卒業論文

頻出パターンを用いたコンピュータ囲碁候補手の選定と並列化の検討

氏　　名：中川　聖也
学籍番号：2260070068－9
指導教員：山崎 勝弘　教授
提 出 日：2011 年 2 月 18 日

立命館大学　理工学部　電子情報デザイン学科

内容梗概

本論文の目的は、モンテカルロシミュレーションを取り入れた囲碁プログラムの作成である。今回は去年同研究室の上野謙二郎氏が作成した囲碁プログラムをベースに、その棋力を上げるために候補手の思考部分に改良を加えた。具体的には、候補手のパターン化とモンテカルロ法の並列化である。候補手のパターン化はあらかじめ定石、手筋をこちらが指定してプログラムしておくことで、対局中にその場面が現れた場合に指定した手をさすようにする。モンテカルロ法はある局面からコンピュータにランダムにたくさん終局まで打たせ、その中で最も勝率の良い手を次の一手に選ぶという手法で、囲碁プログラムの発達にもっとも適しているといわれるシステムである。

目次

図目次

1. はじめに

本研究の目的は頻出パターンを用いた候補手選定と並列化の検討による囲碁プログラムの棋力向上である。コンピュータ囲碁を始め思考型ゲームのプログラムは人工知能研究の良い題材として長年研究が続けられている。1997 年には Deep Blue というチェスのプログラムが初めて世界チャンピオンを破った。将棋においても 2010 年にあから 2010 とい候補手選定の並列化に特化したプログラムが女流プロ棋士を破っており、その実力は若手プロに並ぶとまで言われるようになった。囲碁においても、モンテカルロシミュレーションを取り入れてからは急速に棋力を向上させ、現在はアマチュアの 3 から 4 段程度とされている。

チェスや将棋、囲碁には序盤から互いの最善手を尽くした手順である定石が存在する。これは長年プロの囲碁棋士によって研究されているものであり、日々新しい結論が出るものであるものの、いわば「その時点での完成された手順」であると言える。もちろん、これら定石の中には手筋、いい形というものが多分に含まれている。それらの頻出パターンをあらかじめデータベースとして用意し、対局中に同じパターンが盤上に現れれば指定しておいた手を指させるという技術は、囲碁プログラムの棋力向上において非常に重要である。また、今回私は囲碁プログラムにおけるモンテカルロシミュレーションの検討も行った。この近年、囲碁プログラム界ではモンテカルロシミュレーションという手法が非常に注目され、モンテカルロシミュレーションを取り入れた囲碁プログラムの実力は急速に発展してきている。モンテカルロシミュレーションとは、ランダムな候補手で終局まで対局をシミュレートし、その中で最も勝率の良い手を選ぶという手法である。これは理論的に好手を指すのではなく、複数の候補手を手当たり次第に調べて最善手を見つけるということである。

思考型ゲームは、囲碁の他に将棋やチェスと様々なものがあるが、囲碁の最大の特徴は『評価関数の作成の難しさ』、『局面数の多さ』である。評価関数とは、局面の状況を分析して、有利不利を数値化する仕組みである。また局面数とは、コンピュータによってゲーム終了まで選択可能なすべての手を調べ尽くすことに必要な手数である。一般的には、囲碁は将棋、チェスに比べ評価関数の作成が難しい。評価関数とはプログラムにおいての絶対的な判断基準であり、プログラムの実力を決定づける最大の要因である。囲碁が評価関数の作成が難しいとされる理由は、対局中の形勢判断が難しいということがあげられる。将棋やチェスでは、駒の損得という比較的重要でかつ簡単に計算可能な基準があり、駒の強さに点数をつけることで大まかな形勢判断が可能であるが、囲碁の場合にはそのような基準はない。このため将棋やチェスに比べて囲碁のプログラムの実力は低い位置にある。それを解決してくれるのがモンテカルロシミュレーションであり、今では多くの囲碁プログラムに取り入れられるようになった。モンテカルロシミュレーションは、極論をいえば評価関数を使用しない手法であり、使用するのは大規模な計算能力と、ランダムに着手を

行う確立的手法だけである。これは局面数が膨大な囲碁だからこそ用いられる手法であり、将棋やチェスではできないことである。この手法をといれた結果、囲碁プログラムは急速に発展し、従来のプログラムでは実現不可能だった人間の思考らしい手を打つまでに進化した。また現在では、モンテカルロシミュレーションに新たな手法を取り入れたプログラムも登場し、その実力を着実に伸ばしている。

2. コンピュータ囲碁システム

2.1 囲碁のルール

ルール 1. 黒白交互に打つ

ルール 2. 上下左右の点を囲まれた石は取り除かれる

ルール 3. 4 方向を囲まれているところへは置けない

図 1 の×のあるところには打つことができない。なぜなら×のところはすでに上下左右の点が囲まれており、そこへは存在することができないからである。このような場所（×）のことを着手禁止点（眼）という。

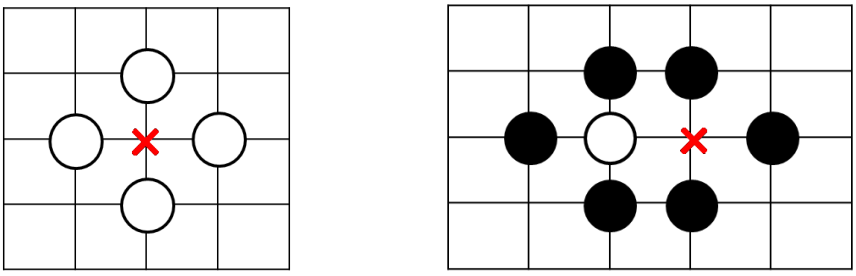


図 1：着手禁止点の例

ただし、図 2 のような場合黒は打つことができる。なぜなら、打った黒石で相手の白石を取ることができ、石の上下左右を囲まれた形にならないからである。

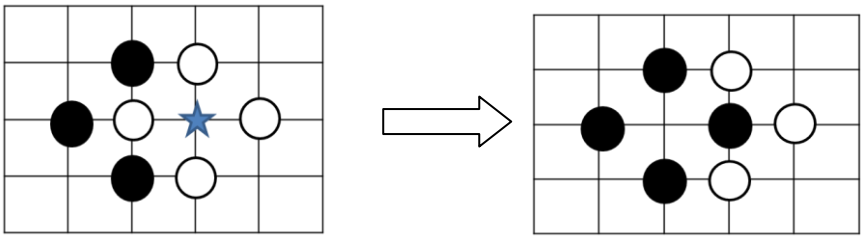


図 2：着手禁止点の例外

ルール 4. 同型反復禁止（コウ）

図 2（右）で黒が取った後の場合を考える。同型反復禁止（コウ）というのはこのような場面ですぐ直後に白が黒を取り返せないというルールである。なぜなら取り返すとまた図 6（左）のような同じ場面になりまた黒が取り返して・・・・という繰り返しになり、終わらなくなってしまうからだ。

このような繰り返しを防ぐためのルールを「コウ」と呼び、コウになるとしかたなく他の場所に打つことになり、それに相手がつきあってくれると次には取り返すことができる。

2.2　システム構成

図 3 にコンピュータ囲碁のシステム構成を示す[10]。

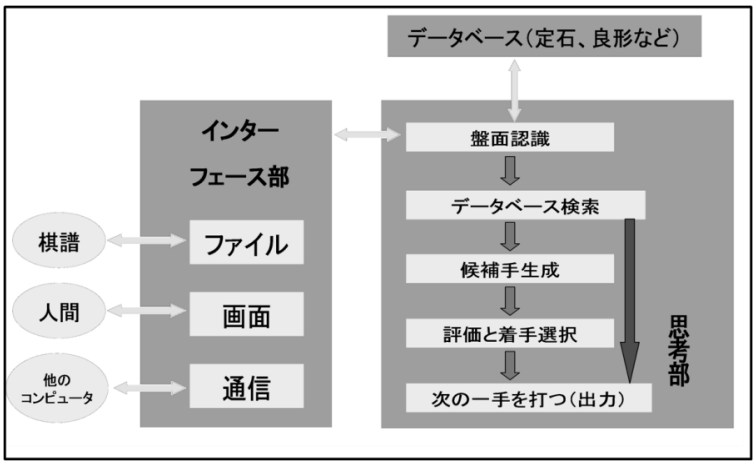


図 3：コンピュータ囲碁のシステム構成

まず、インターフェース部から棋譜や人間、他のコンピュータの打った碁盤の状況を読み取る。そして、思考部で処理をしていくという流れになる。思考部の中の流れは以下のようになっている。

(1) 盤面認識

まず、インターフェース部から碁盤を読み取り、その盤面から石の状況（群の有無、コウの有無など）を読み取る。

(2) データベース

データベースには石の結線のパターンや定石、好形等のパターンをあらかじめ用意しておく。

そして、盤面認識をした際にデータベース内にあるパターンと一致するものがあった場合、それに対応した処理を行う。例えば、ある石の配置を認識した場合、次の一手を打つ場所を定めたパターン（定石）を用意しておき、もし盤面にそのパターンと一致する場所を発見した場合、あらかじめ定めておいた場所に着手するようにする。これについては次章で説明する。

(3) 候補手生成

盤面にデータベースの中の次の一手を打つパターンと一致するものがなかった場合、次の手の候補を考える必要がある。

石の活き死にや地から次の一手の候補を選ぶ。

(4) 評価と着手選択

各候補手を評価し、その中で最良のものを選ぶ（評価値が最高のものを選ぶ）。

(5) 次の一手を打つ

データベースのパターンや候補手の中から選ばれた手を次の一手として打つ。

2.3　プログラムの処理手順

ここでは囲碁プログラムを動かした際の処理の流れを簡単に説明する。

（1）碁盤初期化

最初に対局を行う碁盤を何も石が行かれていないものにする必要がある。これが碁盤初期化である。

（2）候補手生成

次に候補手生成が行われる。図 4 では候補手生成に使われている地について示した図である。各石から距離 2 以内をその石の地と考える。イメージとしては一つ一つの石がそれぞれ同じだけの力を持っていると考えてもらいたい。図 4 の■が地となり、石一つが最大 12 目分の力を持っていることになる。

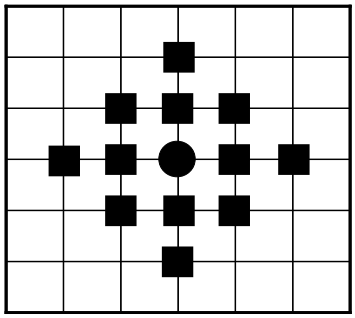
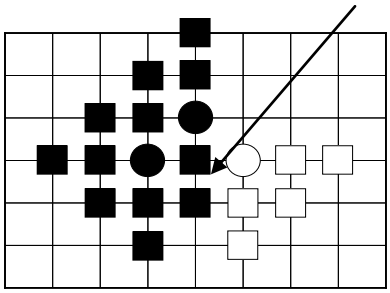
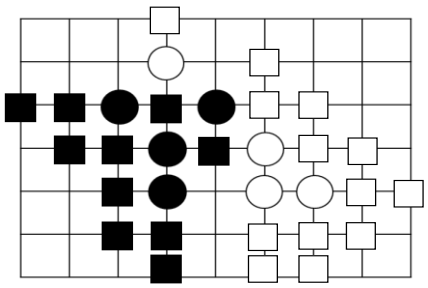


図 4：石一つの地

ここで、石が近くに複数ある場合を考える。ある場所で黒、白両方の地が重なった場合、影響力の強い方をその場所の地とし、力の強さが等しければ、打ち消しあい、そこはどちらの地でもないようにしている。例として図 5 を見てもらいたい。図 5(a)を使って説明していく。左図の矢印の指している地の部分は黒、白両方の力が及んでいる場所である。この場所は 2 つの黒石からそれぞれ力を受け、黒石の力が 2。白石からは石が一つなので力が 1。つまり、黒石の力が白石の力を上回っていることになるので、黒の地となっている。
このように、石を置いたときに増える地の数が多い部分の候補手の優先度が高くなる仕組みになっている。また、これに加え、第 3 章で解説する頻出パターンのデータベースを利用した候補手生成を行う。



(a)複数の石の地 1



(b)複数の石の地 2

図 5：複数の石の地

（３）連の認識

連の認識は候補手生成の内部で行われるものであり、候補手生成のための要素の１つである。連とは縦横につながった同じ色の石の集合のことである。この「連」という表現は囲碁用語ではないが、多くの囲碁プログラムの作者は「連」と呼んでいるので、ここでも「連」と呼ぶことにする。具体的には図 6 の黒石、白石それぞれの集合のことをいう。

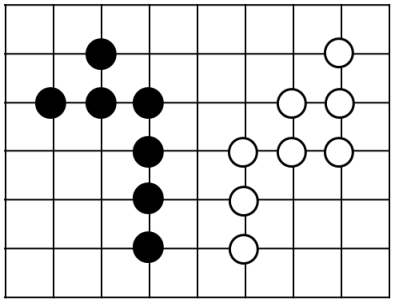


図 6：連

（４）結線の認識

連の次は結線の認識である。結線も連と同じく候補手生成の内部で行われる処理である。結線とは石と石の間に仮想的な線を引いて、石どうしを結んでいるとするデータ構造のことである。具体的には図 7 の丸で囲った石同士のように少し距離を置いた位置にある石をつながっているとするデータ構造である。

この認識ができるようになることで、近くにある石を一つの集合としてとらえることができる。（２）から（４）までで候補手を生成するという一つの処理が終わる。

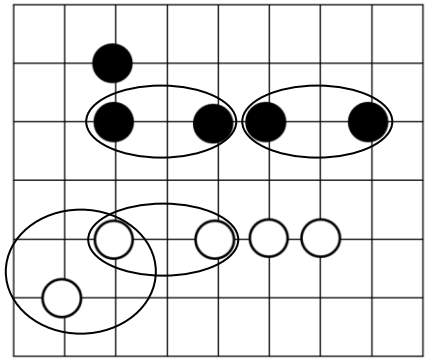


図 7：結線の例

（５）終局時の地の計算

最後に対局終了時の地の計算が行われる。まず、対局が終了した時に碁盤をみて、弱い群を碁盤から削除する。また、削除された群の結線も削除する。すると以下のようになる。図８は弱い群を消去したものである。

＊現在の弱い群の定義は『石の数が４つ以下のもの』としている。

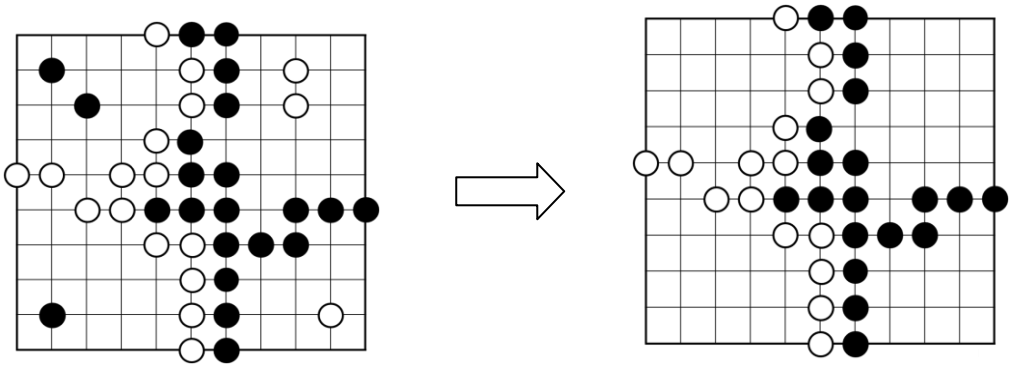


図８：地の作成１

次に結線をそれぞれの地とする。（＆は黒の結線、＄は白の結線）

図９は結線を地に変換する様子である。

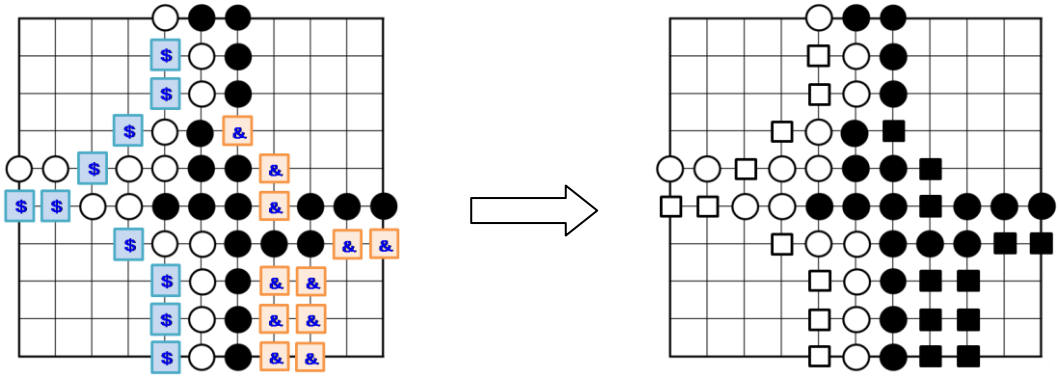


図９：地の作成２

ここからまずは黒の地を認識させる。方法としては碁盤の空点を検索し、空点をみつけた場合、その点から上下左右に手を伸ばして、『黒石、白石、黒の地、白の地、盤外』のどれに当たったかをそれぞれ記憶させる。この４点を調べ、記憶されているものが『盤外』が２つ以下かつ『白石、白の地』が２つ以下なら黒の地とする。例えば、図 10 の☆の点の場合、記憶されるのは以下のようになるため、黒の地となる。

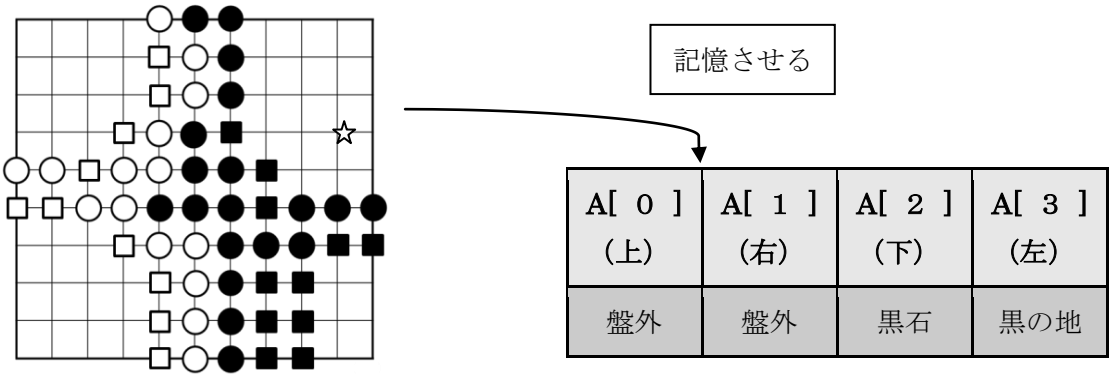


図 10：地の作成 3

白の地の認識も同様にして行うと、次のようになる。
図 11 は終局時の碁盤の認識結果の例である。

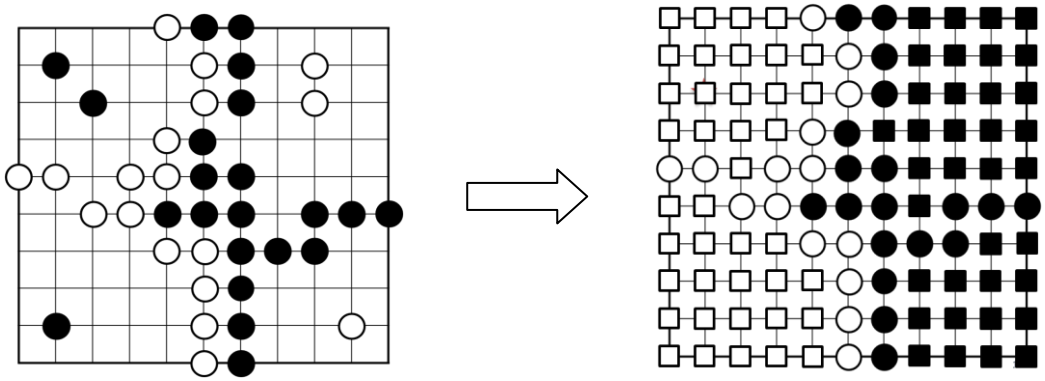


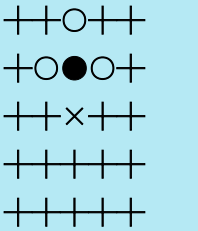
図 11：地の作成 4

3. 候補手の作成方法

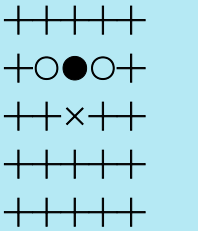
3.1 候補手のパターン化

昨年までの囲碁プログラムでは、一つの石に一定の力（地）を設定し、地が多く増える地点の評価値を高くするという方針で候補手を作成していた[10]。しかし、その方法ではあまりにも抽象的である。そこで、あらかじめ定石、手筋をこちらが指定してプログラムしておくことで、対局中にその場面が現れた場合に指定した手をさすようにした。候補手のパターンの大きさは 5×5 とし、全て白番が着手する場合を考えており、×印が着手点である。石が少ない場合（2 個から 4 個）の候補手のパターンを図 12 の（A）から（R）に、石が多い場合（5 から 9 個）の候補手を図 13 の（a）から（t）に示す。また、黒番が着手する際で、なおかつパターンが白黒反転していた場合や、パターンが 90 度、180 度回転した場合も×の地点に着手できるようにプログラムしている。

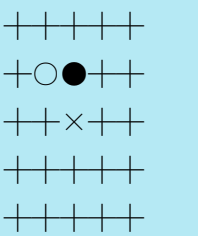
(A)



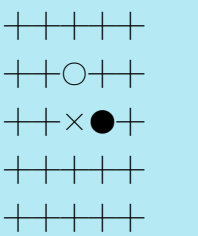
(B)



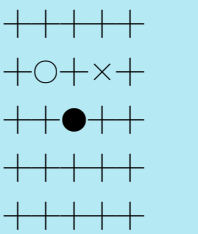
(C)



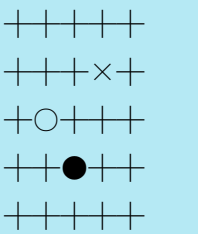
(D)



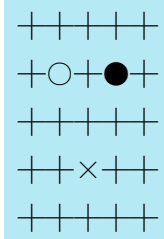
(E)



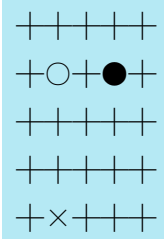
(F)



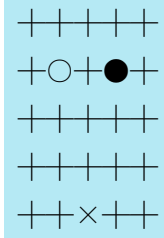
(G)



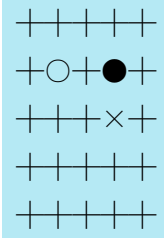
(H)



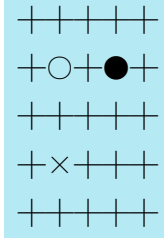
(I)



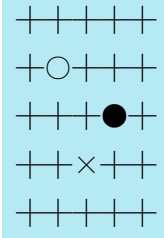
(J)



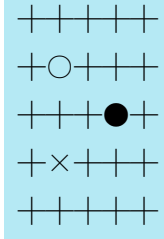
(K)



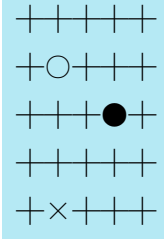
(L)



(M)



(N)



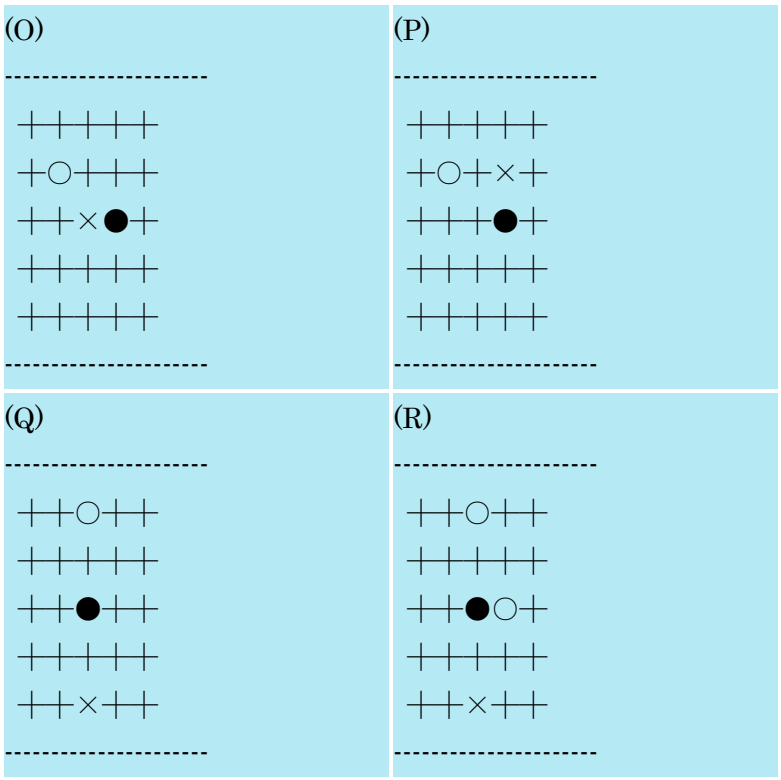
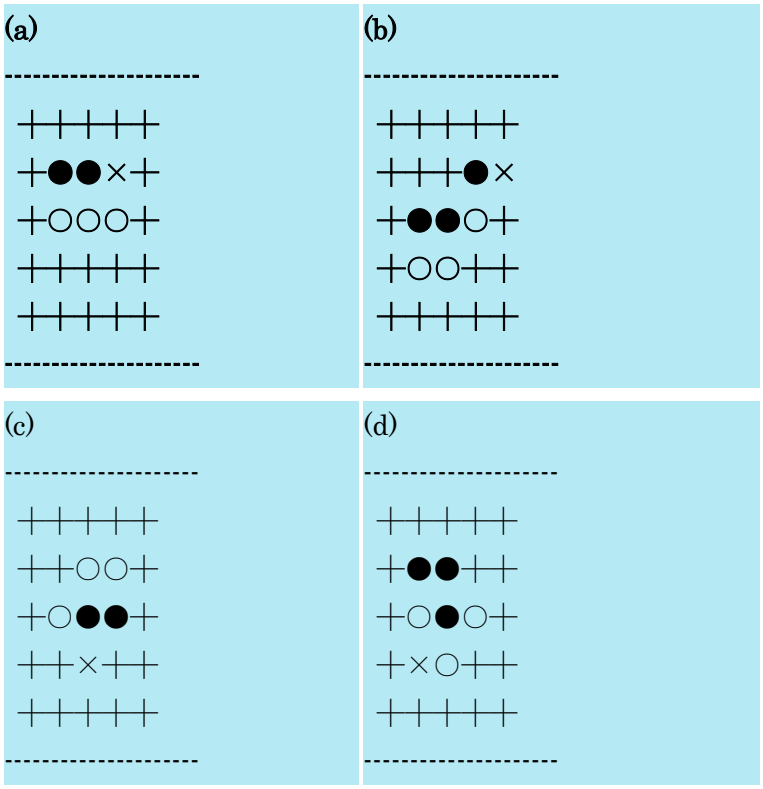
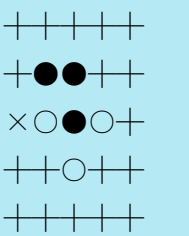


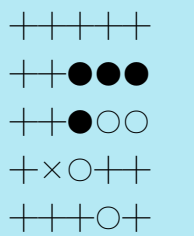
図 12：石が 2 個から 4 個の場合の候補手のパターン



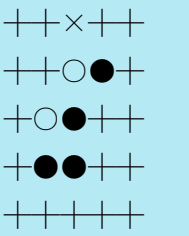
(e)



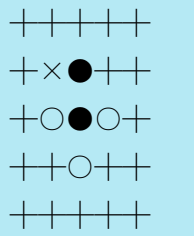
(f)



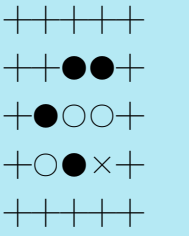
(g)



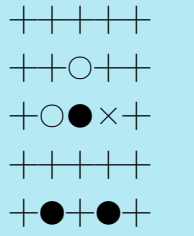
(h)



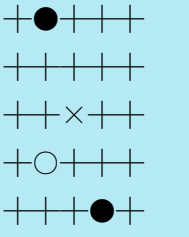
(i)



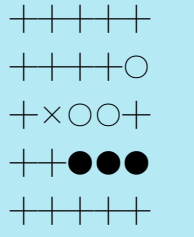
(j)



(k)



(l)



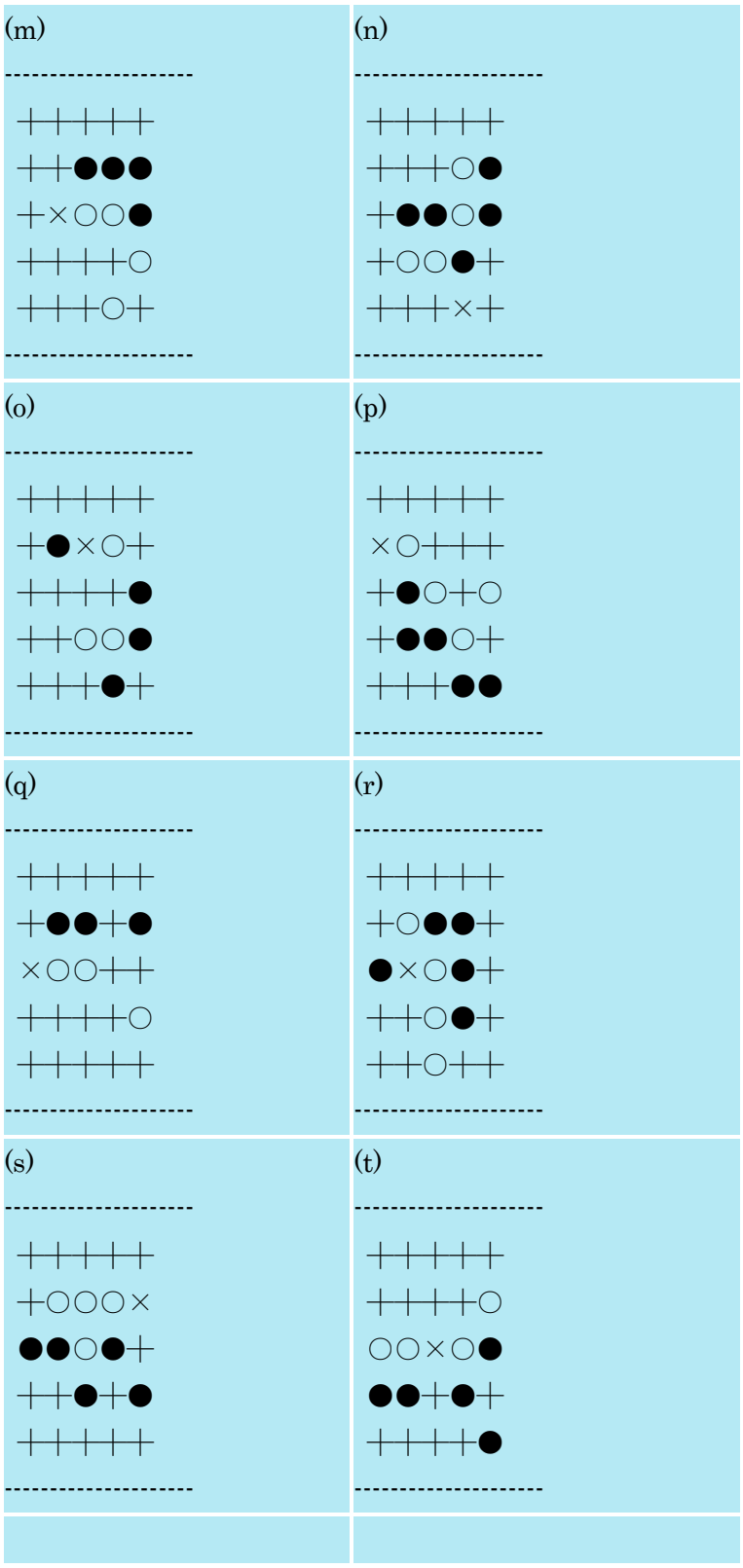


図 13：石が 5 個から 9 個の場合の候補手のパターン

3.2　実装

プログラム内でのパターンは黒石を BLACL、白石を WHITE、空点を SPACE と表示している。また、5×5 のパターンにおける着手点の座標を上に記入する。例を挙げると図 14 のようになる。

```
{   3, 3,                                                    /*  候補手の座標  */
 {{ SPACE, SPACE, SPACE, SPACE, SPACE },      /* ┼┼┼┼┼ */
  { SPACE, BLACK, BLACK, SPACE, SPACE },     /* ┼●●×┼ */
  { SPACE, WHITE, WHITE, WHITE, SPACE },     /* ┼○○○┼ */
  { SPACE, SPACE, SPACE, SPACE, SPACE },      /* ┼┼┼┼┼ */
  { SPACE, BLACK, SPACE, SPACE, SPACE }}},   /* ┼┼┼┼┼ */
```

図 14：候補手のパターンを表すプログラムの例

今回は 9×9 路盤上においてコンピュータ同士で終局まで囲碁を打ち合った場合に対局中に何回くらい図 12、図 13 で示したパターンが現れるかどうかを試した。コンピュータ同士が打ち合うプログラムを 10 回動作させて実験を行ったところ、図 12 の（B）が 2 回、（C）が 3 回、（D）が 2 回、（H）が 2 回、（J）が 5 回、（L）が 3 回、（Q）が 3 回、（R）が 1 回現れるという結果になった。なお、これらは全て 1 回の対局の中にそれぞれのパターンが 1 回ずつ現れた。また、図 13 の中では 10 回の対局中に（a）のパターンが 1 回出てくるのみとなった。これにより、石が少ない場合のパターンを実現させることは可能になったが、石が多いパターンを実現させることはまだ難しいという結果が得られた。

3.3　考察

候補手のパターン化の実相によって、囲碁においてある程度手筋となる指し手、好手となる手をこちらが意図して指させることができるようになった。パターンの形が現れれば、評価値はそのパターン通りの指し手を優先させるためである。問題は、パターンが現れる局面以外は去年までの着手選択方法になってしまうという点である。これによって、パターンの実相はできたものの、残念ながら大幅な棋力向上には至らなかった。今後は序盤の定石に基づくパターンを取り入れていきたいと考えている。また、取られそうになった自分の石を守れるようなパターンも取り入れていきたい。

4.モンテカルロ法の並列化

4.1　モンテカルロ法のシステム

モンテカルロシミュレーションとは、ランダムな候補手で終局まで対局をシミュレートし、その中で最も勝率の良い手を選ぶという手法である。これは理論的に好手を指すのではなく、複数の候補手を手当たり次第に調べて最善手を見つけるということである。囲碁には局面ごとに複数の候補手があり、それに対する相手の応手も複数存在することが多い。よって、先を読み進むにつれて候補手は枝分かれしていくことになる。この枝分かれをモンテカルロ木探索という。図 15 はモンテカルロ木を示した図である。この考え方は mini-max 探索というものから来ている。これは双方が最善とされる手段を模索していく考え方である。しかし、全ての枝を深く読み進めていくと膨大な時間がかかってしまう。そこで、探索の途中で明らかに最善ではないものの考慮を途中で省いてしまう方法がある。それが alpha-beta 探索である。これにより、囲碁の膨大な候補手とその後の変化を極めて効率的に探索していくことができる。図 15 を見ると、1 番左の木からさらに枝が分かれしている。これは一番左の候補手が有力と分かり、さらに深く読もうとしているのである[11]。

この考え方は将棋やチェスなど他のゲームに当てはめても囲碁ほどの効果が出ない。それはなぜか？例えば将棋における形勢判断の基準には駒の損得、玉の堅さ、位の厚み、駒の働きなどが挙げられるが、これらはある程度将棋が分かる人間なら将棋盤を見た瞬間分かることであり、形勢判断もしやすい。対して囲碁は陣地を取り合うゲームであるため、盤を見ただけではどちらが有利か分かりにくい。囲碁の形勢判断はある程度囲碁ができる人間にも難しいと言われる。このように囲碁は有利不利の基準が大雑把であいまいなものであるために、手当たり次第に候補手を探す方法が有効になるのである。モンテカルロ法の実装によって、囲碁プログラムは大幅に棋力を上げることとなった。

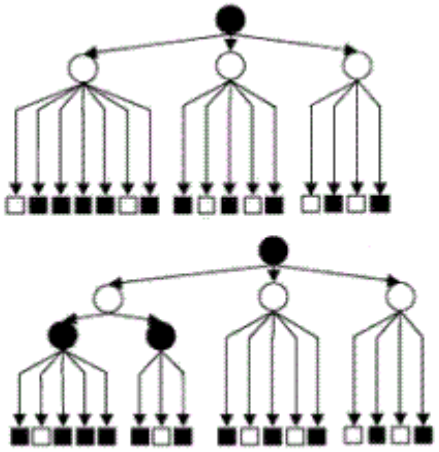
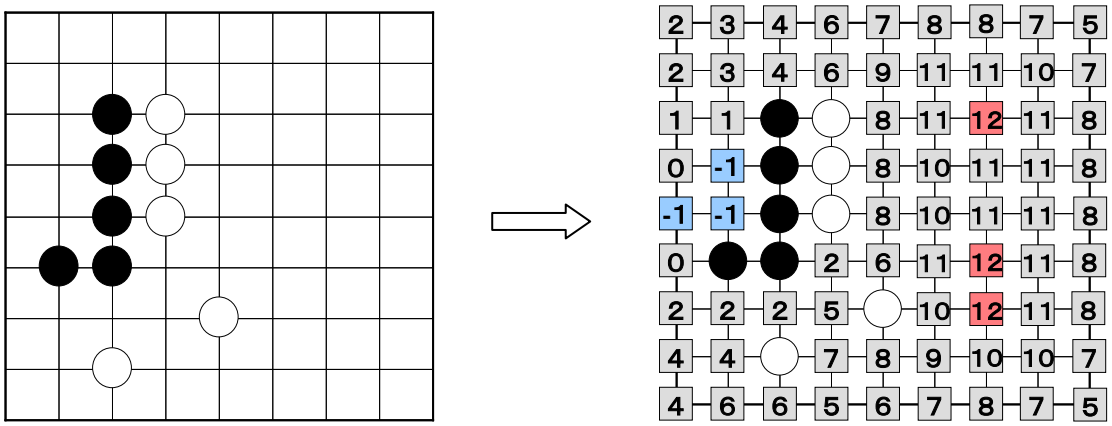


図 15：モンテカルロ木

4.2　モンテカルロ法の並列化の検討

ただモンテカルロシミュレーションを用いて囲碁プログラムを起動するだけでは、計算に膨大な時間がかかってしまう。そのため、モンテカルロシミュレーションを行う際は処理を並列化させる必要がある。以下に並列化を行う場合の手法を示す。

モンテカルロシミュレーションはランダムな候補手で終局まで対局をシミュレートし、その中で最も勝率の良い手を選ぶという手法であるが、実際は効率よく候補手を探索するために最初にある程度評価値の高い手を厳選し、その中から最も勝率の良い手を着手するのが理想的である。例として図 16 (a) の局面を挙げる。この局面で候補手の作成を行うと、評価値の高い手は図 16（b）の 12 と書かれた点となる。なぜ 12 の点の評価値が高くなるかは 2.3 の（３）候補手生成の欄を参照してほしい。12 と書かれた点は 3 つあるが、どの手が最善かは見ただけではわからない。ここでモンテカルロシミュレーションを使い、この 3 点からそれぞれ着手した場合の終局までを読み進め、もっとも勝率の高い手を着手する。



(a)碁盤　　(b)地の増減

図 16：候補手の作成

5.考察

今回は頻出パターンを用いた候補手選定とモンテカルロ法の並列化の検討を行った。頻出パターンを用いた候補手選定は一定の成果が得られた。今回用意したパターンは 20 から 30 個くらいの数であったが、今後さらに数を増やしていきたい。パターンの数が増えれば増えるほど、プログラムはいろんな局面に柔軟に対応できるようになるはずである。また、複数のパターンに当てはまる局面になったとき、どちらを選んだほうがいいのかを選べるようになるということもできるようになればよい。今回は前年度までの候補手選定法に頻出パターンを用いた候補手選定を組み合わせることで次の一手を指すというプログラムを目指した。この問題についてはモンテカルロ法を取り入れることに成功すれば改善されるのかもしれない。ある程度選りすぐった候補手を読み進めていき、一番勝率のいい手を着手するのがモンテカルロ法のシステムだからだ。この選りすぐった手というのも、今のプログラムの基準で選りすぐった手ということになると思うので、より本筋の候補手を思考するプログラムを目指そうと思う。

6.おわりに

今回は主に頻出パターンを用いた候補手選定のプログラムを作成したが、残念ながらこれだけで大幅な棋力アップという結果にはならなかった。前にも言ったが、パターンに当てはまらない局面では直接の効果がないからである。この棋力アップに関する解決策としてはやはりモンテカルロ法が挙げられる。モンテカルロ法が囲碁の棋力アップに適したシステムであることは第 4 章で言ったとおりである。しかし、まだ問題は存在する。それは、「群の認識」と「生死判定」である。群とは何かという点に関しては去年の上野謙二郎氏の卒業論文を参照してもらいたい[10]。群については結線を用いた群の認識プログラムがすでにできている。問題はどの群が強くてその群が弱いのかを認識できていないということである。群は囲碁の対局の中盤から終盤にかけて必ず複数個盤上に出来上がる。この群の強弱を考慮したうえで候補手を思考するプログラムができれば、棋力は上がると思われる。もう一つの生死判定については石が生きているか死んでいるかを判定することであり、勝敗結果に直接響く要素である。現在はまだきちんとした生死判定がつくれていなく、「2 目作ることができれば生きる」ということをプログラムしているだけである。先ほど言った群の生死も勝敗を左右する強い要素である。これらの問題を解決できれば、囲碁プログラムの棋力は大きく伸びることが期待できる。

謝辞

本研究の機会を与えて下さり、貴重な助言、ご指導を頂きました山崎勝弘教授、ならびに上野謙二郎先輩に深く感謝いたします。また, 本研究に関して様々な相談に乗って頂き、貴重なご意見を頂きました高性能計算研究室の皆様に深く感謝いたします。

参考文献

[1]清慎一、山下宏、佐々木宜助：コンピュータ囲碁の入門，共立出版社, 2005.

[2]中村貞吾：“コンピュータ囲碁研究の現状と展望, FIT2008 コンピュータ囲碁最前線, 2008.

[3]山下宏：囲碁プログラム　彩の仕組み, FIT2008, 2008.

[4]村松正和：コンピュータ囲碁の現状, FIT2008, 2008.

[5]飯田弘之：ゲーム情報学の中のコンピュータ囲碁, FIT2008, 2008.

[6]清慎一：コンピュータ囲碁の歴史と将来の展望, FIT2008, 2008.

[7]村松正和：プロ棋士対コンピュータ　FIT2008 における囲碁対局報告，情報処理 Vol.50 No1, Jan.2009, pp70-73, 2009.

[8]山下宏：YSS と彩のページ, http://www32.ocn.ne.jp/~yss/index_j.html

[9]Fuego, : http://www.perfectsky.net/fuego/index.html

[10] 上野 謙二郎 : コンピュータ囲碁の思考部の作成[2]，立命館大学理工学部電子情報デザイン学科卒業論文, 2010.

[11] 美添一樹 : モンテカルロ木探索，情報処理 Vol.49, No.6, June.2008, pp.686-693, 2008.