人工知能

第5回講義課題 課題番号15

提出締切: 2013.12.18

提出日:　2013.12.15

工学部電子情報工学科

03-123006 岩成達哉

1　概要

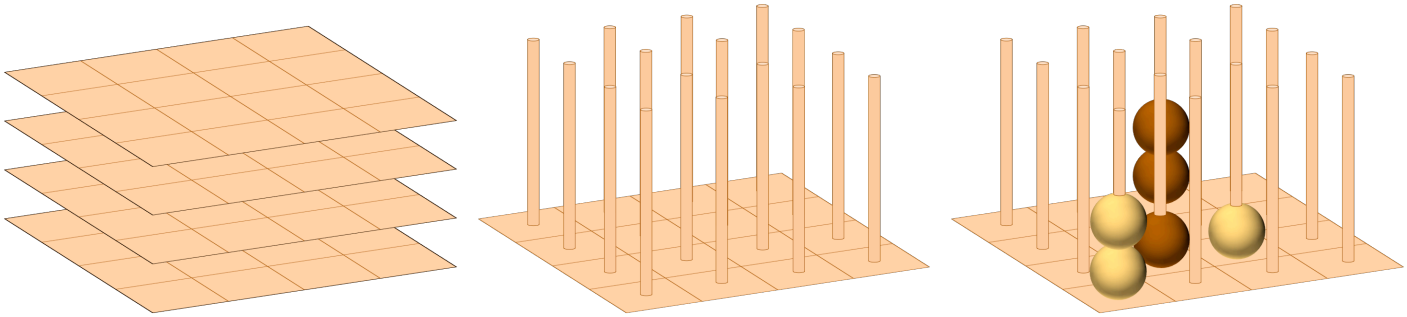
立体 4 目並べに対して，UCT アルゴリズムを適用し，試行回数によって，勝敗がどのように変化するかを調べた．また，Negamax に $\alpha\beta$ カットを用いたプログラムと対戦させ，勝敗や探索の効率 (展開されたノード数，実行時間など) を比較した．

2　立体 4 目並べ

題材とした立体 4 目並べについて説明する．

2.1　道具

立体 4 目並べは，4×4 の盤面が 4 つ重なった図 1(a) のような盤面上でゲームが進行する．実際には図 1(b) のように 16 本の棒が立っており，図 1(c) のように石を置いていく．石は，図 1(c) のように黒と白の 2 種類があり，それぞれ 32 個ずつある．



(a) 立体 4 目並べの盤面　(b) 実際の立体 4 目並べの構造　(c) 石の設置

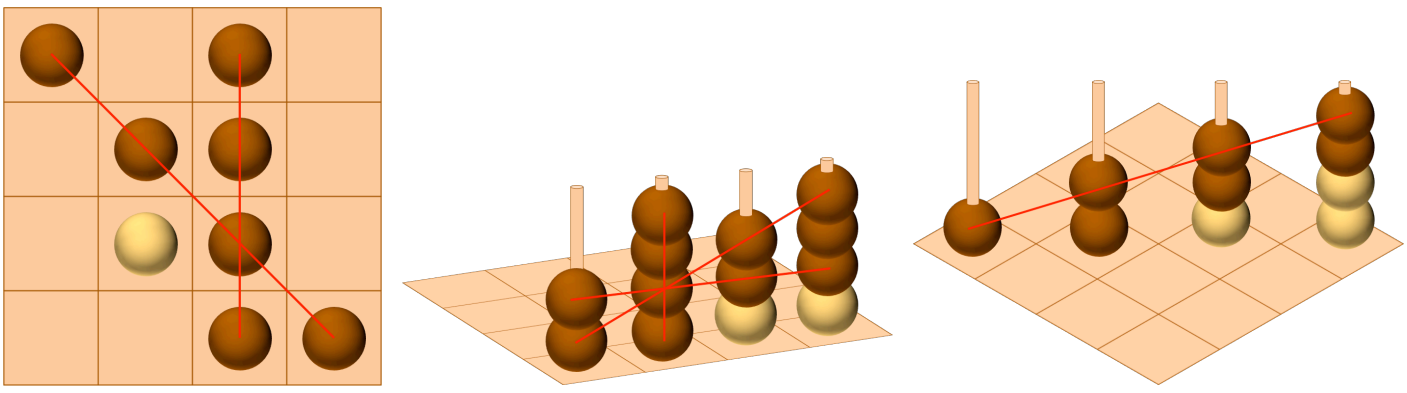
図 1: ゲームの盤面

2.2　ゲームの流れ

立体 4 目並べは 2 人で行う対戦ゲームである．2 人のプレイヤーが交互に自分の色の石を置いていき，先に勝利条件を満たした者が勝ちとなる．石は，図 1 で示した盤面に置いていくが，石を浮かせることは出来ず，一番下から順に積んでいく．

2.3　勝利条件

立体 4 目並べは，3 次元的に縦，横，斜めに同じ色の石を 4 つ揃えた者の勝利である．例えば，図 2 のような並びで黒が勝利となる．図 1(a) でも示したように 4 つの平面によって成り立つため，図 2(a) の条件がそれぞれの面で成立し得る．また，図 2(b) や図 2(c) のような場合も 3 次元的に縦，横，斜めに配置できているため，黒の勝利といえる．



(a) 平面での勝利　(b) 横の面での勝利　(c) 対角線での勝利

図 2: 黒が勝利する例

3　実装したプログラムの仕様

3.1　実行環境

プログラムの動作環境を表 1 に示した．動作環境が Mac OS，Linux となっているのは，時間を図るためにシステムコールを用いる gettimeofday 関数を使用しているからである．提出したファイルでは，コメントアウトしているため，Windows でも実行は可能である．

表 1: プログラムの動作環境

言語	開発環境	コンパイラ	動作環境
C	Mac OS X (Mavericks)	gcc	Mac OS , Linux(Windows)

3.2　実行方法

プログラムのコンパイルは，添付した Makefile によって行える．コマンドラインで，AI_123006_15 のフォルダに移動し，下記のコマンドを打ち，Enter キーを押す．

リスト 1: make

```
1 > make
```

正常に動作すれば，main というファイルが生成されるため，

リスト 2: 実行

```
1 > ./main
```

と入力し，Enter キーを押すことで実行できる．

本アプリケーションは，コマンドラインからの引数で対戦方法を指定できる．これをリスト 3 に示した．

リスト 3: 実行

```
1 > ./main [先手:0, 後手:その他(デフォルト:0)] [UCTの総プレイアウト回数(デフォルト:10000)]
```

3.3　使用方法

アプリケーションの使用方法を以下に示した．

先手の場合を考える．アプリケーションを実行すると，リスト 4 のような表示がプロンプトに現れる．

リスト 4: 手の打ち方

```
1  --- 4段目---
2    |  |  |  |
3    |  |  |  |
4    |  |  |  |
5    |  |  |  |
6  --- 3段目---
7    |  |  |  |
8    |  |  |  |
9    |  |  |  |
10   |  |  |  |
11 --- 2段目---
12   |  |  |  |
13   |  |  |  |
14   |  |  |  |
15   |  |  |  |
```

```
--- 1段目---
  |  |  |  |
  |  |  |  |
  |  |  |  |
  |  |  |  |
--- guide---
 0| 1| 2| 3|
 4| 5| 6| 7|
 8| 9|10|11|
12|13|14|15|
-----------
input next pos(0 )[0-15]:
```

これは，上から 4 段目，3 段目... の 16 マスを表している．プレイヤーは，guide にしたがって，0〜15 の数値を入力し，Enter キーを押す．すると，指定した位置に石が置かれ，その様子をリスト 5 に示した．

リスト 5: 置いた結果

```
input next pos(0 )[0-15]: 0
--- 4段目---
  |  |  |  |
  |  |  |  |
  |  |  |  |
  |  |  |  |
--- 3段目---
  |  |  |  |
  |  |  |  |
  |  |  |  |
  |  |  |  |
--- 2段目---
  |  |  |  |
  |  |  |  |
  |  |  |  |
  |  |  |  |
--- 1段目---
0 |  |  |  |
  |  |  |  |
  |  |  |  |
  |  |  |  |
```

次に，PC が UCT を用いて，手を打つ．これをリスト 6 に示した．

リスト 6: 相手の手番

```
--- next turn ---
input next pos(O )[0-15]: 1
--- 4段目---
  |  |  |  |
  |  |  |  |
  |  |  |  |
  |  |  |  |
--- 3段目---
  |  |  |  |
  |  |  |  |
  |  |  |  |
  |  |  |  |
--- 2段目---
  |  |  |  |
  |  |  |  |
  |  |  |  |
  |  |  |  |
--- 1段目---
0 |O |  |  |
  |  |  |  |
  |  |  |  |
  |  |  |  |
```

次に，リスト 4 のような guide が表示され，プレイヤーが打つ．以上を繰り返し，どちらかが勝利するか引き分けになればその旨が表示される．

4　アルゴリズムの概要

4.1　勝利条件の判定

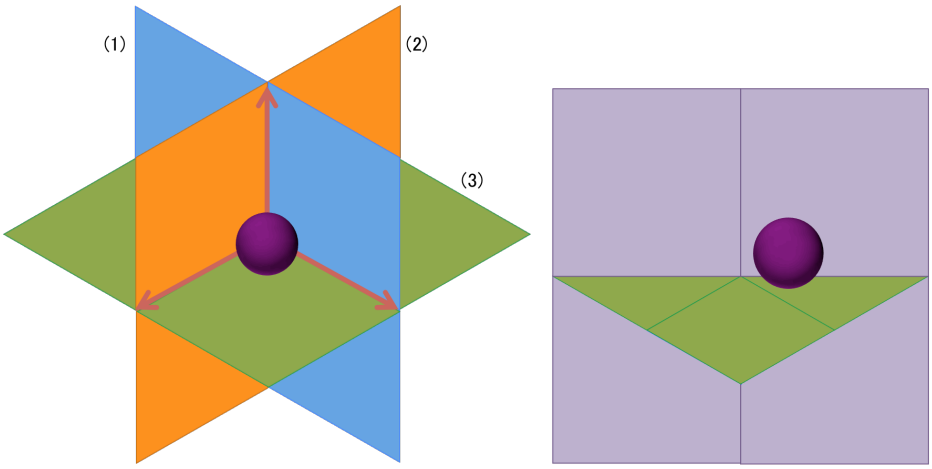
立体 4 目並べでは，勝利パターンが数多く存在し，それを判定する必要がある．

まず，4×4 の面においての勝利判定を用意する．これは，置かれた位置を中心に，その前後左右，斜めに 4 つ同じ色の石が並んでいるか判定すればよい．

次に，置いてある位置を中心とした，最大で 4 つの面を作る．この様子を図 3 に示した．

図 3(a) は，置かれた位置がどこであっても生成する判定すべき面である．置かれた位置に対して 3 つの方向に面を作り，それぞれの面に対して，前述の勝利判定を行えばよい．

一方，置かれた位置が図 1(a) で表した 4 つの面のいずれかの対角線上に置かれた場合は，図 2(c) のようなパターンで勝利する可能性があるため，この面についても前述の勝利判定を行う．



(a) 基本となる 3 つの面　　(b) 斜めの面

図 3: 判定を行う面

4.2　合法手の作成

UCT や Negamax 法では，合法手の作成が必須である．立体 4 目並べでは，一番上の面 (下から 4 番目の面) の，すでに石が置かれている位置には石が置けないという以外，白と黒の石を置く位置に制限はない．よって，一番上の面の 16 個を確認すれば良い．

4.3　UCT の概要 [1]

UCT(UCb applied to Trees) は，モンテカルロ木探索に UCB 値を用いて，探索するノードの優先度をつける手法である．UCB 値は，以下の式で与えられる．

$$UCB = (\pm)X_i + C\sqrt{\frac{\ln n}{n_i}} \tag{1}$$

ただし，　符号は自分の手番の手なら +，相手の手番の手なら −，

X_iはそのノードの勝率, C は定数, n は総プレイアウト回数, n_iはそのノードのプレイアウト回数

この値が最も大きくなる手をランダムにゲームオーバーまで探索する．また，あるノードのプレイアウト回数が一定値を超えた際は，そのノードをさらに展開 (次の手も子として持つ) し，プレイアウトを行う．その結果は，親に向かって伝播させる．

これによって，有力な手は次々に展開されていく．また，展開した手が，相手の手番の手である場合は，UCB 値の勝率の符号を判定させて，相手が相手にとって有利な手を打つことを想定するようにした．

参考文献 [1] を基に，これを実装した．

4.4 Negamax $+\alpha\beta$ カットの概要 [1][2]

Negamax 法は，MinMax 法を改良した方式である．

MinMax 法は,「自分は自分の評価値が最大となる手を指し，相手は自分の評価値が最小となる手を指す」という仮定でアルゴリズムを組んでおり，自分と相手の手を指すのは別の関数となる．一方，NegaMax 法は「自分も相手も自身の評価値が最大となる手を指す」ということを仮定し，相手の評価値は符号を反転して評価に加える．これによって，同じ関数を用いることができ，実装が容易となる．

この Negamax 法に，$\alpha\beta$ 枝刈りを導入したプログラムを作成した．Negamax の評価関数は，$4\times4\times4$ の評価値を経験から入力しておき，それを用いて計算する単純な方式を用いた．リーチの数を求めて評価に加える方法もあったが，後述のように十分強い AI となったため，この実装は行わなかった．

4.5 工夫した点

UCT では，勝率を計算するために，勝ったときのみそれをカウントするため，式 (1) の勝率 X_i はもちろん正である．しかし，今回扱った立体 4 目並べでは，負ける組み合わせ (同時に勝つ組み合わせ) が膨大であり,「負ける」という情報を大きく扱うべきと考える．

よって，今回実装した UCT アルゴリズムでは，負けた時は勝った回数を -1 することで，X_i が負の値を取りうるように変更した．つまり，Negamax の考え方を UCT に導入したといえる．

5 実験方法

実験は，学科 PC(Ubuntu 12.04 [Intel Core i7 @2.40GHz，8 コア，メモリ 16GB]) によって行った．

5.1 C の決定

UCT で用いる式 (1) では，定数 C を用いている．この値は実験によって決める値となっているため，最も勝率が高くなるような C を求めるために実験を行った．実験方法は以下のとおりである．UCT を先手とした．

1. Negamax 法のプログラム (探索の深さ 5) のプログラムを対戦相手とした
2. UCT の総プレイアウト回数を 1000，展開の閾値を 50 として，C を変化させて 500 回戦わせ勝率を求めた
3. 総プレイアウト回数が 2000，5000，10000，20000，50000 でも C を変化させて勝率を求めた

展開の閾値を 50 としたのはあらかじめ実験を行って求めた結果からである．

5.2 UCT 同士の対戦

UCT アルゴリズム同士の対戦でプレイアウト回数を変化させた時の勝率の変化を調べた．実験方法は以下のとおりである．勝率は先手のものを求めた．

1. UCT の総プレイアウト回数を 1000，展開の閾値を 50，$C = 0.6$ を敵とした
2. UCT の総プレイアウト回数を変化させたものと 500 回戦わせて勝率を求めた
3. 敵の総プレイアウト回数を 2000，5000，10000，20000，50000 と変化させて同様の操作を行った

5.3 UCT と Negamax の対戦と探索効率

UCT と Negamax の対戦を行い，その勝率を求めた．このとき，C は 5.1 節の実験から $C = 0.6$ を採用した．実験方法は以下のとおりである．UCT を先手とした．

1. Negamax 法 (探索の深さ 1) のプログラムを対戦相手とした
2. UCT の総プレイアウト回数を 1000，展開の閾値を 50 として C を変化させて 500 回戦わせ勝率を求めた
3. 総プレイアウト回数が 2000，5000，10000，20000，50000 でも C を変化させて勝率を求めた
4. Negamax 法の探索を $2, 3, .., 6$ と増やし，同様の実験を行った．

また，探索の効率を UCT と Negamax で比較を行った．比較は，展開されたノード数と実行時間を調査することで行った．UCT の探索効率を測る実験の方法は以下のとおりである．

1. 1 手目での展開されたノード数 (探索の深さ) と実行時間を比較した
2. 試行回数 1000 の UCT を用いて，1 手目を打つことを 100 回繰り返し，その合計時間を求めた
3. 手を繰り返す際に，一番深く進んだ深さの 100 回の平均を求めた
4. 同様の内容を試行回数を変化させて行った

Negamax の探索効率を測る実験の方法は以下のとおりである．

1. 深さ 1 の Negamax を用いて，1 手目を打つことを 100 回繰り返し，その合計時間を求めた
2. 同様の内容を深さを変化させて行った

6 結果

6.1 C の決定

C を求める実験の結果を図 4 に示した．結果から，$C = 0.5$〜0.7 の時に勝率が大きくなっていることがわかった．

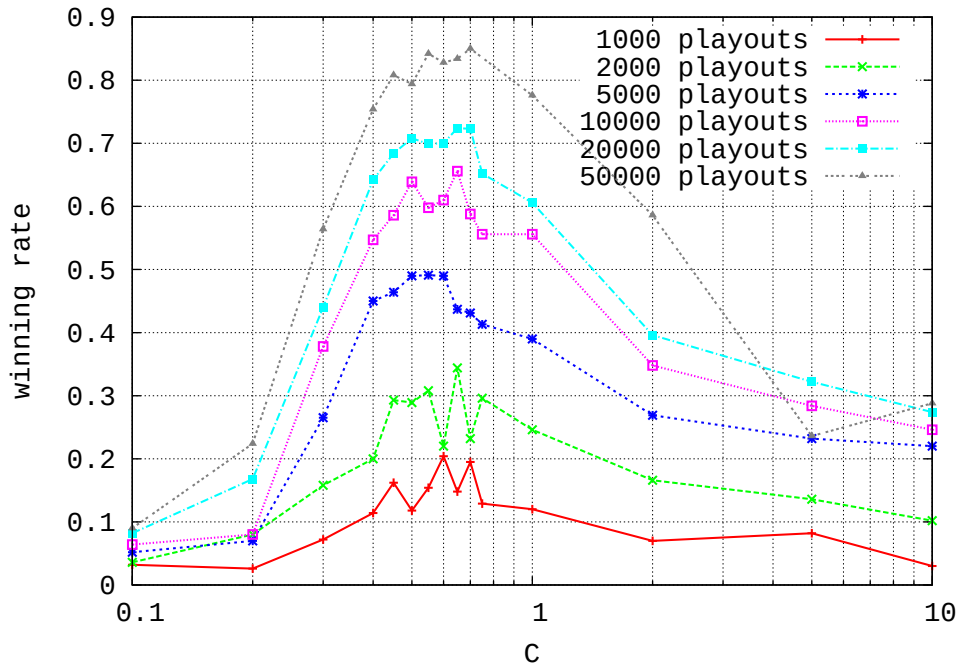


図 4: 最適な C の調査

6.2 UCT 同士の対戦

UCT 同士の対戦結果を図 5 に示した．x 軸は先手のプレイアウト回数であり，後手のプレイアウト回数を固定して実験を行い，先手の勝率を y 軸にとって示した．この結果から，試行回数を増やすと勝率が向上していることがわかった．

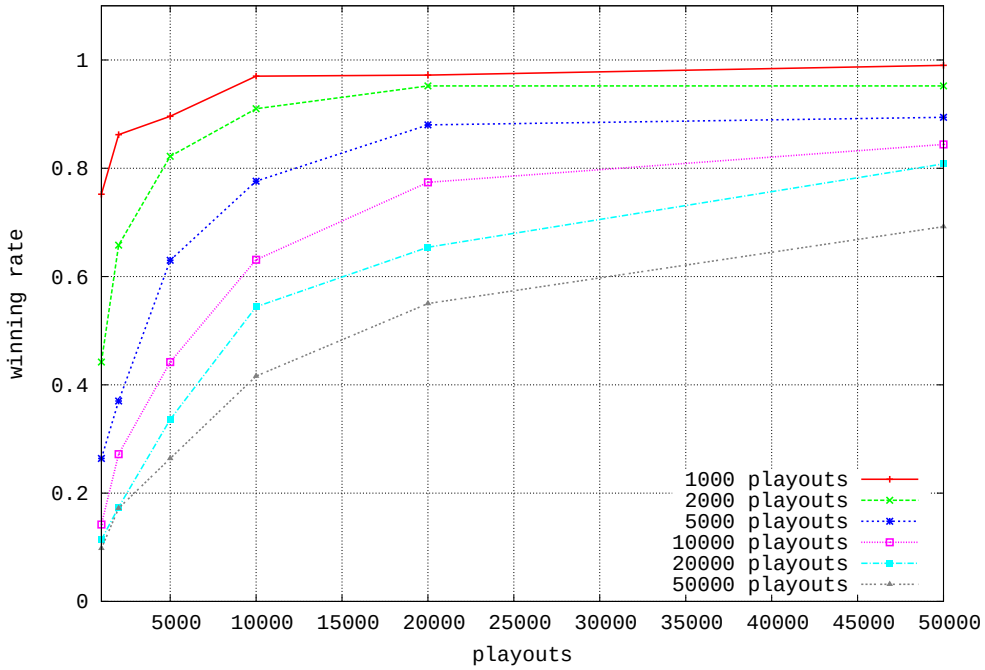


図 5: UCT 同士の対戦

6.3 UCT と Negamax の対戦と探索効率

UCT と Negamax の対戦結果を図 6 に示した．この結果から，深さ 1 の Negamax 法に対しては，負けることがなく，深さ 6 の Negamax 法に対しても試行回数が 10000 を超えれば勝率が 50% 以上となることがわかった．

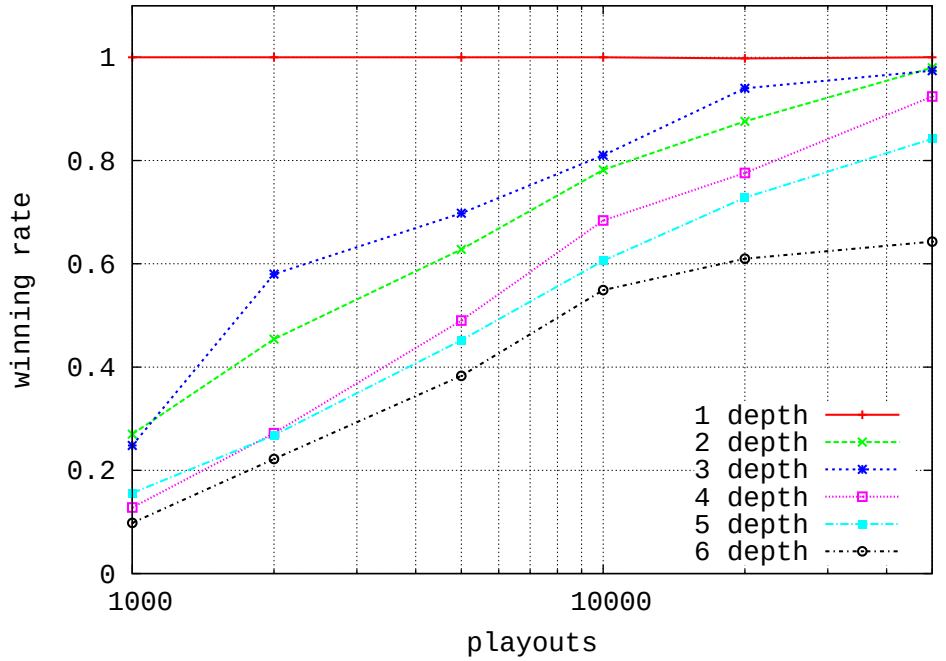
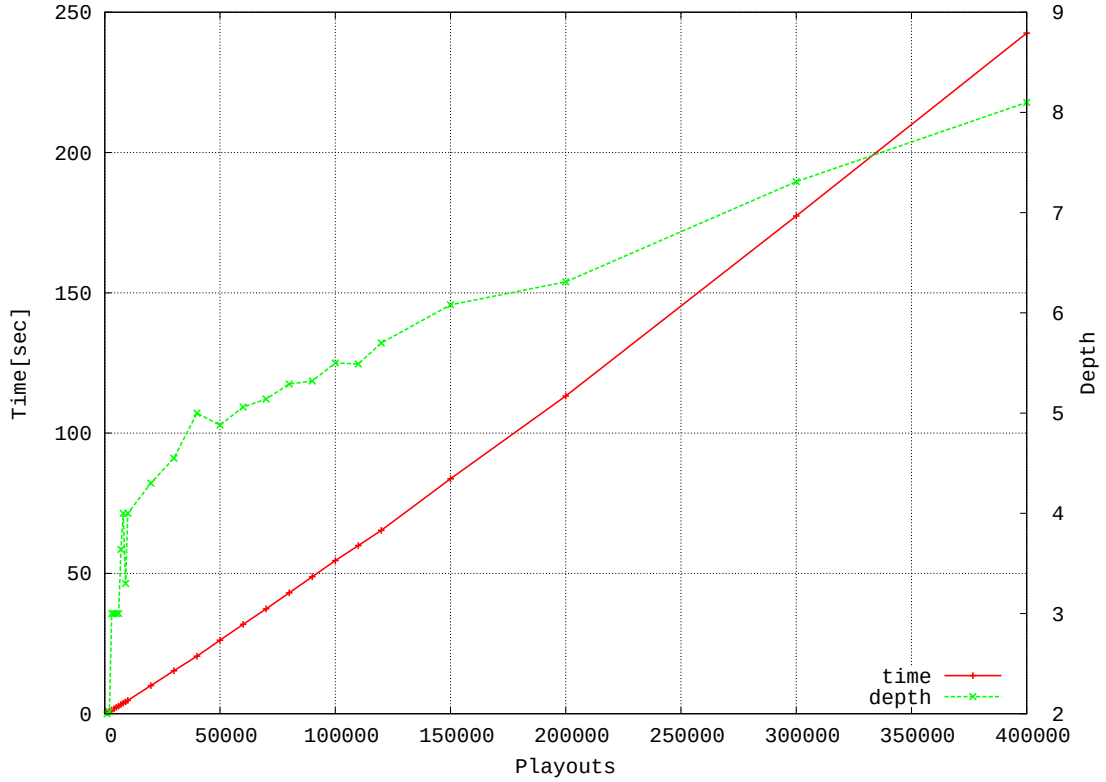
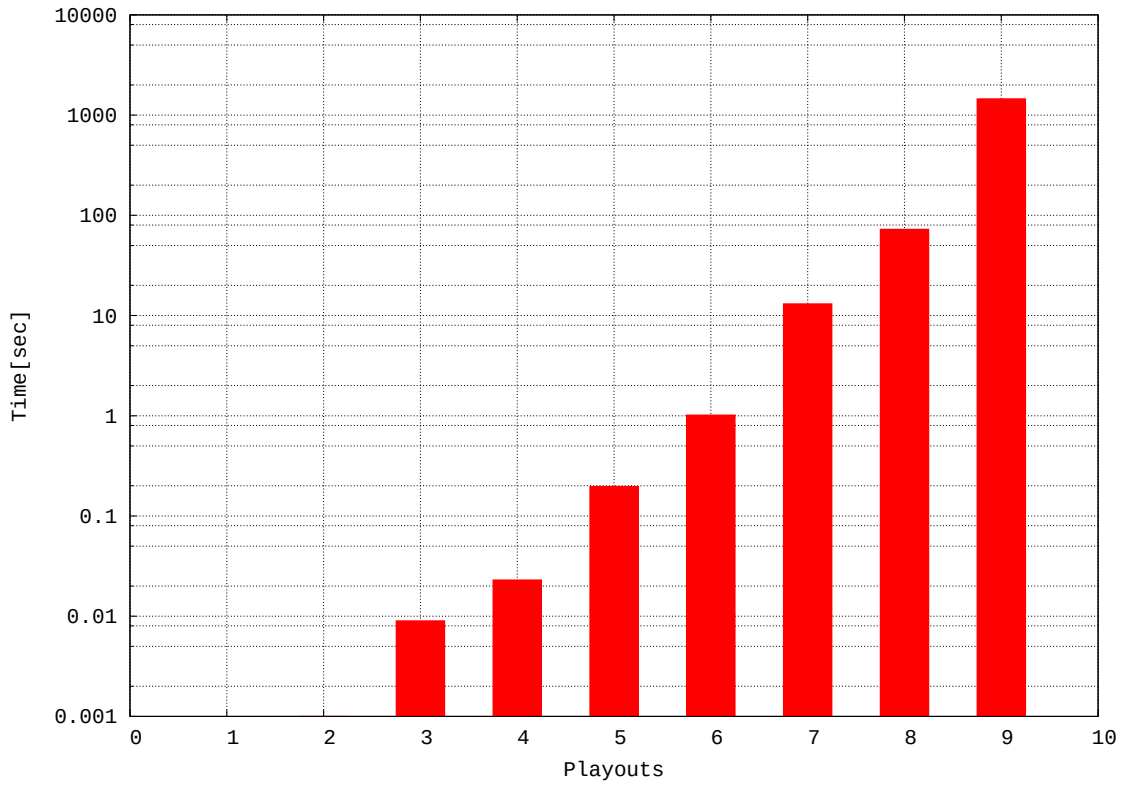


図 6: Negamax との対戦

また，探索効率の実験結果を図 7 に示した．これによって，UCT は試行回数にともなって線形に探索時間が変化し，探索の深さは徐々に大きくなっていくことがわかった．400000 回の試行で深さの平均は 8 となった．一方，Negamax は探索の深さを変化させると，指数関数的に増加することがわかった．これらは，アルゴリズム通りの結果である．



(a) UCT の探索効率



(b) Negamax の探索効率

図 7: 探索効率

7　考察

7.1　C の決定

C を求めた結果，C が 0.5 よりも小さい時や，0.7 よりも大きい時は勝率が低くなった．そもそも C は，UCB 値の第 2 項の影響をどの程度考慮するかを決める値である．UCB 値の第 1 項は勝率の高さ，第 2 項は探索した回数が少ないことをそれぞれ優先する．

したがって，C が小さければ勝率のみを重視して探索を行うため，偶発的に勝利，あるいは敗北した手に対して，探索が効率よく行われない．また，逆に C が大きすぎれば，勝率よりも探索の回数が少ないものが重視されるため，全ての手をまんべんなく試行するようになる．つまり幅優先に近い動作をすると考えられる．これによって，探索の深さが浅くなり，良い結果が得られない．

今回は，およそ 0.5〜0.7 の間で勝率が良くなったため，その平均の 0.6 を使用して他の実験を行った．

7.2　UCT 同士の対戦

UCT 同士の対戦では，試行回数を増やしていくと，勝率が上昇していく様子がわかった．したがって，プログラムが正しく動作しているとかんがえられる．

試行回数が同じときを見てみると，プレイアウト回数が 1000 回のときに 0.75 である以外は，いずれも 0.6 程度となっているとわかる．つまり，このゲームは先手のほうが勝率が高くなるゲームであると考えられ，実力が同程度であれば 0.6 程度の勝率になると推察される．

7.3　UCT と Negamax の対戦と探索効率

UCT と Negamax の対戦では，試行回数を増やすと Negamax に対して勝率が上昇した．前述のとおり先手が有利であると考えると，深さ 6 の Negamax は，プレイアウト回数が 2〜50000 回の UCT の強さに相当すると推察できる．

一方で，探索の効率の結果を見ると，深さ 6 の Negamax は 1 秒で 100 回探索が行えるが，プレイアウト回数が 50000 回の UCT では，26 秒が 100 回探索にかかっているため，UCT のほうが効率が悪いといえる．

一方で，UCT の展開の深さの平均は，50000 回のときに 5 となっている．よって，平均すると 5 手先まで見ることで深さ 6 の Negamax と同等の強さを持つこととなり，この点では効率が良いといえる．これは，有望な手を UCB 値によって正しく定められているからであると推察される．

また，Negamax は試行にかかる時間が指数的に上昇する．Negamax は，深さを 7 とすると，深さ 6 のときのおよそ 10 倍の時間がかかる．一方で，UCT では，試行回数 50000 回から 150000 回，あるいは 200000 回から 270000 回などのようにおよそ 100000 回で深さが一つ増えるが，試行にかかる時間は線形であるため，強くするには UCT のほうが有利であることが推察される．

8　結論

UCT アルゴリズムを用いて，立体 4 目並べの AI を実装し，Negamax+$\alpha\beta$ アルゴリズムと対戦させた．その結果，UCT の効率の良さと強くしやすさが分かる結果となった．立体 4 目並べは，比較的評価関数が作りやすいため，Negamax 法でも十分な強さとなったが，評価関数が作成しにくいゲームでは，UCT アルゴリズムのほうがより有利になると考えられる．

参考文献

[1] 伊庭 斉志 :『人工知能と人工生命の基礎』，オーム社 (2013.5.24)

[2] 『MinMax 法と NegaMax 法』(last accessed 2013.12.13)，http://www.es-cube.net/es-cube/reversi/sample/html/2_4.html

付録

ソースリスト

ソースのリストを以下に示した．

- Makefile
  make をするためのファイル
- board.c / board.h
  盤面についての処理をまとめたもの．盤面の表示や勝利判定，盤面のコピー，Undo などが定義されている．
- uct.c / uct.h
  UCT アルゴリズムを実装したもの．
- negamax.c / negamax.h
  Negamax 法 +$\alpha\beta$ カットを実装したもの．
- main.c
  ゲームの進行をするもの．ユーザからの入力などを受け付ける．