修 士 論 文

有望さに基づくUCTの選択的探索手法とMonteCarlo囲碁への応用

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

矢島 享幸

2011年3月

修 士 論 文

有望さに基づくUCTの選択的探索手法とMonteCarlo囲碁への応用

指導教員　飯田　弘之 教授

審査委員主査　飯田　弘之 教授
審査委員　池田　心　准教授
審査委員　鶴岡　慶雅 准教授

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

s0810063 矢島 享幸

提出年月: 2011 年 2 月

概 要

オセロやチェスといったゲームでは既にコンピュータプレイヤが人間のトッププロに勝利している．しかし，これらのゲームで成功した評価関数とゲーム木探索を用いて最善手を判別する従来型のアプローチは，囲碁では充分な性能を発揮することができなかった．これは囲碁の探索空間がオセロやチェスに比べて非常に大きいことだけではなく，知識表現が難しく高精度な評価関数を設計することが困難であることが原因として挙げられる．

しかし 2006 年にモンテカルロ法を用いた Crazy Stone が登場したことで，以前よりも容易に強力なコンピュータ囲碁プレイヤを製作することが可能となった．Crazy Stone が用いた手法は局面の評価に評価関数ではなくランダムシミュレーション（プレイアウト）を用い，これと木探索を組み合わせることで最善手を判別するアルゴリズムである．

以来囲碁のコンピュータプレイヤではモンテカルロ (MC) 木探索法が広く使われ，また研究されている．現在では，充分な時間を与えれば minmax の結果に一致することが保証されている UCT アルゴリズム [3] が主に使われている．UCT ではあるノードから最も期待値の高いノードを辿って末端ノードまで探索木を下りる．末端ノードへの到達回数がある閾値を超えると末端ノードを展開する．最後に，末端ノードからプレイアウトを行う

UCT の研究では木探索，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．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なされてきた．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．そこで本研究では，展開条件に焦点を当て木探索の効率化を図る．

木探索におけるノード展開の方法として，訪問回数がある一定の閾値を超えた場合に展開するのが一般的である．本研究では，まず閾値とコンピュータプレイヤの強さの関係について調べた．その結果，大きすぎる閾値では深い探索ができず，小さすぎる閾値ではノードが増えすぎてメモリの制限に引っかかるおそれがあるので中程度が良いことがわかった．よって，次に必要なところだけ早く展開することで UCT の探索効率化を図った．本論文では探索効率化法として，評価値の突出度と勝率の突出度，訪問回数の推定を用いた展開手法の 3 つを提案する．さらに，3 つの提案手法それぞれについて提案した手法が有効であるか検証を行った．

目 次

図 目 次

表 目 次

第1章　はじめに

1.1　背景と目的

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ログラムは飛躍的な進歩を遂げた．以降，囲碁プログラム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．なお，現在ではモンテカルロ木探索の中でも十分な時間を与えれば minmax の結果に一致するという意味で探索の正しさが保障されている UCT が主に使われている [3].

UCT では，現時点で得られている探索木のルートノードからある末端ノードまで探索を行い，そこからプレイアウトを行う．末端ノードの訪問回数が閾値を超えるとノードを展開して子ノードを探索木に加え，新たに探索木に加えられた子ノードの 1 つからプレイアウトを行う．一般的にプレイアウトとは，ある確率に従って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経由した全ノードに繰り上げられ，再び探索が行われる．この作業を繰り返して探索木を成長させていく．

UCT アルゴリズムにお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，現在世界最強クラスの囲碁プログラムである MoGo では，展開の閾値に 1 を用いて木探索で末端ノードまで下る毎に新しいノードを展開する手法を用いている．また，富田 [4] らは 10 という値を用いていた．しかし，この展開に用いる閾値がコンピュータ囲碁プログラムの強さにどの様な影響を及ぼすのかはよく分かっていない. 本研究ではまず閾値がコンピュータ囲碁プログラムの強さに及ぼす結果を調べた．その結果, 大きすぎる閾値では深い探索ができず，小さすぎる閾値ではノードが増えすぎてメモリの制限が発生する恐れがあるので中程度が良いことが判った.

よって，次に必要なところだけ早く展開することで UCT の探索効率化を図った．展開に用いる閾値が中程度の場合，当然の手が容易に判明するような局面でノードの展開が直ぐに行われない．そこで探索過程でいずれ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，より効率的なアルゴリズムを実現できると考える．本稿ではこのような考えに基づき，UCT 実行過程いずれ展開されるノードを予測し早めに展開する手法を 3 種類提案し，それぞれについて検証を行う.

また，有望さを用いた効率化手法として既に Proguressive Pruning[20] をはじめとした手法が提案されている．3 章では，それら既存の有望さを用いた UCT 効率化手法の紹介を行う．4 章では本研究で着目した展開条件について検証を行う．5 章では提案する手法の解説を行う．6 章では本論文で提案する手法の実験と検証を行う．7 章で結論，今後の課題た展望について述べる．

1.2 論文構成

本稿の論文構成は以下のようになる．

2 章 コンピュータ囲碁の発展と課題

モンテカルロ木探索法の元となった手法や N 腕バンディット問題についても触れた上で，モンテカルロ法を用いたコンピュータ囲碁プログラムのアルゴリズムを解説する．また，現在提案されている UCT をはじめとしたモンテカルロ木探索の問題点についても述べる．

3 章 モンテカルロ木探索の関連研究

前章で述べた UCT をはじめとしたモンテカルロ木探索の問題を解消するために現在提案されている，モンテカルロ囲碁の改良手法とその特徴や問題点を説明する．

4 章 展開条件の検証

展開の条件とコンピュータ囲碁プログラムの強さの関係について実験と検証をする．このとき，合法手数展開手法についても実験と検証をする．

5 章 探索効率化手法の提案

前章を踏まえた上で，モンテカルロ囲碁の探索効率化を図る手法の提案する．提案する探索効率化は “局面の推移確率”，“勝率の突出度”，“訪問回数の推定” という 3 つのアプローチに基づいている．

6 章 提案手法の実験と考察

6 章で提案した手法の有効性について検証する．

7 章 おわりに

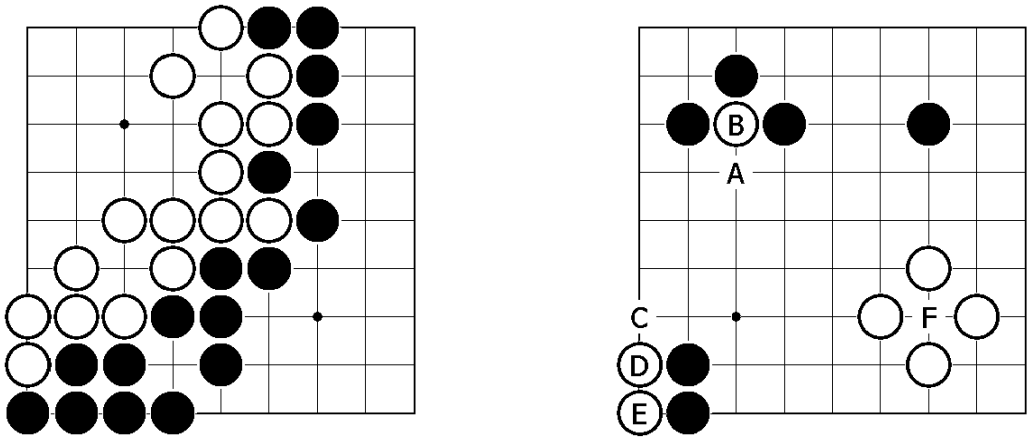
提案手法の実験結果・考察を踏まえて，今後の課題や展望について述べる．

第2章　コンピュータ囲碁の発展と課題

本章では囲碁の簡単な解説と，囲碁をコンピュータに打たせる際に何がどうして難しいのかを説明する．次に，現在コンピュータ囲碁の分野で主流となっているモンテカルロ木探索法と UCT アルゴリズムについて解説する．

2.1　囲碁とは

囲碁とはプレイヤ 2 人と 9 路盤（9x9），13 路盤（13x13），19 路盤（19x19）のいずれかの盤，黒/白の石を用いて行う．2 人のプレイヤは交互に黒/白の石を盤面に置いて行き，自分の石で囲んだ領域の広さを争う．つまり，このゲームの目的は自分の色の石によって盤面のより広い領域を確保する（囲う）ことである．図 2.1 左の例では黒が白よりも 7 目多く値を囲っている．なお，一度置かれた石は相手の石に全周を取り囲まれない限り取り除いたり移動することはできない．石および石の一団はその周囲の交点全てに相手の石を置かれると取られる．図 2.1 右の例では，A の場所に黒が石を置くと，B の石が取られてしまう．同様に黒が C の場所に黒が石を置くと，D と E の石が取られてしまう．また，自分の石を置くとその石が取られる状態になる場所は石を置くことが出来ない．図 2.1 右の例では，黒は F に石を置くことができない．



終局の例　　石を取る例（A，C），石を置けない例（F）

図 2.1: 囲碁のルール

2.2 囲碁の困難性

オセロやチェスといったゲームでは既にコンピュータ囲碁プログラムが人間のトッププロに勝利している．また，将棋についても既に女流プロに勝利するほど強力なコンピュータ囲碁プログラムが生み出されている [5]．しかし，これらのゲームで成功したゲーム木探索と評価関数を用いて最善手を判別する従来型のアプローチは，囲碁では充分な性能を発揮することができなかった．

この原因は探索空間が膨大であることと高精度な評価関数の設計が困難であることの 2 つが考えられる．本節では，コンピュータ囲碁の発展を妨げているこれら 2 つの要因について説明する．

2.2.1 探索空間の大きさ

囲碁の困難さとして判りやすいものとして，探索空間の広さが挙げられる．つまり，囲碁は他のゲームに比べて非常に合法手が多く，終局までの手数が長いのである．表 2.1 に囲碁とその他の代表的ななゲームの探索空間の広さを示す．

表 2.1: 探索空間の大きさ [2]

ゲーム	探索空間の広さ
オセロ	10^{58}
囲碁（9 路）	10^{70}
チェス	10^{123}
囲碁（13 路）	10^{180}
将棋	10^{220}
囲碁（19 路）	10^{360}

9 路盤における囲碁の探索空間はチェスよりも小さい．にもかかわらず 2005 年以前には 9 路盤でも 19 路盤でも変わらず強力なコンピュータ囲碁プログラムを実現できずにいた．このことは，探索空間の広さ以外にも囲碁の困難さの要素があることを示唆している．探索空間の広さ以外の囲碁の困難さの要因については次の節で述べる．

2.2.2 評価関数の設計とコマの価値

論理的には，2 人零和有限確定完全情報ゲームは minmax 探索により最善手を求めることができ，三目並べの様な単純なゲームならば勝敗が決まる終局までの探索木を全て探索して最善手を求めることが可能である．しかし，より複雑なゲームでは探索範囲が大き

すぎて終局までの全ての探索木を探索することができない．そのため，実際には探索を途中で打ち切りその時点でわかる範囲で最も良さそうな手を選択する．この時，探索を打ち切った局面のスコアを近似的に数値化する評価関数を用いる．

チェスや将棋などのゲームではコマの価値が非常に重要な要素であるため，コマの価値を中心として利用することで精度の高い評価関数を作成することができる．また，評価関数は充分に高速であることが求められるが，コマの価値を用いた評価関数は構造が簡単なため高速に算出することができる．

しかし，チェスなどのゲームと異なり囲碁では各石に価値大差が無い．占領した領域の広さ（地）を競うゲームであるからには地の大きさを評価関数に用いることが望ましいが，地が確定するのは終局間際であるためそれ以前に地の広さを評価関数として用いられるほど高速に求めるのは非常に難しい．

その様な中，ゲーム木探索＋評価関数に代わる新たなアプローチとして現れたのがモンテカルロ囲碁 [1] である．モンテカルロ囲碁では設計の困難な静的局面評価関数の代わりに，ランダムゲームを用いたシミュレーションを行うことで局面評価を行う．モンテカルロ木探索が登場して 2 年余りで 9 路盤でプロ棋士を破るほどの強さを獲得した．モンテカルロ木探索の出現は従来のゲーム木探索＋評価関数というアプローチの他に新たな選択肢を与えたという意味や，多くのゲームでルールや知識とは独立的に実装できるという点でも非常に意義があると言える．

2.3 モンテカルロ法

モンテカルロ法は，乱数を用いてシミュレーションや数値計算を行なう手法の総称である．モンテカルロ法で最も有名な例は，円の範囲内に収まる点の個数から円周率を求めるアルゴリズムである．

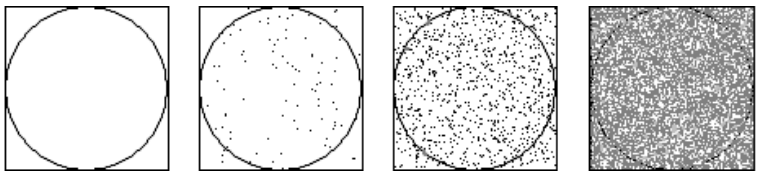


図 2.2: モンテカルロ法を使った円周率の計算法

図 2.2 のように決められた正方形の中に適当に点を打つとき，その点が正方形の一辺を直径とする円の円内にある確率は

$$\text{円内にあった点の数} \div \text{打った全数} = \text{円の面積} \div \text{正方形の面積} = \frac{\pi}{4} \tag{2.1}$$

であるため，この比から円周率を算出することができる．理論的には点を打つ数を増やせば増やすほど円周率に近づく．

この手法を囲碁に応用したものがモンテカルロ囲碁（以降，原始モンテカルロ囲碁）である．モンテカルロ法を囲碁の局面評価に応用しようという研究は1993年にB.Brüegmann[6]らによって始められた．このとき提案された原始モンテカルロ囲碁は乱数を用いて深さ1のノードから終局までプレイした結果によって局面を評価し着手を選択するというものであった．この ”乱数を用いて適当に終局までプレイ ”を用いて勝敗を得る手法はプレイアウトと呼ばれる．囲碁というゲームでは手番が進むにつれて合法手が制限されるため，ランダムな着手でも必ず終局に至ることが可能である．このプレイアウトを図2.3の様に複数回（各合法手に対して同じ回数）行うことで，評価したい合法手の勝率を求め，評価値とする．これは人間から見ると非常に不正確に見えるが，それでも回数を積み重ねることによってそれなりに良い評価値を得ることができる．

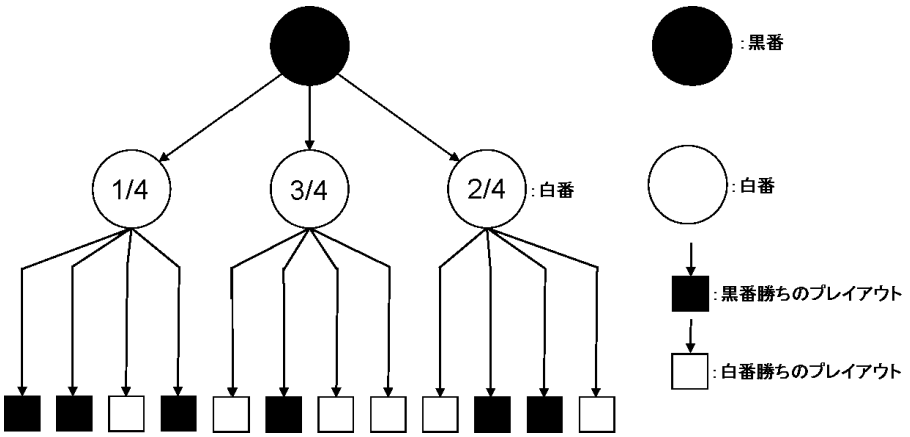


図 2.3: 原始モンテカルロを用いた着手の選択

図2.3の場合では白丸の中の数字がプレイアウトの勝利数を意味し，白番の評価値になる．この場合は中央の手（ノード）が勝利数が最も高いため，選択されることになる．しかし，囲碁にモンテカルロ法を応用するだけでは強力なコンピュータ囲碁プログラムは実現できない．これは，囲碁のプレイアウトは円周率とは異なり，手がランダムに選択されるという “仮定” に基づいているからである．実際に囲碁の手の選択がランダムに行われるということは無い．そのため，相手のミスを期待する手を高く評価してしまう等の問題が起こり得るためと考えられる．

2.4 K腕バンディット

K腕バンディットとは，複数のスロットマシン（もしくは，スロットマシンに似たもの）を対象とした単純な機械学習問題である．複数のスロットマシンはそれぞれ固有のある分布から定まる「報酬」を提供する。その「報酬」の総和を最大化する戦略を考えるのがK腕バンディット問題である [7]．プレイヤはスロットマシン固有の分布に関する知識を持た

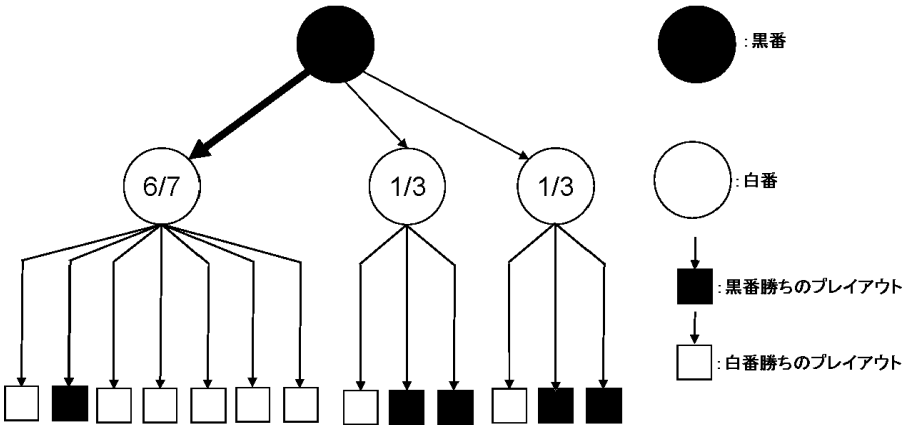


図 2.4: UCB を用いた着手の選択

ず，試行を繰り返す必要がある．任意の対象に対して試行（プレイアウト）を行い,「報酬」に関する固有の分布を推定する行ためはモンテカルロ法によく似ている．

この問題で重要なことは得られた知識に基づいて報酬を多く得られそうなスロットマシンを選ぶことと，知識を増やすためにより良い報酬を得られるかもしれないスロットマシンを調べることのバランスを取ることである．強化学習では「収穫と探検のジレンマ〈exploitation-exploration dilemma〉」として知られている．つまり，バンディット問題における「収穫」はこれまでに集めた知識から現時点で最善の腕を選ぶことであり，他方「探検」は他の腕を選んでその腕に関する知識を増やすことに対応している．

K 腕バンディット問題に対する戦略は既に提案されていたが [8]，計算量，消費メモリともに大きくなるため現実の問題に用いる場合には適していなかった．それに対し，2002 年に Auer らによって提案された UCB1 という戦略 [9] は計算量を小さく保ちつつ高い報酬を期待することができるものである．この UCB1 では期待値＋バイアスで表現される UCB 値を計算し，全マシン（腕）で最も UCB 値の高いマシンを選択する．UCB 値は以下の式 2.2 で与えられる．

$$\overline{X_i} + C\sqrt{\frac{2\log N}{n_i}} \tag{2.2}$$

ここで i はスロットマシン（バンディットの腕）の番号を示す．N は今までの総プレイ回数，n_i はスロットマシン i が選択された回数を意味する．また，$\overline{X_i}$ はスロットマシン i のその時点までの報酬の期待値である．C は探索の傾向を定める定数で実験的に定める．

UCB は以下の考え方にもと基づいている．

- 基本的には，期待値の高い所にリソースを集中
- ただし，試行回数の少ないマシンは運悪く真の期待値よりも小さい期待値になっている可能性があるので，それを考慮し優遇する

各合法手について UCB1 値を計算し代際の値を返す合法手を選択すると，図 2.4 に示したように有望な手に対してプレイアウトが偏るようになる．しかし深さ 1 でしか探索を行わないため，2.3 章で述べた「相手のミスを期待する手を高く評価してしまう」という問題は相変わらず発生する．

2.5 モンテカルロ木探索法

モンテカルロ法だけで囲碁の最善手を選ぶことは難しい．これを解決するために原始モンテカルロ囲碁と木探索法を組み合わせた手法がモンテカルロ木探索である．現在はモンテカルロ木探索に更に UCB1 を応用した UCT が主流となっている．図 2.5 に，UCT のアルゴリズムを図示する．

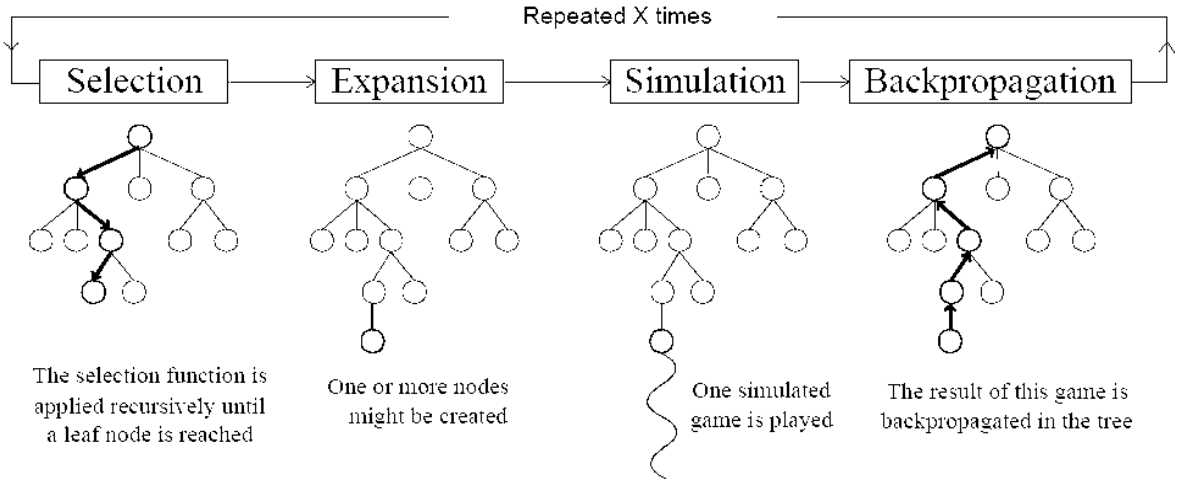


図 2.5: UCT アルゴリズムのイメージ [10]

UCT の考え方は，各ノード（囲碁の局面）を独立した K 腕バンディット問題と見なし，その子ノードを独立した腕（マシン）と見なすことである．ルートノードから順に UCB1 値の高いノードを辿っていく（図 2.5．Selection）．ここで，末端ノードの訪問回数が閾値を超えると子ノードが展開される（図 2.5．Expansion）．探索木の末端ノードまで到達したらそこからプレイアウトを行う（図 2.5．Simulation）．プレイアウトの結果はルートノードから辿って来た全ノードに共有される（図 2.5．Backpropagation）．これを繰り

返すことで，探索木中にある全ノードの勝率や期待値を得ることができる．各ノードでの子ノードの選択はUCB値またはそれに準じる値を持って行われる．現在の主流は，UCBを改良したUCB1-Tuned[11]の値を計算してノードを選択する．UCB1-Tunedは以下の式2.3で与えられる．

$$\overline{X_i} + C\sqrt{\frac{\ln N}{n_i}\min\left\{\frac{1}{4},\ x_i - x_i^2 + \sqrt{\frac{2\ln N}{n_i}}\right\}} \tag{2.3}$$

この条件は様々であるが，普通は閾値に固定値を用いて1回目の訪問で展開する場合が多い．これによって探索木が成長し，より深く正確な探索が行えるようになる．

原始モンテカルロ囲碁では深さ1の手しか読まないため，探索時間をいくら増やしても最善手を選択することができない．それに対しUCTでは木が成長していくために探索時間を充分にかけさえすれば，訪問回数の偏りがminmax探索と同様の効果を導くことで最善手を得ることができる．図2.6にUCTの擬似コードを示す

2.6 モンテカルロ木探索の課題

UCTを初めとするモンテカルロ木探索法は静的な局面評価関数が必ずしも必要でないという利点があるが，大数の法則に基づくために局面の評価精度はプレイアウト回数に依存する．一般的に対局で1手選ぶ時間は数秒から数分であり，探索木中の全ノードに対して高い精度を得られるまでプレイアウトを行うことは難しい．UCTとは元々有望なノードに対して探索が偏るように作られているが，それでも無謀な（不必要）な手に対する探索を打ち切る機能を追加することには意味があると思われる．

また，原始モンテカルロ囲碁ではプレイアウトを増やしても棋力が頭打ちになる[12]ことからも判るように，より探索木を成長させ深いノードに対してプレイアウトを行った方が正確な評価を行える．しかし，探索時間は有限であるため全合法手から延びる探索木を正確な評価を得られるようになるまで成長させることも難しい．さらに，正確な評価を行いたい有望な手も悪手と同一の条件で探索木を成長させるとするなら，有望な手から延びる探索木が正確な評価を得られるようになる前に探索が終了してしまうことも起こりうる．こうなった場合には有望な手に対して精度の良い評価を行うことが出来ない．しかしプログラムを実行するコンピュータのリソースは有限であるため，無制限に探索木を成長させれば良い訳ではない．また，プレイアウト部分に比べて木探索部分はプログラムの実行速度が遅い．そのため，無制限な探索木の成長は全探索速度の低下にも繋がる．つまり有望な手に対して的確に探索木を成長させる必要があるが，これもUCT単独では実現できない．

そこで，無謀な手への探索を打ち切る手法や，有望な手を展開条件が満たされる前に率先して展開する手法が求められる．

```
1.  function playOneSequence(rootNode)                    //図 2.5 を繰り返す
2.    node[0] := rootNode; i = 0;
3.    while(node[i] is not leaf) do
4.      node[i+1] := descendByUCB1(node[i]);              //図 2.5.Selection
5.      i := i + 1;
6.    end while;
7.    if node[i].visit ≥ TH                             //図 2.5.Expansion
8.      node[i+1] := descendByUCB1(node[i]);
9.      i := i + 1;
10.   Playout(node[i+1])                                //図 2.5.Simulation
11.   updateValue(node, - node[i].value);               //図 2.5.Backpropagation
12. end function;

13. function descendByUCB1(node)
14.   for i := 0 to node.childNode.size() -1 do
15.     if node.childNode[i].nb = 0
16.       do v[i] := FPU;
17.     else v[i] := CalculateUCB
18.   index := argmax(v[j]);
19.   return node.childNode[index];
20. end function;

21. function updateValue(node, value)
22.   for i := node.size()-2 to 0 do
23.     node[i].value := node[i].value + value;
24.     node[i].nb := node[i].nb + 1;
25.     value := 1 - value;
26.   end for;
27. end function;
```

図 2.6: UCT アルゴリズムの擬似コード [16]

第3章　モンテカルロ木探索の関連研究

UCT の提案以降，囲碁プログラムは飛躍的な性能向上を遂げた．この UCT 法における効率化の研究はプレイアウト部と木探索部，それぞれについて効率化と質の向上についての研究が行われている．

プレイアウトの質の改良では，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 [19] や Bonanza メソッドを用いた学習手法 [17] が提案されている．また，プレイアウトの効率化については RAVE[24] などが挙げられる．木探索部分の質の向上は QuickUCT[13] や評価関数を用いた枝刈り手法の一種である Progressive Widening[20][21] 等に代表される．

木探索部分の効率化では，2 章で述べたモンテカルロ木探索の問題点の解決手法として枝刈りやノードの展開に付いての研究が行われている．枝刈りとは，なんらかの意味で明らかに有望でない手への探索を省略する手法を指す．我々が主に注目しているのは UCT におけるノード展開の効率化ではあるが，それには有望な手とそうでない手を判断する必要があるのは枝刈りと変わらない．そこで，“有望な手と無謀な手の判別を行う”ための関連研究として枝刈りを本章で説明する．UCT における枝刈りとして主なものは Progressive Pruning[20] や但馬ら [22]，北川ら [23] の手法が挙げられる．その上で，現在提案されているノード展開に関する関連研究についても説明する．

3.1　選択的枝刈り

枝刈りの研究はモンテカルロ木探索が提案される以前から行われてきた，不要と判断した探索木の一部分への探索を省略することで全体の効率を向上する手法の総称である．minmax 法において最も有名な枝刈り手法である $\alpha\beta$ 法では枝刈りをしない場合と全く同じ結果を保証しながら，計算量を省略することができた．

モンテカルロ木探索は確率的な挙動をするため，$\alpha\beta$ 法の場合と異なり同じ結果を保証することができない．しかし，それぞれの合法手の勝率の確率分布が正規分布に従うと仮定して，勝率と得られた報酬の標準偏差から勝率の取りうる範囲を推定，そこから高確率で無駄な合法手について枝刈りを行う手法が提案されている．

本節では，現在提案されている UCT における枝刈りを紹介する．

3.1.1 Progressive Pruning

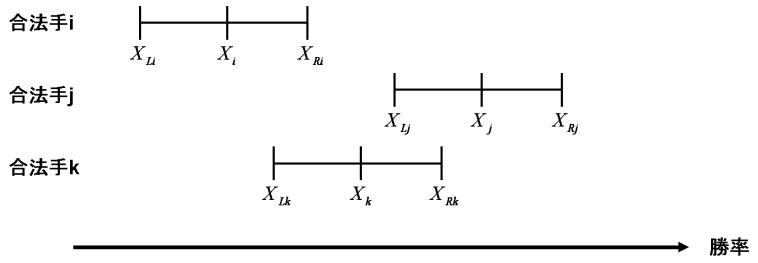
B. Bouzy ら [20] は UCT の基本となる多腕バンディット問題を効率よく解くために Progressive Pruning（前向き枝刈り）を提案した．この手法は多腕バンディット問題について提案された手法であるが，モンテカルロ木探索へ応用が可能である．Progressive Pruning では合法手 i の勝率 X_i の信頼上限 X_{Ri}・下限 X_{Li} を以下の式 (3.1)・(3.2) を用いて計算する．

$$X_{Li} = X_i - C \cdot \frac{\sigma_i}{\sqrt{n_i}} \tag{3.1}$$

$$X_{Ri} = X_i + C \cdot \frac{\sigma_i}{\sqrt{n_i}} \tag{3.2}$$

ここで σ_i は標準偏差，n_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，この式は勝ち数と負け数がともに 10 以上になった場合にのみ適用する．が成立する場合である．それ以外の場合は式 3.2，3.1 の算出は行わない．

勝率の信頼上限・下限を計算してある合法手 i, j について $X_{Li} > X_{Rj}$ 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．



i は j に劣るので省略，k は i に劣るかもしれないが省略しない

図 3.1: ProgressivePruning のイメージ

図 3.1 の場合はノード i への探索を省略する．ノード k への探索は省略しない．この手法では信頼区間 $[X_{Li}, X_{Ri}]$ を狭めるために多くのプレイアウトを繰り返す必要があるため，プレイアウト回数が少ない場合は効果を得られない．また，探索木を深く辿っていくにつれてプレイアウト回数の割り振りが減少するため，枝刈りの効果が得られに難くなる．

3.1.2 但馬らの手法

但馬らが提案した手法は，実際のコンピュータ囲碁プログラムでは一定のシミュレーション回数で探索を打ち切る必要があるという考えに基づいている．この手法は，UCB1,

UCT 双方に対して提案された手法である．UCT においては，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 (3.3) で求められる．x^* とは，ルートノードの子ノード中で最大の勝率を示す．また勝率 x^* を持つノードを合法手 i^* とする．

$$F(X) \geq \frac{8 \log N'}{(x^* - X)^2} + 1 + \frac{\pi^2}{3} \tag{3.3}$$

この式 3.3 を用いてノード i の残りの訪問回数を推定する．その結果，勝ち続けたとしても探索終了までに最大の勝率 x^* を逆転出来ないノードについて探索を打ち切る．

この手法は探索終了時に最良と判断される可能性の無い手について枝刈りを打ち切る．そのため，枝刈りまでの猶予が大きく効果が得られるのは探索の終盤になってしまう．

3.1.3 北川らの手法

実際のコンピュータ囲碁プログラムでは探索を無限に続けることは不可能であり，一定のシミュレーション回数で探索を打ち切る必要がある．このとき各合法手に分配される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のが北川らの手法 [22] である．

北川らの手法では，合法手 i の残りプレイアウトで到達し得る上限の勝率（到達上限確率 P_{Ri}）及び下限の確率（到達下限確率 P_{Li}）を以下の式で求める．

$$P_{Ri} = \frac{X_i + e_i X_{Ri}}{n_i + e_i} \tag{3.4}$$

$$P_{Li} = \frac{X_i + e_i X_{Li}}{n_i + e_i} \tag{3.5}$$

ここで e_i は残りプレイアウト中でのノード i の選択回数の推定値で，X_{Ri}，X_{Li} はノード i の勝率の信頼上限 3.2，信頼下限 3.1 を用いる．

式 (3.4)・(3.5) を用いることで，残り探索回数に因ってどの範囲でノード i の勝率が変化するかを ProgressivePruning よりも正確に推定することが出来る．北川らの提案した枝刈り手法のイメージを図 3.2 に示す．図 3.2 の場合では，Progressive Pruning では合法手 k は枝刈り対象にはならなかった．しかし北川らの手法を用いることで，合法手 k も枝刈り対象とすることができる．

北川らの検証によると，この手法を従来のモンテカルロ木探索法に組み込んだ場合，従来手法に対して最大 84 %の勝率を得られることが判明している．しかし，逆に枝刈りをしてはいけないノードについても枝刈りをしてしまい，合法手を見逃してしまうことが起こり得る．

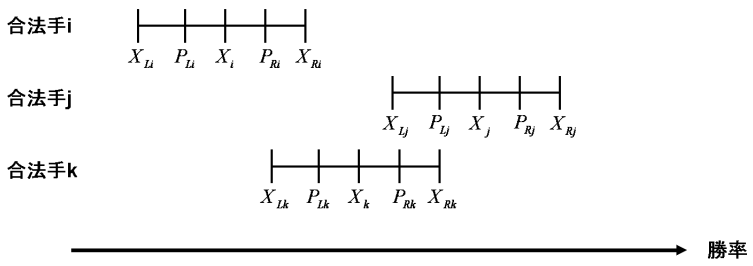


図 3.2: 北川らの手法のイメージ

3.2 枝刈りの課題

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．枝刈りの問題点として共通して挙げられるのは，枝刈りの効果を得るためには一定以上の探索を行う必要がある点である．北川の手法では効果を発揮するまでに最低 2000 回の探索が必要であった．特に，探索の序盤で明らかに悪手があることが判った場合では，判明した悪手への探索を終了させたい．しかし枝刈り手法では悪手への探索を打ち切って，それ以外の候補手に探索を集中させるには一定以上の探索を必要とする．結果，その悪手を枝刈りして良手に探索を集中させるには時間がかかる．場合によっては，探索終了間際にならないと枝刈りの効果が得られないことが起こりうる．

3.3 ノード展開・探索延長

ノード展開・探索延長に関する研究も枝刈りと並んでモンテカルロ木探索が提案される以前から盛んに研究されてきた．探索中に有望と思われる合法手をその時点の限界の深さよりも深くまで探索することで有望手の勝率推定の精度を上げることを狙った手法である．探索延長の手法には有望と思われる合法手を判別するためにゲームの知識に依存した手法が多く見られる．本節では，モンテカルロ木探索法に対して提案された手法について紹介する．

3.3.1 モンテカルロ囲碁におけるシーケンシャルパターンマッチの試み

囲碁では,「ハネツギ」（図 3.3 参照）の様な「こう打ったら多くの場合こう打つ」というようなある程度決まった打ち方が存在する．図 3.3 の場合 1 と打って 2 と打たれた後に，3 以外の場所に打つことで得をすることはなく，通常この順番通りに打たれる．眞鍋らはこの様な一定範囲内での連続した石の運びを 1 つのパターンとして取り扱う．また，このパターンのことを「シーケンシャルパターン」（以下 SP）と呼ぶ.

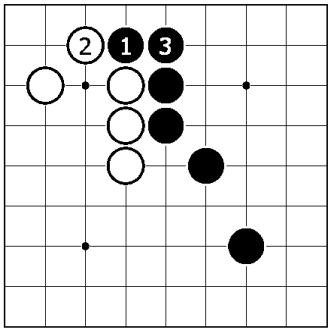


図 3.3: 囲碁におけるシーケンシャルパターン，ハネツギの例

眞鍋らが提案する手法 [14] では，本来 UCT は最大でも 1 手ずつしか探索木を成長することができない所を SP を用いることで，SP の 1 手目ににマッチした場合にパターンの長さ分だけ探索木を成長させる．これによって UCB 値を計算することなく早めに探索木を成長させることができる．図 3.4 にシーケンシャルパターンを取り入れた探索木の成長の例を示す．A，B，C の枝はパターンにマッチしたため UCT とは関係無しに展開されたノードである．

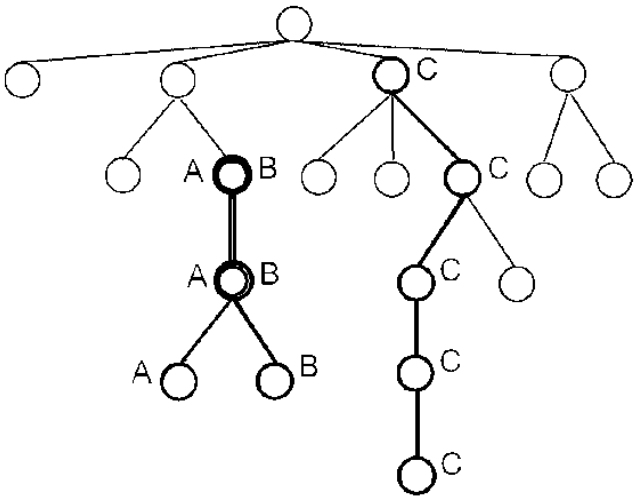


図 3.4: シーケンシャルパターンを取り入れた探索木

眞鍋らはシーケンシャルパターンとして，プロ棋譜 9535 局からマンハッタン距離 2 以内での一定回数以上出現している連続した石の運びを抽出した．表 3.1 に文献 [14] によるシーケンシャルパターンマッチ＋モンテカルロ木探索とモンテカルロ木探索の対戦結果を載せる．この実験から，シーケンシャルパターンを用いて探索木を成長させる手法が有効

であるという明確な結果は得られていない．

モンテカルロ木探索は勝利が確定している安全手を打ち，負けが確定している局面では人間では考えないような悪手を打つ．シーケンシャルパターンを用いた探索木の成長が有効ではなかったのは，人間にとって打ち易い手がモンテカルロ木探索を用いたコンピュータ囲碁プログラムにとって打ち易いとは限らないからだと考えられる．

表 3.1: シーケンシャルパターンマッチを用いた実験結果 (勝率：95 %信頼区間)

対戦数	勝利数	勝率	勝率下限	勝率上限
300	157	0.523	0.465	0.581

3.4 ノード展開・探索延長の課題

前章で枝刈りとは別の探索効率化手法である，展開されるノードを予測・展開する手法を解説した．しかし，この手法は評価関数を用いるためゲームの知識に依存している．よって前提として知識を蓄積する必要があるため，新しく知識の集積のないゲームでは適用が難しい．更に，モンテカルロ木探索におけるこの分野の研究は前例が少なく，また効果は明確な優位性があるとは言いがたい点でも新しい手法が必要である．

また，UCT のノード展開条件については非常に関連研究が少ない．多くは，1 探索毎にノードを展開する手法 [1] [15] を用いているようである．例外的に 10 回訪問した場合にノードを展開する [4] 手法を用いる場合もある．しかし，これらの文献には 1 回ないし 10 回にした理論的・実験的理由が明確には記述されていなかった．そこで，次章でまず UCT の展開条件について検証を行う．

第4章　展開条件の検証

3章でUCTアルゴリズムを効率化する方向性として，枝刈りと探索木の成長の促進について紹介した．しかし，枝刈りは様々な手法が提案されているが，効果を発揮するまでに一定量のプレイアウトが必要不可欠であるという問題点を解消できずにいる．また，ノードの展開に関しては展開に用いるパラメータと強さについて知識の蓄積が充分とは言えず，わずかにある先行研究でもその効果は明確な優位性があるとは言いがたい．そこで本研究では効果的な手法が余り提案されていない，UCTにおけるノード展開・探索延長について着目する．

4.1　展開条件と強さの検証

まずノードを展開する際に用いる最低訪問回数を設定し，展開条件とする．コンピュータ囲碁プログラムについて，このパラメータの影響を調べる．コンピュータ囲碁プログラムは我々が開発を行っているNomitanを使用した．Nomitanの詳細については付録Aに掲載する．また，比較対象としてフリーのコンピュータ囲碁プログラムである“GNU Go”を利用した．

4.1.1　実験環境

- Process：1（ただし，Nomitanは本研究の提案も含めマルチスレッドに対応している）
- OS：Linux
- CPU：Intel(R) Xeon(R) CPU L5520 @ 2.27GHz
- メモリ：24.7GByte
- 開発言語：C++

4.1.2　実験条件

展開条件と強さの関係を検証する際の実験条件を以下に記す．なお，ノード展開を行う最低訪問回数が最小の場合（閾値=１）にすると，1探索につき1つのノードが作られる．Nomitanの探索回数が秒間約1000回であるため，最大記憶ノード数は$1000 * 5 * 60 = 30$万を超える値に設定した．

- コミ：6.5
- 手番：先後入れ替え
- 囲碁盤：9 路盤
- ルール：中国ルール
- 試合数：200 戦
- 持ち時間：1 局 5 分
- 対戦相手：GNU Go
- 最大記憶ノード数：50 万

4.1.3 実験結果・考察

図 4.1 に Nomitan がノードを展開する際の訪問回数条件（Constant）を変えた場合に，Gnugo に対してどの位の勝率を得られるかを実験した結果を示す．横軸が Nomitan がノードを展開する際の閾値で，縦軸が Nomitan の勝率を表す．

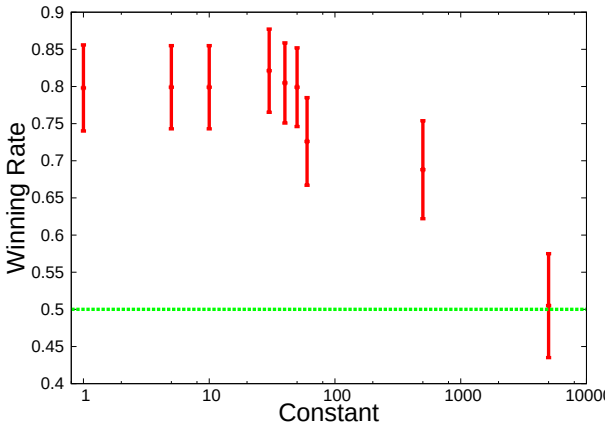


図 4.1: 展開の閾値を変えた場合の勝率

図 4.1 から判るように，Constant が 50 までは Constant が 1 の場合とあまり勝率が変わらずに．80 %という非常に高い勝率を示している．しかし，Constant が非常に大きい場合は Nomitan の勝率は 50 %まで低下する．これは，Constant が大きくなるにつれて原始モンテカルロに近づき，深い探索が出来なくなっているためであろう．

しかし，一定値までは Constant が増加しても勝率にあまり影響が出ていない．これは，プレイアウトの精度の向上によって一定以上の深さは無理に展開しなくても，充分なプレイアウトを行えれば信頼できる評価を行えるためであると推測している．以降，この Constant が強さに影響を与えない限界の値を境界訪問回数と呼ぶ．

4.2　最大記憶ノード数と強さの関係

4.1 章で 50 までノード展開を行う最低訪問回数を増やしても強さに影響が無いことが判明した．次に，最大記憶ノード数がコンピュータ囲碁プログラムにどのような影響を与えているのかを調べる．ノード展開を行う最低訪問回数が最小の場合（閾値= 1 ）と境界訪問回数を用いた場合を対象に実験を行う．また，Nomitan が従来から採用していた展開を行う最低訪問回数を “局面の合法手とする手法”（以降，合法手数展開手法）についても同時に検証する．最低訪問回数が小さい場合には展開が多くなるため，最大記憶ノード数が小さい場合には性能の低下が予想される．

4.2.1　実験結果・考察

実験の結果を図 4.2，4.3，4.4 に示す．実験結果から最低訪問回数が 1 の場合は，最大記憶ノード数を小さくすると 10 %台まで低下することが判明した．最低訪問回数が 1 の場合に対して，他の 2 つの条件では最大記憶ノード数を小さくしても強さに影響が見られなかった．この事から実験前に予想したとおり，展開に用いる最低訪問回数が大きくする事で強さを維持したまま最大記憶ノード数の節約が期待できることが判った．

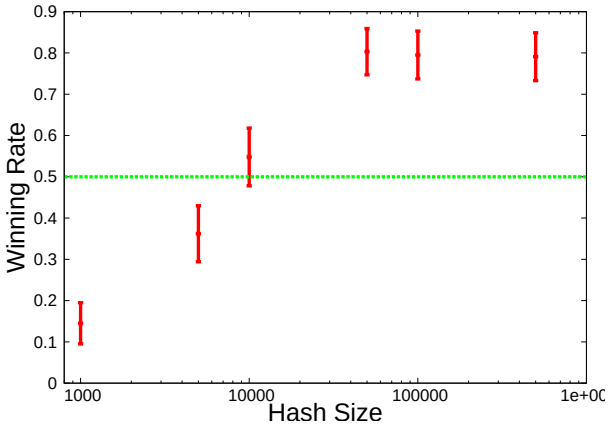


図 4.2: 最低訪問回数=1 を用いた Nomitan の勝率

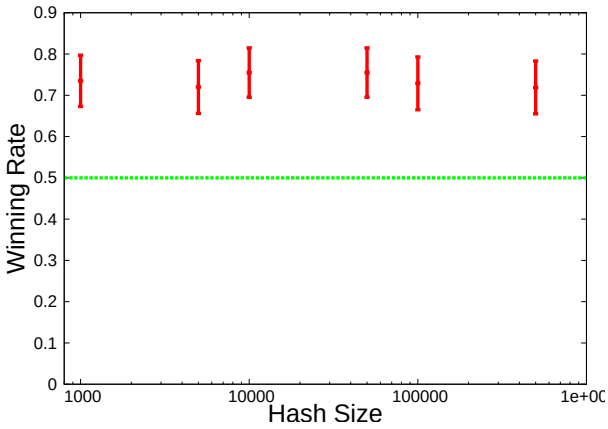


図 4.3: 最低訪問回数=50 を用いた Nomitan の勝率

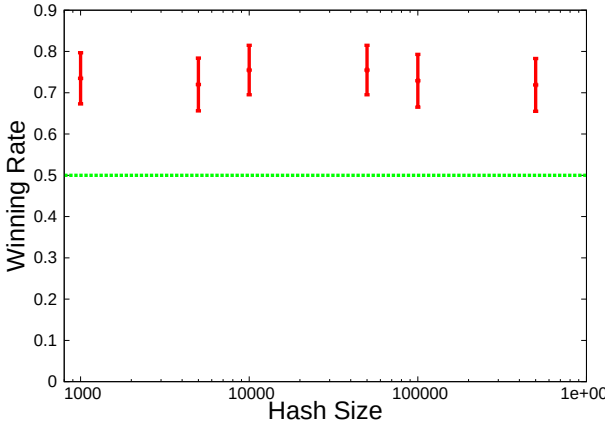


図 4.4: 合法手数展開手法を用いた Nomitan の勝率

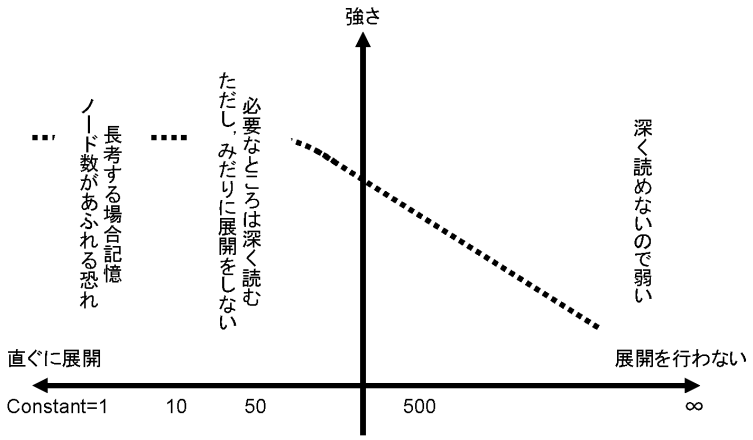


図 4.5: 展開条件と強さの関係

展開条件と強さの関係は図 4.5 のようになる．現在は Nomitan の動作が信頼できる 9 路盤で検証しているが，13，19 路盤，12〜16 スレッド，持ち時間 30 分などと探索の規模が大きくなっていくことを考えると，記憶ノード数の全展開法での限界はずっと早く訪れることが予想される．よって本研究では数十回の訪問が展開に必要となる境界閾値や合法手数展開手法を用いたノード展開法をベースとして用いることにする．

第5章　展開ノードの推定とUCTアルゴリズムの効率化法の提案

4章ではノードの展開条件について実験と検証を行った．結果として，展開に用いるパラメータが大きすぎる場合は探索が浅くなることで，性能が低下することが判明した．小さすぎる場合では記憶ノード数を超える恐れがあり，中程度が良さそうであることがわかった．マルチスレッド化や探索時間の増加から起こる探索規模の増大が見込まれるため，最大記憶ノード数の不足は回避すべき課題である．よって，本章では合法手数展開手法や展開に用いる最低訪問回数が大きい場合についての効率化手法を提案する．展開に用いるパラメータが大きい場合，小さい場合よりも展開スピードが劣るため，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アルゴリズムの効率化を実現できる．ただし，不要なノードを展開することは記憶容量の無駄であるため，展開するべきノードをいかに選択・決定するかが重要となる．

本研究では，その方法として3種類の手法を提案した．3つの提案手法は，それぞれ**評価値の突出度**，**勝率の突出度**，**訪問回数の推定**を用いている．

5.1　提案手法を用いたノード展開の流れ

本論文で提案する3つの手法は，全てノードの展開を早める手法である．それぞれ異なった判断基準を用いているが，展開に至る流れは基本的に同一である．本章ではまず，ノード展開に関する共通の枠組みを図5.1を用いて説明する．

提案手法は従来手法での展開の有無に関わらず末端ノードにおいては毎回適用される．つまり，例えば評価値の突出した手が数手続くような場合には，一回の探索で複数の展開が一気に行われることもありうる．

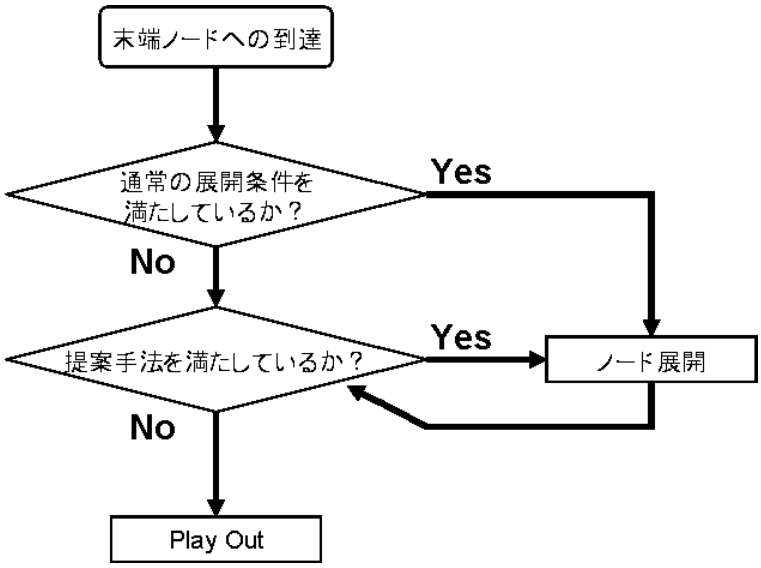


図 5.1: 提案手法を用いたノード展開の流れ

5.2 手法1：評価値の突出度を用いたUCTアルゴリズムの改良

本節では，1 番目に提案する “評価値の突出度を用いて末端ノードを展開する手法” について解説を行う．この手法は対象とするゲーム固有の知識を評価関数として用いた手法である．

UCT アルゴリズムにおけるプレイアウトは，原始モンテカルロ囲碁の頃から純粋な乱数を用いたゲームではなく，自分の目を潰さない，あと一手で取られそうな石は逃げる等の制約を与えられてきた．現在のプレイアウトでは，さらにパターン等を用いた評価関数を使って正確なシミュレーションを実現しようとしている．つまり，より有望そうな手をプレイアウトの中でも判断し，それに着手を偏らせるということを木探索部分だけではなくプレイアウト内部でも行っているのである．

本節の評価値の突出度を用いて末端ノードを展開する手法（手法 1）は，本来プレイアウト中で “有望そうな手” の判断に用いてる評価関数を木探索部分でも利用することで，十分な訪問がされていないノードに対しても展開を行おうという手法である．

5.2.1 展開に用いるパラメータの制御

提案する手法と基本となる展開条件と組み合わせて探索木を成長させていく. 局面の遷移確率（言い換えれば “着手確率”）がある値Th以上ならばつまり全着手中のその手の評価値が十分大きい割合を占める場合，それは主要な候補手であり，いずれは十分なプレイアウトが行われるであろうと判断して，即座に末端ノードを展開する. Thは実験的に定めるものとする.

5.2.2 局面の遷移確率

本稿で提案する展開法において問題となるのが, 評価値の突出度の算出である. 突出度の定義としてはプレイアウトで用いている着手選択確率と同様の計算式を採用した. 以下の式を用いて評価値の突出度を求める.

$$P(i) = \frac{\gamma(i)}{\sum_{i \in B} \gamma(i)} \tag{5.1}$$

ここでiとは任意の合法手であり, $\gamma(i)$が合法手 i に対する評価関数の値を表す. またBが合法手の集合を意味する. 合法手iが突出した手であるほど$P(i)$が 1.0 に近い大きい値を持つ. 評価関数の学習については, 事前に行っておくものとする. 使用する機械学習は, Elo レーティング [19] や勾配法を用いた手法 [17] が考えられる. 今回は Nomitan プロジェクトのメンバーである土井 [18] が研究している学習結果を利用した.

5.2.3 予想される挙動

評価関数がある程度精度よく学習できれば，アタリには逃げるといった「まっさきに考える手」「定石的な一連の手」の評価値は高くなることが期待できる. この場合，こういう手をまずノード展開することは効率よい展開や正しい勝率推定に貢献すると予想できる.

5.3　手法2：勝率の突出度を用いたノード展開手法

5.3.1　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ある場合はどの手に探索が集中するか判断できないため，十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[25] の様に 1 つだけ突出しているノードは有望であると判断して早めにノードを展開する（図 5.2 左）．勝率が突出したノードが複数ある場合には基本となる展開条件よりも展開を早めることはしない（図 5.2 右）．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．

ノードを展開する場合　　ノードを展開しない場合

図 5.2: 手法 2 を用いた展開ノードの判別

5.3.2　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ず，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．ただし，Progressive Pruning などは探索木中の各ノードの勝率の確率分布が正規分布に従う範囲でしか計算を行わないが，本手法では正規分布に従う範囲外でも区間推定を行いたい．そこで，正規分布に従う範囲内での信頼上限 X_{Ri} と信頼下限 X_{Li} の算出は，Progressive Pruning と同様に式 3.1，3.2 を用いる．また，正規分布に従う範囲外での信頼上限 X_{Ri} と信頼下限 X_{Li} の算出は F 分布を利用した以下の式 5.2，5.3 を用いた．

$$X_{Ri} = \frac{v_1 F_{1-\alpha/2}(v_1, v_2)}{v_2 + v_1 F_{1-\alpha/2}(v_1, v_2)} \tag{5.2}$$

ただし，$v_1 = 2(x_i + 1)$, $v_2 = 2(n_i - x_i)$，n_i:ノード i の試行回数，x_i:ノード i の勝利数をそれぞれ意味する．

$$X_{Li} = \frac{v_2'}{v_2' + v_1' F_{1-\alpha/2}(v_1', v_2')} \tag{5.3}$$

ただし，$v_1' = 2(n_i - x_i + 1)$, $v_2' = 2x_i$，n_i:ノード i の試行回数，x_i:ノード i の勝利数をそれぞれ意味する．

例を挙げると，最大勝率勝率を持つノード i と二番目の勝率を持つノード j の訪問回数がそれぞれ 21 回と 8 回だったとする．勝利数はそれぞれ 20 回と 3 回とする．ノード i, j は勝率の確率分布が正規分布で近似可能な条件を満たしていないため，式 3.2，式 3.1 を用いて信用できる区間の推定が出来ない．つまり，提案手法を適用できない．しかし，式 5.2 と 5.3 を用いることで信頼上限 X_{Ri} と信頼下限 X_{Li} を算出できる．α が 0.05 とした場合の式を示す.

$$\begin{aligned} X_{Li} &= \frac{v_2'}{v_2' + v_1' F_{1-0.05/2}(v_1', v_2')} \\ &= \frac{2 * 20}{2 * (21 - 20 + 1) + (2 * 20) F_{0.975}(2 * (21 - 20 + 1), 2 * 20)} \\ &= \frac{40}{4 + 40 F_{0.975}(4, 2 * 20)} \\ &= 0.86 \end{aligned} \tag{5.4}$$

$$\begin{aligned} X_{Rj} &= \frac{v_1 F_{1-0.05/2}(v_1, v_2)}{v_2 + v_1 F_{1-0.05/2}(v_1, v_2)} \\ &= \frac{2 * (3 + 1) * F_{0.975}(2 * (3 + 1), 2 * (8 - 3))}{2 * (8 - 3) + 2 * (3 + 1) * F_{0.975}(2 * (3 + 1), 2 * (8 - 3))} \\ &= \frac{8 * F_{0.975}(8, 10)}{10 + 8 * F_{0.975}(8, 10)} \\ &= 0.755 \end{aligned} \tag{5.5}$$

計算の結果 $X_{Li} > X_{Rj}$ となっているためこの場合は，本手法を適応してノード i を展開することが出来る．

5.3.3 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 i_1，i_1 を除いた兄弟ノード中で最も高い勝率をもつノードを i_2 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．このとき，どの程度の精度で勝率の区間を推定するのかを決定するパラメータ C, α は実験によって定める．

$$X_{Li_1} \geq X_{Ri_2} \tag{5.6}$$

提案する手法と基本となる展開条件と組み合わせて探索木を成長させていく．基本となるノード展開条件には最低訪問回数が大きめの場合と局面の合法手数の両方を試す．ただし，ノードの訪問回数が少ない場合は勝率の信頼性が低く，勝率の信頼区間 $[X_L, X_R]$ の幅が広いため，本提案手法 2 の効果を発揮したノードの展開は発生し難い．つまり，提案手法 2 で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ことが予想される．

5.3.4 予想される挙動

本手法では，突出して高い勝率を持つノードを優先して展開する．同じ探索回数で有望な手に対してより探索木を成長させることができるため，より正確な手の評価が期待できる．また，一見勝率が高く良手だと考えられたノードが，より詳しく探索を行うと悪手であった場合には，より早く悪手であると気が付くことが期待できる．

一方で，手法 1 が場合によっては 1 回の訪問で展開できるのに比べ，本手法では最低 11 回程度の訪問が必要であるという特徴もある．

5.4　手法3：訪問回数の推定を用いたノード展開手法

5.4.1　概念

本手法ではUCT探索終了時の訪問回数を探索途中で推定し，十分に訪問すると予想される場合には、ただちにノードを展開する．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ある）．そして，それらの情報をもとに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最大訪問回数は，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したもので抑えられる事が判っている [9]．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．

$$E(T_i(N)) \leq \frac{8\log N}{(x^* - x_i)^2} + 1 + \frac{\pi^2}{3} \tag{5.7}$$

$T_i(N)$: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

つまり，兄弟ノード中の最大勝率と自身の勝率の差が小さい手ほど多くの訪問が期待できる．そこで，ノードiの最大推定訪問回数を

$$F(i) = \frac{8\log N}{(x^* - x_i)^2} + 1 + \frac{\pi^2}{3} \tag{5.8}$$

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．しかし，式5.8のままでは x^* と x_i の差が非常に小さい場合に N よりも大きくなってしまう．より極端な例では，x^* と x_i が同じ勝率を持っていた場合に式5.8の結果が無限大になってしまう．また，全体の訪問回数の中の現時点までの訪問回数は自明であるため推定する必要が無い．そこで判明しているノード i の訪問回数 n_i，ノード i の親ノードの現時点の訪問回数 N 'を用いて各ノードの推定訪問回数を以下の式で求める．ここで，Bとは合法手の集合である．

$$F'(i) = n_i + \frac{\min(N - N', F(i))}{\sum_{i \in B} \min(N - N', F(i))} \cdot (N - N') \tag{5.9}$$

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．

5.4.2 ノード展開条件

本提案手法 3 では，式 5.9 を用いて各ノードの推定訪問回数を求め，推定値が基本となる展開条件を満たすなら十分な訪問を行う前にノードを展開する (図 5.3)．逆に式 5.9 で求めたノード i の推定訪問回数 $F(i)$ が基本となるノード展開条件を満たさない場合は，最後まで展開の閾値を満たさないと判断して，展開は行わない (図 5.3)．基本となる展開条件は固定閾値と合法手数展開手法の両方を試す．

ノードを展開する場合　　ノードを展開しない場合

図 5.3: 手法 3 を用いた展開ノードの判別

また，現時点での勝率差 $x^* - x_i$ を使うと 1 回訪問して 1 勝したノードでは展開がされてしまうため，最低 5 回以上訪問したノードに対してのみ本手法を適用する．

5.4.3 予想される挙動

提案手法 2 は末端ノード中に良手が 1 つしか無い場合にのみ展開を促進する．しかし，提案手法 3 では探索木の成長を予想することで全体的に効果を発揮することが期待できる．また，残りの探索時間が長ければ長いほど式 5.9 は後半部分が支配的になるため効果が得られやすい．逆に，訪問が進むにつれて式 5.9 の値はノード i が既に探索された回数 n_i に近づく．そのため，探索の終盤では効果が得られないことが予想される．

第6章　実験・検証

6.1　目的

本章では，5 章で提案した 3 つの手法についての実験を行い，結果を検証する．実験条件は 4.2.1 に準じる．

6.1.1　実験環境

実験環境は 4 章での実験で用いたものと同じ環境を用いた．

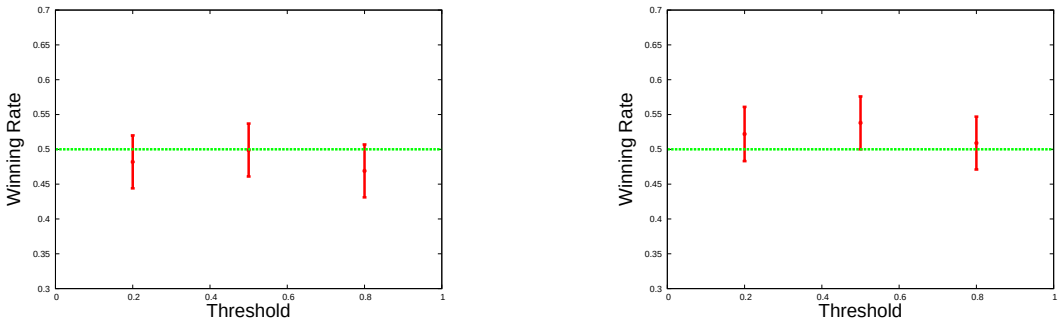
- process：1（ただし，Nomitan は本研究の提案も含めマルチスレッドに対応している）
- OS：Linux
- CPU：Intel(R) Xeon(R) CPU L5520 @ 2.27GHz
- メモリ：24.7GByte （実際には数 GB 程度しか利用しない）
- cashe size：8192 KB
- 開発言語：C++

6.2　対戦実験

提案手法を実装した Nomitan と未実装の Nomitan を対戦させることで，提案手法の有効性を検証する．基本となる展開条件は，最低訪問回数が 50 回以上で展開するものと，合法手数展開手法の両方について試した．本実験では最大記憶ノード数は基本となる展開条件が，最低訪問回数が 1 回の場合よりも充分に有効であると判断できる 10000（１万）に設定した．

6.2.1　手法 1：評価値の突出度を用いた場合の実験結果

今後十分なプレイアウトが行われるであろうと判断する値 Th を変えた場合の従来手法に対する勝率を図 6.2 に示す．Th を横軸に, 提案手法の勝率を縦軸に取る．自己対戦は各条件で 600 回行った．



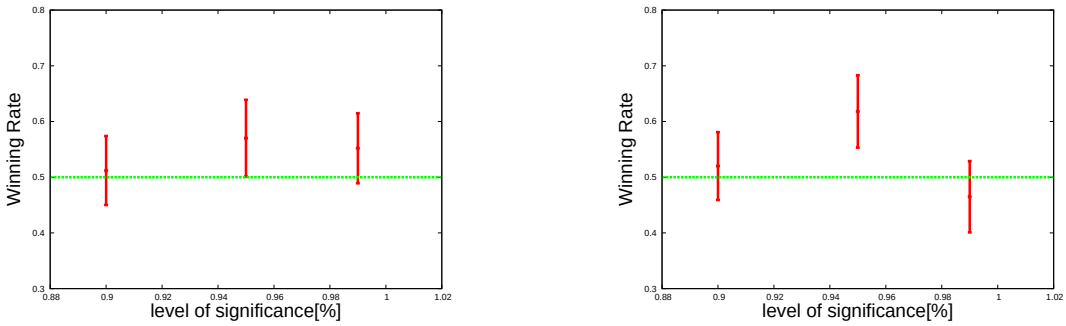
合法手数展開方式　　最低訪問回数 50

図 6.1: 手法 1 の従来手法に対する勝率

最低訪問回数が 50 回で展開する場合，Th が 0.5 のときに従来手法よりも有意に強くなることが判った．しかし，Th が大きい場合と小さい場合は有意に強くなったとは言えなかった．これは，Th が小さいと無条件にノードが展開されるため最低訪問回数が 1 回で展開する手法の挙動に近づく．また Th が大きいと，提案手法そのものの効果が得られ難くなり，結果，強さに影響を与えないことになる．

6.2.2　手法 2：勝率の突出度を用いた手法の実験結果

手法 2 では各ノードにおける勝率の区間を推定する．推定する精度（有意水準）を変えて，従来手法に対する有効性を検証した．このとき，式 3.1，3.2 で用いる係数 C 及び式 5.2，5.3 で用いる α は勝率の区間の有意水準に準じる．自己対戦は各条件で 200 回行った．



合法手数展開方式　　最低訪問回数 50

図 6.2: 手法 2 の従来手法に対する勝率

勝率の区間を 95 %の有意水準で推定した場合，2 つの展開条件について有意に強くなっていることを確認した．有意水準が 90 %と 99 %の場合では有意に強くなったとは言えなかった．これは，精度を減らすと不要なノードを展開してしまい，探索の効率が上昇しなかったと考えられる．また，逆に推定の精度が高すぎると提案手法そのものの効果が得られ難くなったからだと考えている．

6.2.3 手法 3：訪問回数の推定を用いた手法の実験結果

5.4 節で提案した訪問回数の推定を用いた手法を実装し，提案手法の有効性を検証した．表 6.1 に対戦結果を示す．実験の結果，提案手法 3 は従来手法について有意に強くなっていると言える充分な結果は得られなかった．ただし最大記憶ノード数が 10000 の場合の，展開に用いる最低訪問回数が 1 回の場合の様に弱くなることはなかった．この事から決して無駄なノードを展開しているのではなく，今後展開されるであろうノードをある程度正確に予測できていると考えられる．自己対戦は各条件で 200 回行った．

表 6.1: 手法 3 の従来手法に対する勝率

	勝率	95 %信頼区間
合法手数展開手法+提案手法 3	0.525	± 0.069
最低訪問回数 50+ 提案手法 3	0.4600	± 0.069

6.2.4 探索速度

前節までで行った対戦実験の際の各手法の 1 秒当たりのプレイアウト回数の平均値を表 6.2 と表 6.3 にまとめた．展開条件が異なれば探索の挙動そのものも変化するため，正確に比較することは困難でありこれはあくまで実験的な数値である．表 6.2，6.3 を見ると各手法の探索速度に大差はないように見受けられる．このことから，探索速度が強さに影響を与えているわけではないといえる．

表 6.2: 各提案手法の速度比較 1

手法	プレイアウト回数/秒
合法手数展開手法	1483.67
合法手数展開手法+評価値の突出度 (Th=0.5)	1481.86
合法手数展開手法+勝率突出度	1398.74
合法手数展開手法+訪問回数推定	1585.74

表 6.3: 各提案手法の速度比較 2

手法	プレイアウト回数/秒
最低訪問回数 50	1419.32
最低訪問回数 50 +評価値の突出度 (Th=0.5)	1481.86
最低訪問回数 50 +勝率突出度	1398.98
最低訪問回数 50 +訪問回数推定	1586.01

6.3 展開の進行度

図 6.3 に示す局面を提案手法を実装した Nomitan に与えて，探索の進行と提案手法の効果の進行具合について調べた．基本となる展開条件は，最低訪問回数が 50 回以上で展開するものと，合法手数展開手法の両方について試した．本実験では 10000 回の探索を 10 回繰り返し，提案手法が適用され，ノードが展開された回数の平均を調べた．手法 1 においては，自己対戦で最も良かった $Th = 0.5$ を超える場合にノードを展開することにする．同様に手法 2 については，勝率の区間推定は最も実験結果の良かった有意水準 95 %で行った．最大記憶ノード数は基本となる展開条件が，最低訪問回数が 1 回の場合よりも充分に有効であると判断できる 10000（１万）に設定した．

6.3.1 実験条件

展開条件と強さの検証を検証する際の実験条件は，条件 4.2.1 に準じる．Nomitan に与える局面は図 6.3 に示す．この問題は，手番が黒番で右下の黒石が危険で，手を抜くと取られてしまうため，深く読むことによりその死活を正確に評価する必要がある．

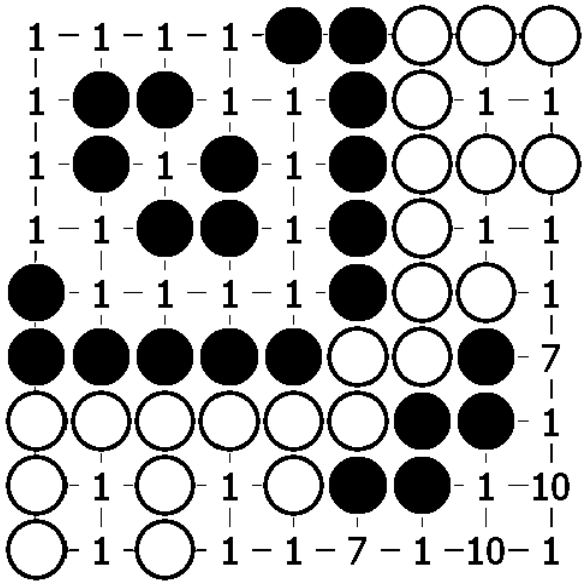


図 6.3: 死活問題の問題（数字が 10 となっている箇所が正解）

6.3.2 実験結果

図 6.4〜図 6.8 に実験結果を示す．本実験において生成された探索木の各深さのノード数を表したグラフが図 6.4 と図 6.5 である．横軸が探索木の深さを表し，縦軸が各深さにおける生成されたノード数を示す．また，図 6.6〜図 6.8 は探索が進むにつれて，提案手法がどのような頻度で効果を発揮しているかの関係を表す．横軸が探索回数を表し，縦軸が提案手法によって展開が行われた回数の合計を示す．

図 6.4 が基本の展開条件として合法手数展開手法を用いた場合の探索木のノード数を表す．図 6.5 は基本の展開条件として最低訪問回数が 50 回以上でノードを展開した場合の探索木のノード数を表す．図 6.4 と図 6.5 からは，提案手法 3 がその他の提案手法に比べて，特に深さ 2 と 3 でノードの展開を促進していることがわかる．その他 2 つの提案手法については，主に深さ 4 から 7 で弱手の展開促進が確認できる．

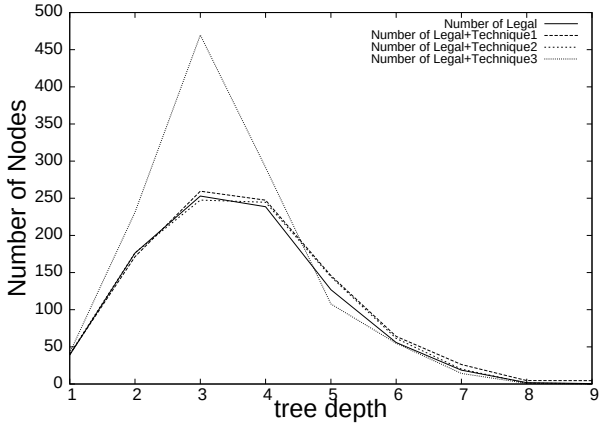


図 6.4: 探索木の各深さのノード数
（合法手数展開手法）

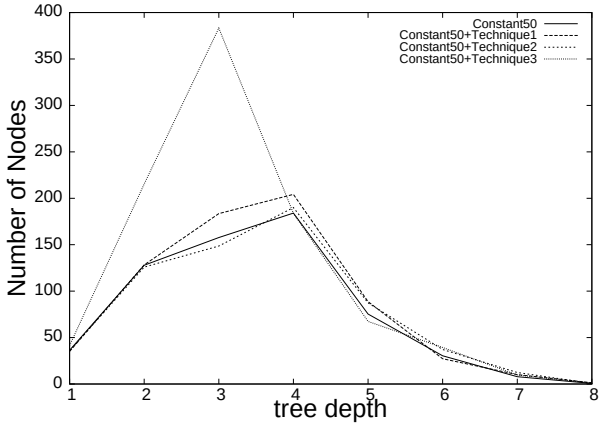


図 6.5: 探索木の各深さのノード数
（最低訪問回数=50）

また図 6.6〜図 6.8 から，枝刈りは効果が発生するまでにある程度のプレイアウトが必要であったが，本稿で提案する手法は全て探索開始直後から効果を発揮していることが判る．また，図 6.6 からは提案手法 1 の効果が探索開始直前から終盤まで均一に効果が得られていることが，図 6.7 からは提案手法 2 は最低でも 1000 回弱ほど探索を行わなくては効果が得られないという予想通りの結果が得られた．また提案手法 2 は各合法手の勝率の区間推定を用いているため，その区間が狭まらなくては効果が得られ難い．図 6.7 のグラフからおよそ 6000 回の探索を行うことで，勝率の区間推定が狭まり効果が発揮され易くなることが判った．図 6.8 からは提案手法 3 がその他の提案手法に対して非常に多くの効果が得られていることがわかる．これは，提案手法 3 で用いた式 5.8 は訪問の上限回数を求める式であったため，実際の訪問回数よりも多く訪問されると推定してノードを展開している事が考えられる．また，図 6.8 から提案手法 3 は探索終盤で効果が発揮されていない．これは 5.4.3 節で予想した通り式 5.9 の値が n_i に近づくため，効果が得られ難くなったと考えられる．

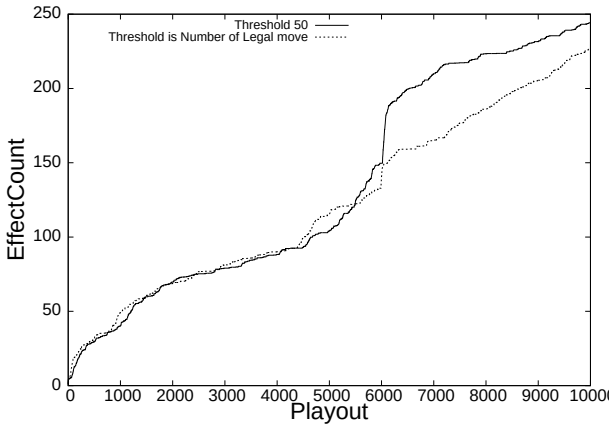


図 6.6: 手法 1 の効果の現れ方

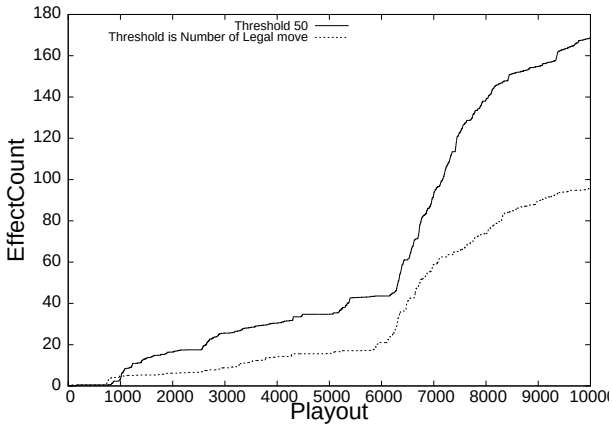


図 6.7: 手法 2 の効果の現れ方

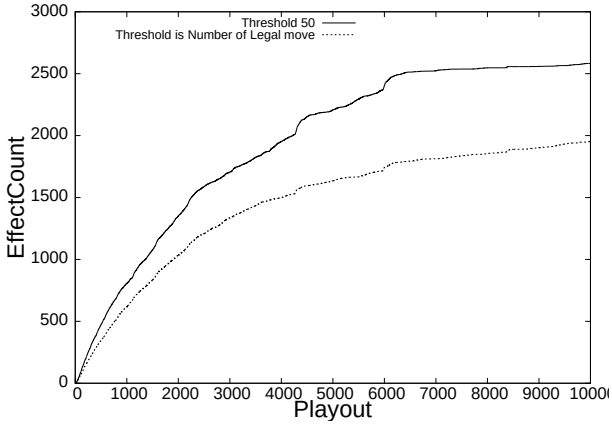


図 6.8: 手法 3 の効果の現れ方

6.3.3 結果のまとめと考察

本章では合法手数展開手法と展開に境界閾値を用いた場合において，3 つの提案手法が有効であるか（探索を効率化できたか）を検証した．実験結果から，提案手法 1 と 2 を用いて有望な手を推定し探索木を成長させる手法が，従来の展開手法に比べて有意に強くなったと言うことが出来た．また，今回提案した 3 つの手法は探索の初期段階から効果を発揮することが判った．これは，既存の枝刈り手法との挙動の大きな違いである．

今後の課題として，UCT の展開手法条件に用いる閾値が小さい場合については検証が不十分である．そのため，展開の閾値が小さい場合についても検証する必要があると考える．また，合法手数展開手法と展開に境界閾値を用いた展開が 19 路盤など本当に探索規模が増大した場合に有効であるかを検証する必要がある，加えて，今回提案した手法は枝刈り手法とは共存可能であるため，2 つの手法を組み合わせることで更なる性能の向上が期待できる．

第7章　おわりに

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最低訪問回数を用いた手法について検証を行った．結果，最低訪問回数の閾値の違いが強さに与える影響が判った．また，最低訪問回数の閾値に合法手数を用いた手法が有用であることを確かめた．次に，最低訪問回数値が大きい場合や合法手数の場合について探索の効率化法を3つ提案し，内2つについて有効性を確認した．

しかし，普遍的なノード展開手法について効率化する手法を提案することは出来なかったため，より普遍的な探索効率化手法について研究を行っていきたい．研究の方向性としては，近年評価関数の自動調節法の研究が進みプレイアウトの精度が向上している．このことが木探索に与える影響は無視できないはずである．精度の向上したプレイアウトに適応したモンテカルロ木探索を考えることが今後重要となるであろう．また，本研究の展開手法を踏まえ，探索の深さをより柔軟に制御することで探索の効率化を実現する手法を検討して行きたい．

謝辞

本研究を進めるにあたり，大変多くの方にお世話になりました．飯田弘之教授や池田心准教授，橋本剛准教授には論文作成などで的確なご指導，ご教授を頂きました．

また，私が本論文を書き上げることが出来たのは，たくさんの助言・ご協力を頂きました橋本隼一氏，松井利樹氏，野口陽来氏，他飯田・池田研究室の皆様のおかげであります．

ここに，心よりの感謝の意を表します．

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] Victor L. Allis. Searching for Solutions in Games and Artificial Intelligence. PhD thesis, University of Limburg, 1994.

[3] L.Kocsis，C.Szepesvari，Bandit Based Monte-Carlo Planning，*LNCS* vol．4212，pp．282-293，2006.

[4] 大崎泰寛, 小谷善行， 選択的シミュレーションに基づいた評価関数の学習，情報処理学会 第 14 回ゲームプログラミングワークショップ GPW2009,pp.135-141, Nov. 2009.

[5] コンピュータ将棋、初勝利　女流王将を下す　情報処理学会「あから」，http://www.itmedia.co.jp/news/articles/1010/11/news005.html，アクセス日時:2011.02.06 14:00

[6] Bernd Brüegmann，Monte Carlo Go (1993)

[7] B.Bouzy and T.Cazenave. Computer go: An AI oriented survey. *Artificial Intellegence*, 132(1): 39-103, 2001

[8] T. L. LAI AND H. ROBBINS, Asymptotically efficient adaptive allocation rules, *Adv. Appl. Math.,* 6 (1985), pp. 4.22.

[9] P.Auer，N.Cesa-Bianchi，P.Fischer，Finite-time analysis of the multiarmed bandit problem，*Machine Learning*，vol．47，pp．235-256，2002.

[10] G. Chaslot, M. Winands, J. Uiterwijk, H.J. van den Herik, and B. Bouzy, "Progressive strategies for Monte-Carlo Tree Search", New Mathematics and Natural Computation, 4(3), 2008, pages 343-357. *2008 Chess Base best publication award.*

[11] B. Bouzy and G. Chaslot. Bayesian generation and integration of k-nearest-neighbor patterns for 19x19 go. In G. Kendall and Simon Lucas, editors, *IEEE 2005 Symposium on Computational*

[12] H. Yoshimoto, K. Yoshizoe, T. Kaneko, A. Kishimoto, and K. Taura: Monte Carlo Go Has a Way to Go, Twenty-First National Conference on Artificial Intelligence (AAAI-06), pages 1070-1075, 2006

[13] Junichi Hashimoto, A Study on Domain-Independent heuristice in Game-Tree Search, 博士論文，北陸先端科学技術大学院大学，2011.

[14] 眞鍋和子，村松正和，モンテカルロ碁におけるシーケンシャルパターンマッチの試み，卒業論文，電気通信大学情報工学科，2008.

[15] Maarten P. D. Schadd, Mark H. M.Winands, H. Jaap van den Herik, Guillaume Chaslot, and Jos W. H. M. Uiterwijk. Single-player monte-carlo tree search. In Computers and Games, pages 1.12, 2008.

[16] Sylvain Gelly, Yizao Wang, Re'mi Munos, and Olivier Teytaud. Modification of UCT with patterns in Monte-Carlo Go. Technical Report 6062, INRIA, France, November 2006.

[17] 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.

[18] 土井，局所化と汎化を両立させる囲碁パターンマッチング，修士論文，北陸先端科学技術大学院大学，2011.

[19] 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.

[20] B.Bouzy，Move-Pruning Techniques forMonte-Carlo Go，CG 2005 *LNCS*，vol. 4250，pp. 104-119，Springer，Heidelberg，2006.

[21] B. Bouzy, B. Helmstetter. Monte-Carlo Go Developments. In 10th Advances in Computer Games, pp, 159 . 174, 2004.

[22] 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.

[23] 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

[24] 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

[25] T.Anantharaman, M.S.Campbell, F.H.Hsu, Singular Extensions:Adding Selectivity to Brute-Force Searching, Artificial Intelligence, vol. 43, 99-109, 1990.

付 録A　Appendix

A.1　囲碁プログラムNomitanの解説

本節では，我々が開発を行っているコンピュータ囲碁プログラム「Nomitan」に組み込まれている特徴的な手法について説明を行う．

A.1.1　探索：UCT

Nomitan は近年提案されたモンテカルロ木探索法に基づいて探索を行う．現在合法手の選択には UCB1 ではなく UCB1-Tuned を用いている．

A.1.2　探索ヒューリスティック：Progressive Widening[19]

新しいノードを展開し末端ノードを選択する際，一度にすべての合法手を探索対象とするのではなく，ゲームの知識を利用してよさそうな指し手から探索対象を広げていくという，枝刈り手法の一種である．Nomitan では土井 [18] の研究成果の評価値を用いている．評価値の高い合法手から順に探索が行われる．

A.1.3　評価関数

Nomitan ではプレイアウトや先述の ProgressiveWidening などに評価関数を用いている．詳しくは土井 [18] を参照されたい．

A.1.4　FPU[16]

FPU（先頭打着緊急度）とは，未探索のノードにおける UCB 値の代わりとなる値である．一般的に固定値が与えられる．一度でも末端ノードがプレイアウトされると，選択されたノードの評価値は UCB 値や UCB1-Tuned の値などに切り替わる．FPU の値が大きければ，既に探訪済みの手がさらに探訪される前に全ての手が一度探訪されることを保証しする．FPU の値が小さければ，UCB 値が大きいノードがあればそちらを優先する．この場合他の未探訪ノードは選ばれ難くなる．