Vol.2011-MPS-83 No.13
2011/5/17

深さに応じたバイアスによるモンテカルロ木探索の効率化

但馬康宏[†1]

ゲームにおける着手選択の方法として，モンテカルロシミュレーションに基づいた手法が注目されている．このシミュレーションの制御には UCT アルゴリズムが効果的であることが知られているが，これは k-armed バンデット問題に対する UCB1 アルゴリズムをゲーム木の各節点に適用したアルゴリズムである．しかし，木構造に対する探索アルゴリズムの特徴として，深い子節点に対して十分効果的な探索を行うには膨大な計算量が必要となってしまう．本研究では，UCT アルゴリズムにおける子節点の展開をその節点の深さに応じて経験的な評価関数を用いて制限することにより，実行の効率化を図った．その結果，従来の UCT アルゴリズムに対して同じ計算時間でより深い節点まで探索を行えることを確認し，着手の強さに向上がみられることが確認された．

A Monte-Carlo Game Tree Search with Bias which Depends on the Depth

Yasuhiro TAJIMA[†1]

In game tree search, the monte-carlo simulation method is a hot topic for selecting the best move. Especially, computer GO is growing rapidly with this method. UCT algorithm is one of the most effective algorithm to control random similations and UCT is based on UCB1 algorithm which is an effective algorithm for k-armed bandit problem. Neverthless, searching deep nodes is difficult because of the combinatorics problem. In this paper, we propose a restriction method for tree nodes exprosion by a huristic method for deep nodes creatioin. We evaluate this method on the game named Othello, and confirm that it works faster than the normal UCT algorithm.

1. はじめに

ランダムシミュレーションに基づいた着手決定手法は，モンテカルロ法と呼ばれ評価関数の作成が難しいゲームにおいて近年大きな成果をあげている．一般にゲームの着手決定アルゴリズムは，ヒューリスティックや機械学習により獲得された評価関数を用いて，ゲーム木を min-max 探索することにより実現されることが多い．しかし，ランダムシミュレーションの結果による手法はそのアルゴリズムの簡潔さに比べ強い手を選択する場合が多く，ゲームに対する新たなアルゴリズムとして注目を集めている．

数年前までは，ランダムシミュレーションの制御にヒューリスティック[8] や正規分布とみなした勝率の分布[6]，並列化やヒューリスティックとの組合せ[5]，さらに局所戦略をランダムシミュレーションで評価づけし，着手選択を行う方法[7] など様々な試みが行われていた．しかし近年になって，UCT アルゴリズムを中心にヒューリスティックの適用や機械学習の利用[3] を行う研究が一定の成果を上げている．

モンテカルロシミュレーションは，可能手すべてに対して同じ回数だけシミュレーションを行っていたのでは得られる勝率のばらつきや，明らかに強くない手に対するシミュレーションなど無駄が多くなる．そこでシミュレーションの制御を行うアルゴリズムが必要となる．この制御に対して，k-armed バンデット問題に対する解法の応用が広く知られている．k-armed バンデット問題とは，0 から 1 の値をとる k 個の確率変数 $X_i(i = 1, 2, \cdots, k)$ から一つを選択する操作 (試行) を n 回行い，そのとき得られる値の総和をできるだけ大きくする問題である．ここで，各確率変数は互いに独立で，それぞれの期待値は全試行を通じて固定されているものとする．この問題に対するアルゴリズムとして，UCB1 アルゴリズムが知られている[1]．UCB1 アルゴリズムは，任意の n 回の試行において，もっとも期待値の高い確率変数を選びつづけた場合との差 (取りこぼし) の上界を保証している．このアルゴリズムを思考ゲームに適用する場合，ゲームの各局面における可能手を k-arm とみなし，一つの可能手でゲームを進めた局面からランダムシミュレーションを行った場合の勝率を各確率変数の値として利用する．この適用の発展として，ゲーム木探索と組み合わせたアルゴリズム UCT[2] も多くの研究で実装されている．

以上のように UCB1 アルゴリズムとそのゲーム木への応用である UCT アルゴリズム

†1 岡山県立大学 情報システム工学科
Department of Systems Engineering, Okayama Prefectural University

はゲームの着手決定における有益なアルゴリズムであるが，特に UCB1 アルゴリズムはシミュレーションの総試行回数に関する仮定を持っていない．すなわち，任意の試行回数において獲得値の最大化を目指すアルゴリズムとなっている．しかし，ゲームの着手選択においては，一定の時間内にもっとも獲得値の高い arm ，すなわち勝率の高い着手を見つけることが重要である点が k-armed バンデット問題との相違となっている．

ここで，UCT アルゴリズムにおいては，子節点の展開方法もその効果に大きな影響を及ぼす．一般に UCT アルゴリズムでは，一定の深さのゲーム木をあらかじめ作り，根から UCB1 アルゴリズムにしたがって子節点を選択してゆく．作成した木の葉に到達するとそこからランダムシミュレーションを行い勝敗を葉に到達するまでの各節点に反映させ，この操作を複数回繰り返すことで最善手を見つけ出す．ゲーム木は，あらかじめ定めた深さで固定してもよいが，より良い手についてはより深く動的に子節点を展開する手法が一般的である．しかし，深い子節点に対する展開は，不利な手に対しても評価が定まるまではシミュレーションを重ねる必要があるため，その節点数の多さも手伝って膨大な計算時間を必要とする．そこで本研究では，UCT 木の深い部分ではヒューリスティックな評価関数を用いてその上位の手のみを展開し，木の大きさを制限することにより効率的な探索を行う．ヒューリスティックな関数を組み合わせることにより木の展開，子節点の選択を行う方法はいくつか既存の研究でも提案されている[4] が，その深さに対して効果を調節する手法は現在見当たらない．

本手法をオセロゲームに適用することにより，一般的な UCT アルゴリズムよりもより深い探索が可能となり，強い着手を選択できることが確認された．

2. UCB1, UCT アルゴリズムのゲーム探索への適用

k-armed バンデット問題は以下のように定義される問題である．

- k 個の確率変数 $X_1, X_2, \cdots, X_k$ は $0-1$ の範囲の値をとる．それぞれの期待値はあらかじめ定まっているが未知であり，アルゴリズム実行中に変化しないものとする．これらを arm と呼ぶ．X_i の期待値を μ_i で表す．X_i の値は分布 P_i にしたがうものとする．
- 1 回の試行とは，$X_i\ (i = 1, 2, \cdots, k)$ を 1 つ選択し，その値を得ることにより終了する．
- n 回の試行を繰り替えした後の総獲得値 $\sum_{j=1}^{n} X_{\Lambda(j)}$ を最大化することが目的である．ここで，$\Lambda(j)$ は，j 回目の試行における選択を表す．すなわち，j 回目の試行で X_i を選択した場合，$\Lambda(j) = i$ である．

$\mu_i\ (i = 1, 2, \cdots, k)$ の中で最大のものを μ^* と表す．同様に $\mu^* = \mu_i$ である i について，X_i を X^* と表す．あるアルゴリズムにおいて，合計 n 回の試行の中で X_i を試行した回数を $T_i(n)$ と表す．また，$\bar{x_i} = (1/T_i(n)) \sum_{1 \le j \le n, \Lambda(j)=i} X_i$ とする．さらに，$\Delta_i = \mu^* - \mu_i$ とする．

UCB1 アルゴリズムは，以下のとおりである．

(1) すべての $X_i\ (i = 1, 2, \cdots, k)$ について 1 度づつ試行する．

(2) 以下の値がもっとも大きな j について試行を行う．

$$\bar{x_j} + \sqrt{\frac{2 \ln n}{n_j}}$$

ここで n は現時点での総試行回数であり，n_j は現時点までに X_j を試行した回数である．

(3) 前ステップを繰り返す．

このとき，以下の定理が成り立つ[1]．

定理 1 UCB1 アルゴリズムを用いて n 回の試行を行ったときの選択の系列を $\Lambda_u(k)$ $(k = 1, 2, \cdots, n)$ とする．このとき，$n \cdot \mu^* - \sum_{k=1,\cdots,n} X_{\Lambda_u(k)}$ は以下の値で抑えられる．

$$\left(8 \sum_{i:\mu_i < \mu^*} \frac{\ln n}{\Delta_i}\right) + \left(1 + \frac{\pi^2}{3}\right) \left(\sum_{j=1}^{k} \Delta_j\right)$$

さらに，任意の $i = 1, 2, \cdots, k$ について，

$$E(T_i(n)) \le \frac{8 \ln n}{\Delta_i^2} + 1 + \frac{\pi^2}{3}$$

が成り立つ．すなわち，ある arm i が試行される回数の期待値は，Δ_i の二乗に反比例する．

囲碁や将棋などの二人確定完全情報零和ゲームの過程は，初期局面を根，可能手を枝に持ち，その可能手により進められた局面を子節点にもつゲーム木で表現することができる．上記 UCB1 アルゴリズムは，以下のようにゲームにおける着手選択に適用される．

- 現在の局面に対応するゲーム木の節点から，直接枝が張られているすべての子節点を k-armed バンデット問題の arm に対応させる．
- UCB1 アルゴリズムにおける試行は，対応する子節点からランダムに着手を選択し対戦結果をシミュレートし，その結果が勝ちならば $X_i = 1$ 負けならば $X_i = 0$ とする (図 1).

UCT アルゴリズムは図 2 に示すアルゴリズムで行われるゲーム木探索である． すなわ

Vol.2011-MPS-83 No.13
2011/5/17

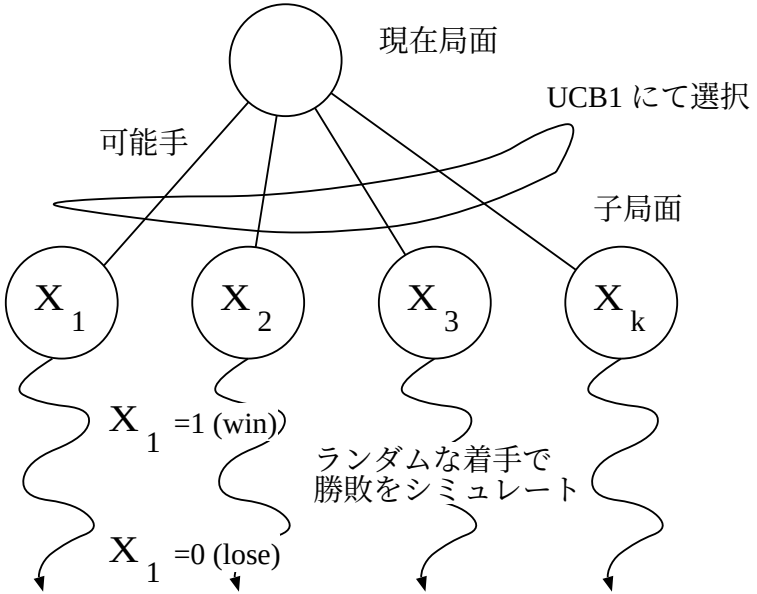

図 1 ゲーム木における UCB1 アルゴリズム

ち，ランダムシミュレーションを行うにあたり，部分的なゲーム木を展開し，木に含まれる節点においては UCB1 アルゴリズムにて次の手を選択し，葉に到達したらランダムに次の手を選択することで 1 回のシミュレーションとする手法である．この方法により，1 回のランダムシミュレーションの前半は過去の統計情報に基づいた選択でゲームが進められ，木の葉に到達した時点でランダムに着手が選択されてゲームが進められる．過去の統計情報に基づくことにより，木の探索においては min-max 探索と同じ効果をもたらしている．

3. 木の深さによる子節点の展開制限

ゲーム木の大きさはその終局まで展開すると膨大な節点数となるため，一般的には現在局面を根とし，展開できる範囲のみを保持することとなる．UCT アルゴリズムにおいては，初めはある少ない定数段数の木を作成し，シミュレーションの到達回数が増えるにしたがって，葉を逐次追加してゆき大きな木とすることが一般的である．しかし，この方法でも末端の節点数は指数関数的に増大するため，ある程度の大きさをこえると木の展開が困難と

```
UCT algorithm
初期ゲーム木 t を作成する; (ex. 根 1 段の子節点)
各節点の勝敗数を初期化;
while (制限時間まで繰り返す) begin
    注目節点 := 根;
    while (注目節点が t の葉となるまで) begin
        注目節点から UCB1 アルゴリズムにより子節点 s を 1 つ選択;
        注目節点を s とする;
    end
    注目節点からランダムシミュレーションを行う;
    結果を根から注目節点までの各節点に反映させる;
    if (注目節点の訪問回数が展開条件を満たした) then
        注目節点のすべての可能手に対する子節点を t に付け加える;
    fi
end
根の子節点のうち勝率が最大となる手を選択する;
```

図 2 UCT アルゴリズム

なる．

本研究では，ヒューリスティックな評価関数を用いて，ある節点から展開する子節点の数を制限することにより，より有望な手のパスにつながる葉を多く展開し，探索の効率を高める (図 3)．

すなわち，より深い節点は，着手の選択に与える影響が相対的に低くなるため，ランダムシミュレーションの回数を評価が十分定まるまで積み重ねるためのコストがより高くなる．そのため，深い節点から展開を行う場合は，ヒューリスティックな評価関数による評価が低い手は展開しないことで，そのコストを抑える．

具体的には，以下の手順で節点を展開する．ゲーム木の葉 l について，可能手が m 手存在するとする．このとき，すべての可能手を展開するならば子節点 $v_1, v_2, \cdots, v_m$ が生成される．評価関数 f を用いて $f(v_1), f(v_2), cdots, f(v_m)$ を値の低い順 (節点 l にとって評価の高い順) に並べ替え，そのときの節点の順を $u_1, u_2, \cdots, u_m$ と表す．節点 l から u_i の上

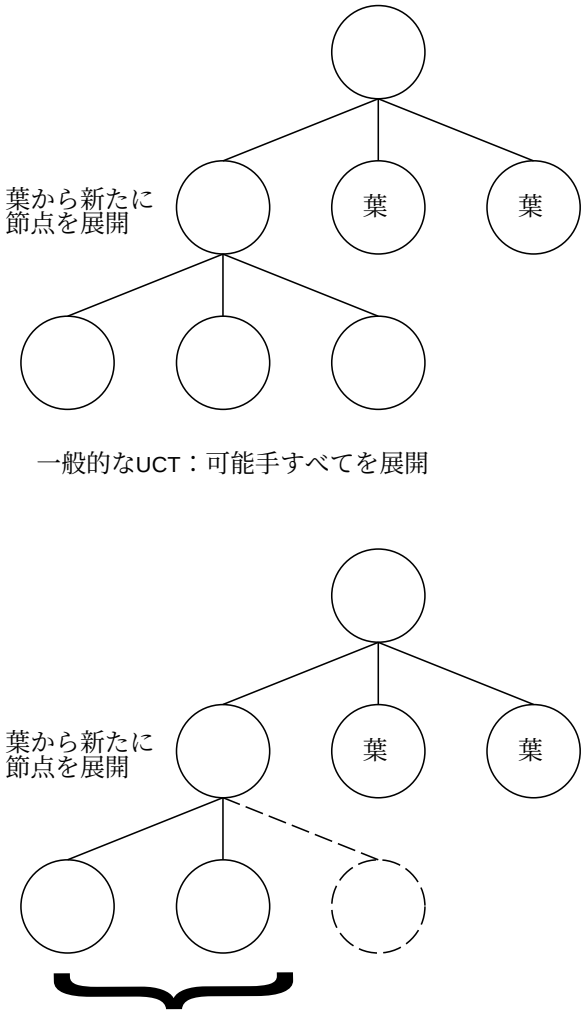

図 3 本手法による節点の展開

位 p の子節点のみを探索木に追加し，アルゴリズムを続行する．このとき，展開する節点の割合 p は，深さ i に依存して以下の値とした．

$$p = \max\{(1 - \frac{i}{n}), 0\}$$

ここで，n はこの値よりも深い節点については展開しないようにする定数である．

過去の研究においても，Prograssive Pruning として，評価関数を用いて展開の幅を制御した例はあるが，本研究においては，探索木の深さに依存する点がポイントである．

4. 評価実験

評価実験はオセロを題材とし，以下の仕様の UCT アルゴリズムを比較対象とした．

- 1 手あたりの思考時間 5 秒および 10 秒．
- 葉においてランダムシミュレーションが 10 回到達すると子節点の展開が行われる．
- 初期探索木は，根と 1 段めの子節点すべて．
- 対戦数は，1 手 5 秒では 100 ゲーム，1 手 10 秒では 68 ゲーム．
- 評価関数は，その局面の得点 (コマの数) と可能手の数の線形和であり，得点の重みを 1 ，可能手数の重みを 10 とした．

以上の条件において，本手法によるアルゴリズムと，子節点の生成に制限を設けない一般的な UCT アルゴリズムとの対戦実験を行った．

結果は，1 手 5 秒の場合，本手法が先手では本手法の 56 勝 42 敗 2 引き分け．本手法が後手では 52 勝 46 敗 2 引き分けとなった．いずれの場合も勝ち越していることがわかる．また，1 手 10 秒の場合は，本手法が先手では本手法の 22 勝 42 敗 4 引き分け，本手法が後手では 37 勝 29 敗 2 引き分けとなり，わずかに負け越している．これは，1 手あたりの計算時間が長い場合，評価関数を用いて展開しなかった節点に最善手が含まれている確率が上がるためだと思われる．改善策としては，評価関数の制度を良くする方法が考えられる．本手法における評価関数は，UCT のシミュレーションの最中に利用されるものであり，制度よりも計算速度が求められるが，利用されるのは子節点の展開時のみであるため，ある程度の時間をかけても制度よく評価のできる評価関数のほうが有効であると思われる．

5. おわりに

モンテカルロ法を用いたゲーム木探索において，深い節点ほど展開に制限を加えた UCT 法を提案し，オセロによる評価実験で思考時間が短い場合に従来の手法よりも有利であるこ

Vol.2011-MPS-83 No.13
2011/5/17

表 1 対戦結果

	本手法勝数	UCT 勝数	引き分け
1 手 5 秒 (本手法先手)	56	42	2
1 手 5 秒 (本手法後手)	52	46	2
1 手 10 秒 (本手法先手)	22	42	4
1 手 10 秒 (本手法後手)	37	29	2

とを示した．今後の課題として，本手法に適した評価関数の設計方法，子節点の展開に制限をかける割合の算出法の工夫などが挙げられる．

参 考 文 献

1) P.Auer, N.Cesa-Bianchi and P. Fischer, Finite-time analysis of the multiarmed bandit problem, Machine Learning, vol.47, nos.2,3, p.235–256, 2002.
2) Levente Kocsis and Csaba Szepesvari, Bandit based monte-carlo planning, Lecture Notes in Computer Science, vol. 4212, pp.282-293, 2006.
3) Sylvain Gelly and David Silver, Combining online and offline knowledge in UCT, Proc. of the 24th International Conference on Machine Learning, pp.273-280, 2007.
4) Bruno Bouzy, Move-Pruning Techniques for Monte-Carlo Go, LNCS 4250, pp.104–119, 2006.
5) Haruhiro Yoshimoto, Kazuki Yoshizoe, Tomoyuki Kaneko, Akihiro Kishimoto, and Kenjiro Taura, Monte Carlo Go Has a Way to Go, Proc. of the 21st National Conference on Artificial Intelligence (AAAI-06), pp. 1070-1075, 2006.
6) Remi Coulom, Efficient selectivity and backup operators in Monte-Carlo tree search, Proc. of the 5th International Conference on Computer and Games, CG2006, pp.72–83, 2006.
7) Tristan Cazenave, Bernard Helmstetter, Combining tactical search and Monte-Carlo in the game of Go, Proc. of IEEE conference on Computational Intelligence and Games, CIG2005, 2005.
8) Bruno Bouzy and Bernard Helmstetter, Monte Carlo go developments, Advances in Computer Games conference (ACG-10), pp. 159-174, 2003