社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－GI－15 (1)
2006／3／7

Bouzy's 5/21 algorithm を用いたDf-pn+探索の詰碁への適用

石井宏和† 横山大作† 近山隆†

コンピュータゲームプレイヤは、人工知能の黎明期から盛んに研究され続けており、現在まで多くのゲームで成功を収めているが、一方で将棋や囲碁など一部の複雑なゲームではたくさんの課題が残っている。それらゲームの問題の一部として、詰めを考える問題がある。詰将棋は、近年大きな進歩を遂げており、現在では、300手以上の手数を要する長編詰将棋が全て解かれている。一方詰碁の方では、まだ有効的な手段は見つかっていない。そこで、我々は現在AND/OR探索木で最も成功している探索手法の一つであるDepth-First Proof-Number(Df-pn)+探索に、盤面の形勢を評価するBouzy's 5/21 algorithmを適用して、詰碁の一眼問題に対して実験を行った。

Applying Df-pn+ using Bouzy's 5/21 algorithm to Tsumego

HIROKAZU ISHII,† DAISAKU YOKOYAMA† and TAKASHI CHIKAYAMA†

Checking mates in Shogi and Go have been studied for a long time as a field of computer game players. Great success has been achieved in Shogi, but there are still many challenges in Go. We apply a static evaluation function "Bouzy's 5/21 algorithm" which can estimate a state of position to Depth-First Proof-Number(Df-pn)+ search which is an advanced approach of AND/OR tree search have achived great successes in checking mates of Shogi and confirms an effect in Tsumego. This thesis presents the result of using this approach to a oneeye problem as a field of Tsumego.

1. はじめに

コンピュータゲームプレイヤは、人工知能の黎明期から盛んに研究され続けている[1),2)]。人工知能にとって、ゲームをコンピュータに行わせることは人間の「思考」に関する重要な研究課題であり、「思考」実現のためには、機械学習や探索・並列処理・認知科学など様々な要素技術を統合して扱う必要がある。現在、計算能力の向上や最新の手法を通して、いくつかのコンピュータエージェントの技術は、オセロやバックギャモン・チェスなどの複雑なゲームにおいて人間の世界チャンピオンのレベルか、あるいはそれ以上の領域にまで到達した。また、五目並べのようないくつかのゲームにおいては完全に解決された。しかし、探索空間が大きいこと、あるいは知識や判断基準の定義と優先順位の決定を正確に行うことの難しさなどにより、いまだアマチュアレベルに留まっているゲームもある。その中でも、将棋や囲碁などは難しいゲームとして知られている。

ゲームでは、そのゲームの問題の一部として、詰めを考える問題がある。将棋や囲碁では、その研究が始まったのと同時期の1970年代に、詰めの問題についても研究が始まった。詰将棋は、近年大きな進歩を遂げており、1997年に現存する最長手数の詰将棋である「ミクロコスモス」(1525手詰)が解かれ、現在では、300手以上の手数を要する長編詰将棋が全て解かれている。一方詰碁の方では、まだ有効的な手段は見つかっていない。

詰碁がコンピュータにとって難しいのにはいくつかの理由がある[3)]。一つは前述した探索空間の大きさである。囲碁は最大で盤の大きさが19×19の大きさを持ち、駒である石は特徴を持たない。そのため、ゲーム木が非常に大きくなってしまう、いわゆる組合せ爆発という現象を引き起こしてしまう。囲碁の探索空間はある試算では10^{360}にまで上ると言われており、チェスの探索空間10^{120}・将棋の探索空間10^{220}と比べるとその大きさが分かる。詰碁では探索する範囲が限られるためこれ程まで探索空間は大きくはならないが、それでも探索空間は依然大きく、難しさの原因の一つとなっている。もう一つに判断基準の曖昧さがある。将棋の場合、王を取ったものが勝者という絶対的な基準がある。しかし、囲碁の場合このような指標はなく、また将棋で判断の助けになっている駒の特徴のようなものもない。そのため、評価基準が曖昧性を帯びてしまい、難しさの原因の一つとなっている。そこで、我々は現在AND/OR探索木で最も成功している探索手法の一つ

† 東京大学
University of Tokyo

である Depth-First Proof-Number(Df-pn)+探索に、盤面の形勢を評価する Bouzy's 5/21 algorithm を適用して、詰碁の一眼問題に対して実験を行った。

2. 詰　　碁

詰碁とは、白黒の石が置かれた囲碁の盤面の一部または全部と、攻め手と受け手に手番 (「白番」、「白先」もしくは「黒番」、「黒先」) が示され、どのように打てば受け手は自分の石を生きにもちこめるか、あるいは攻め手は相手の石を取ることができるかという死活を考えるものである。詰碁の目的は黒先黒生・黒先白死・白先白生・白先黒死の 4 種類がある。本論文では、目的を達成できる場合 (黒先黒生ならば、黒が最終的に生きること) を詰みと呼び、達成できない場合を不詰みと呼ぶこととする。将棋では詰将棋がそれに対応する。詰碁は概して以下の三つに大分される。

- 攻め合い問題
  攻撃側と防御側の区別をしないでの勝者の決定が主目的
- 脱出・切断問題
  石の連絡の有無の判定が主目的
- 死活問題
  囲まれた領域内での、被攻撃側の石の眼の有無の判定が主目的

詰碁で詰むか詰まないかの判断は、眼の数を持って行われる。目的が生きの場合は、生かす石が眼を二つ以上持てば生きで詰みとなり、眼が一以下であれば死にで不詰みとなる。目的が死にの場合は、死なせる石が眼を一つ以下しかもたないなら死にで詰みとなり、二つ以上持てば生きで不詰みとなる。黒先白死の例を図 1、図 2 に示す。左の図 1 が問題で、右の図 2 がその答えとなる。この場合は、図 2 にあるように、黒が 1 の場所に打てば、その後白はどう打ったとしても二眼を持つことができず、死にとなる。この場合、黒の 5 が急所の一着となる。

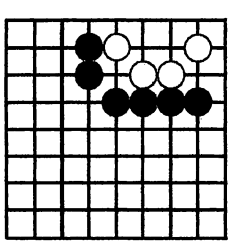

図 1　詰碁の例 (問)

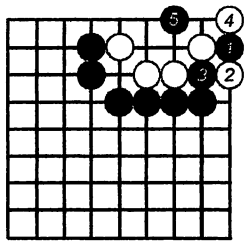

図 2　詰碁の例 (答)

詰碁の探索の行き着く結果は、生き・死にの他にも、セキやコウ・ヨセコウ・長生など循環を伴う状態もあり、問題は複雑となっている。そのため、本研究では以下に述べる通常の問題にさらに制限をかけた一眼問題に対して実験を行った。

2.1 一眼問題

一眼問題とは、詰碁の問題にさらに制限をかけた問題を指す[13]。詰碁は前述の通り、目的の石が二眼を持つかどうかを焦点とするが、一眼問題では目的の石が眼を一つ持った状態からスタートする。そこからさらにその目的の石がもう一つ眼を持つことができるかどうかを計算する。つまり、詰碁の問題としては生きだけを扱うことになる。一眼問題が前提とする事項は以下の通りである。

- 始めから生かすべき石が決まっている
- 目的の石は始めから眼を一つ持つ
- 受け手側の石は攻め手側の生きた石に囲まれている
- 打てる範囲が決まっている

一眼問題の例を図 3 と図 4 に示す。前述の詰碁の例と同様、左の図 3 が問題で右の図 4 がその答えとなる。両図の盤の左上隅にある眼が、一眼問題の前提となる、石が初めから持つ眼であり、生かすべき黒石は △ の印が付いた石である。また、黒の石を囲む白の石は既に二眼を作っており、生きの状態になっている。よって、これによって囲まれた × の付いた点が着手できる交点である。この場合は、図 4 にあるように、黒が 1 の場所に打てば、その後は白がどう打ったとしてもさらに一眼を作り、二眼を持つため生きることができる。この場合、黒の 1 が急所の一着となる。

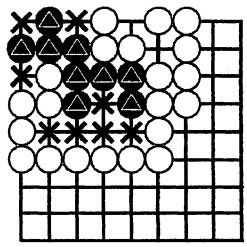

図 3　一眼問題の例 (問)

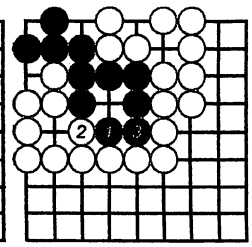

図 4　一眼問題の例 (答)

3. 関 連 研 究

本章では、関連研究として Df-pn+探索と Bouzy's 5/21 algorithm という 2 つの手法を紹介する。

3.1 Df-pn+探索

Df-pn+探索とは、現在 AND/OR 木探索で最も成功している探索手法の 1 つである。Df-pn+探索は、証明数探索と同じ振る舞いを保ちつつ深さ優先にした Df-pn 探索に、さらに評価関数を組み合わせたアルゴリズムである。Df-pn+探索について述べる前に、まず証明数探索と Df-pn 探索について紹介する。

3.1.1 証明数探索

証明数探索は、それぞれのノードに証明数と反証数という 2 つの閾値を設けて探索を行う手法である。証明数・反証数の元となったのは、1980 年代中頃に McAllester によって提案された共謀数という概念で、$\alpha\beta$ 法などの探索方法の対象とする Minimax 木に対して提案されたものである[4]。Minimax 木の探索においては、先端ノードの静的評価に適用される評価関数は一般に正確ではなく、その評価誤差が上のノードに伝播されて、ルートでの評価の誤りにつながる。しかし、一般には一つの局面だけでの評価誤差がルートでの評価誤差に直接結びつくわけではなく、複数のノードが共謀しなければ、ルートの評価値を変えることができない場合が多い。つまり、共謀しなければならないノードの数が多ければ多いほどルートでの評価値が誤っている可能性が高くなる。この考え方を AND/OR 木に適応したものが証明数・反証数の概念である。

証明数とは、各ノードにおいて、その詰みを証明するために必要な子ノードの最小の数を意味する。ノードが OR ノードであった場合、その各子ノードで示される証明数の値は、ノードを探索する時に最低限必要なリソースの量と考えることができる。よって、ノードを展開して次に選ぶ子ノード中での最も有望なノードを選ぶ判断基準として、有効に働くと考えられる。反証数も同様に有効であると考えられる。

証明数・反証数は、ノードの種類により計算方法が異なる。ノード n が末端ノードである場合の評価値が**true** つまりそのノードが詰みである時は、詰みであることが証明できているので証明数は 0 となり、不詰みであることは証明できないので、反証数は ∞ となる。末端ノードの評価値が**false**、つまりそのノードが不詰みである場合は、詰みの場合と逆なので、証明数が ∞ で反証数が 0 となる。また、評価値が不明の時は、どのくらいリソースをつぎ込めば詰み、あるいは不詰みが証明できるか分からないので、共に 1 とする。次に、ノード n が内部ノードでかつ OR ノードである場合は、そのノードの詰みを示す場合、子ノードのうちどれか一つが詰みであればよいので、その証明数は子ノードで証明数が最小のノードの証明数となる。また、不詰みであることを証明するためには、子ノード全てが不詰みでなければならないので、子ノードの反証数を全て足したものをそのノードの反証数とする。n が OR ノードの場合の証明数と反証数は、ノードが AND ノードの場合のそれらの計算と逆になる。このような計算を再帰的に行う。AND/OR 木探索における最終的な評価値 (「詰み」と「不詰み」) を**true** と**false** で一般化すると、証明数・反証数の定義は以下のようになる。

定義 [証明数 (反証数)]

(1) ノード n が末端ノード

(a) 最終的な評価値が**true** のとき

$$\mathbf{pn}(n) = 0 \quad (1)$$

$$\mathbf{dn}(n) = \infty \quad (2)$$

(b) 最終的な評価値が**false** のとき

$$\mathbf{pn}(n) = \infty \quad (3)$$

$$\mathbf{dn}(n) = 0 \quad (4)$$

(c) 最終的な評価値が不明のとき

$$\mathbf{pn}(n) = 1 \quad (5)$$

$$\mathbf{dn}(n) = 1 \quad (6)$$

(2) ノード n が内部ノード

(a) n が OR ノードのとき

$$\mathbf{pn}(n) = \min_{n_c \in children\ of\ n} \mathbf{pn}(n_{child}) \quad (7)$$

$$\mathbf{dn}(n) = \sum_{n_c \in children\ of\ n} \mathbf{dn}(n_{child}) \quad (8)$$

(b) n が AND ノードのとき

$$\mathbf{pn}(n) = \sum_{n_c \in children\ of\ n} \mathbf{pn}(n_{child}) \quad (9)$$

$$\mathbf{dn}(n) = \min_{n_c \in children\ of\ n} \mathbf{dn}(n_{child}) \quad (10)$$

3.1.2 Df-pn 探索

Depth-First Proof-Number(Df-pn) アルゴリズムとは、前述の証明数と反証数という二つの概念を用いる最良優先探索の証明数探索を深さ優先探索に変換し、多重反復深化法 (Multiple Iterative Deepning) という手法を用いたアルゴリズムである[6)~11)]。前述したように、深さ優先探索は、メモリの使用量や時間の効率が良く、詰碁のように大きな探索空間を持つ問題には適している。しかし、深さ優先探索では、解の存在する深さが深い場合、はまってしまう可能性があるので、そのような状況を避けるために探索をする時には閾値を設ける。多重反復深化法とは、この与えた閾値をそのまま用いるのではなく、各ノードで段階的に閾値を増やして探索を行う方法である。与えた閾値をそ

のままで探索を行った場合、不必要なノードを探索してしまうことも多いが、この方法を取ることで探索するノード数を抑えることが期待できる。これらの方法により、比較的少ないメモリ量で最良優先探索と同様の効率的な探索を実現している。

Depth-First Proof-Number 探索は、以下のようなアルゴリズムで探索を行う。

ルートノードrでの閾値を

$$r.th_p = \infty,\ r.th_d = \infty \tag{11}$$

とする。

(1) 各ノードnにおいては、上記の定義に則って計算したノードの証明数 $n.pn$ が閾値 $n.th_p$ 以上になるか、反証数 $n.dn$ が閾値 $n.th_d$ 以上になるまで (これを終了条件と呼ぶ) 自分の下を探索し続ける。

(2) nがORノードの場合、証明数最小の子ノード n_c、二番目に証明数の小さい子ノード n_2 を選ぶ (証明数最小の子ノードがもう一つあれば、そのノードを n_2 とする)。

$$n_c.th_p = \min(n.th_p, n_2.pn+1) \tag{12}$$
$$n_c.th_d = n.th_d + n_c.dn - \sum n_{child}.dn \tag{13}$$

のように閾値を割り当て、n_c を探索する。終了条件を満たすまでこの操作を繰り返す。

(3) n がANDノードの場合、反証数最小の子ノード n_c、二番目に反証数の小さい子ノード n_2 を選ぶ (反証数最小の子ノードがもう一つあれば、そのノードを n_2 とする)。

$$n_c.th_p = n.th_p + n_c.pn - \sum n_{child}.pn \tag{14}$$
$$n_c.th_d = \min(n.th_d, n_2.dn+1) \tag{15}$$

のように閾値を割り当て、n_c を探索する。終了条件を満たすまでこの操作を繰り返す。

(4) 終了条件を満たしたら、親ノード n_{parent} に処理を戻す。

閾値は、コンピュータがそのノードの探索につぎ込むリソースの量を表したものと考えることができる。ルートノードでは、リソースの全てをつぎ込むと言う意味で閾値を ∞ とする。ノードnがORノードの場合、証明数は子ノードのうちどれかが詰みであることを証明できればよい。よって、ノード n_c に許される証明数のためのリソースの量は、許されるリソースの全てか、あるいは二番目に証明数が少ないノード n_2 を証明するためにつぎ込まなければならないリソースの量に1を加えたもののうちより少ない方を取ればよい。反証数は子ノードすべてを不詰みであることを示さなければならないので、ノード n_c につぎ込める反証数のためのリソースの量は、許されたリソースの量から自分以外の子ノードの反証数を引いた値となる。ノードnがANDノードの時はORノードの逆を考えればよい。

3.1.3 Df-pn+探索

人間が局面を見て先読みをする時は、いくつかの見込みのある手を絞り、その手を深く先読みする。一方、あまり見込みのないような手は、早々に読むのを止めてしまう。コンピュータによる探索でも、このように見込みのある手・見込みのない手を判別することができれば、探索をより効率的に行うことができる。Depth-First Proof-Number+とは、Df-pnにヒューリスティックスと呼ばれる問題領域固有の知識を使って人間の見込みを表現できるようにした探索法である[14]。

ノードnからその子ノード n_{child} までにかかるコストを $cost_{(dis)proof}(n, n_c)$、あるヒューリスティックスを用いて静的に評価したノードnの評価値を $h_{(dis)proof}(n)$ と定義し、上記の式(5)〜(10)・(12)〜(15)を以下のように変更する。また、Df-pnではノードの閾値を決める時に、証明数最小のノードと二番目に小さいノードを選んだが、Df-pn+では、証明数にcostを加えたものの中で最小のものを n_c に、同様の基準で二番目に小さいものを n_2 として選ぶ。

$$\mathbf{pn}(n) = h_{proof}(n) \tag{16}$$
$$\mathbf{dn}(n) = h_{disproof}(n) \tag{17}$$
$$\mathbf{pn}(n) = \min_{n_c \in children\ of\ n} (\mathbf{pn}(n_{child}) + cost_{proof}(n, n_{child})) \tag{18}$$
$$\mathbf{dn}(n) = \sum_{n_c \in children\ of\ n} (\mathbf{dn}(n_{child}) + cost_{disproof}(n, n_{child})) \tag{19}$$
$$\mathbf{pn}(n) = \sum_{n_c \in children\ of\ n} (\mathbf{pn}(n_{child}) + cost_{proof}(n, n_{child})) \tag{20}$$
$$\mathbf{dn}(n) = \min_{n_c \in children\ of\ n} (\mathbf{dn}(n_{child}) + cost_{disproof}(n, n_{child})) \tag{21}$$
$$n_c.th_p = \min(n.th_p, n_2.pn + cost_{proof}(n, n_2) + 1) - cost_{proof}(n, n_c) \tag{22}$$
$$n_c.th_d = n.th_d + n_c.dn - \sum(n_{child}.dn + cost_{disproof}(n, n_c)) \tag{23}$$
$$n_c.th_p = n.th_p + n_c.pn - \sum(n_{child}.pn + cost_{proof}(n, n_c)) \tag{24}$$
$$n_c.th_d = \min(n.th_d, n_2.dn + cost_{disproof}(n, n_2) + 1) - cost_{disp}(n, n_c) \tag{25}$$

今まで探索の途中で評価値が不明なノードは評価値を1としてきたが、各ノードの局面によって静的に評価を与え、その差異で各ノードの差別化を行う。この評価値を判断するものとして、hを用いる。つまり、hは兄弟ノードの優劣を計る指標と言い換えることができる。また、costを用いて、各ノードにおける証明数・反証数や閾値の計算では、次に探索するノードを選ぶ際に他の兄弟ノードに対して局面に評価値を与えることにより差別化を行い、次に探索するノードの順番を変える。つまり、costは当該ノードの展開の重さを計る指標と言い換えることができる。このようなhとcostという二つのヒューリスティックスを用いて、人間が行っている見込みの判断を表現する。

3.2 Bouzy's 5/21 algorithm

Bouzy's 5/21 algorithmは、Brono Bouzyによって作られた、囲碁の局面を静的に評価するアルゴリズムである[15]。このアルゴリズムは、基本的にDilationとErosionという2つの操作で構成される。アルゴリズムの流れは以下の通りである。

(1) 盤上で死んでいる石を取り除く
(2) 盤上で残った石のある交点に対して、高い評価値を与える(例えば、黒ならば128、白ならば-128)
(3) 空点に0の評価値を与える
(4) Dilationをn回繰り返して行う
(5) Erosionをm回繰り返して行う

DilationとErosionの操作は以下の通り。

- Dilation(膨張)
  - 交点の評価値が0以上かつ負の評価値を持つ交点が周りに無い
    - 現在の評価値に周りにある正の交点の個数を足す
  - 交点の評価値が0以下かつ正の評価値を持つ交点が周りに無い
    - 現在の評価値から周りにある負の交点の個数を引く
- Erosion(侵食)
  - 交点の評価値が正
    - 現在の評価値から周りにある0以下の交点の数を引く
  - 交点の評価値が負
    - 現在の評価値に周りにある0以上の交点の数を足す
  - それぞれ評価値が0になったらErosionの操作を止める

これらの操作を碁盤上の交点全てに対して行う。以下にDilationを3回、Erosionを7回行った流れを例として示す。

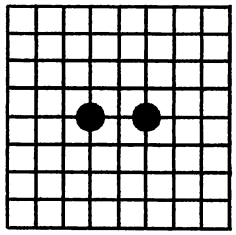

図5 初期状態

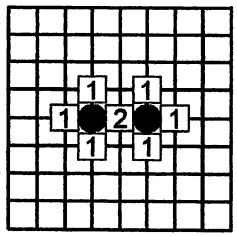

図6 1 Dilation

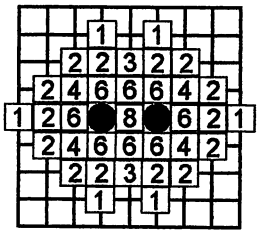

図7 3 Dilation

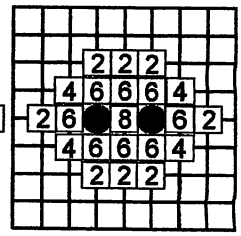

図8 3 Dilation + 1 Erosion

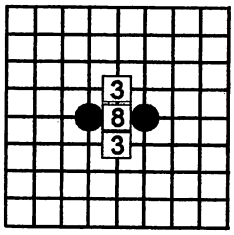

図9 3 Dil + 6 Ero

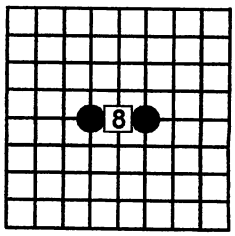

図10 3 Dil + 7 Ero

Erosionの回数は、m = 1 + n(n - 1)の計算式[16]から、Dilationの数nで表現される。また、このアルゴリズムは*GNU Go*[21]のバージョン2で使われており、5Dilation+21Erosionを目算、5Dilation+10Erosionを模様、4Dilation+0Erosionを勢力図として用いられており、目的によってDilationやErosionの回数を変化させる。また、現行のバージョン3でも連(石の連なり)の呼吸点(ダメ)の算出に用いられている。

4. 提案手法

4.1 研究の方針

評価関数は、探索手法とともにコンピュータゲームプレイヤの振る舞いを決める重要な要素である。この評価関数は、囲碁を始めとして将棋やチェスなどの比較的難しいゲームにおいて、人手で作成するのが一般

的となっている。評価関数がより正確であれば探索はより正しい結果を導くことができるが、評価関数の正確さは処理速度とトレードオフの関係にある。特に囲碁に関しては、駒に個性が無いなどゲームの特徴から、正確さを増すために *GoTools*[12),17)~20)] のようにかなりの部分を人手で書いており、処理速度の無視できない割合を評価関数の部分が占めている。先行研究として岸本が行った詰碁の解析プログラムに Depth-First Proof-Number 探索を用いている例[13)] も、評価関数として囲碁の知識を用いて作成したものを使っている。そこで、本研究では前章で紹介した一定の計算式から局面を評価でき、計算時間も比較的軽くて済む Bouzy's 5/21 algorithm を Depth-First Proof-Number+探索の評価関数として適用して、人手で作られたものだけではなく定式化された評価関数でも有効であることを示し、現状よりも良い結果を出す詰碁解析プログラムを作ることを目指した。

4.2 システムの実装

一眼問題に Bouzy's 5/21 algorithm を適用することを考える。当然、眼となっている部分は評価値が最も高いはずである。詰碁では二眼を作れば生きなのであるから、Bouzy's 5/21 algorithm を適用した時に、始めからある眼の評価値と同じ評価値を他の交点が持てば、そこがもう一つの眼となる可能性が非常に高い。このことから、本研究では局面に Bouzy's 5/21 algorithm を適用し、その中で二番目に高い評価値を最大化するような着手に最大の評価を与える評価関数を作成した。また、探索は置換表[22)] を用いて探索の効率化を図った。

5. 実験結果

5.1 実験環境

実験は、Intel Pentium 4 1.90GHz・メモリ 512MB のマシン上で行った。実装には C++言語を用いた。

5.2 実験結果

提案手法との比較検討のために、A:反復深化法に子ノードの優先順位付けとして Bouzy's 5/21 algorithm を適用したもの、B:Depth-First Proof-Number 探索、C:Depth-First Proof-Number 探索で pn・dn が同じだった場合に Bouzy's 5/21 algorithm を使って子ノードの優先順位付けをしたものの 3 つを用意し、これと本研究で提案する提案手法:Depth-First Proof-Number+探索に Bouzy's 5/21 algorithm を適用したものの計 4 つに対して先行研究である[13)] に用いられていた一眼問題の一部と、それに訂正を加えたものの計 40 問に対して実験を行った。手法 A～手法 C の探索ノード数をそれぞれを縦軸とし、提案手法の探索ノード数を横軸とした散布図を以下の図 11～図 13 に示す。散布図の 1 プロットが 1 つの一眼問題に相当する。

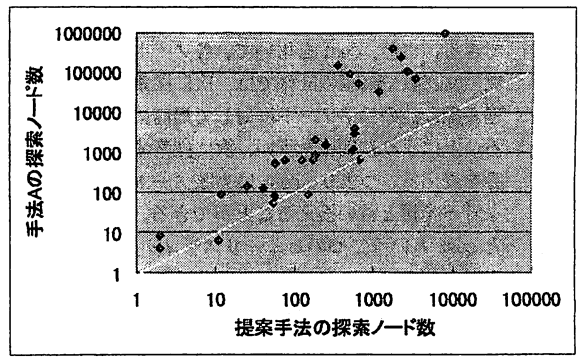

図 11 手法 A と提案手法の探索ノード数の散布図

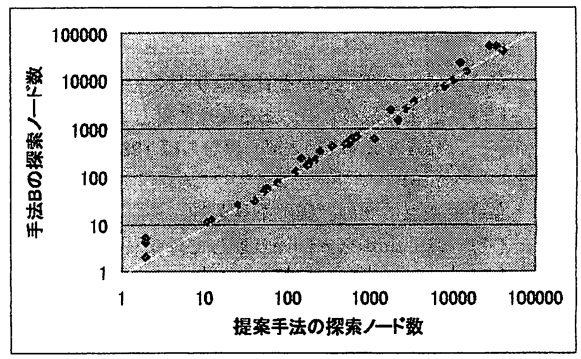

図 12 手法 B と提案手法の探索ノード数の散布図

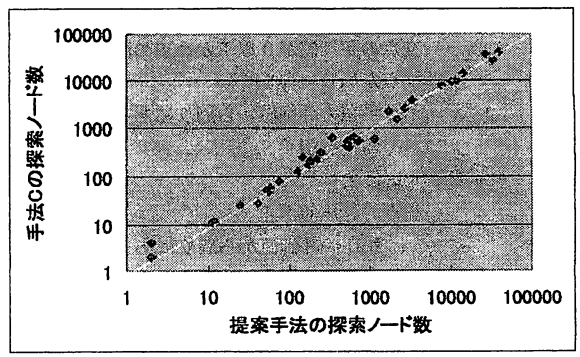

図 13 手法 C と提案手法の探索ノード数の散布図

6. 考　　察

本章では、実験の結果の考察を行う。なお、今回の実験では、碁盤プログラムが完成には遠いため、一手の生成や着手の戻しの処理速度が遅く、計算の時間では正確な比較検討が出来ないことから、検討事項から計算時間を除いて考える。

図 11 を見てみると、特に探索ノード数が増えれば増えるほど提案手法の方が良い結果となっている。一方、図 12 と図 13 を見てみると、どちらも散布図からはあまり変化をみることは出来ない。

次に、手法 A～C のそれぞれと提案手法で各問題毎に探索ノード数を比較し、提案手法の方が探索ノード数が少なかった問題の数・同数だった問題の数・多かった問題の数を次の表 1 に示す。

表 1　探索ノード数から見た問題の数

	手法 A	手法 B	手法 C
少なかった問題の数	36	21	12
同数だった問題の数	1	12	16
多かった問題の数	3	7	12

手法 A を見てみると、提案手法の方が探索ノード数が少なかった場合が圧倒的となっており、散布図からみても提案手法の方が優れているということができる。手法 B を見てみると、問題のうち約半分が提案手法の方が探索ノード数が少なくなっている。しかし、一方で提案手法の方がかった問題の数も 4 分の 1 に上り、おおむね提案手法の方が良い結果を出しているが、局面によっては悪い結果が出ることがあるという結果になった。手法 C を見てみると、少なかった問題の数と多かった問題の数は同数であり、提案手法との差は見らず、手法 C とは優劣がほぼないという結果になった。

これらの問題の探索ノードの数から、提案手法の出す結果についていくつかの傾向が見えてきた。探索する領域がある程度まとまった大きさの空点である場合、特に問題 14 のように、探索領域の中に受け手の石がいくつかある場合は提案手法が優れた結果を出した。これは、本研究の方針が問題によくはまった場合であると考えられる。しかし、一方で提案手法では上手くいかない場合も見つかった。

上手くいかなかったのは、その問題の解が受け手の守ろうとしている石と離れている場合である。問題 15 がこのような場合に当たる。Bouzy's 5/21 algorithm は、目算や領地の計算として用いられているように、自分の陣地であると見なせるような場所、つまり自分の石がたくさんあるような場所に高い評価値を示す。そのため、石と解が離れていると、その交点に対しては高い評価値は出ず、その場所に対する着手は他の着手より低い評価値となり、余計な部分を探索してしまうことになる。

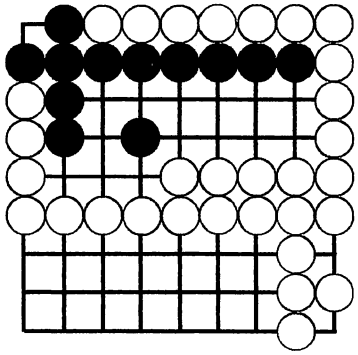

図 14　成功したら例

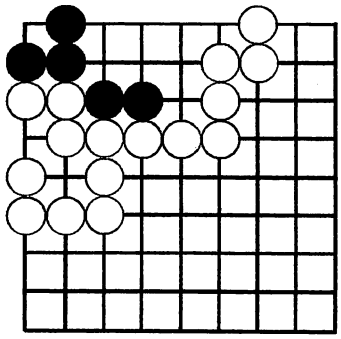

図 15　失敗した例

7. おわりに

7.1 ま と め

本研究は、詰碁を効率的に解くことを目的とし、現在最も成功している AND/OR 木探索である Depth-First Proof-Number 探索をさらに改良した Depth-First Proof-Number+探索に、一定の計算式から局面を評価でき、計算時間も比較的少なくて済む Bouzy's 5/21 algorithm を評価関数として適用して、詰碁の一分野である一眼問題に対して実験を行った。提案手法の比較検討のために、手法 A:反復深化法に探索順序制御として Bouzy's 5/21 algorithm を用いたもの・手法 B:Depth-First Proof-Number 探索・手法 C:Depth-First Proof-Number 探索に探索順序制御として Bouzy's 5/21 algorithm を用いたものの三つを

用意した。その結果、提案手法は手法 A よりも優れた結果を出すことができた。手法 B と比べるとおおむね良い結果を出したが局面によっては悪い結果も出していた。手法 C とはほぼ優劣がつかないという結果になった。

7.2 今後の課題

今後、まずは碁盤プログラムの改善をしなければならない。その上で、今回プログラムの速度が出なかったため検討できなかった計算時間についても再度検討していきたい。

次に、今回 Bouzy's 5/21 algorithm では Dilation を 5 回、Erosion を 21 回行ったが、この計算の回数を変えて実験を行うことを考えている。前述したように、Erosion の回数は Dilation の回数から計算されているが、これは経験則に基づいたものであるので、この計算方法を崩して色々な回数で実験を行い、Dilation5 回と Erosion21 回よりも良い結果が出せる回数の組合せを探してみたい。また、Erosion を行わず、Dilation だけを使った評価関数というのも考えてみたい。

次に、Bouzy's 5/21 algorithm の評価関数への適用の仕方の変更も考えてみたい。本研究では評価する局面と一手前の局面に Bouzy's 5/21 algorithm を適用し、一手前の局面の全ての交点の中で二番目に高い評価値を持つ交点の評価値を最大化するような着手に対して最大の評価を与える評価関数を用いて実験を行った。今度は、局面に対して Bouzy's 5/21 algorithm を適用した後、探索可能領域の全ての交点の評価値の和を最大化するような着手に最大の評価を与えるという評価関数を作ってみたいと考えている。

参 考 文 献

1) 松原仁, 竹内郁雄 編著:ゲームプログラミング, 共立出版,1998.
2) Bruno Bouzy and Tristan Cazenave. *Computer Go: an AI Oriented Survey*. June 2001.
3) Martin Muller:*Not Like Other Games - Why Tree Search in Go is Different*,In Proceedings of Fifth Joint Conference on Information Sciences (JCIS 2000), pages 974-977, 2000.
4) 脊尾昌宏: 共謀数を用いた詰将棋の解法, コンピュータ将棋の進歩 2, pp.1-28, 共立出版, 1998.
5) Ayumu Nagai, Hiroshi Imai: Applicaqtion of df-pn+ to Othello Endgames. In *The 5th Game Programming Workshop (GPW'99)*, pages 16-23, 1999.
6) Ayumu Nagai: A NEW DEPTH-FIRST-SEARCH ALGORITHM FOR AND/OR TREES, A Master Thesis, the University of Tokyo, February 1999.
7) 長井歩: df-pn アルゴリズムと詰将棋を解くプログラムへの応用, アマ 4 段を超える -コンピュータ将棋の進歩 4, pp.96-114, 共立出版, 2003.
8) A. Kishimoto, M. Muller: Df-pn in Go: Application to the one-eye problem. In *Advances in Computer Games. Many Games, Many Challenges*, pages 125-141. Kluwer Academic Publishers, 2003.
9) Akihiro Kishimoto: A Correct Algorithm to Prove No-Mate Positions in Shogi, 9th Game Programming Workshop in Japan (GPW2004), pages 1-8, November, 2004.
10) Akihiro Kishimoto: About the Tsume-Shogi Solver of ISshogi, at Computer Shogi Association Bulletin Vol. 16, 2004.
11) Akihiro Kishimoto and Martin Muller: A General Solution to the Graph History Interaction Problem, Nineteenth National Conference on Artificial Intelligence (AAAI'04), pages 644-649, AAAI Press, 2004.
12) Thomas Wolf: *GoTools*, `http://alpha.qmw.ac.uk/~ugah006/gotools/t.wolf.html`
13) Akihiro Kishimoto: CORRECT AND EFFICIENT SEARCH ALGORITHMS IN THE PRESENCE OF REPETITIONS, A Doctor Thesis, the University of Alberta, Spring 2005.
14) Ayumu Nagai, Hiroshi Imai: Applicaqtion of df-pn+ to Othello Endgames. In *The 5th Game Programming Workshop (GPW'99)*, pages 16-23, 1999.
15) Brono Bouzy: Mathematical morphology applied to computer go. IJPRAI, vol 17, no.2, March 2003.
16) Brono Bouzy: [coumputer-go] Bouzy/Zobrist algorithms, `http://computer-go.org/pipermail/computer-go/2004-June/000593.html`
17) Thomas Wolf: Investigating tsumego problems with "RisiKo". In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence2*, pages 153-160. Ellis Horwood, 1991.
18) Thomas Wolf: About problems in generalizing a tsumego program to open positions. July 12, 1996.
19) Thomas Wolf: Foward pruning and other heuristic search techniques in tsume go. *Information Sciences*, 122(1): 59-76, 2000.
20) Thomas Wolf, Matthew Pratola: Optimizing GoTools' search heuristics using Genetic Algorithms. ICGA Journal March 2003.
21) GNU Project: *GNU Go*, `http://www.gnu.org/software/gnugo/gnugo.html`
22) Seal Software: アルゴリズム解説, `http://fujitake.dip.jp/sealsoft/thell/algorithm.html\#transpositiontable`