ゲーム情報学 1−11
（1999. 6. 25）

将棋におけるゲーム木探索アルゴリズムの比較

鈴木 豪　乾 伸雄　小谷 善行
東京農工大学大学院工学研究科
{go,nobu,kotani}@fairy.ei.tuat.ac.jp

概要

ゲーム木探索アルゴリズムはαβ探索を中心としてさまざまなものが提案されいる。これらはコンピュータチェスやコンピュータオセロにおいてはその探索効率の比較が行われているが、コンピュータ将棋において比較はほとんど行われていない。本稿ではαβ探索、aspiration探索、PVS探索、negascout探索を実装した将棋プログラムにより、将棋のゲーム木に対し探索効率の比較を行った。その結果、PVS探索とnegascout探索においてはチェスと同様に将棋のゲーム木探索においてもαβ探索よりも効率的であることが確認できた。

A Comparison of Game-Tree Search Algorithm in Shogi

Tsuyoshi SUZUKI　Nobuo INUI　Yoshiyuki KOTANI
Tokyo University of Agriculture and Technology
{go,nobu,kotani}@fairy.ei.tuat.ac.jp

Abstract

There are many game-tree search algorithms based on the alpha-beta search. In this paper, we compared with alpha-beta search, aspiration search, PVS search and negascout search in Shogi-game-tree. We examined these game-tree search algorithms in Shogi program. As the result, we find the effectiveness of PVS-search and negascout search in Shogi.

1 はじめに

将棋のようなゲームをプレイするプログラムは可能手を列挙したゲーム木を探索して次の指し手を決定する。将棋のゲーム木の場合、平均で８０の可能手があり、これがゲーム終了まで１００手以上続くことになる。このような巨大なゲーム木を効率よく探索するためにゲーム木探索アルゴリズムはさまざまなものが提案されている。これら探索アルゴリズムの比較は、ランダムな木、チェスのゲーム木、オセロのゲーム木などに対しては行われているが、将棋のゲーム木を使っての比較はほとんど行われていない。

本稿では代表的なゲーム木探索アルゴリズ

ムである α β探索、aspiration 探索、PVS 探索、negascout 探索を取り上げ、将棋のゲーム木に対してその探索効率の比較を行った。

2 ゲーム木探索アルゴリズム

ここでは実験を行う α β探索、aspiration 探索、PVS 探索、negascout 探索を概説する。以下、ノード p の子ノードを p.1,…,p.b などと書くことにする。

2.1 minimax 探索

ゲーム木探索の基本は、自分は自分にとって一番都合のいい手を指し、相手は自分にとって一番都合の悪い手を指すことからなる。すなわち、自分の手番では評価の一番高い指し手を選択し、相手の手番では評価の一番低い指し手を選択する。図1のゲーム木において、max ノード p では p.1、p.2、p.3 の中から評価の一番高い p.1 の指し手を選択し、min ノード p.1 では p.1.1、p.1.2 の中から評価値の低い p.1.2 の指し手を選択する。

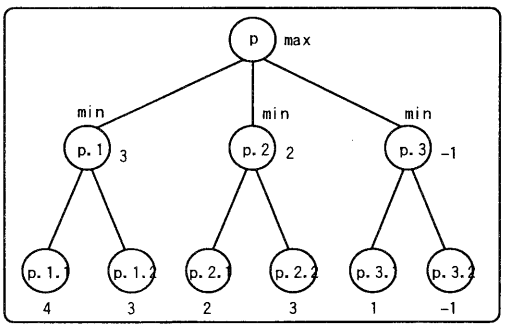

図1　深さ2の木の探索

この手続きをプログラムであらわすと図2の minimax 探索となる。minimax 探索の場合、深さが d で分岐数が b の木の探索には d^b 個のノード数を探索する必要がある。

```
minimax(position,depth)
{
   if( depth <= 0 ) return( evaluate(position) );
   width = generate(position);
   if( width == 0 ) return( evaluate(position) );
   score = -∞;
   for( i = 0 ; i < width ; i++ ){
      value = -minimax(position.i,depth-1);
      if( value > score ) score = value;
   }
   return( score );
}
```

図2　minimax 探索

2.2 α β探索

minimax 探索では探索不要な枝まで探索を行っている。すなわち、図1のゲーム木の場合、min ノード p.2 は p.2.1 を探索した時点で p.1 を超えないことが分かるので p.2.2 は探索する必要がない。このようにして探索不要な枝を探索しないアルゴリズムとして α β探索が提案された。α β探索は、minimax 探索と同じ値を得るにもかかわらず、不要な枝の探索を行わないため効率的である。この探索効率は探索する子ノードの順番に依存する。子ノードの並びが理想的な場合その探索ノード数は、深さが d、分岐数が b のゲーム木に対しては $b^{d/2}$ 程度であることが知られている[4]。α β探索を図3に示す。

```
alphabeta (position,alpha,beta, depth)
{
   if( depth <= 0 ) return( evaluate(position) );
   width = generate(position);
   if( width == 0 ) return( evaluate(position) );
   score = alpha;
   for( i = 0 ; i < width ; i++ ){
      value = - alphabeta(position.i,-beta,-score,depth-1);
      if( value > score ) score = value;
      if( score >= beta ) return( score );
   }
   return( score );
}
```

図3　α β探索

2.3 aspiration 探索

α β探索では α 値と β 値の幅であるウインドウ (α, β)が小さいほど探索ノード数が小さくなる。aspiration 探索ではあらかじめ設定し

たウインドウで探索を行い、その探索が失敗したときには失敗時の情報を使って再探索を行う。aspiration 探索はルートノードのみで行う場合や各ノードで再帰的行うなどなどなと考えられる。aspiration 探索を図4に示す。

```
aspiration(position,error)
{
   score = evaluate(position);
   alpha = score - error;
   beta  = score + error;
   score = alphabeta(position,alpha,beta,depth);
   if( score >= beta )
      score = alphabeta(position,beta,+∞,depth);
   else if( score <= alpha )
      score = alphabeta(position,-∞,v,depth);
   return( score );
}
```

図4　aspiration 探索

2.4 PVS 探索

PVS 探索は Campbell と Marsland により提案された[1]。PVS 探索は pvs と alphabeta という2つの関数からなる。これは最初の子の探索を行った後、その子の評価値を使ってそれ以降の子を幅が1のウインドウで α β 探索を行う。探索が失敗した場合は、そのときの情報を使って再探索を行う。PVS 探索を図5に示す。

```
pvs( position, alpha, beta, depth )
{
   if( depth <= 0 ) return( evaluate(position) );
   width = generate(position);
   if( width == 0 ) return( evaluate(position) );
   score = -pvs(p.1,-beta,-alpha,depth-1);
   for( i = 1 ; i < width ; i++ ){
      value = -alphabeta(position.i,-score-1,-score,depth-1);
      if( value > score ){
         value = -alphabeta(position.i,-∞,-score,depth-1);
   }
   return( score );
}
```

図5　PVS 探索

2.5 negascout 探索

negascout 探索は Reinefeld により提案された[7]。これは PVS 探索と同様に、最初の子を探索した後は、その値を使って幅が1のウインドウで探索を行う。これは、PVS 探索のように2つの手続きを必要としない。negascout 探索を図6に示す。

```
scout (position,alpha,beta,depth)
{
   if( depth <= 0 ) return( evaluate(position) );
   width = generate(position);
   if( width == 0 ) return( evaluate(position) );
   score = -∞; n = beta;
   for( i = 0 ; i < width ; i++ ){
      value = -scout(position.i,-n,-max(alpha,score),depth-1);
      if( value > score ){
         if( n == beta || depth < 2 ) score = value;
         else score = -scout(p.i,-beta,-value,depth - 1);
      }
      if( score >= beta ) return( score );
      n = max(alpha,score) + 1;
   }
   return( score );
}
```

図6　negascout 探索

3 ゲーム木探索のテクニック

ここでは、ゲーム木探索に使われているいくつかのテクニックについて述べる。

3.1 指し手の順序づけ

α β 探索では指し手が理想的な順番で並んでいる場合には最も効率よく探索が行われる。たとえば、図1においてノード p.2.1 と p.2.2 が逆に並んでいた場合、p.2.2 以下の部分木を探索しなければならない。しかしノード p.2.1 と p.2.2 がこの順序で探索されたときには p.2.2 は探索不要であり効率の良い探索となる。指し手の順序つけテクニックは、このような理想に近い順番で子ノードの探索を行うために、探索を始める前に指し手を仮評価しておき、その評価の高い順に探索を行うものである。

3.2 反復深化

反復深化は探索をはじめから深く行うのではなく、浅いところから順次深くしていく。反復深化の利点は探索を途中で止めても悪い影響が少ないことである。これにより時間的に効率よく探索が行うことができる。また、浅い深さで探索の最善応手の指し手を使って次の深

さの探索を開始できるので、指し手の順序付けの意味からもα β探索の効率も向上に貢献する。反復深化はルートノードのみで行う場合や、各ノードで再帰的に行うなどが考えられる。

4 比較実験

ここでは、2節にあげたゲーム木探索アルゴリズムを使って、将棋ゲーム木の探索を行った比較実験について述べる。

4.1 実験内容

全幅探索の将棋プログラムを使って、将棋のゲーム木を探索し、その探索ノード数の比較を行った。実験は

(1)指し手の順序付けを行わない場合のα β探索、aspiration 探索、PVS 探索、negascout 探索の比較

(2)指し手の順序付けを行う場合のα β探索、aspiration 探索、PVS 探索、negascout 探索の比較

(3)反復深化を行わないα β探索と、ルートノードでのみ反復深化を行うα β探索行うプログラムの比較

からなる。それぞれのアルゴリズムを実装したプログラムに400～500局面を与えて、探索ノード数の比較を行った。実験結果は全てα β探索で探索されたノード数を1として他のアルゴリズムで探索されたノード数との比をとっている。(1)(2)では探索する木は全く同一であるが、(3)では探索しているノードの子ノード中に評価値が同じものがある場合には、探索する指し手の順番により探索を行う木は異なることもある。また、実験に使用した aspiration 探索では図2中の評価の誤差 error を1としている。

4.2 指し手の順序付けを行わない場合

指し手を仮評価順に順序付けを行わない全幅探索プログラムを使って、α β探索、aspiration 探索、PVS 探索、negascout 探索を行った。同一の深さで探索しているゲーム木は同一である。深さ3，4，5のときに各探索アルゴリズムが探索したノード数とα β探索が探索したノード数の比を図7に示す。ここではα β探索により探索されたノード数を1としている。

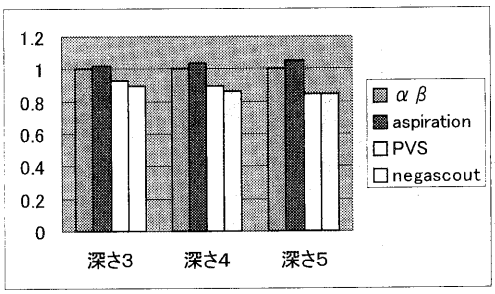

図7 順序付けを行わない場合の探索ノード数

探索の深さが3のときのゲーム木の平均分岐数による探索ノード数の比較を図8に示す。

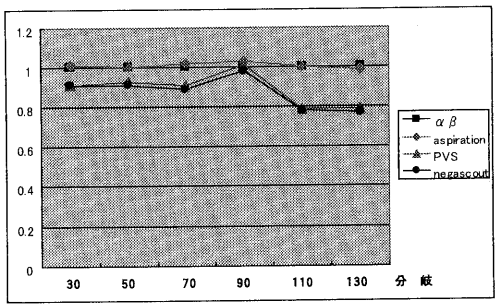

図8 平均分岐数による探索ノード数

4.3 指し手の順序付けを行う場合

探索を行う前に指し手の仮評価を行い、指し手の順序付けを行う全幅探索プログラムを使って、α β探索、aspiration 探索、PVS 探索、negascout 探索を行った。指し手の仮評価は駒の損得・絶対位置・自王敵王との相対位置を考慮している。また、末端での静的関数は駒の損得・絶対位置・自王敵王との相対位置、飛角の自由度などを評価要素としている。実験結果を図9に示す。同一の深さで探索しているゲーム木は同一である。

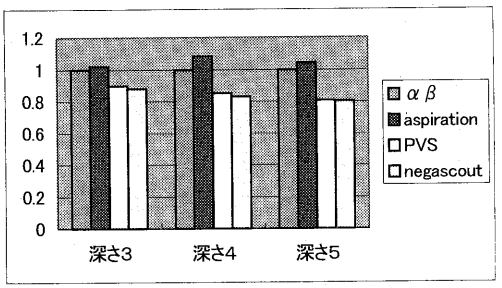

図 9　順序付けを行う場合の探索ノード数

深さが 4 のときにゲーム木の平均分岐数による探索ノード数の比較を図 10 に示す。

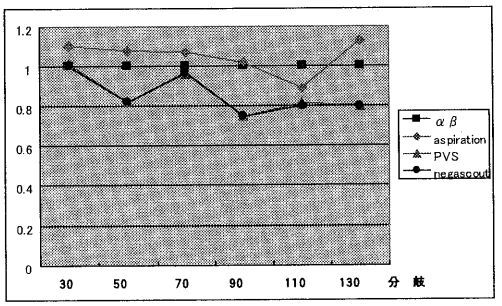

図 10　平均分岐数による探索ノード数

αβ 探索を基本とする探索アルゴリズムは指し手の順序付けにより、その効率が変化する。このため指し手の順序付けを駒得のみで行い、評価関数も駒得のみで行う別プログラムで αβ 探索、PVS 探索、negascout 探索の比較を行った。実験結果を図 11 に示す。

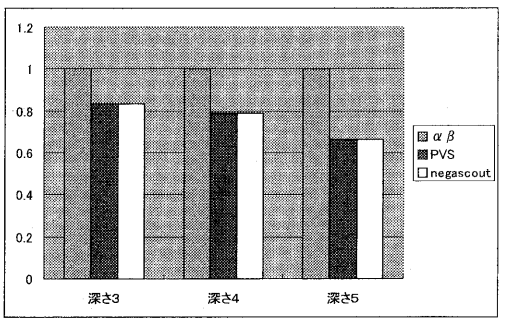

図 11 差し手の順序付けを変更した比較

4.4 反復深化を行う場合

αβ 探索において反復深化をルート局面で行うプログラムと、反復深化を行わない αβ 探索の探索ノード数の比較を行った。反復は深さ 1，2，3…と順番に行い、浅い深さでの評価値を利用して次の探索の深さでの指しての順序付けを行っている。実験結果を図 12 に示す。

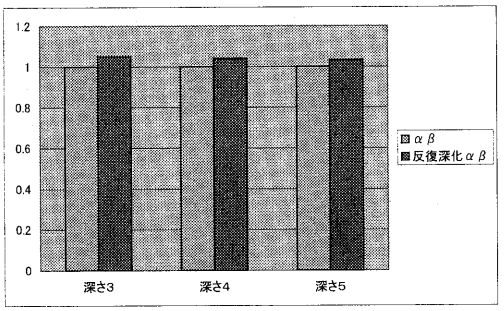

図 12　反復深化 αβ との比較

4 考察

aspiration 探索は、指し手の順序付けを行わない場合のと順序づけを行った場合のいずれでも探索ノード数は αβ 探索よりも多くなった。また、深さが 3，4，5 の全ての場合についても同じ結果となった。これは最初の仮のウインドウを設定するときの評価値が適切でなかったことと、そうでもあるにも関わらず評価値の誤差の幅を狭く設定したことによる。局面によっては αβ 探索と比較して 4 割以上の探索ノード数の減少がみられるものもあった。しかし、ほとんどの場合は探索開始直後に失敗し、新たなウインドウで探索を行う結果となった。このため本実験では aspiration 探索は全体として αβ 探索による探索ノード数よりも若干多くなっている。また、指し手の順序づけを行う場合と行わない場合の αβ 探索と比較しての探索効率はいずれも違いがみられなかった。仮の評価値とウインドウの幅が適切に選択されれば aspiration 探索も有効であると考えられる。

PVS 探索と negascout 探索は指し手の順序づけを行う場合と行わない場合のいずれでも

$\alpha\beta$探索よりも探索ノード数は少なくなっている。これらは深さが深くなるごとに少しずつではあるがより少なくなる傾向にある。局面ごとに探索ノード数の比較を行うと、PVS探索、negascout 探索のいずれもほとんどの局面で$\alpha\beta$探索よりも探索ノード数が少なくなっている。$\alpha\beta$探索よりも探索ノード数が増えた局面はわずかではあるが、それらの局面においてもその探索ノード数は$\alpha\beta$探索でのノード数を大きく超えることはなかった。PVS 探索、negascout 探索は最初の探索に失敗しても幅が1のウインドウによる探索の効率の方が勝り、全体的には問題にならないことを示している。また、探索失敗回数は全体の探索ノード数からみると無視できる。PVS探索、negascout探索では、どの局面においてもほとんど探索ノード数の差はみられなかった。PVS 探索とnegascout探索において、指し手の順序付けを行う場合は、順序づけを行わない場合と比較して探索ノード数の減少がみられた。これは有力な指し手が先に探索されることにより、その探索効率が向上したためと考えられる。指し手の仮評価と評価関数を簡単にした実験では、上記の複雑な仮評価と評価関数を実装したプログラムよりも探索効率が良くなっている。これは複雑な仮評価を行っているプログラムにおいては、始めから小さな木を探索しているため大きな差がでなかったと考えられる。探索効率は局面により大きく異なるが、分岐数が増えるにしたがい向上する傾向が見られた。

反復深化を使った実験では、反復深化を行った方が直接$\alpha\beta$探索を使ったものよりも若干ではあるが探索ノード数が増えた。これは全幅探索を行っていることと、指し手の順序づけにより探索される木がはじめからある程度小さなものであるためと考えられる。本実験では反復時に浅いところでの探索の評価を使って指し手を絞ることを行っていないた。この絞込みを組み込んだ場合には(その絞込みの幅にも依存するが)反復深化は結果的に$\alpha\beta$探索よりも効率的になると考えられる。

5 おわりに

本稿では$\alpha\beta$探索、aspiration 探索、PVS探索、negascout探索を実装した将棋プログラムにより、将棋のゲーム木に対し探索効率の比較を行った。その結果、PVS探索、negascout探索においては$\alpha\beta$探索よりも効率的であることが確認できた。ここで実験を行った探索効率はプログラムの実装により結果が大きく異なる。本実験で使用したプログラムは、指し手の順序付けテクニックなど改良の余地があり、これらの改善を行えばよりよい結果が得られると期待できる。

6 参考文献

[1] M.S. Campbell and T.A. Marsland, Parallel Search of Strongly Ordered Game Tree, Computing Surveys 14(4), (1982), 533-551

[2] M.S. Campbell and T.A. Marsland, A Comparison of Minimax Tree Search Algorithms, Artificial Intelligence 20(4), (1983), 347-367

[3] A.Junghanns, Are There Practical Alternatives to Alpha-beta? ICCA Journal 21(1), (1998), 14-32.

[4] D.E. Knuth and R.W. Moore, An Analysis of Alpha-beta Pruning. Artificial Intelligence 6(4), (1975), 293-326.

[5] T.A. Marsland, A Review of Game-Tree Pruning. ICCA Journal 9(1), (1986), 3-19.

[6] J. Pearl, Asymptotic Properties of Minimax Trees and Game Searching Procedures, Artificial Intelligence 14(2), (1980), 113-138.

[7] A. Reinefeld, An Improement of the Scout Tree-Search Algorithm, ICCA Journal 6(4), (1983), 4-14.