モンテカルロ囲碁における探索木の情報を利用したシミュレーション方策の動的更新

板持 貴之[1]　三輪 誠[2]　鶴岡 慶雅[1]　近山 隆[1]

概要：多くのコンピュータ囲碁プレイヤにおいて，モンテカルロ木探索，特に UCT が広く持ちいられている．UCT では局面の評価を行うためにシミュレーションを行うが，近年では，シミュレーション方策を用いることで囲碁プレイヤの強化を行うことが多い．その際，多くの棋譜から事前に学習した結果を用いることが多いが，その方策は対戦中には変化せず，固定されたままである．そこで本稿では，対戦中に状況に応じてシミュレーション方策を動的に更新する手法を提案する．本提案手法を，静的な方策を利用したベースラインプレイヤと比較した結果，GNU Go との対戦実験では両者に差が見られなかったが，直接対戦により評価した結果，パラメータを正しく調整すればベースラインを上回る傾向を示す結果となった．

キーワード：コンピュータ囲碁, モンテカルロ木探索, シミュレーション方策

Dynamic Adaption of Simulation Policies using Information from the Search Tree in Monte-Carlo Go

Takayuki Itamochi[1]　Makoto Miwa[2]　Yoshimasa Tsuruoka[1]　Takashi Chikayama[1]

Abstract: Most of the recent strong computer Go players use Monte-Carlo Tree Search (MCTS), especially UCT. Recent UCT-based players often employ simulation policies to enhance the players. Most of simulation policies are learnt from many records in advance, but the policies are not updated during a game. In this paper, we propose a method to update simulation policies dynamically during a game. We compared our method with a baseline player with a static simulation policy. Although the baseline player and the player with our method did not show any significant differences when they played games against GNU Go, the player with our method showed a tendency to be better than the baseline player when they played games against each other.

Keywords: Computer-Go, Monte-Carlo Tree Search, Simulation Policy

1. はじめに

近年のコンピュータ囲碁プレイヤでは，Monte-Carlo Tree Search (MCTS) [1] の一種である Upper Confidence bound applied to Tree (UCT) [4] がよく用いられている．UCT は局面の評価を行うために，自己対戦により終局までゲームを進めるシミュレーションを行う．そのシミュレーションは局面の推定勝率を決めるものであり，その推定の精度はプレイヤ全体の強さに大きく影響する．そのため，シミュレーションを行う際の方策 (シミュレーション方策) を改善するため，プロや高段者の棋譜から様々な特徴を抽出し，棋譜中の着手を再現するように方策の改善を行う手法 [2] や，深い探索による評価値との誤差を減らすように最適化を行う手法 [7] が提案されてきた．

これらの手法は，棋譜から抽出された特徴を特徴ベクトルとして持ち，それに対する重みを目的関数に応じて最適化することで方策の改善を行っている．この事前に学習しておいた方策をシミュレーションに用いることによりプレイヤの改善を行っているが，対戦中 UCT によってゲーム木がどのように成長しようとシミュレーション方策そのものは変化しない．しかし，ある局面について UCT による

[1] 東京大学大学院工学系研究科
Graduate School of Engineering, The University of Tokyo
{itamochi, tsuruoka, chikayama}@logos.t.u-tokyo.ac.jp
[2] マンチェスター大学コンピュータ科学科
School of Computer Science, The University of Manchester
makoto.miwa@manchester.ac.uk

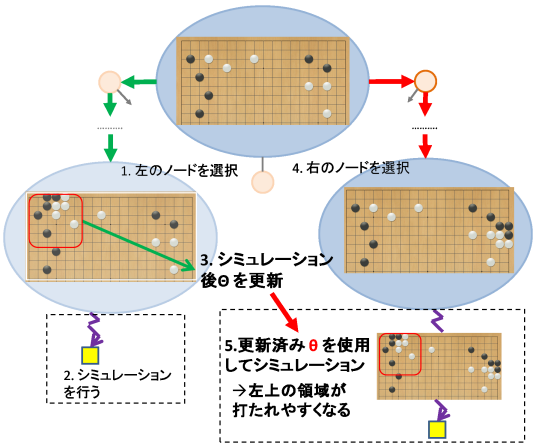

図 1　提案手法の概念図

探索が進めば，ゲーム木の情報 (どの着手がどのくらい選択されているか，等) からその局面についてある程度信頼性の高い着手が推定できる．このような高い信頼性を持つ着手・勝率や訪問回数といった情報は，木探索の上だけでなくシミュレーションにおいても有効だと考えられる．

そこで本稿は，対局中の状況に合わせたシミュレーション方策の改善を目的として，木探索の結果を用いて対戦中に動的にシミュレーション方策を更新する手法を提案する．

2. 関連研究

Rosin は MCTS を用いた一人用ゲームのプレイヤにおいて，動的にシミュレーション方策を更新する手法を提案している [5]．しかし，Rosin は方策の初期値が無い状態から動的に更新を行うことで改善を行っている．一方，本研究においては，その方策の初期値に予め棋譜から学習しておいたものを用い，そこから動的な更新を行うことで方策の改善を行う．これにより，過去に打たれた棋譜から強いプレイヤを得ることができ，そこからさらにプレイヤの強化を行うことができる．

また，MCTS による囲碁プレイヤで，Silver は強化学習によってゲーム中に動的にシミュレーション方策を更新する手法を提案している [6]．この研究では，木の上とシミュレーションで選択された一連の局面と着手について特徴量を取り出し，強化学習によって方策を動的に更新している．本研究では，シミュレーション上で選択された着手は信頼性が薄いと考え，探索木のみに注目して動的な学習を行う．

3. 提案手法

最初に，提案手法の基本的な考え方を説明する．例えば，図 1 のルートノードの局面について打つべき手を探索する場合を考える．まず，図 1 の 1, 2 のように，UCT によって木の左側のシミュレーションが行われたとする．シミュレーションが終わった後，3 で示しているように，UCT によって選択された状態と着手 (図の左側の盤面上で囲われている範囲) について，これらを表した特徴量を用いて方策 ($\boldsymbol{\theta}$) が更新される．続いて，4, 5 のように手順が選択され，シミュレーションが行われたとする．このシミュレーションでは，「3 で更新された方策 $\boldsymbol{\theta}$」を使ってシミュレーションを行うことになる．つまり，方策に何も変更を加えない従来の手法に比べ，「1 の木探索で選択された手順 (囲われている領域) と同様の着手を行いやすい」シミュレーションが行われる．また，局面と着手そのものに対して重みを持たず，特徴量に対しての重みを更新するため，着手位置の周辺パターン等が特徴量に入っていれば，「以前，木の上で選択された着手周辺と似たようなパターン」があれば，着手位置が異なっても打ちやすくなる．逆に，局面や着手そのものを示す特徴量が入っていた場合，以前打たれた位置と同じ局面・位置に打ちやすくなる．

Algorithm 1 Proposed Method

```
1: function SearchUCT(s_0, θ)
2:     while until no left time do
3:         s_l, seq ← TreeSearch(s_0)
4:         sim_result ← SimulationWithPolicy(s_l, θ)
5:         BackUp(s_l, sim_result)
6:         θ ← UpdatePolicy(seq, θ)
7:     return BestMove(s_0), θ
8: function TreeSearch(s)
9:     seq ← []
10:    while s is nonterminal do
11:        if visited(s) ≤ threshold then
12:            return child, seq
13:        else if v not fully expanded then
14:            child, action ← Expand(s)
15:            seq.append(s, action)
16:            return child, seq
17:        else
18:            child, action ← MaxUcbChild(s)
19:            seq.append(s, action)
20:            s ← child
21:    return s, seq
22: function UpdatePolicy(seq, θ)
23:    θ' ← θ
24:    for (s_m, a_m) ∈ seq do
25:        if visited(s_m) > threshold then
26:            θ' = θ' + Δ_θ(s_m, a_m)
27:    return θ'
```

続いて，具体的な手法の説明を行う．前述したように，本手法では「UCT の探索中に選択された手をより打ちやすくするように方策を更新する」という方針をとる．その目的関数は式 (1) となる．

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \left\{ \sum_{m=1}^{M} \ln \pi_{\boldsymbol{\theta}}(s_m, a_m) - \frac{\lambda}{2} |\boldsymbol{\theta} - \boldsymbol{\theta}_0|^2 \right\} \quad (1)$$

s, a は局面と着手 (s_m, a_m は木探索で探索された局面と着手) を表し，$\pi_{\boldsymbol{\theta}}(s, a)$ はその局面でその着手を行う確率 (方策) を示す．本手法では，方策に式 (2) の Softmax-Policy を用いている．

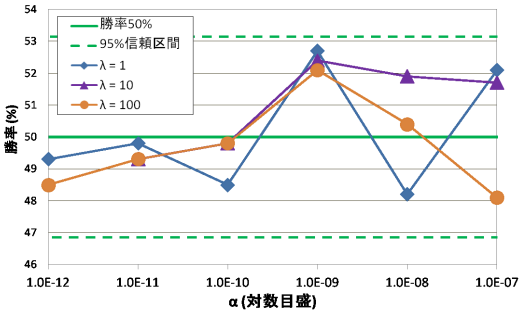

図 2　Static との対戦結果

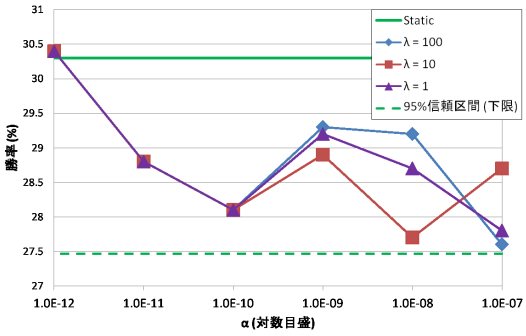

図 3　GNU Go との対戦結果

$$\pi_{\boldsymbol{\theta}}(s,a) = \frac{e^{\boldsymbol{\phi}(s,a)^T\boldsymbol{\theta}}}{\sum_b e^{\boldsymbol{\phi}(s,b)^T\boldsymbol{\theta}}} \qquad (2)$$

$\boldsymbol{\phi}(s,a)$ は局面と着手に対する特徴ベクトル，$\boldsymbol{\theta}$ は特徴ベクトルに対する重みを示す (式 (1) の $\boldsymbol{\theta}_0$ は $\boldsymbol{\theta}$ の初期値)．分母の $\sum_b$ は，与えられた局面 s の合法手ついて合計をとっており，正規化を行っている．この目的関数では打たれた手の対数尤度の最大化を行っている．また，$-\lambda|\boldsymbol{\theta}-\boldsymbol{\theta}_0|^2/2$ という項により，パラメータ λ を大きくすれば初期値からあまり離れすぎないように，逆に小さくすれば初期値を無視するようになる．実際に更新する際は，式 (1) の微分をとった式 (3) を用いる．

$$\Delta_{\boldsymbol{\theta}}(s_m,a_m) = \alpha\left\{\frac{d}{d\boldsymbol{\theta}}\ln\pi_{\boldsymbol{\theta}}(s_m,a_m) - \lambda(\boldsymbol{\theta}-\boldsymbol{\theta}_0)\right\} \qquad (3)$$

α は更新量を制御するためのパラメータである．

本手法では以上の更新式を用い，UCT の各イテレーションにおいて，各ノードにシミュレーション結果を伝搬した後に方策の更新を行う．Algorithm 1 に，提案手法の一部を擬似コードで示す．SearchUCT が最初に呼ばれる関数であり，探索対象の局面 s_0 と現在の方策の重み $\boldsymbol{\theta}$ を引数で受け取り，その局面の最善手と新しく更新された方策を返している．22 行目からが提案手法の部分であり，ここで「UCT で選択された (報酬を得た) 手順に関して $\Delta_{\boldsymbol{\theta}}$ で $\boldsymbol{\theta}$ を更新」している．ただし，25 行で示しているように，訪問回数の多いノードの方がより信頼できると考え，訪問回数が一定回数を超えたノードのみを更新対象としている．

4. 評価

自作の UCT による囲碁プログラムにおいて，9 路盤上での対戦実験を行った．比較対象として用いたアルゴリズムは以下のようになる．

- Static: Kombilo[*1] の棋譜を用い，Rémi による手法 [2] で求めた $\boldsymbol{\theta}$ で Softmax-Policy を利用したもの
- Dynamic: 提案手法．$\boldsymbol{\theta}_0$ には Static の $\boldsymbol{\theta}$ を用いる

特徴量には，Rémi が用いている特徴量 [2] のうち，3×3 パターン，Capture，Extension，Self-atari，Atari のシチョウ知識を用いないものに加え，1 手前，2 手前との距離 ($|dx|+|dy|+\max(|dx|,|dy|)$) が 3 以下かどうか，という特徴を使っている．また，シミュレーションを実行する時間に比べ，方策を更新する部分にかかる時間はわずかなものであるため，1 手あたりのシミュレーション回数は両方共 5,000 回とし，先手後手を入れ替えて合計 1,000 局対戦実験を行った．さらに，両プレイヤとも序盤については定石を利用し，UCT のパラメータ C には 0.3 を用い，50 回以上訪問されたノードのみ子ノードを展開し，Dynamic での更新対象とした．

以上の設定で Static と Dynamic の直接対戦による実験，GNU Go ver.3.8 (level 10) との対戦実験を行った．図 2，3 がそれぞれの結果となる．横軸は Dynamic のパラメータである α を対数目盛りで示しており，縦軸は勝率となっている．結果を見ると，Static との対戦においては，統計的に有意では無いものの，λ に関わらず $\alpha = 10^{-9}$ で提案手法が若干勝ち越している．しかし，GNU Go との対戦では勝率が Static を上回らず，わずかに負け越す (1%弱) 結果となった．

5. 考察

結果としては，直接対戦では Static を上回るが，GNU Go との対戦による間接的な比較では上回ることができなかった．本手法は「ツリー上で選択された手をシミュレーション中で優先的に打つ」という目的で，「更新時に用いるノードの選択」「方策の更新」を同時に行っていた．そのため，そもそも更新に用いているノードが正しいのか，という問題点に加え，用いた特徴が局面を抽象化しすぎており，本来の目的とは関係ない局面や着手に関しても更新を行っている可能性がある．そこで，この方針そのものの検証を行うため，以下の補足実験を行った．

与えられた局面に対して長時間かけて UCT を行い，その結果ルートノードで最も良かった手を「事前に分かっている良い手」とする．そして，シミュレーションにおいてその着手を行う確率を高くする．ただし「1 手目で打たなかったが 3 手目ではその手を打つ」といったパターンをシミュレーション中で実現したいため，ルートノードを深さ 1 とした時，深さ 3 以下のノードでのみ，その着手の確率

[*1] http://www.u-go.net/gamerecords/

Algorithm 2 Preliminary Method

```
1: function SelectBestMove(s_0)
2:     table ← [], table[all moves] ← 1.0
3:     best ← SearchUCT_LongTime(s_0, null)
4:     table[best] += α
5:     return SearchUCT(s_0, table)
6: function SimulationPolicyWithTable(s, table)
7:     while s is nonterminal do
8:         if the depth of s ≤ 3 then
9:             probs ← ConvertTableToProbs(table)
10:            move ← SelectMoveByProbs(probs)
11:        else
12:            move ← SelectOneMoveFromLegalMoves(s)
13:        s ← Move(s, move)
14:    return EvaluateWinning(s)
```

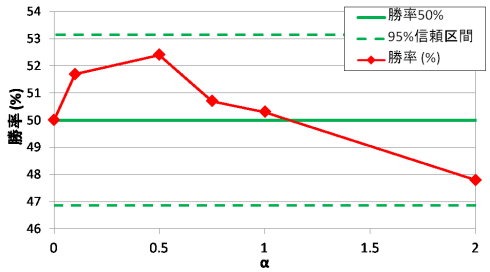

図 4　補足実験 - Plain との対戦結果

を高くする．RAVE [3] の考えと同様に，良い手は多少手順が異なっても結局打たれる可能性が高い，という考えを反映したアルゴリズムとなっている．Algorithm 2 が擬似コードでこれを表した例である．示していないコードは通常の UCT と同じである．3 行目で通常の探索に比べて長い時間をかけて「良い手」を決める．その次の 4 行目で，その手の重みを α だけ増やし，通常の探索で行われるシミュレーション中でこの手を打つ確率を高くしている．

このアルゴリズム (Cheated) について，合法手からランダムに指し手を決めるシミュレーション方策 (Plain) との直接対戦，GNU Go との対戦により評価を行った．その結果が図 4，5 となる．なお，1 手あたりのシミュレーション回数は 10,000 回とし，良い手を決めるためのの事前探索では 1 手あたり 100,000 回シミュレーションを行った．また，Cheated，Plain 両方共，序盤は定石を用いており，対戦は 1,000 回ずつ行った．

Plain との対戦結果を見ると分かるように，統計的に有意ではないが少し勝ち越す傾向を示している．また，GNU Go との対戦では，α 次第ではほぼ Plain と同程度の勝率を示した．この補足的な実験では，深さ 3 以下のノードのシミュレーションでのみ着手確率が偏るので，実際にこの偏った確率に従って着手を行う回数は，全体で行われる着手に比べて遥かに少ない．それでも Plain に勝ち越す傾向を示したため，本稿での提案手法の方針はある程度有望であると考えられる．これより，本提案手法の問題点として

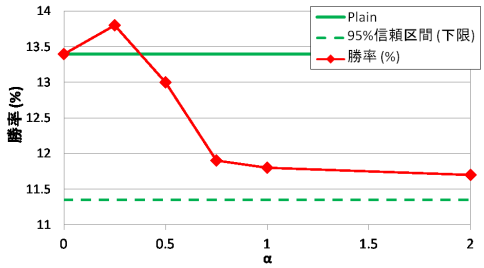

図 5　補足実験 - GNU Go との対戦結果

は，更新対象とするノードの選択方法正しくない，特徴量が局面と着手を抽象化しすぎている，といったことが原因である考えられる．

6. おわりに

本稿では，UCT を用いたコンピュータ囲碁プレイヤにおいて，シミュレーション方策の動的更新が重要とし，Softmax-Policy で用いられる重みを UCT で選択されたゲーム木上のノードの情報を用いて更新する手法を提案した．本手法は，「UCT で選択された手を後のシミュレーションでより打ちやすくする」という方針に基づいている．この手法を，方策をゲーム中に更新しないベースラインプレイヤ (Static) との対戦，GNU Go との対戦を通し，評価を行った．その結果，GNU Go との対戦では同等，直接対戦では 53% 近くの勝率となった．統計的に有意ではないものの，ベースラインプレイヤより強くなる傾向を示した．また，「打つべき良い手が事前に分かっている」場合，その手をシミュレーション中に打つ確率を上げる，という実験により，提案手法の方針自体の検証を行った．その結果，パラメータ次第ではベースラインプレイヤ (Plain) より強くなると思われる傾向を示し，本研究の方針自体は有望である可能性が残る結果となった．

今後は，更新対象とするノードの選択方法の考案，特徴量の追加，19 路盤での評価，などを行っていく予定である．

参考文献

[1] Coulom, R.: Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search, *CG*, pp. 72–83 (2006).

[2] Coulom, R.: Computing Elo Ratings of Move Patterns in the Game of Go, *Computer Games Workshop*, pp. 198–208 (2007).

[3] Gelly, S.: Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer Go, *Artificial Intelligence*, Vol. 175, No. 11, pp. 1856–1875 (2011).

[4] Kocsis, L. and et al.: Bandit based Monte-Carlo Planning, *ECML*, pp. 282–293 (2006).

[5] Rosin, C. D.: Nested Rollout Policy Adaption for Monte Carlo Tree Sesarch, *IJCAI*, pp. 649–654 (2011).

[6] Silver, D. and et al.: Sample-Based Learning and Search with Permanent and Transient Memories, *ICML*, pp. 968–975 (2008).

[7] Silver, D. and Tesauro, G.: Monte-Carlo Simulation Balancing, *ICML*, pp. 945–952 (2009).