UCT アルゴリズムにおけるノード展開手法の提案

矢島享幸[†1] 橋本剛[†1]

近年囲碁プレイヤではモンテカルロ (MC) 木探索法が広く使われている．その木探索，MC 部分それぞれで質の向上と効率の向上に関する研究が盛んに行われている．本研究では，木探索の効率化に焦点を当てる．

木探索の効率化を実現する方法として，これまで枝刈りに関する研究が多くされてきた．しかし木探索にとって重要なノード展開の条件については，あまり研究がされてこなかった．

木探索におけるノード展開の方法として，全末端展開手法と訪問回数展開手法がある．実験によって訪問回数展開手法が優位であることを確認した．

次に訪問回数展開手法の改良として，勝率の突出度と訪問回数の推定を用いた展開手法の 2 種類提案する．

検証実験から提案した 2 つの効率化手法は従来の訪問回数展開手法に対して有意に強いことを確認した．また各枝刈りとの比較実験を行い，枝刈りと比べて本提案手法は探索の早い段階から効果が得られることを確認した．

Proposal of node expansion methods on the UCT algorithm

TAKAYUKI YAJIMA[†1] and TSUYOSI HASHIMOTO[†1]

Monte-Carlo tree search method flourishes in computer Go in recent years. Both the tree search part and the MC part are respectively widely studying concering improvement of quality and improvement of efficiency. This paper focuses on the efficiency of search tree.

Pruning were well studied as a method for tree search efficiency. However node expansion has not well been studied despite its importance for search tree.

As methods of node expansion for tree search, each leaf node expansion method and visiting number expansion method have been proposed. We confirm the superiority of visiting number expansion method by experiments.

Moreover we propose two methods for improving the visiting number expansion method, prominent winning percentage expansion and visiting number estimation expansion.

We confirmed that both two proposed efficiency methods have significantly stronger strength than the existing visiting number expansion method by experiments. Furthermore, we performed comparing experiments with pruning methods and confirmed that proposed expansion methods work in the earlier stages than pruning.

1. はじめに

モンテカルロ法と木探索アルゴリズムを組み合わせたモンテカルロ木探索アルゴリズムの登場[1] により，コンピュータ囲碁プレイヤは飛躍的な進歩を遂げた．

以降，囲碁プレイヤにおいてモンテカルロ木探索アルゴリズムが主流となっており，盛んに研究が行われている．現在では数学的な裏付が明確な UCT アルゴリズムが主に使われている[2]．

UCT では，あるノードから末端ノードまで木を下り，そこからプレイアウトを行う．条件によっては新たな末端ノードを展開し，そこからプレイアウトを行う．一般的にプレイアウトとは，ランダムに手を選ぶモンテカルロシミュレーション (以降，MC) を指す．プレイアウトの結果は上に繰り上げられ，木探索が行われる．この作業を繰り返して探索木を成長させていく．

UCT アルゴリズムに置いて末端ノードを展開して新しいノードを作る条件は極めて重要であるが，その条件について言及した研究は少ない．数少ない例として，Coulom は末端ノードの訪問回数が当該局面の兄弟ノード数を超えた場合に新しいノードを展開する手法[1](以降，訪問回数展開手法) を紹介している．この手法は現在世界最強クラスの囲碁プログラム CrazyStone が用いていることもあり，広く使われていると思われる．また，CrazyStone 同様の強豪囲碁プログラムである MoGo では，木探索で末端ノードまで下る毎に新

†1 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology

しいノードを展開する手法 (以降，全末端展開手法) を用いている．ただし，自分たちのプログラムで比較実験を行った結果，前者の手法が優勢であった (この実験結果は 4.2 節で述べる).

よって，本研究では CrazyStone が用いるノード展開手法に焦点を当て，展開条件を改良することで UCT アルゴリズムの効率化を目標とする．

CrazyStone が用いる訪問回数展開手法では展開条件が満たされるまでプレイアウトが繰り返される．UCT で用いられるプレイアウトは一般的にランダムゲームを終局まで行うため，通常の minmax ベースの評価関数よりも計算量を要する．そのため，ノードが展開される迄に多大な計算量を必要とする．そこで UCT の実行過程で展開されるノードを早めに予想し展開することが出来れば，いわゆる当然の一手があるような簡単なノードでのプレイアウトを少なくし，難しい局面に多く時間を割ける．結果，計算量が節約出来るため，効率的なアルゴリズムを実現できると考えられる．

本稿では，UCT の実行過程で展開されるノードを予想し展開する手法を 2 種類提案し，それぞれについて検証を行う．

ノード展開手法に焦点を当てた効率化手法として，筆者が以前提案した手法[3] がある．また，ノード展開手法とは異なる UCT 効率化手法として，既に Progressive Pruning[4] をはじめとした手法が提案されている．2 章では，それら既存の UCT 効率化手法の紹介を行う．3 章では提案する手法の解説を行う．4 章では提案手法や既存の枝刈り手法について検証実験を行う．5 章で実験結果から提案手法と既存の探索効率化手法を比較・検証する．

2. 関 連 研 究

UCT+MC 法が提案され，囲碁プレイヤは飛躍的な性能向上を遂げた．この UCT+MC 法における効率化の研究は MC と UCT の改良があり，それぞれに対して効率化と質の向上についての研究が行われている．

MC の質の改良では，プレイアウトでの精度を上げるための評価関数が研究されている．評価関数を自動調整する手法として，少数化-最大化アルゴリズム[5] や Bonanza メソッドを用いた学習手法[6] が提案されている．また，MC の効率化については RAVE[7] などが挙げられる．

UCT の質の向上は，評価関数を用いる Progressive Widening[8] に代表される．UCT の効率化では，枝刈りの研究が多く行われている．UCT における枝刈りとして Progressive Pruning[4] や，但馬ら[9] や北川ら[10] の手法が挙げられる．

本研究では UCT の効率化に焦点を当てるため，本章では現在提案されている UCT 効率化手法である枝刈りや筆者が以前提案したノード展開手法について述べる．

2.1 Progressive Pruning

B. Bouzy ら[4] は UCT の基本となる多腕バンディット問題を効率よく解くために Progressive Pruning を提案した．Progressive Pruning では合法手 i の勝率 X_i の信頼上限 X_{Ri}・下限 X_{Li} を以下の式 (1)・(2) を用いて計算する．

$$X_{Li} = X_i - C * \frac{\sigma_i}{\sqrt{s_i}} \quad (1)$$

$$X_{Ri} = X_i + C * \frac{\sigma_i}{\sqrt{s_i}} \quad (2)$$

ここで σ_i は標準偏差，s_i は合法手 i の訪問回数，C は勝率の信頼上限と下限の精度を決める値である．ただし，この時 $s_i > 0$ である．

勝率の信頼上限・下限を計算して以下の式が成り立つならば合法手 j の勝率が合法手 i の勝率を逆転する可能性が低いと判断して合法手 j の探索を省略する．

$$X_{Li} > X_{Rj} \quad (3)$$

この手法ではプレイアウトを繰り返す必要が有るため，プレイアウト回数が少ない場合は効果を得られない．つまり，総プレイアウト回数が多い場合は枝刈りの効果が得られやすい．逆に総プレイアウト回数が少ない場合は枝刈りの効果が得られにくい．

2.2 北川らの手法

実際のコンピュータゲームプレイヤでは探索を無限に続けることは不可能であり，一定のシュミレーション回数で探索を打ち切る必要がある．このとき各合法手の残りプレイアウト回数を予測することで，より精度の高い信頼上限と下限を計算できる．この上限と下限に基づいて，今後プレイアウトを続けたとしても選ばれる可能性の少ない手の探索を打ち切る．

ある程度のプレイアウトが行われた探索木におけるルートノードでの合法手 i について考える．合法手 i のプレイアウト回数を s_i，探索における総プレイアウト回数を n，合法手の数を k とすると，UCT では勝率の高い手ほど多くプレイアウトが行われるため，合法手 i の残りプレイアウト回数 e_i は以下の式で求められる．

$$e_i = \frac{X_i}{\sum_{j=1}^{k} X_j} n \quad (4)$$

式 (1)・(2) と式 (4) を用いて，到達上限確率 P_{Ri} と到達下限確率 P_{Li} は以下の式で求められる．

$$P_{Ri} = \frac{X_i + e_i X_{Ri}}{s_i + e_i} \quad (5)$$

$$P_{Li} = \frac{X_i + e_i X_{Li}}{s_i + e_i} \quad (6)$$

式 (5)・(6) を用いることで，ProgressivePruning よりも高い精度で枝刈りが可能となる．

2.3 但馬らの手法

但馬らが提案した手法も，実際のコンピュータゲームプレイヤでは一定のシュミレーション回数で探索を打ち切る必要が有るという考えに基づいている．

この手法も，探索木におけるルートノードでの合法手について考える．現時点の総プレイアウト回数を n とすると，勝率 X をもつ合法手 i の訪問回数の推定値 $F(X)$ は以下の式 (12) で求められる．

$$F(X) = \frac{8 \ln n}{(x^* - X)^2} + 1 + \frac{\pi^2}{3} \quad (7)$$

x^* とは，ルートノードの子ノード中で最大の勝率を示す．また，合法手 i の勝率が $X > x^*$ となるためには，合法手 i が最低

$$\frac{x^* - X}{1 - X}(n_i + 1)$$

回数プレイアウトを行われる必要が有る．n_i は合法手 i の現時点での訪問回数を示す．

この勝率 x^* を持つ合法手を手 j とすると，合法手 i が 1 回試行されると，j は $\max(1, \frac{n_j}{F(X)})$ 回試行される．つまり，$X > x^*$ まで変化する間に j は少なくとも Z 回試行される．Z の値は以下の式 (8) で求められる．

$$Z = \sum_{k=1}^{\frac{x^* - X}{1 - X}(n_i + 1)} \max(1, \frac{n_j}{F(X + \frac{1 - X}{n_i + 1})k}) \quad (8)$$

この時，残りの探索回数を P とすると，以下の式が成り立つ場合，手 i の探索を省略する．

$$Z + \frac{x^* - X}{1 - X}(n_i + 1) > P \quad (9)$$

2.4 枝刈りの問題点

これまでに，現在提案されている枝刈り手法について説明をしてきた．枝刈り手法の中で Progressive Pruning は，枝刈りを行うためにはある程度の回数プレイアウトを行う必要がある．そのため，探索時間が短い場合は枝刈りの効果が得られにくいと考えられる．その他の手法では残りプレイアウト回数を基準としているため，残りプレイアウト回数が少ない時に枝刈りの効果を発揮する．よって，探索時間が短い場合は枝刈りの効果が得られやすいと考えられる．しかし，探索時間が長いと枝刈り用の計算が負担となって，逆に効率が悪化してしまうと考えられる．

これら枝刈りの特徴については 4.5 節の実験で検証する．

2.5 遷移確率を用いた UCT アルゴリズムの改良

筆者らは以前に評価関数を用いるゲームの知識を必要とするノード展開手法[3] を提案した．

この手法では，評価関数から合法手 i への遷移確率 $P(i)$ を求める．そして遷移確率の高い，打たれる確率が非常に高いノードについて展開を行う手法である．展開条件を式にすると以下のようになる．

$$P(i) \geq Th \quad (10)$$

この条件と訪問回数展開手法[1] を組み合わせて探索木を成長させていく．評価関数には勾配法を用いた学習[6] の結果を用いた．ただし，この手法はゲームの知識に依存しているという欠点がある．

3. 展開ノードの推定と UCT アルゴリズムの効率化

UCT の実行過程で展開されるノードを早めに推定し展開することが出来れば効率的なアルゴリズムを実現できると考えられる．ここで問題となるのが展開されるノードをいかに予測するかである．

ただし，既存のノード展開手法[3] はゲームの知識に依存しているため，知識表現の難しいゲームや新しく知識の集積のないゲームでは適用が難しい．

そこで本研究では知識に依存しない，各ノードにおける勝率を用いて展開されるノードを推定する．

本研究では，展開されるノードを推定するにあたって 2 種類の方法でアプローチを行った．まず，勝率の突出度を用いたノード展開手法を提案する．次に，訪問回数の推定を用いたノード展開手法を提案する．

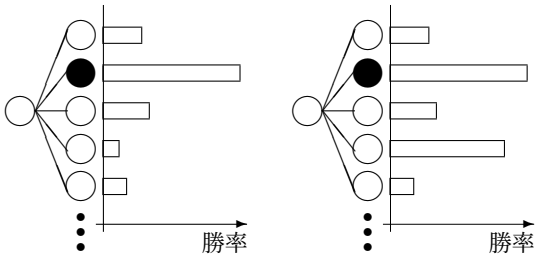
3.1 勝率の突出度を用いたノード展開手法

3.1.1 概　　念

UCT で探索木を下った先の末端ノードについて考える．

末端ノードとその兄弟ノードに複数有望なノードが有る場合はどれが展開されるか判断出来ないため，充分に探索とプレイアウトを行って展開するノードを決める必要があると思われる．対して，末端ノードの勝率が兄弟ノードに比べて突出している場合は展開される可能性が高いため，プレイアウトをあまり行わずに展開したい．

そこで，本提案手法では勝率の信頼区間を計算し，Singular Extensions[11] の様に 1 つだけ突出しているノードは展開される可能性が高いと判断して早めにノードを展開する (図 1(a))．勝率が突出したノードが複数有る場合にはノードの展開を行わずに探索を終了し，プレイアウトを行う (図 1(b))．



(a) 展開を行う場合　　(b) 展開を行わない場合

図 1　勝率の突出度を用いたノード展開の概念

以降で勝率の突出度を用いて展開するノードを決定する方法を述べる．

3.1.2　勝率の区間推定

本手法ではノードの勝率はプレイアウトの結果をそのまま用いていない．より信頼性を上げるために勝率の信頼上限 X_{Ri} と信頼下限 X_{Li} を用いる．X_{Ri} と X_{Li} は，Progressive Pruning と同様に式 (1)・(2) を用いて計算した．

3.1.3　ノード展開条件

UCT で選択された末端ノードの勝率が突出しているか調べる．

UCT で選択された末端ノードを $i1$，$i1$ を除いた兄弟ノード中で最も高い勝率をもつノードを $i2$ とする．閾値を Th とすると，以下の条件が成立する場合にノードを展開する．

$$X_{Li1} - X_{Ri2} \geq Th \tag{11}$$

この条件と訪問回数展開手法[1] を組み合わせて探索木を成長させていく．

ノードの訪問回数が少ない場合は，勝率が信頼性が低い．しかし本手法では，突出度を調べる時に勝率の区間推定を用いている．訪問回数の少ないノードの場合は勝率の信頼区間 $[X_L, X_R]$ の幅が広いため，式 (11) に照らし合わせたとしてもノードの展開は発生しない．式 (11) を満たしノードが展開されるには，ある程度のプレイアウトが行われ推定される区間が狭まらなければならない．

3.2　訪問回数の推定を用いたノード展開手法

3.2.1　概　　念

UCT は探索木中の各ノードを訪問する．本手法では UCT の探索が終了した時点で，ノードを訪問した回数の推定を試みる．

モンテカルロ木探索アルゴリズムにおいては，各ノードは勝率と訪問回数に関する情報を保有している (場合によってはそれ以外の情報を保持していることも有る)．そして，それらの情報を元に UCB 値を計算し，最も大きな UCB 値を持つノードを辿っていく．

この UCT アルゴリズムの探索について各ノードの訪問回数は，兄弟ノード中最大の勝率と自身の勝率の差の二乗に反比例する事が判っている[12]．ノード i の推定訪問回数の期待値 $E(T_i(n))$ は以下の性質を持つ．

$$E(T_i(n)) \leq \frac{8 \ln n}{(x^* - x_i)^2} + C \tag{12}$$

$T_i(n)$:ノード i の訪問回数
n :ノード i の親ノードの推定訪問回数
x^* :ノード i の兄弟ノード中最大の勝率
x_i :ノード i の勝率

ここで，ノード i の最大推定訪問回数を

$$F(i) = \frac{8 \ln n}{(x^* - x_i)^2} + C \tag{13}$$

と置く．ルートノードの推定訪問回数が全体の総プレイアウト回数の推定値であり，各ノードの n の値は帰納的に計算される．

探索が時間打ち切りの場合，総プレイアウト回数は指定された時間で実行されるプレイアウト回数を推定して用いる．探索がプレイアウト回数で打ち切られる場合，総プレイアウト回数は設定された総プレイアウト回数を用いる．

式 (13) で各ノードの最大訪問回数が推定出来る．この式を用いて得られた最大推定訪問回数をノード展開に使用する方法を以下に述べる．

3.2.2　ノード展開条件

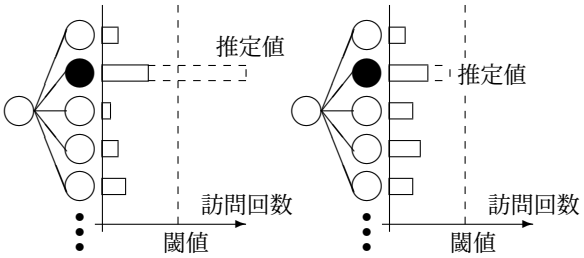
本提案手法では，式 (13) を用いて各ノードの推定訪問回数を求め，訪問回数展開条件[1] を満たすなら充分な訪問を行う前にノードを展開する (図 2(a))．つまり，末端ノード i の兄弟ノード数を N_i とすると，以下の条件の場合が成立する場合にノードを展開する．

$$F(i) \geq N_i \tag{14}$$

逆に末端で選択されたノード i の推定訪問回数 $F(i)$ が式 (14) を満たさない場合は，最後まで訪問回数展開手法の展開条件を満たさないと判断して，展開は行わない．(図 2(b))．

この条件と訪問回数展開手法を組み合わせて探索木を成長させていく．

この手法は，探索時間が長ければ長いほどルート局面に近い深さのノードはすぐに展開される．つまり，探索時間が長いほど効率化の効果が得られやすいと考えられる．



(a) ノードを展開する場合　　(b) ノードを展開しない場合

図 2　訪問回数の推定を用いたノード展開の概念

4. 実　　験

3.1・3.2 節で提案した 2 種類のノード展開手法を既存の囲碁ゲームプレイヤに実装した．

9 路盤において提案手法を実装したプログラムと，従来の手法のみのプログラムで対戦を行い，提案手法の有効性を検証した．囲碁ゲームプレイヤは Nomitan[13] を用いた．Nomitan の詳細は文献[13] を参照していただきたい．

囲碁ゲームプレイヤの特徴は以下の通りである．

- UCT+MC 法
- 評価関数を用いたプレイアウトの改良[6]
- RAVE[7]
- Transposition Table
- ノード展開条件：訪問回数展開手法[1]

4.1 実 験 条 件

実験は以下の環境で行った．

- 実験環境 1
  - OS ：Windows XP SP2
  - CPU ：Intel(R) Core 2 Duo　2.66GHz
  - メモリ ：2.00GB
- 実験環境 2
  - OS ：Windows XP SP2
  - CPU ：Intel(R) Core i7　3.2GHz
  - メモリ ：3.00GB

なお，並列計算は行っていない．また，定石は使用していない．

提案手法は UCT 末端ノードで適用され，条件を満たす場合はノードを展開し，一手深く読む．一手深く読んだ際には新たな末端ノードで再度提案手法を適用し，条件を満たすならばノードを展開する．

4.2 訪問回数展開手法と全末端展開手法の比較

囲碁ゲームプレイヤの Nomitan に訪問回数展開手法と全末端展開手法を実装し，どちらがより強力な手法かを検証する．

実験環境 1 を用い実験を行った．対戦では 9 路盤を用い．一手 5 秒の探索を行った．一戦毎に先後を入れ替えてそれぞれ 100，計 200 局の対戦を行った．以下に対戦結果を示す．

表 1　訪問回数展開手法と全末端展開手法の比較結果

	勝利数	勝率
訪問回数展開手法	133	66.5 %
全末端展開手法	67	33.5 %

実験結果より，訪問回数展開手法が有意に強いことが判った．よって囲碁ゲームプレイヤ Nomitan のノード展開条件には訪問回数展開手法を用いる．

4.3 勝率の突出度を用いた手法の実験結果

実験環境 1 を用い，突出度を用いた手法の式 (11) における閾値 Th を 0.1〜0.5 まで変化させた時の従来手法に対する有効性を検証した．

実験では 9 路盤を用い．一手 5 秒の探索を行った．一戦毎に先後を入れ替えてそれぞれ 50，計 100 局の対戦を行った．このとき，式 (1)・(2) の係数 C は，1.96 を用いた．図 3 に対戦結果を示す．

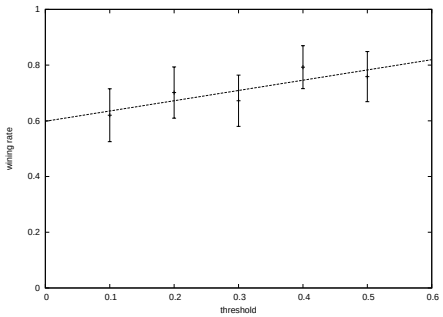


図 3　勝率の突出度を用いたノード展開手法の実験結果

実験結果から閾値 Th が高いほど，高い勝率が期待出来ることが判った．

閾値 Th が小さいと勝率が下がってしまうのは，突出度の少なく実際に展開されるか判らないノードについても展開してしまったためと考えられる．この結果は筆者が以前提案した手法[3] の実験結果に合致する．

4.4 訪問回数の推定を用いた手法の実験結果

実験環境 1 を用い，3.2 節で提案した訪問回数の推定を用いた手法を実装し，9 路盤において提案手法の有効性を検証した．探索時間は一手 5 秒の時間打ち切りとした．また，対戦は先後を交互に入れ替えてそ

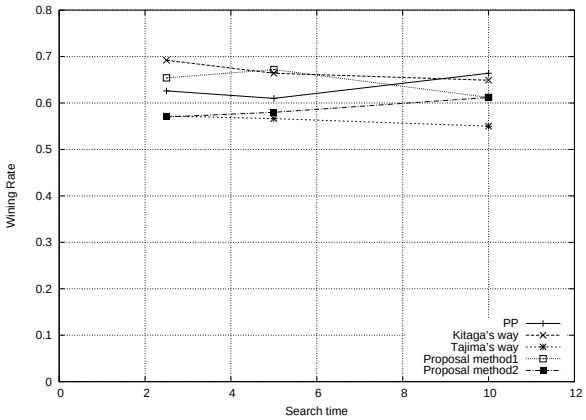


図 4 探索時間を変えた場合の各効率化手法の勝率の変化

れぞれ 500，計 1000 局行った．この際，式 (14) の定数 C は以下の値を用いた．表 2 に対戦結果を示す．

$$C = 1 + \frac{\pi^2}{3}$$

表 2 訪問回数の推定を用いたノード展開手法の実験結果

	勝利数	勝率 [%]	95 %信頼区間
提案手法	610	61.0	± 3.0
従来手法	390	39.0	± 3.0

実験の結果，提案した手法の勝率は 61(±3.0) %で有意に強くなっていることが判る．

4.5 枝刈り手法との比較

実験環境 2 を用い，各枝刈り[4)9)10)] や 3.1 節と 3.2 節で提案した手法の探索時間を変えた場合に，従来プログラムに対する勝率がどのように変化するかを検証した．この時，従来プログラムも各手法と同じ時間だけ探索を行った．対戦は先後を交互に入れ替えてそれぞれ 250，計 500 局行った．

探索時間と各効率化手法の勝率の変化を示したグラフが図 4 である．横軸が探索時間 [sec]，縦軸が勝率である．また勝率の突出度を用いた手法では，式 (11) の閾値 $Th = 0.3$ を用いた．

図 4 から判るように，全効率化手法とも従来手法に対して高い勝率を示している．この中で但馬や北川の手法は探索時間が増えると勝率が低下している．これは，2 つの手法が総探索回数を考慮した枝刈り手法であるため，探索終了間際に最も枝刈りが発生する．結果，探索の終盤でしか枝刈りが起こらず，探索時間が増えると枝刈りが起こる探索の割合が減って効率が悪化するためと考えられる．

但馬らや北川らの手法に対し，ProgressivePruning は探索時間が増えると勝率が向上している．これは 2.1 節での予想通りであるが，探索時間が短い場合でも比較的優秀な勝率を示すことが判った．

図 4 から訪問回数の推定を用いた展開手法 (Proposal method2) は探索時間が延びると勝率が向上することが判る．これは 3.2 節で述べた通り，探索時間が延びると式 (13) の n の値が大きく大きくなるためノード展開の効果を得られるやすくなるからだと考えられる．

対して勝率の突出度を用いた手法 (Proposal method1) は全般的に高い勝率を示しながらも，探索時間が延びると勝率の低下が見られた．

4.6 効率化手法の進行

提案手法・既存の枝刈り手法の進行を調べる．実験環境 1 を用い，探索は時間ではなくプレイアウトが 25000 回行われた後に打ち切ることにした．

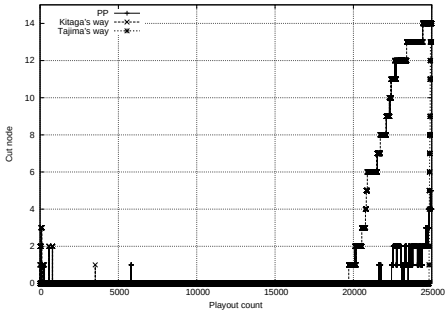


図 5 枝刈りの進行 (一手目・探索回数 25000)

図 5 に 1 手目のシミュレーション回数と枝刈りの進行を示すグラフを載せる．横軸がプレイアウト回数を表す．縦軸はプレイアウト $n-1$ 回目が終わってから n 回目が行われる間に枝刈りが行われた回数を表す．

図 5 から，枝刈りはプレイアウトがある程度行われ

るか，探索終盤でしか行われていないことが判る．

次に勝率の突出度を用いた展開手法の 1 手目のシミュレーション回数とノード展開の進行の関係を示すグラフを載せる．図 6 の横軸がプレイアウト回数を示す．縦軸は n 回目のプレイアウトを行う前に，末端ノードで式 (11) を満たして展開された回数である．

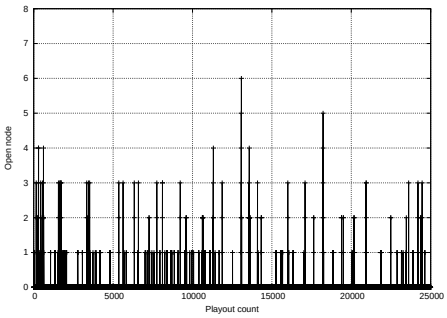


図 6 勝率の突出度を用いた展開手法のノード展開の進行 (1 手目)

図 6 を見ると，プレイアウトの開始間際から勝率の突出度を用いた展開が行われていることが判る．

以降に探索訪問回数の推定を用いた展開手法の 1 手目のシミュレーション回数とノード展開の進行の関係を示すグラフを載せる．図 7 の横軸がプレイアウト回数，縦軸は n 回目のプレイアウトを行う前に，末端ノードで式 (14) を満たして展開された回数を示す．

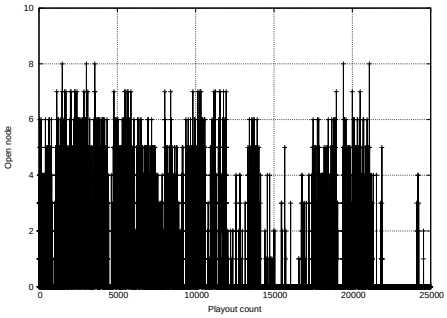


図 7 訪問回数の推定を用いた展開手法のノード展開の進行 (1 手目)

図 7 を見ると訪問回数の推定を用いた展開手法は，勝率の突出度を用いた展開と同様にプレイアウトの開始間際からノードの展開が行われている．また，図 6 と図 7 を比較すると，訪問回数の推定を用いた展開手法の方がより多くノード展開を行っていることが判る．

最後に効率化手法の進行を検証した際の，各手法の 1 秒当たりのプレイアウト回数を比較する．対戦における各探索の 1 秒当たりのプレイアウト回数の平均値をまとめた表が表 3 である．

表 3 各手法の探索速度比較

効率化手法	プレイアウト回数/秒
なし	6502.53
PP	5688.29
北川	6032.40
但馬	5857.50
訪問回数推定	4502.31
勝率突出度	6323.06

表 3 を見ると訪問回数の推定を用いた手法の 1 秒当たりのプレイアウト回数が最も低くなっていることが分かる．効率化手法を実装していないプレイヤに対して，30 %以上も低下している．

5. 考　　察

4 章の実験結果から，2 種類の提案手法についての考察を行う．

図 5，図 6，図 7 から，今回提案した 2 つの手法と既存の枝刈り手法[4)9)10)] との挙動の大きな違いとして，初期段階からノード展開が発生するという点が挙げられる．

また，図 4 を見ると訪問回数の推定よりも勝率の突出度を用いた手法の方が全体的に良い結果が得られている．2 つの提案手法のノード展開の進行は図 6，図 7 に示されているが，訪問回数の推定を用いた手法の法が多くノードが展開されている．このことから，訪問回数の推定を用いた手法は無駄なノード展開を行っている可能性が考えられる．また，表 3 から 1 秒当たりのプレイアウト回数の低下が訪問回数の推定を用いた手法の勝率に悪影響を及ぼしていることが考えられる．それでも従来手法に対して有意に強い結果を示しているため，探索速度の改善を行うことで，より良い結果が得られるだろう．

6. ま と め

本研究では UCT アルゴリズムのノード展開に焦点を当てた．現在主流のノード展開手法である訪問回数展開手法と全末端展開手法を対戦させ，より強力な訪問回数展開手法について 2 種類の効率化手法を提案した．

また，囲碁を題材として提案手法を実装したプログラムと未実装のプログラムを対戦させて有効性を検証した．実験結果より提案手法を実装したプログラムが未実装なプログラムに対して充分に強力であることが判った．さらに，訪問回数の推定を用いたノード展開手法では，探索時間が長くなると従来手法に対して勝率が向上することが判った．

7. 今後の課題

今回提案したノード展開手法は既存の枝刈り手法と両立が可能である．既存の枝刈り手法，特に Progressive Pruning と本提案手法を組み合わせることで，より良い効率化が実現できると考えられる．今後の問題としては，今述べたように提案手法と既存の枝刈り手法を組み合わせた場合の検証を行う必要がある．特に探索時間の長さに依る効率的な組み合わせ方を探す必要がある．

また 5 章で触れたが，訪問回数の推定を用いた展開について，勝率の突出度を用いたノード展開よりも勝率が悪いのは，無駄なノード展開をしているからか，探索回数の遅さが原因か調べる必要がある．探索回数の遅さについては，総訪問回数 $F(i)$ を求める式 (12) を簡略化することで，探索速度の向上を図る．

さらに，CGOS サーバーでのレーティングの検証を行っていきたい．

参 考 文 献

1) Remi.Coulom, Eficient selectivity and backup operators in Monte-Carlo tree search, LNCS vol. 4630, pp. 72-83, 2007.
2) L.Kocsis, C.Szepesvari, Bandit Based Monte-Carlo Planning, LNCS vol. 4212, pp. 282-293, 2006.
3) 矢島享幸，松井利樹，野口陽来，橋本剛，遷移確率を用いた UCT アルゴリズムの改良，第 3 回エンターテイメントと認知科学シンポジウム，pp. 32-35，2009.
4) B.Bouzy, Move-Pruning Techniques forMonte-Carlo Go, CG 2005 LNCS, vol. 4250, pp. 104-119, Springer, Heidelberg, 2006.
5) David Stern, Ralf Herbrih, Thore Graepel, Bayesian pattern ranking for move predition in the game of Go. In Proeedings of the 23rd international onfereneon Mahine learning, pages 873-880, Pittsburgh, Pennsylvania, USA, 2006.
6) 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，ゲーム情報学（GI），Vol. 2009 No. 27, pp. 33-40, 2009.
7) S.Gelly, D.Silver, Combining online and offline knowledge in UCT, Proceeding sof the 24th International Conference on machine Learning, PP. 273-280, OmniPress, 2007
8) Remi.Coulom, Efficient Selectivity and Backup Operators inMonte-Carlo Tree Search, Proceedings of the 5th International Conference on Computers and Games, Italy, 2006.
9) 但馬康宏，小谷善行，UCT アルゴリズムにおける確率的な試行回数削減方法，ゲーム情報学（GI），vol. 2008, No. 59, pp. 23-30, 2008.
10) 北川竜平，栗田哲平，近山隆，投入計算量の有限性に基づく UCT 探索の枝刈り，第 13 回ゲームプログラミングワークショップ 2008，pp. 46-53, Nov. 2008
11) T.Anantharaman, M.S.Campbell, F.H.Hsu, Singular Extensions:Adding Selectivity to Brute-Force Searching, Artificial Intelligence, vol. 43, 99-109, 1990.
12) P.Auer, N.Cesa-Bianchi, P.Fischer, Finite-time analysis of the multiarmed bandit problem, Machine Learning, vol. 47, pp. 235-256, 2002.
13) 野口陽来，モンテカルロ法を用いたコンピュータ囲碁の性能向上に関する研究，修士論文，北陸先端科学技術大学院大学，2009.