win の 4 つである。数字の 0 や 18 はタブーサイズである。プレイアウト総数は対局同様すべて 8000 で統一した。重複した上位 50 局面を横軸に取った結果を図 5 に示す。

タブーリストを内包しない gTabu0 に比べ、タブーリストを内包した他の 3 つの提案手法の方が局面重複数が少ない。よってタブーリストを内包すれば、19 路盤でもプ

[*1] GNU Go http://www.gnu.org/software/gnugo/

表 2 19 路盤囲碁で gnugo と対局した結果

手法	黒手番での 勝数	勝率	白手番での 勝数	勝率	合計勝数 勝数	勝率
gnugo	513	—	487	—	513/1000	51.3%
gnugo-uct	266/500	53.2%	253/500	50.6%	519/1000	51.9%
gTabu18	239	47.8%	252	50.4%	491	49.1%
gTabu18-hash	264	52.8%	259	51.8%	523	52.3%
gTabu18-hash-win	277	55.4%	263	52.6%	540	54.0%

表 3 19 路盤囲碁での対 gnugo に対する二項検定の結果

手法	p 値	信頼区間	p 値 $<$ 有意水準 α
gnugo	0.429	0.482 – 0.544	No
gnugo-uct	0.242	0.488 – 0.550	No
gTabu18	0.591	0.460 – 0.522	No
gTabu18-hash	0.155	0.492 – 0.554	No
gTabu18-hash-win	0.012	0.509 – 0.571	Yes

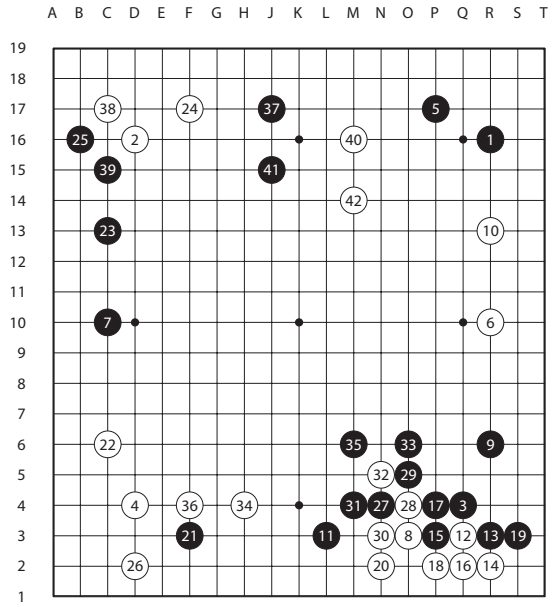


図 4 19 路盤の初期局面

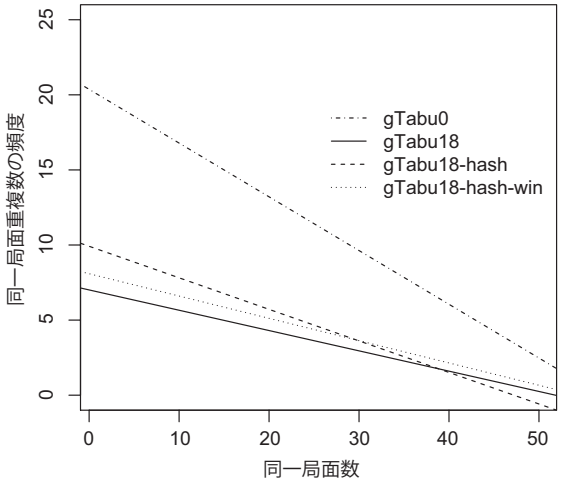


図 5 上位 50 局面の同一局面重複数

レイアウトの多様性をある程度確保できることが確認できた。gTabu18 は盤上の石の配置を考慮せず、手のみを管理している。そのため他の 2 つの手法と比べて、同一局面重複数は少なくなる傾向にある。

文献 [6] では 9 路盤で適当な初期局面から次の一手を探索する状況で同一手重複数を計測している。実験状況は異なるが、同一局面重複数は同一手重複数と比較し、その出現頻度は低くなる傾向がある。理由は 2 つ考えられる。

一つ目の理由は 19 路盤の探索空間の広さである。9 路盤の探索空間が 10^{38} 程度なのに対し、19 路盤の探索空間は 10^{171} もある。探索空間が大きくなれば、候補手数も増加する。候補手数の増加により同一局面の出現頻度は相対的に減少することになる。もう一つの理由は、同一手よりも同一局面の方が出現し難いからである。局面を比較する場合、盤面にある石の配置が少しでも異なると同一重複とはならない。局面が同一でも手順の途中にパスがあった場合、ハッシュ値が変更されてやはり同一局面とは判定されない。以上 2 つの理由により同一局面重複数の方が同一手重複数よりもその出現頻度は低くなる。

5・4 1 回のプレイアウトでタブーになる平均候補手数の評価

図 4 を初期局面として、● 43 手目を探索したときに 1 回のプレイアウトでタブーになる平均候補手数を計測した。タブーになる候補手総数は、総プレイアウト数 8000 回のうち手が生成されたにもかかわらず破棄された候補手の総数で評価する。よって 1 回のプレイアウトでタブーになる平均候補手数は、破棄された候補手総数を総プレイアウト数の 8000 で除した平均値で評価する。タブーサイズが 0 の gTabu0 では、候補手がタブーリストに阻まれるこ

表 4 19 路盤囲碁で 1 回のプレイアウト中にタブーになる候補手総数と平均候補手数

Method+ Tabu size	タブーになる候補手総数	タブーになる平均候補手数
gTabu0	0	0
gTabu1	375	0.047
gTabu2	685	0.086
gTabu3	941	0.118
gTabu4	1114	0.139
gTabu5	1288	0.161
gTabu6	1440	0.180
gTabu7	1655	0.207
gTabu8	1700	0.213
gTabu9	1835	0.229
gTabu10	1860	0.233
gTabu11	2065	0.258
gTabu12	2098	0.262
gTabu13	2031	0.254
gTabu14	2179	0.272
gTabu15	2348	0.293

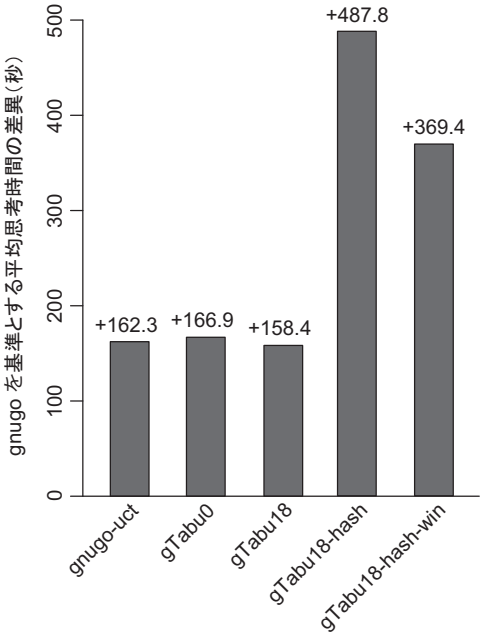


図 6 19 路盤対局で gnugo を基準とする平均思考時間の差異

とはないのでタブーになる候補手総数は当然 0 である。

実験結果を表 4 に示す。タブーサイズの増加に伴いタブーになる候補手総数も増加する。タブーリストは着手禁止点を意図的に増加させることに寄与するので、この結果は至極当然である。意外なのは平均候補手数である。

文献 [6] によれば、N 路盤でプレイアウト初手から M 手目までにタブーサイズ L のタブーリストを導入するとき、1 回のプレイアウト中にタブーになる平均候補手数 $\overline{T}(N,L,M)$ の上界は式 (12) で与えられる。

$$\overline{T}(N,L,M) < \frac{ML}{n}\left(\ln n + \gamma + \frac{L\pi^2}{6}\right). \qquad (12)$$

ただし、$n = N^2 - L - M$ とおいた。$\gamma \simeq 0.57721$ はオイラー定数（Euler's constant）[15] である。

19 路盤（$N = 19$）でタブーサイズ 12 のタブーリスト（$L = 12$）をプレイアウト開始 5 手まで（$M = 5$）に導入したとき、タブーになる平均候補手数は、$\overline{T}(19, 12, 5) < 4.562$ となる。数値実験で得られた表 4 からタブーサイズ 12 のそれを拾うと 0.262 である。総プレイアウト数 8000 回で実験しているので、1 回のプレイアウトにつき平均 0.262 手だけ候補手を無駄にしていることになる。整数に丸めても高々 1 である。この数値実験からモンテカルロ木探索にタブーリストを内包しても 1 回のプレイアウトでタブーになる平均候補手数は 19 路盤では非常に少ないことが判る。

手をタブーリストに登録する限りにおいて、モンテカルロ木探索の木探索処理にあまり負荷を掛けることはない。しかし、表 2 に示したように勝率という観点から、gTabu18 でさえ gnugo に対し勝率 50%に 0.9%届かない。

5・5 思考時間の検証

gTabuX のタブーリストに手を登録する場合、非常に効率が良いことは節 5・4 の数値実験で確認した。しかし、gnugo と対戦させると勝率は予想するほど伸びない。局面タブーリストを導入した gTabu18-hash や gTabu18-hash-win ならば、元になった gnugo-uct と互角かそれ以上の棋力になる。今度は gTabu18-hash や gTabu18-hash-win の思考時間について考察してみる。

UCT を実装していない gnugo と 1 局あたりの平均思考時間で比較してみる。実験環境を以下に示す。

オペレーティングシステム: Ubuntu 14.04.1 (x86_64)
プロセッサ: Intel Core i7-4790 CPU @3.6 GHz
メモリ: 32 GB

節 5・2 では先手 500 局、後手 500 局の計 1000 局対戦させ、その勝敗から二項検定を使って棋力を統計的に処理した。そのときの 1 局あたりの gnugo を基準とする平均思考時間の差異をグラフ化したのが図 6 である。

gnugo の平均思考時間を基準値 0.0 とすると、すべての手法で gnugo よりも思考時間が長い。gnugo はモンテカルロ木探索を搭載しておらず、知識ベースのみで手を探索するため 1 手にかかる思考時間が短い。しかし、それ以外の手法はモンテカルロ木探索でプレイアウトを 8000 回行っているので必然的に思考時間が長くなる。

gnugo-uct, gTabu0, gTabu18 の 3 つの手法の間に思考時間の差はほとんど見られなかった。この結果から手を登録するタブーサイズ 18 の gTabu18 は思考時間に大きな影響を与えることはないといえる。

一方で gTabu18-hash と gTabu18-hash-win の平均思考時間が大幅に増加している。図 5 から gTabu18-hash や gTabu18-hash-win がプレイアウト中にタブーリストに阻まれて禁じ手とせざるを得ない候補手生成数は gTabu18 と大差ない。違いはタブーリストの使い方である。手のみをタブーリストに追加する gTabu18 では、候補手それ自

体をタブーリストに登録する。局面をタブーリストに登録する gTabu18-hash や gTabu18-hash-win では、直前の局面と候補手から一手打った後の局面のハッシュ値を計算する。しかもプレイアウト初手から第 5 手目まで毎回ハッシュ値を計算することになる。おそらくこのハッシュ処理が gTabu18-hash と gTabu18-hash-win で思考時間がこれ程までに増加する原因になったと考えられる。

さて、図 6 で gTabu18-hash-win の思考時間が gTabu18-hash よりも減少している。この理由について考察してみる。gTabu18-hash がプレイアウト初手から第 5 手目まですべての局面を無条件にタブーリストに逐次登録するのに対し、gTabu18-hash-win はプレイアウトの勝敗に基づいてその局面をタブーリストに一括登録するか否かを取捨選択する。つまり、タブーリストに登録する回数が減少したため、それに伴い思考時間も減少した、と考えられる。

6. おわりに

モンテカルロ木の葉ノードに局面タブーリストを内包させることを提案した。タブーリストにはプレイアウト中の局面が逐次登録されていく。局面の記録にはゾブリストハッシュを利用した。局面タブーリストの更新では、プレイアウトの勝敗に基づいて負けたときのみ局面を一括登録する方法も併せて提案した。タブーリストを葉ノードに内包させる目的はそこから実行されるプレイアウトに多様性を持たせるためである。探索木を深さ方向だけでなく、より均一的に成長させて好手を逃さないようにするためである。タブーリストに登録された局面はタブー期間だけタブーになる。つまりその局面を生成する手を禁じ手にする。これによりプレイアウトの多様性を確保した。

19 路盤対局での数値実験から、勝敗に基づいて局面をタブーリストに一括登録する手法は二項検定の結果からも優位であり、gnugo-uct の勝率を最も改善した。逐次更新法は勝率こそ改善したが、統計的な有意差は得られなかった。手を登録する従来法は勝率 5 割にも満たなかった。

19 路盤囲碁で初期局面を与え● 43 手目を探索させて、同一局面重複数の出現頻度を計測した。結果からタブーリストへの登録は手だけでなく局面でも概ね良好であることを確認した。1 回のプレイアウトでタブーになる平均候補手数、これもかなり少ないことを数値実験で確認した。

平均思考時間については提案した局面タブーリストの内包は不調であった。局面タブーリストの一括更新法でも思考時間は長くなるが、逐次更新法はさらに劣る結果となった。ハッシュ処理の一部を見直す必要があるだろう。

謝　辞

本研究は文部科学省科研費基盤研究 (C) No. 25330441 の助成を受けて達成された。ここに謝意を表する。

参考文献

1) 村松正和: “コンピュータ囲碁の現状”, 情報処理, Vol. 53, No. 2, pp. 133–138, 2012.
2) 美添一樹: “モンテカルロ木探索 — コンピュータ囲碁に革命を起こした新手法”, 情報処理, Vol. 49, No. 6, pp. 686–693, 2008.
3) R. Coulom: “Computing Elo Ratings of Move Patterns in the Game of Go”, Computer Games Workshop 2007 (CGW 2007), 2007.
4) P. Auer, N. Cesa-Bianchi, and P. Fischer: “Finite-time Analysis of the Multiarmed Bandit Problem”, *Machine Learning*, Vol. 47, Issues 2–3, pp. 235–256, 2002.
5) L. Kocsis and C. Szepesvári: “Bandit based Monte-Carlo Planning”, Proceedings of the 17th European Conference on Machine Learning (ECML 2006), pp. 282–293, 2006.
6) 太田雄大, 伊藤雅: “タブーリストを内包したモンテカルロ木探索の詰碁と 9 路盤囲碁への応用”, 電気学会論文誌 C, Vol. 135, No. 3, pp. 331–339, 2015.
7) F. Glover: “Tabu Search — Part I”, *ORSA Journal on Computing*, Vol. 1, No. 3, pp. 190–206, 1989.
8) F. Glover: “Tabu Search — Part II”, *ORSA Journal on Computing*, Vol. 2, No. 1, pp. 4–32, 1990.
9) A. Zobrist: “A New Hashing Method with Applications for Game Playing”, Technical Report #88, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1970, reprinted in *International Computer-Chess Association (ICCA) Journal*, Vol. 13, No. 2, pp. 69–73, 1990.
10) S. Gelly and D. Silver: “Combining online and offline knowledge in UCT”, Proceedings of the 24th International Conference on Machine Learning (ICML 2007), pp. 273–280, 2007.
11) S. Gelly and D. Silver: “Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer Go”, *Artificial Intelligence*, Vol. 175, No. 11, pp. 1856–1875, 2011.
12) G. M. J.-B. Chaslot, M. H. M. Winands, and H. J. van den Herik, “Parallel Monte-Carlo Tree Search”, in Proceedings of the 6th International Conference on Computers and Games (CG2008), Springer, *Computers and Games*, Vol. LNCS5131, pp. 60–71, 2008.
13) S. Gelly, Y. Wang, R. Munos, and O. Teytaud: “Modification of UCT with Patterns in Monte-Carlo Go”, INRIA, Technical Report, RR-6062, 2006.
14) H. Baier and P. D. Drake: “The Power of Forgetting: Improving the Last-Good-Reply Policy in Monte Carlo Go”, *IEEE Trans. on Computational Intelligence and AI in Games*, Vol. 2, No. 4, pp. 303–309, 2010.
15) J. Havil: GAMMA Exploring Euler’s Constant, Princeton University Press, New Jersey, 2003.

（受理　平成 28 年 3 月 19 日）