記号処理 43－3
（1987. 11. 13）

候補手記述言語‘拡張Gopal’を用いた囲碁プログラムの試作

大西 諭　　井上 克郎　　鳥居 宏次
（大阪大学基礎工学部）

囲碁のパターン知識を記述するための言語Gopalを、定石などの一連の手順や手順の分岐、手抜きなどを記述しやすいように拡張した‘拡張Gopal’を定義し、実際にその実行系を実現した。この拡張Gopalの実行系を用いて、囲碁プログラムを作成中である。一方、この囲碁プログラムが行う探索の方法として、探索の深さが一定でなく、真の値の存在する範囲の上下限をノードの値として使用するB*サーチアルゴリズムを採用し、現在実現中である。拡張Gopal実行系が出力する候補手をB*サーチアルゴリズムで探索することにより、探索の回数を少なくなり、深い先読みができると考えている。また、探索の際に使用する評価関数についても試作している。

An Extension of Go Pattern Language (Gopal) and an Experimental Go Program Using Extended Gopal

Satoshi Onishi, Katsuro Inoue, and Koji Torii

Department of Information and Computer Sciences,
Faculty of Engineering Science, Osaka University
1-1 Machikaneyama, Toyonaka, Osaka 560, Japan

Gopal is a language for describing pattern knowledge of Go. We have extended Gopal so that we can easily describe a sequence of moves such as Jyoseki. Using extend Gopal, branches of Jyoseki and Tenuki are also described easily. We have actual implemented extended Gopal processor which outputs candidate moves for a pattern of the Go board. We are developing an experimental Go program which includes the extended Gopal processor. As a search algorithm for the next move, B* search has been implemented. The candidate moves generated by the extended Gopal processor are only investigated precisely by B* search. We are developing evaluation function used with then B* search.

1．まえがき

一般に、チェスやオセロなどのゲームプログラムでは、すべての着手可能な手について探索を行い、主に駒得などで評価する方法で、次の一手を選んでいる。しかし、囲碁の場合、ほぼすべての空点が着手可能であるため、すべての着手可能な手について探索を行うことは、時間がかかりすぎてその実現が不可能に近い。また、探索の時には各状態での評価値を求めるが、その中間評価の基準も一般の駒取りゲームのように単純ではなく、「地」「厚み」などの各種の直感的、知識的な要因が絡まっていることが多い。このような点が囲碁プログラムの作成を難しくする要因になっている。

現在、我々は囲碁プログラムを作成している。そこでは、知識としてパターンを使用して候補手を選択する方法を用いている。パターン知識とは、ある囲碁の盤面の一部分を取り出し、それを一般的な条件などで表わした盤面状況の知識である。現在の局面をパターン知識で表わし、各パターンに対する次の手を候補手として記述すれば、次の手の選択が可能である。このパターン知識と候補手を記述する言語としてGopal(Go Pattern Language)という言語仕様・処理形がすでに作成されている（今後、これを旧Gopalと呼ぶ）。今回は、このGopalをさらに仕様拡張し、定石や手筋といった知識を簡単に記述できるようにした。

また、現在、探索の実現について検討中である。先ほどの理由からすべての着手可能な手についての探索は不可能に近いので、パターン知識を活用した探索を行うことにした。これにより、人間と同様に候補手を絞った狭く深い探索を行うようにする。その方法として、パターン知識から得られた候補手のみを先読みの対象として絞り込むことを考えた。また、探索アルゴリズムとしてB*探索アルゴリズムをインプリメントした。このアルゴリズムは、実際にチェスに適用して$\alpha-\beta$法より探索回数が減少することが確かめられており[1]、我々はこれは囲碁においても有効な探索方法ではないかと考えている。また、評価関数についても試作を行った。

現在、我々のプログラムは主にGopalによって記述されたパターン知識部と、それを実行するための部分の二つによって構成されている。Gopalプログラムはトランスレータによって Pascalプログラムに変換され、他の部分と結合される。実行部には、メインルーチンの他にデータ・メモリ管理部、Gopalで使用する各種関数のライブラリなどがある。現在、実行部に探索実行部も付加中である。

2．パターン記述言語Gopalの拡張

2．1　囲碁におけるパターン

各ゲームにはそれぞれのルールがあり、一般に先読みを行うことで、ゲームに勝つ条件を達成しようとする。しかし、特に囲碁では深い先読みが必要な部分が多いため（石の死活など）、ルールと探索だけではよい対局プログラムが実現できない。そのため、戦略的な知識を用いることが必要になる。ここでは、プログラム中に知識の一部としてパターンを使用した。

パターン知識は、現在の盤面の一部分の状態を表わしたものである。この盤面とそれに対する候補手を指定することにより、各状態での適当な手を選択できるようになる。

2．2　旧Gopal

旧Gopalは電総研で開発された。旧Gopalは次のような特徴を持つ[2]。

1)非手続き的であること
2)関連する種々のパターンを系統的に記述できること
3)対象形の扱いは処理系が行い、基本系の記述のみで良いこと
4)適用時期、適用場所の指定ができること
5)その状況では必然的に生ずるであろう着手系列に対して、それらの予測可能な先読みを行えること
6)属性値の参照等のための豊富な組み込み関数が用意されること
7)Gopalプログラムは手続き的な既存言語へコンパイルされること

Gopalプログラムは次のような構成になっている。

- プログラム頭部：プログラム名、引数の宣言
- プログラムの適用時期宣言部：プログラムの適用時期（序盤、中盤など）の宣言
- 定数宣言部
- 変数宣言部
- 関数宣言部：利用者定義関数の宣言
- パターン記述部：盤面のパターンとそれに対する候補手を記述する部分

この中で、パターン記述部は、

- パターン頭部：パターン名の宣言
- パターンの適用時期宣言部：パターンの適用時期（序盤、中盤など）の宣言
- パターンの適用場所宣言部：パターンの適用場所（隅、辺など）と対象形を考慮するかどうかの宣言
- 定数宣言部

・変数宣言部
・関数宣言部：利用者定義関数の宣言
・pickup部：対象物の特定
・碁盤の状態記述部：碁盤の状態（パターン）を記述する部分
・候補手記述部：上のパターンに対する候補手を（次の一手のみ）記述する部分

からなる。図１、図２は隅の序盤のパターン記述の例で、図１は空き隅への先着、図２は星へのカカリを示している。

「pickup部」では、パターン中に現われる１つの対象物を特定するもので、たとえば、「あるパターンにマッチするときのその中の一つの石」、「ダメが１の石の連なり」とかが記述できる。これにより、対象となる場所が限定されるので、実行が高速化される。図１、図２ではpickup部は使用されていない。(pickup none;)

「碁盤の状態記述部」では、どのような状態のときにこのパターンを取り上げればよいかを示す部分で、たとえば、「隅があいている状態」とか、「石を取ることができる状態」とかを記述できる。もちろん、一つパターンを記述しておけば、対象形なども自動的に考慮されるようになっている。この中では、限定された先読み（打つ石を指定しておいてその盤面において状況をみる）も行うことができる。また、各種の関数も使用でき、その中で、「シチョウで逃げることができるか」といった先読みを行う関数も用意されている。図１では、隅の部分が空であるという状態を emparea(pa1,4,4) で記述している。図２では、星に敵がいるという状態を state(pd4)=enemy で、その他が空であるという状態を emplarea(pa1,7,4,pd4) で示している。

「候補手記述部」では、上での関係式を満たす局所的局面に対して、候補手をその意図や重みとともに記述する。意図は、その石を打つ目的で、たとえば「戦い」や「逃げ」などがある。重みは、その石を打つ価値を数字で表わしたものである。意図や重みは候補手の選択のときに使用する（ただし、意図は現在使用していない）。さらに、この部分にはサブパターンとして追加する条件を記述でき、類似の条件式を持ついくつかのパターンを系統的に記述できるようになっている。なお、各パターンから複数の候補手があがった場合には、重みを基準に選び、重みが等しいときは乱数で選ばれる。図１では(c,3)(三三)、(d,3)(c,4)(小目)、(d,4)(星)を、その意図が隅の序盤、重みを８０点で候補手とする。図２では、(f,3)(f,4)の点を意図が隅序盤、重みを７２点で候補手とする。なお、pd4,pf3などは定数宣言部であらかじめ宣言された、場所を示す定数である。

```
pattern akisumisen;
 applied_place                {適用する場所}
   corner non_symmetric;      {隅で，対称形は考慮しない}
 pickup                       {注目する場所はなし}
   none;
 condition
   emparea(pa1,4,4);          {(a,1)から4×4の範囲が空である}
 candidate
   with (sumijoban,80.0) pc3, pd3, pc4, pd4;
   {(c,3)(d,3)(c,4)(d,4)の点を，意図が序盤，点数80で候補とする}
end_pattern;
```

図１．旧Gopalでの空き隅先着の記述

```
patter hkakari;
 applied_place
   corner;                    {適用する場所は隅}
 pickup
   none;
 condition
     state(pd4) = enemy       {(d,4)の点が敵である}
   & emplarea(pa1,7,4,pd4);
          {かつ，(a,1)から7×4の範囲が(d,4)を除いて空である}
 candidate
   with (sumijoban,72.0) pf3,pf4;
          {(f,3)(f,4)の点を，意図が序盤，点数72で候補とする}
end_pattern;
```

図２．旧Gopalでの星へのかかりの記述

２．３　拡張Gopalの仕様

旧Gopalの欠点の一つは、候補手が次の一手しか記述できないことであった。このため、定石を記述しようとしたとき、一手進むごとの各盤面のパターンを別々に記述する必要があった。また、相手の打った手に対する応対の手順についても同じである（たとえば、のぞきに対するつなぎなど）。また、このような記述は手間がかかるだけでなく、実行スピードにも大きな影響を与える（パターンの数に比例して遅くなるため）。

このような点を改善するため、新たに拡張Gopalを設計した。今回の改善点として次のようなことを考慮した。

1)手順の直接記述ができること
2)手順に対する中断や再開、手抜きに対する手なども記述できるようにすること
3)手順の分岐なども記述できること
4)できるだけ実行効率を下げないこと
5)今までの旧Gopalで記述したパターンもなるべく手直しせずに使用できること

これらの点を満たし、かつ実現も簡単にするため、主に候補手記述部を拡張し、手順の記述ができるようにした。

図３が拡張Gopalプログラムの候補手記述部の構造の概略である。また、図４の隅の定石（星－コゲイマガカリ－二間高バサミ）の定石を拡張Gopalを用いて記述したのが図５である。次に、簡単に構造を説明する。

・候補手記述部

図３の1)～20),9)～12),13)～16)

candidateで始まりend_candidateで終わる。

中に手順列挙部を書くことができる。手順列挙部は複数個書くことができ、複数ある時にはその数だけ手順があることを意味する。図５では手順列挙部は一つ

```
1)  candidate
2)    tarea(点１:点２;,,,);
3)    with(意図,重み) 候補手;
4)      next(意図,重み) 候補手;     tenuki(意図,重み) 候補手;
                .
5)              .
6)      branch
7)        condition
                .
8)              .
9)        candidate
10)         with(意図,重み) 候補手;
                .
11)             .
12)       end_candidate;
13)       candidate
14)         with(意図,重み) 候補手;
                .
15)             .
16)       end_candidate;
17)     end_branch;
18)   with(意図,重み) 候補手;
                .
19)             .
20) end_candidate;
```

図３．拡張Gopal候補手記述部の構造の概略

```
pattern hosi_kogeima_nikentakabasami;
  applied_place
    corner;                     {適用する場所は隅}
  pickup
    corner(0);                  {隅から6×6の範囲に石は0個}
  condition
    emparea(pa1,10,10);         {(a,1)から10×10の範囲が空である}
  candidate
    tarea(pa1:pi9);
       {手抜きの範囲を(a,1)と(i,9)で囲まれる長方形の外と定める}
    with(sumijoban,20.0) pd4;       {最初の手は(d,4)}
     next(sumijoban,20.0) pc6;      {二番目の手は(c,6)}
     next(sumijoban,20.0) pd9;      {三番目の手は(d,9)}
     next(nigedashi,20.0) pe6;      {四番目の手は(e,6)}
            tenuki(hokaku,20.0) pd6;
                                {四番目の手が手抜きされたら(d,6)}
     next(kakucho,20.0) pf4;        {五番目の手は(f,4)}
            tenuki(shinryaku,20.0) pf4;
                                {五番目の手が手抜きされたら(f,4)}
     next(shinryaku,20.0) pb4;      {六番目の手は(b,4)}
            tenuki(kakucho,20.0) pb4;
                                {六番目の手が手抜きされたら(b,4)}
     next(kakucho,20.0) pc3;        {七番目の手は(c,3)}
            tenuki(shinryaku,20.0) pc3;
                                {七番目の手が手抜きされたら(c,3)}
     next(kakucho,20.0) pb9;        {八番目の手は(b,9)}
            tenuki(shinryaku,20.0) pb8;
                                {八番目の手が手抜きされたら(b,8)}
     next(kakucho,20.0) pj3;        {九番目の手は(j,3)}
  end_candidate;
```

図５．拡張Gopalでの星・小ゲイマ・二間高バサミの定石の記述

・手順列挙部

図３の2)～17),18)～19)

手順列挙部は手順を記述する部分

まず、with部で最初の手を記述し、その後の手をnext部またはbranch部で記述する。

・with部、next部、

with文、next文を記述する部分。その前に手抜きの範囲を指定するtarea文、後ろに手抜きの場合の候補手を指定するtenuki文を記述できる。

・branch部

図３の6)～17)、図５にはない

branchで始まり、end_branchで終わる。

中に盤面の条件を記述するcondition部、候補手を記述する候補手記述部を書く。

以上の仕様により手順の記述が容易にできるようになる。さらに、序盤の隅の定石のマッチングを早くするために、pickup部に新たな文を加え、隅の部分の石の数（６×６の部分）を記述するcorner(n)を仕様に加えた。

図４に対する図５の記述では、直接手順が記述できていて簡潔である。もしこれを旧Gopalで記述すれば（図

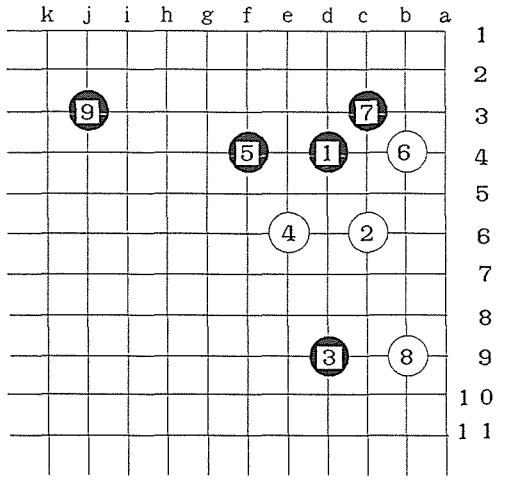

図４．星－コゲイマ－二間高バサミの定石

```
pattern hosi;
  applied_place
    corner non_symmetric;       {隅で対象形は考慮しない}
  pickup
    none;
  condition
    emparea(pa1,10,10);         {(a,1)から10×10の範囲が空である}
  candidate
    with (sumijoban,20.0) pd4;
end_pattern;

pattern hosi_kogeima;
  applied_place
    corner;                     {隅}
  pickup
    none;
  condition
    state(pd4)=enemy & emplarea(pa1,10,10,pd4);
                                {(d,4)が敵で、10×10のその他の部分が空}
  candidate
    with (sumijoban,20.0) pc6;
end_pattern;

pattern hosi_kogeima_nikentakabasami;
  applied_place
    corner;                     {隅}
  pickup
    none;
  condition
    state(pd4)=friend & state(pc6)=emeny
    & states(difference(range(pa1,10,10),pset1(pd4,pc6)))=empty;
                {(d,4)が味方、(c,6)が敵で、10×10のその他の部分が空}
  candidate
    with (sumijoban,20.0) pd9;
end_candidate;
        .
        .
        .                       （全部で１４個のパターン）
        .
```

図６．旧Gopalでの星・小ゲイマ・二間高バサミの定石の記述

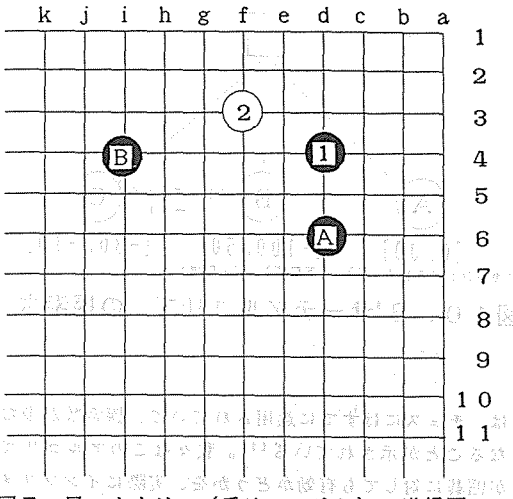

図７．星ーカカリー（受け、ハサミ）の進行図
（三手目はＡかＢのどちらか）

```
pattern akisumisen;
  applied_place
    corner;                          {適用する範囲は隅}
  pickup
    corner(0);                       {隅から6×6の範囲に石がない所に注目}
  condition
    emparea(pa1,8,8);                {(a,1)から8×8の範囲が空である}
  candidate
    tarea(pa1:ph8);                  {手抜きの範囲は(a,1)-(h,8)の外}
    with (sumijoban,20,0) pd4;       {最初の手は(d,4)}
      next (sumijoban,20,0) pf3;     {二番めの手は(f,3)}
      branch
        candidate
          with (sumijoban,20,0) pd6;   {三番目の手は(d,6)
          with (sumijoban,20,0) pi4;                と(i,4)}
        end_candidate;
      end_branch;
  end_candidate;
end_pattern;
```

図８．拡張Gopalでの星ーカカリー（受け、ハサミ）の記述

6）、図５の各with,next,tenuki文に対して一つずつのパターンが必要となり、全部で１４個のパターンを記述しなければならない。また、この場合、手が進むにしたがって盤面の状態の記述が複雑になる。

２．４　拡張Gopalの実現

Gopalはもともと現在の盤面の状態とそれに対する次の一手を記述するものであり、手順といったものを記述するのには適していない。これに手順を付け加えたので、実現には工夫を必要とする。それは、同時に複数のパターンの中から候補手を選ぶという旧Gopalに、以前から進められている手順の中から候補手を選ぶという方法が加わったためである。旧Gopalでは、一手ごとに、Gopalプログラムからトランスレートされた Pascalプログラムを実行し、もしパターンにマッチすれば候補手を列挙するという方法をとっていた。今回の拡張Gopalには手順が加わったので、単純に候補手を列挙するといった方法はできなくなる。そこで、ここでは候補手の手順などの情報を木を使って記憶させる方法をとることにした。

図７が星ーカカリー（受け、ハサミ）の手順で、図８が図７を拡張Gopalで記述したものである。これに対応する候補手順木の例を図９に示す。次の手はnextpntを、分岐する手はwithpntを使って木を構成している。もしGopalプログラムのパターンにマッチすれば、プログラムはこの木を生成し、候補手の木として記憶しておく。そして、この木の群の中から次の手を選び出すようにする。その手の取り出し方は、

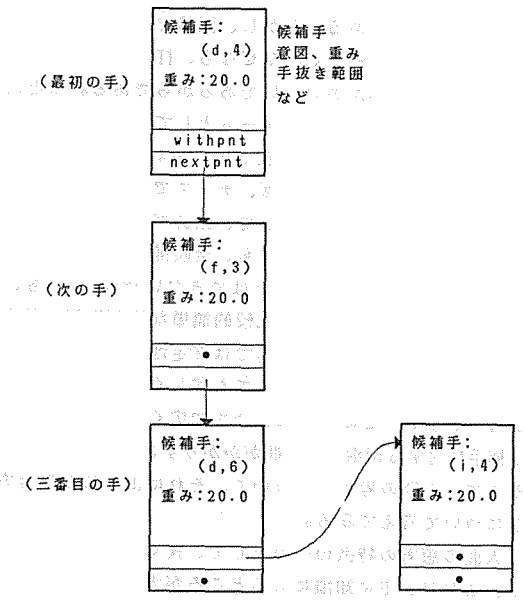

図９．候補木の構造例（図８）

①ノードの色が現在打とうとしている手と同じ場合は、手順進行の途中と考え、その候補手を取り出す。
②色が異なる場合は、前の手が手抜きされたものと考え、手抜きの手を取り出す。
といった方法をとる。次に、一手進むごとにこの木を一つ先に進める必要がある。すなわち、
①実際に打たれた手が記述してある木は、その手順が進められたと考え、一番上を取り除き、一手進める。
②その木にとって手抜きの範囲にある場合は、手抜きされたものと考え、そのまま。
③手抜きの範囲でない場合は、その手順が中止されたものと考え、その木を全部取り除く。

なお、ここで注意することは、手抜きをした場合はそのパターンにより生成された候補手順木は残っているということである。この場合には、次の自分の手番のときに同じパターンにマッチングして再び同じ木が生成される可能性がある。これは別に害はないものの、空間や時間の無駄である。そこで、特に木が大きくなる可能性のある隅の定石に対してはそのパターンについて木が生成されたかどうかを記憶しておいて同じものができないようにした。

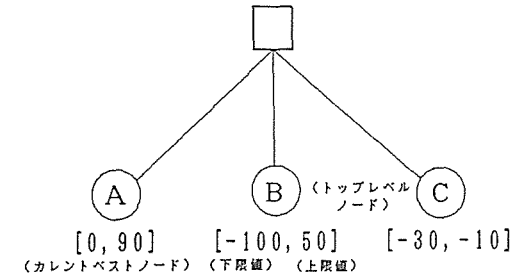

図１０．B*サーチアルゴリズムの探索木

３．探索の導入

３．１　囲碁における探索

囲碁に限らず、ゲームにおいて先読みは非常に重要な問題解決の手段である。しかし、囲碁では一般の探索を行うことは非常に難しい。なぜなら、打着可能な手の数が多く、評価基準があいまいであるからである。たとえば、チェスはコンピュータゲームとして非常によく研究され、強いプログラムも存在している。たとえば、100万回のサーチを行うとすれば、チェスでは７～８手先までサーチできる。それに対して、囲碁プログラムでこれと同じ回数のサーチを行っても、全候補手探索を行えば３手ほどしか先読みすることはできない[3]。しかも、チェスでは駒の損得などの比較的簡単な値が評価の基準として使えるのに比べ、囲碁では確定地の大きさや石の強弱、厚みなど、人間にとっても難しく経験を必要とするものが多い。このように、今まで広く使われてきた全候補手に対する探索では時間がかかりすぎて問題が多い。そこで、人間の考え方について、それによく似た探索方法について考えてみる。

人間の思考の特徴の一つとして、狭く深い先読みがある。これは、主に知識によるところが大きい。今までの豊富な知識により先読みが必要な部分と先読みをする必要がない部分（悪手）、先読みをしなくても結果がわかっている部分とに分けることができるからである。その場合、先読みが必要な手は多くても５～１０程度に限られることが多く、一連の手順の中ではさらにもっと少なくなる。

そこで、我々が今回取り上げるのは、パターン知識と探索を結合した方法である。つまり、Gopalで記述されたパターンを積極的に活用して探索を行う候補手の絞りこみを行う。Gopalであげられた候補手のみに探索を絞り込む方法である。現在、Gopalでは一回に５～２０程度の候補手を出している。パターンはまだ無駄なものや不足しているものがある。しかし、５～２０程度の数であればチェス等のゲームと比較して同程度の数であり、探索を行える数ではあろう。また、「深い先読み」の実現方法として、B*サーチアルゴリズムを採用した。これは、チェスにはすでに適用されていて、探索数が少なくなることが示されている[1]。我々はこのアルゴリズムが囲碁に対しても有効かどうかを、実際にインプリメントして確かめる予定である。

３．２　探索アルゴリズムB*の探索方法

各ゲームにおいて、各種のサーチアルゴリズムが使用されている。その中で、我々はB*サーチアルゴリズムを採用した。

B*サーチアルゴリズムの探索方法は次の通り

①最初に、現在の状態から一手先のノードを作り（これをトップレベルノードという）、これらの各ノードについて評価する。その時、各ノードの値の上下限を求める。（図１０）

②次に、トップレベルノードの中から上限値が最大のノードを求める。このノードをカレントベストノードという。（図１０ではノードA）

③カレントベストノードの下限値が、他のどのトップレベルノードの上限値よりも大きければ、探索が終了する。その時、最良の手はベストノードによって表わされる手である。

④探索の戦略を選ぶ。カレントベストノードの下限値をあげる戦略をProvebest、その他のノードの上限値を下げる戦略をDisproverestという。探索の終了する可能性の高い方の探索方法を選ぶ。

⑤探索を行う。Provebestの場合は、カレントベストノードの下を探索、Disproverestの場合はその他のノードの下を探索する。探索の終了する確率の高いノードを選んでノードの葉まで行き、その下に新たにノードを加える。そして、各ノードの新しい上下限値を計算する。

⑥②へ行き、終了するまで繰り返す。

これが基本的なB*サーチアルゴリズムの探索方法である。なお、B*には、評価値に確率分布を使う方法など各種のバリエーションが存在する[1]。

B*サーチアルゴリズムは一回ごとに探索するノードを決定しているので、それぞれの部分が異なる深さの探索になるし、探索の回数も決まっていない。B*は同じような値を持った部分が深く探索されるので、人間の思考とかなり似通った部分があると考えている。なお、囲碁ではどの手が一番とも決められない状態が多く存在し、その場合はサーチが終了しないので、時間または探索の回数によって強制終了させる必要がある。

3．3　B*サーチアルゴリズムの特徴

B*サーチアルゴリズムはチェスにおいて他のアルゴリズムより探索回数が少ないことが報告されている[1]。我々は以下の理由により、B*サーチアルゴリズムは囲碁においても有効であると考えた。ここでは、B*の特徴について他の方法と比較して述べる。

・探索の順番

α－β法はdepth-firstの順番で、ノードの値に関係なく最初から決まった順番で探索を行う。一方、B*サーチでは、ノードの値により次に探索するノードを決める。最近の効率がいいといわれるアルゴリズム（SSS*など）は順番をダイナミックに決定している[4]。この方が効率がよいことが期待できる。これにより、必要な部分のみの深い探索が実現できる。

・探索の目的（終了条件）

B*サーチでは次の一手の中から最良の手を見つけることが目標であるが、B*以外ほとんどの探索方法は、探索木の葉の値から探索木の値を求め、それにより葉までの最適手を見つけることを目的にしている。ゲームでの探索の目的は、一番よい次の一手を探すことである。この点で、B*は他の探索よりゲームに適していると考えられる。

・使用する評価関数

B*では、真の値の存在する範囲の上下限を用いるが、B*以外のほとんどの探索では、評価関数で求められた最もふさわしいと思われる一つの値を用いる。一般に、常に一つの適切な評価値を得るのは困難と思われる。従って、誤まった結果が求められる場合もある。しかし、B*では上下限という範囲のある値を用いるので、誤る可能性は少ない。さらに、囲碁では盤面評価があいまいであるという点がB*の評価関数の上下限という概念にあっていると考えている。

以上のような理由により我々はB*サーチアルゴリズムを選んだ。

3．4　評価関数

以前に述べたように、囲碁の盤面評価は難しい点が多い。人間同士の一般の対局の時、、「確定地」、「壁、厚み」、「石の強弱」、「アジ」などといったものが存在する。確定地の場合はまだコンピュータで計算しやすい部分であるが（それでも、どれが確定しているかどうかは高度な知識が要求される）、さらに他の部分は人間でも難しく、それが何目の利益になるのかはプロでも難しい場合が多い。

しかし、探索には評価関数が不可欠であり、取りあえず作成する必要がある。ここでは、B*サーチアルゴリズムの評価関数が評価値の上下限を使用することを利用して、評価関数のあいまいさの部分をその幅を使用して表現することにした。評価の方法は次のように定めた。

①盤面のポテンシャルを計算する。
②ポテンシャル値からそれぞれの石のつながりを計算する。
③ダメの数、地の大きさ、回りのスペースを計算し、石の強さを計算する。
④石の強さから、取られている石を見つける。
⑤石のつながりや強さ、地の広さなどを再計算する。
⑥これらの値を目数に換算して評価値を求める。

これが現在実現している評価方法である。ポテンシャルというのは囲碁プログラムにおいてはよく使用される方法で、その部分が白の勢力か黒の勢力かの度合いを数字で表わすものである。この値により、どの値以上ならつながっている部分かを計算する。また、地についても同様である。生死についても石の強さの値で決定する。この値のとり方を自分に有利にする（自分は小さい値で地になったり活きたりし、相手は大きくする）と、評価値の上限値が求められる。下限値についてはB*の文献[1]にもいくつかの方法があげられているが（上限の方は、チェスで行っているため最大の駒得で行っている）、簡単な方法、つまり、一つ前の手のノードでの評価の最大値から求めるという方法をとった。

以上のような評価関数を実際に作成した。これは、B*にあうように作成したものである。しかし、値のとり方などを調節して改良すれば他の探索方法でも使用可能である。現在の評価関数はまだ試作の段階であり、改善する要素が多い。それは、

①実行スピードが遅い
②強さの判定が単純すぎる

などがある。①の方は、現在のプログラムがポテンシャルの計算にかなり時間を費やしており、ここを他のアルゴリズムにすればよくなるものと考えている。また、②の方は、目の判定や回りの石との強さの比較などが必要であろうと考えている。

4．あとがき

囲碁プログラムに手順の知識を取り入れる方法と探索を実現するための一案について述べた。拡張Gopalは現在すでに実現済みで、実際に定石などの手順を入力・実行可能である。探索については現在インプリメント中である。ただし、現在の状況では、人間との対戦可能な速度での実行は行えない。その原因については、

①探索を、Gopalで出したすべての候補手について行っているので、まだかなりの手数の先読みが必要なこと

②電総研で作られたシステムを改良してきたので、最初に作られた時に想定した使われ方とは異なってきて、かなり無駄な部分があること

などが考えられる。特に、①については、改善の余地が十分ある。人間でも一般に先読みの時は一部分の対象について考えることが多い。そこで、本システムでも盤面の一部を切り出してその部分のみを探索の対象とすればスピードも上がり実用性が出てくるかもしれない。ただし、問題を切り出す方法については難しい点が多い。たとえば、「捕獲できるか」とか「切ることができるか」など目的の限定された先読みについては検討されている例もあり[5][6]、それについてはその対象部分を決定することも比較的やさしいが、一般の局面についてはそれぞれが複雑に絡まりあっている。その部分の手を進めれば状況が変化して他の部分にも影響を及ぼす。たとえば、ある部分の石が活きるために回りの敵石が強くなって他の部分に影響が出るといったこともあり、単純に盤面を分割することはできない。

また、パターン入力のために、拡張Gopalについての支援ツールについて作成を予定している。現在は人間がパターンを考えてそれをGopalに落として入力している。しかし、囲碁の知識は多く、また、システムに入力する場合には系統立てて入力を行わなければシステムのバランスが悪くなる。そこで、知識をGopalに落とすツール、たとえば、マウスとグラフィック画面で盤面を指定してそれをGopalプログラムに直すツールがあれば便利である。また、各知識の検証、たとえば、各盤面に対する候補手の列挙などのツールがあれば、パターンが十分か、誤っていないかどうかを確かめることの手助けになる。

また、囲碁の手というのは意図を持って打たれるのが普通である。たとえば、「攻める」とか「地を守る」とかの意図がある。現在、Gopalの候補手記述部には意図を記述する部分があるが、それは候補手の選択には使用されていない。人間の着手には継続した戦略と意図があるのが普通であり、意図がある方が以前の手がよくなる可能性が高い。そこで、意図の使用、特に、意図を継続して一貫した戦略をとることについても課題である。

さらに、ここではB*サーチアルゴリズムを取り上げたが、もっと囲碁に適したアルゴリズムにできないかどうかといった検討も今後の課題である。

参考文献

[1] A. J. Palay "Searching with Probabilities",Pitman Advanced Publishing Program (1985).

[2] 真野“囲碁における着手候補記述言語Gopalについて”、情報処理学会第24回全国大会、821(1982).

[3] B. Wilcox "Reflections on Building Two Go Programs",SIGART NEWSLETTER,94,pp.297-312(1985-10).

[4] T. Ibaraki "Generaization of Alpha-Beta and SSS* Search Procedures",Artifical Intelligence,29,pp.73-117(1986).

[5] 実近“知識思考型碁プログラムＧＯ．１における探索モジュール”、情報処理学会第33回全国大会、1325(1986).

[6] ＩＣＯＴ　ＣＧＳタスクグループ“囲碁システム概念設計書”、(1986).