第1版：2010.3.26
第2版：2012.3.25

＜ym将棋の技術的特徴＞

１．概要

ym将棋では、将棋等のチェスライクゲームで一般に用いられるalpha-betaアルゴリズムによる探索（以下、通常探索と呼ぶ）とともに、モンテカルロ木探索を併用している。

以下にその手法の詳細を述べる。

２．通常探索とモンテカルロ木探索の併用とは

モンテカルロ木探索は、一般に、一直線の手順を読むのが苦手とされている。
将棋においては、▲24歩△同歩▲同飛…のような一直線の手順が多く出現するため、
モンテカルロ木探索のみで最善手を見つけるのは困難である。

しかし、駒得していながら有効な攻めが無い、というような、通常探索での評価が難しい局面でも、
モンテカルロ木探索は近似的に勝率を与えることができる。

このように、通常探索とモンテカルロ木探索には各々長所と短所があり、
通常探索の短所をモンテカルロ木探索で補うことを目指したのがym将棋である。

３．実装について（１）－併用

通常探索とモンテカルロ木探索は、別スレッドで探索を行う。（各々1スレッド）
ym将棋はUSIエンジンとして動作するため、USIコマンド受付スレッドが別に存在している。
さらに、時間監視用スレッドを別に設けているため、合計4スレッドで動作している。

探索時の処理の流れは次の通り。

a.USIコマンド受付スレッド：
　思考開始（go）コマンド受付→時間監視用スレッド起動

b.時間監視用スレッド：
　時間監視を開始し、通常探索スレッド・モンテカルロ木探索スレッドを起動

c.通常探索スレッド・モンテカルロ木探索スレッド：
　探索を行う

d.時間監視用スレッド：
　制限時間が到来すると、通常探索スレッド・モンテカルロ木探索スレッドの
　処理を終了させる
　双方の探索結果を比較していずれかを選択し、最善手をGUIに返す（後述）
　自スレッドを終了させる

４．実装について（２）－情報共有

通常探索とモンテカルロ木探索は独立して探索を行っているが、各々の探索で得た情報は共有している。
双方の探索で別々に作成・使用している局面表（ハッシュテーブル）を互いに参照することで、
共有を実現している。

情報共有は、具体的には以下のような場合に行っている。

・詰みに関する情報
　通常探索で発見した情報を、モンテカルロ木探索に提供する。
　（モンテカルロ木探索単独でも、詰みを発見することは理論的には可能だが、
　　一般に通常探索の方が早く発見するため）

５．実装について（３）－指し手の選択

通常探索とモンテカルロ木探索の結果（最善手）が異なった場合、
判定条件に従い、どちらの手を指すか決定する。

以下に判定条件を例示する。

・通常探索で詰みを発見した：通常探索の指し手
・通常探索の指し手の、モンテカルロ木探索での評価（勝率）が低い：モンテカルロ木探索の指し手
・通常探索の評価が低い（一定の評価値を下回った）：モンテカルロ木探索の指し手
…

なお、playout数が少ない場合はモンテカルロ木探索の信頼度が低いため、
制限時間内に実行できたplayout数が一定値を下回った場合は、無条件に通常探索の指し手を選択する。

６．実装について（４）－モンテカルロ木探索の詳細

(1)基本アルゴリズム

モンテカルロ木探索の基本アルゴリズムは、モンテカルロ囲碁で一般的な
UCT（UCB applied to Tree）を用い、これに独自の修正を施している。

ルートノードでは最初からUCBにより指し手の選択を行い、
それ以下のノードではランダムに指し手を決定する。
playoutが一定の回数（現在のym将棋では1回）通過したノードについては、
次回のplayoutからUCBにより指し手選択を行う。
ただし、UCBにより指し手選択を行うのは、ルートから一定の深さ（現在のym将棋では深さ5）
に制限している。
一定時間内に実行できるplayout数には限りがあり、あまり深いノードまでUCBを適用する
意味がないため、また深いノードまでUCB情報を保持するとメモリが不足するためである。

UCBの計算式はUCB1-TUNEDを基本としているが、これも修正を加えている。

(2)指し手の評価

通常探索では、killer、SEE、history heuristic等により指し手の評価を行っているが、
モンテカルロ木探索では、SEE、指し手の静的条件、history heuristicにより評価を行う。
この評価は、指し手選択や前向き枝刈りの際に用いる。

静的条件として使用しているものは次の通り。
・打つ手か否か
・取る手か否か
・移動先の利きの状態
・自玉、相手玉との距離
・駒が前に進んでいるか
・etc.

なお、モンテカルロ木探索で用いるhistory heuristicは、通常探索とは別に管理している。
playoutが終局した場合に、勝利した側の指し手を全て保存する。
打つ手、駒を取る手も保存対象である。

(3)指し手選択（１）－ランダム

ランダムに指し手を選択する場合、次の方式により行う。

a.全ての指し手を生成し、上記の評価値を設定する。
b.評価値を重みとしたルーレット方式により指し手を選択する。

例：指し手A=評価値100、指し手B=評価値50、指し手C=評価値20の場合、
各指し手が選ばれる確率はA:B:C=10:5:2となる。

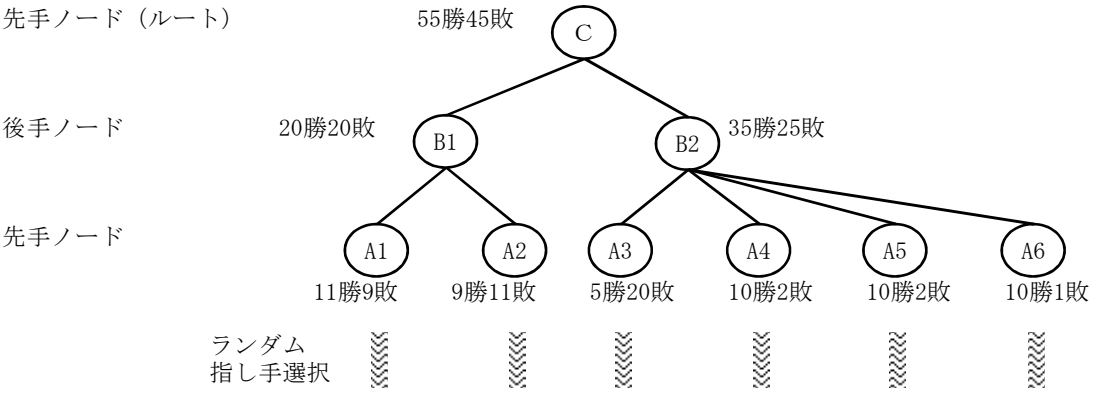
best-of-nアルゴリズムは、現時点では用いていない。

(4)指し手選択（２）－ＵＣＢ

前述の通り、UCB-TUNEDアルゴリズムを基本としている。
大きな修正点は次の点である。

a.ノード勝率の計算法
一般に、ノードの勝率は、そのノードを通過した全playoutの勝敗から算出される。
しかし、そのノードからある指し手Aを選んだ場合の勝率が7割で、別の指し手Bを選んだ場合の勝率が4割であった場合、必ず指し手Aを選ぶべきだとは限らない。

例として、次のような探索木を考える。（勝敗は先手からみたもの）



playoutを100回実施した結果、ルートノードでの先手勝率は0.55（先手有利）であり、
最善手はＢ２となる（Ｂ１より勝率が高いため）。
だが、先手がＢ２を指した後の、後手の最善手はＡ３である。（先手勝率が最も低い）
後手がこれを指すと、ノードＡ３での先手勝率は0.2しかなく、先手不利となってしまう。
本来選択すべき手順は、Ｃ→Ｂ１→Ａ２である。

これは、ノードＢ２の勝率を算出する際、後手にとっての最善手であるＡ３以外に、
後手不利となるＡ４～Ａ６の指し手を全て同等に評価していることが原因である。
つまり、ノードＢ２で先手勝率が高いのは、後手が手を間違うことを前提として
評価を行っているためである。
評価関数とalpha-betaアルゴリズムを使用する探索では、このようなことは発生せず、
単純に勝率だけで指し手を評価するために起きる事象と考えられる。
（なお囲碁では、最善手とそれ以外の差が小さく、この事象の影響が小さいと考えられる）

この問題の影響を低減するため、ym将棋では以下の手法をとっている。

・あるノードでの勝率を計算する際、子ノードの中で最善の勝率を、そのノードの勝率とする。

上記の例でいえば、ノードＢ２の勝率を計算する際、子ノードＡ３～Ａ６の中で
後手にとって最善なのはＡ３であるから、Ａ３の勝率＝0.2をそのままＢ２の勝率とする。
同様にノードＢ１でも、子ノードＡ２の勝率＝0.45をそのままＢ１の勝率とする。
ルートノードＣでは、Ｂ１とＢ２の勝率を比較した結果、Ｂ１の方が高いと判断し、
Ｂ１に多くのplayoutを割り当てる。また指し手選択でもＢ１を優先する。

なお、playout数が少ない間は勝率の誤差が大きいことから、ノード全体の勝率も
加味するよう考慮している。

Discounted UCB(*1)は、現時点では用いていない。
(*1)直近のplayout結果を優先して取り扱う計算法。
遠い過去のplayoutほど、勝率への寄与分を小さくする。

(5)終局の扱い
playoutは終局まで続けず、一定の手数で打ち切っている。
ルート局面とplayout終了局面の静止探索評価値を比較し、その大小によって
playoutの勝ち/負けを決定し処理する。[1]

(6)前向き枝刈り

UCBにより指し手選択を行う場合、ルートノード以外では、指し手の数に比較して
十分なplayout数を確保できない。
単純なUCBアルゴリズムでは、全ての指し手についてまず1度playoutを割り当てることが
前提となっているが、それすらも不可能なことが多い。

このため、前向き枝刈りを行い、見込みの無さそうな手にはplayoutを割り当てず、
有望な手に多く割り当てるようにする。

枝刈りのアルゴリズムはprogressive wideningを基本としている。
すなわち、完全に枝を刈ってしまうことは無く、通過playout数に応じて、
徐々に枝の数を拡大してゆくため、playout数が無限であれば、いずれは全ての手が探索される。
その方式は次の通り。

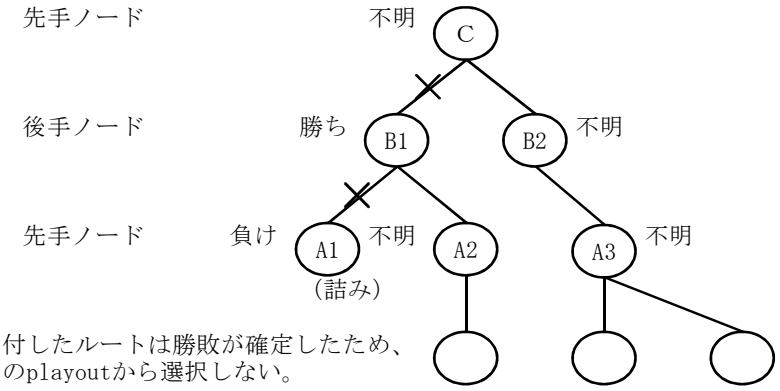
a.ある指し手がまだ1度も選択されていない場合、
その指し手の評価値を基に仮の勝率を決め、
仮の勝率を基に（仮の）UCBを決める。
1度でも選択されていれば、通常通りUCBを算出する。

b.通常通り、最大のUCBを持つ指し手を選択する。

評価値が高ければ仮の勝率が高くなり、したがって（仮の）UCBも大きくなるため、
1度も選ばれていない指し手の中では、評価値の高い手が選ばれやすくなる。
仮の勝率に差をつけることにより、徐々に枝の数を増やすような枝刈りを行うことが
可能である。

(7)確定勝利優先アルゴリズム[2]

モンテカルロ木探索中に、詰みを発見した場合、そのルートを通るplayoutを
再度実行する意味はない。
そのため、あるノードAで詰みを発見したら、ノードAを「負け」とマークしておき、
次回のplayoutから、ノードAに至る指し手を選択しないようにしている。
また、
・あるノードAが「負け」であるとき、その親ノードBは「勝ち」である
・あるノードCの子ノード全てが「勝ち」であるとき、ノードCは「負け」である
ことを利用し、探索の効率化を図っている。



×を付したルートは勝敗が確定したため、
次回のplayoutから選択しない。

７．実装について（５）－通常探索の詳細

基本的には全幅探索＋静止探索。

(1)全幅探索部分

○枝刈り
- beta cut
- Null-move pruning
- Futility pruning
- 歩角飛、2,8段目の香の不成は指し手を生成しない
- 末端からの深さが2までは、SEEが負となる指し手を枝刈り（水平線効果対策）

○反復深化

○Aspiration search

3段階。徐々にwindow幅を広げる。
最初の2段でfail highした場合、3段目でwindowを（-∞、∞）とし探索
残り時間が少ない場合は、1,2段目をスキップする（つまり、Aspiration searchを行わない）

○PV search

○extension/reduction
- extension ： 王手/駒の取り返し
- reduction ： 局面の静的な状況から、重要でないと判断される手

○move ordering
- Transposition table
- 一手詰みの王手
- 駒損にならない駒を取る手
- Killer moves
- SEE(Static Exchange Evaluation)
- History heuristics
- MVV-LVA (Most Valuable Victim - Least Valuable Aggressor)
- 指し手の種類（成る手→成る手以外、盤上の駒を動かす手→打つ手）

○詰将棋
- df-pn
- PV nodeのみ
- ルート局面では全ての王手を生成、ルート局面以外では一部の王手（開き王手等）を生成しない

○水平線効果対策
- 疑似探索延長[3]

  A→B→Cの指し手列が最善手と判断されたとき、A→Bが特定の（水平線効果となりやすい）組合せであれば、その2手を指さずにCを指した場合の評価値と比較する。
  比較の結果、水平線効果と判断された場合、その指し手列は大きなマイナス評価とする。
- 持ち駒の優劣を用いた比較

(2)静止探索

2段階での探索。

○1段目：以下の指し手を探索（SEE、盤面の状況、駒の種類による条件あり）
- 駒を取る手
- 成る手
- 一手詰みの王手
- 取りをかけられた駒が逃げる手

手番側は、手を指すかstand patを返すかを選択する。
stand pat時の評価には、いわゆる「脅威」を考慮している。[4]
（一手パスした時に、取られそうな駒がある場合、評価値をマイナス）

○2段目：そのマス目での取り合いのみを探索

1段目のみ、静止探索専用のTransposition tableを使用している。
評価値のみを保持し、最善手は保持しない。

(3)評価関数
○学習
・TreeStrap（αβ）を使用[5]
・他プログラムとの対戦により学習

○評価要素
・2駒間の関係
・3駒間の関係の一部（王－銀以上の駒－銀以上の駒）

※現バージョンでは、駒割は固定値とし、学習対象としていない

(4)その他
○千日手対策
・基本的には評価値ゼロの扱い
・ただし、残り時間が相手より少ない場合は、プラス評価とする（千日手を歓迎する）

参考文献

[1]竹内聖悟、金子知適、山口和紀：将棋における，評価関数を用いたモンテカルロ木探索、
ゲームプログラミングワークショップ２０１０、pp.86-89（2010）

[2]伊藤毅志、佐藤慎太郎：モンテカルロ木探索における確定勝利優先アルゴリズム、
ゲームプログラミングワークショップ２００８、pp.140-143（2008）

[3]将棋ソフト将皇　http://www14.big.or.jp/~ken1/application/index.html

[4]将棋プログラムKFEnd　http://www31.ocn.ne.jp/~kfend/

[5]宇賀神拓也、小谷善行：将棋におけるTree Strapに基づく評価関数の学習
ゲームプログラミングワークショップ２０１０、pp.114-118（2010）

以上