コンピュータ将棋における全幅探索と futility pruning の応用

保木邦仁（東北大学院理学研究科化学専攻）
khoki@mail.tains.tohoku.ac.jp

5 月に行われたコンピュータ将棋選手権において，拙作の Bonanza が接戦のリーグ戦をすり抜け，幸運に助けられながらも優勝することができた．Bonanza の思考アルゴリズムは，チェスで広く用いられている全幅探索の手法に基づく．将棋においても，全幅探索が有効な手法の 1 つになり得ることが示された．本稿では，このプログラムの仕組みを，探索アルゴリズムの概要と，特に将棋ドメインにおける futility pruning の応用に的を絞り，解説する．Futility pruning を行うことによるプログラムの棋力上昇が，次の一手問題の正答率に基づいて示された．

コンピュータ将棋プログラム Bonanza の特徴

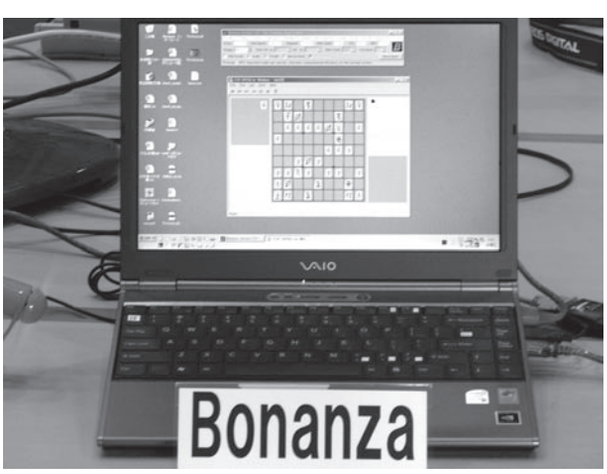
Bonanza（**図 -1**）の思考アルゴリズムは，コンピュータチェスやオセロにおいて広く用いられている全幅探索の手法に基づく．この手法では，minimax 法による探索アルゴリズムを用いて，すべての可能な指手をしらみつぶしに調べ上げることにより局面の最善手を予想する．これとは対照に，現在の主要な商用将棋プログラムの思考アルゴリズムは選択探索に基づく[1]．この手法は，ゲームの知識を考慮して，戦略的に重要と思われる手を重点的に調べる手法である．将棋においては取った駒が再利用できる持ち駒制度が存在し，場合の数が将棋（10^{220}）とチェス（10^{120}）では大きく異なる．したがって，探索範囲の削減を目的とした選択探索は将棋プログラムを強くするのに必要とされてきた．

将棋の場合，可能な指手の数は平均 80 程度であり，すべての可能な指し手を十数手先まで調べ上げるのは一見不可能に思える．しかし，全幅探索においても，枝刈りの手法を用いることにより探索範囲を大幅に削減しつつ，minimax 法とほぼ同じ探索結果を得ることが可能である．Bonanza では以下の枝刈りの手法を用いる．

- Alpha-beta pruning[2]
- Null move pruning[3]
- Futility pruning[4]

本稿では，Bonanza の探索アルゴリズムの特徴の 1 つである，将棋へ応用された futility pruning について解説する．この手法により，“安全に”，平均して 50％から 90% 程度の探索範囲が削減される．Futility pruning とは，alpha-beta 法，静止探索と静的評価関数を組み合わせた全幅探索において，探索の末端付近の振る舞いの性質を利用した枝刈りの手法である．次章以降では，futility pruning を応用するにあたり基本となる alpha-beta 法，静的評価関数，静止探索について概説する．さらに，チェスにおける futility pruning の紹介をした後，将棋プログラム Bonanza において用いた応用法を解説する．

本稿では紙面の都合上 null move pruning に対する説明は行わないが，興味のある方は参考文献を当たられたい．この枝刈りの手法により，さらに，平均して 90%



◘ 図 -1　Bonanza

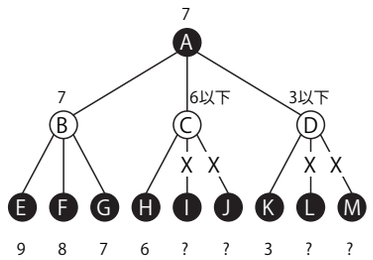
以上の探索範囲が削減される．この枝刈りの手法では，手番を 1 回放棄すると形勢が悪化するというゲームの性質を利用する．この手法はチェスの場合よりも将棋の方がうまく働く．これは，特にチェスの終盤で重要となる zugzwang 局面が，将棋の場合において実質現れないためである．

他の Bonanza のアルゴリズムの特徴としては，詰み探索ルーチンを持たないことが挙げられる．序・中・終盤の区別なく，すべての思考時間を全幅探索に費やす．一方，現在の主要な将棋プログラムは，専用の詰み探索ルーチンを持つ．詰みの発見はそのまま勝ちにつながるため，詰み検索は将棋プログラムを強くするのに必要とされてきた概念の 1 つである．

近年の詰め将棋における探索アルゴリズムは発展目覚しく，証明数という概念を用いた探索アルゴリズムを応用することにより，百手以上の詰め将棋を短時間に解くことが可能である [5]．一方，通常の探索では，王手における探索延長を行っても 21 手程度の詰め手順しか発見することができないが，短所ばかりではない．全幅探索においては，攻め方の指し手として王手以外のすべての可能なものも調べ上げる．終盤においては必死の概念は重要であり，手順に王手以外の指し手を考慮することは終盤力の向上につながる．また，詰み探索においては，局面の詰み・不詰みを把握するのみであるが，通常の探索においては，十数手に及ぶ連続王手の後に見える，駒得等の状況を把握することが可能である．

Bitboard による盤面構造の取り扱いと静的評価関数の自動調整も Bonanza の特徴といえる．Bitboard はコンピュータチェスにおいて開発された技術である．盤面の駒の配置をビットマップとして保持し，駒の利きが論理演算を用いて比較的容易に取得される．利きの保持，更新等の処理を行う必要がなく，盤上の駒を頻繁に動かすプログラムに向いている手法といえる．

静的評価関数の自動調整は変分法の手続きに従い，目的関数を最大にするパラメタを求めることにより行われた．目的関数は，主に 6 万局からなる棋譜の指し手と思考プログラムの指し手の一致率を反映するよう設計する．十分に経験を積んだ人間が手で調整した評価関数と，この手法により自動生成されたそれとの性能の優劣は不明である．しかし，この手法は 1 万を超すパラメタを調整するなど，人の手で作成するのは実質不可能な場合においても安定に動作する．



◘ 図 -2　Minimax 木概念図．黒丸は先手番の局面，白丸は後手番の局面を示す．また，alpha-beta 法に基づき左から深さ優先探索を行った場合，枝刈りされる枝を X で示す．

筆者の知る限り，コンピュータチェスの分野においてでさえ，選択探索 vs. 全幅探索の議論には決着が付いていない．1997 年，当時の世界チャンピオン Garry Kasparov が IBM Deep Blue に敗れるあたりまで全幅探索の手法は全盛を極めた．しかし，2000 年以降に出現した Deep Junior や Fritz 等の主要な商用チェスプログラムは，選択探索のアルゴリズムに基づくと言われている．

5 月に行われたコンピュータ将棋選手権において，拙作の Bonanza が接戦のリーグ戦をすり抜け，幸運に助けられながらも優勝することができた．このプログラムを通した実験により，将棋の分野においても，選択探索 vs. 全幅探索の議論を提唱できたのではないかと考えている．

Alpha-beta 法・静的評価関数・静止探索の概要

図 -2 に示される minimax 法に基づいたゲーム木探索を考える．局面 A をルートとし，2 手先まで探索する．簡単のため，一局面あたりの指し手の数を 3 とした．また，黒丸が先手，白丸が後手番の局面を表す．E 〜 M の末端の局面で静的評価関数を呼び，それぞれの局面に適当な得点をつける．ここで，高い評価値が先手にとって有利な局面である．後手が先手にとって都合の悪い手を選択すると仮定すると，先手の最良の選択は A → B，得点が 7 となる．

全局面数は A 〜 M の 13 であるが，alpha-beta pruning により，局面 I, J, L, M は展開されない．局面 H を展開した時点で C の得点は 6 以下，すなわち B の得点よりも小さいことが判明する．よって，局面 I, J は展開する必要がない．局面 L, M についても同様である．

Alpha-beta 法の威力は，指し手の数が多くなるほど

```
(a)
 1: int alpha _ beta( int depth, int alpha, int
    beta )
 2: {
 3:   if ( depth == 0 ) return evaluate();
 4:
 5:   generate _ all _ moves();
 6:
 7:   while ( move = next _ move() )
 8:     {
 9:       make _ move( move );
10:
11:       value = -alpha _ beta( depth-1, -beta,
    -alpha );
12:
13:       unmake _ move( move );
14:
15:       if ( beta <= value )  return beta;
16:       if ( alpha < value )  alpha = value;
17:     }
18:
19:   return alpha;
20: }

(b)
 1: int quies( int alpha, int beta )
 2: {
 3:   stand _ pat = evaluate();
 4:
 5:   if ( beta <= stand _ pat ) return beta;
 6:   if ( alpha < stand _ pat ) alpha =
    stand _ pat;
 7:
 8:   generate _ tactical _ moves();
 9:
10:   while ( move = next _ move() )
11:    {
12:       make _ move( move );
13:
14:       value = -quies( -beta, -alpha );
15:
16:       unmake _ move( move );
17:
18:       if ( beta <= value )  return beta;
19:       if ( alpha < value )  alpha = value;
20:    }
21:
22:   return alpha;
23: }
```

図 - 3　(a) Alpha-beta 法に基づき探索を行うプログラム例．簡単のため擬似的な C 言語で記述し，文法の厳密性などは考慮しない．3 行目で呼び出す evaluate() は静的評価関数．9 行目の make_move(move) は指し手 move による局面の更新，同様に unmake_move(move) は局面の更新を元に戻す関数．(b) 静止探索を行うプログラム例．手番を持つ側は，stand pat を返すか，一手進めて手番を渡すか選択する．8 行目の generate_tactical_moves() では，駒を取る手等戦略的に重要な手のみを生成．

顕著に現れる．指し手の数が 100 での 2 手先までの探索を，図 -2 の場合と同様に考えると，全局面数は 10,101，枝刈りされる局面の数は 9,801 であることは容易に確かめられる．ただし，図 -2 のように，展開する指し手が順序よく並んでいると仮定した．

この alpha-beta 法と静的評価関数による探索を実現する擬似的な C プログラムを**図 -3(a)** に示す．以下のように関数 alpha _ beta() を呼ぶことにより，3 手先まで探索を行う．

```
score=alpha _ beta (2, INT _ MIN, INT _ MAX);
```

3 行目の evaluate() は静的評価関数であり，深さが 0 に達したときに呼ばれる．11 行目では alpha _ beta() 自身が呼ばれ，再帰的に子局面を探索する．Alpha-beta pruning が実際に行われるのが 15 行目であり，子局面の探索結果が上限値，beta 値以上ならば，指し手の展開を打ち切り beta 値を返す．16 行目では，この局面の下限値 alpha が更新される．

図 -3 (a) で示される例のように，ゲーム木を深さ一定で探索する minimax 法をそのまま将棋に用いる試みでは，安定した結果を得ることは難しい．この不安定性は，駒の取り合い等，静的評価関数の値が大きく変化する手順を一定の深さで突然打ち切った場合，顕著に現れる．

図 -4(a) に示される例は，先手が▲ 2 二角成と後手の角を取り込んだところである．この馬は次の後手の指し手により取り返されることになるため，実質，形勢は駒の損得なしで互角である．しかし，この局面で探索が打ち切られた場合，先手が大きく駒得し優勢と誤った判断をしてしまう．静的評価関数自身が，戦略的に重要な手順を考慮したものになっていれば，このような問題は起きない．しかし，一般に，このような状況を静的に評価するのは非常に難しい．

静止探索は，静的評価関数の値が大きく変化する手順を動的に展開し，安定した結果を得ることを目的とした探索である．通常の探索の末端において，静的評価関数を評価する代わりに，この付加的な探索を行う．

図 -3(b) に静止探索のコードの例を示す．この関数 quies() は，図 -3 (a) 3 行目 evaluate() の代わりに呼ばれる．手番を持つ側は，戦略的に重要と思われる手（駒を

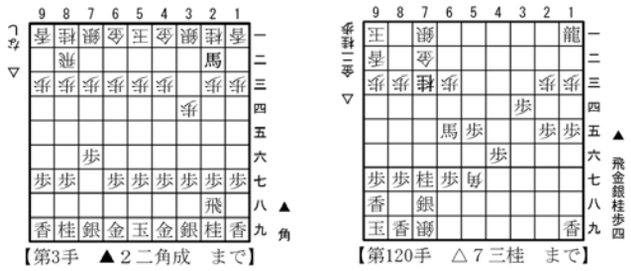


図 -4　(a) 水平線効果の例．後手番で，2 二の馬は次の指し手で取り込まれる．したがって，図の局面で探索を打ち切った場合，信頼のできる評価を得るのは難しい．(b) 次の一手問題の一例．▲７四歩が正解とされる．

取る手等）を指すか，何もせず静的評価関数の値，stand pat を返すか選択する点が，alpha-beta 法の場合と異なる．3 行目において評価された stand pat が上限の beta 値以上の場合，この stand pat によって局面が beta 枝刈りされる．

チェスにおける futility pruning 概要

ゲーム木の末端の親局面（静止探索関数を直接呼ぶ局面と，静止探索中の局面）において枝刈りを行い，計算量を削減することを考える．この親局面は frontier node と呼ばれる．Futility pruning は，図 -3 (b)，5 行目で行う stand pat による beta 枝刈りを，子局面を展開することなく，frontier node で認識することにより行われる．

局面 p の評価値 E(p) が，駒割と駒の位置関係の 2 つの部分からなり，E(p) における駒割の占める割合が位置関係のそれよりも十分大きいという性質に着目する．ここで，以下の不等式を考える．

$$M(p) - Vmax \leqq E(p) \quad (1)$$

ただし，M(p) は局面 p の駒の種類と数，すなわち駒割と，手番のみから決定される評価値を表す．Vmax は正定数であり，E(p) への駒の位置関係の寄与の最大値に対応する．

図 -3 (b)，5 行目の条件，

$$beta \leqq E(p) \quad (2)$$

の真偽を確認するためには，静的評価関数 `evaluate()` を呼ぶ必要がある．一般に，この関数の評価は非常に重い処理となる．そこで，簡単に評価可能な M(p) と正定数 Vmax を用いて条件 (2) の予備テストを行い，可能な限り `evaluate()` を呼ぶ回数を削減する．条件 (1) より，条件 (2) を真とするのに十分な条件は以下の条件式で表される．

$$beta \leqq M(p) - Vmax \quad (3)$$

条件 (3) の p を，親局面 P に書き直すことにより，frontier node における futility pruning の表式が得られる．

$$M(P) + Mcap(m) + Vmax \leqq alpha \quad (4)$$

ただし，Mcap(m) は指し手 m による駒割りの変動を表す．ここで，M(P)+Mcap(m)= -M(p) の関係が成り立つ．指し手に基づき，局面を親 P から子 p へ進める処理，図 - 3 における `make _ move(move)` や `unmake _ move(move)`，は比較的重い処理となる．式 (3) の代わりに式 (4) を用いることにより，さらなる計算量の削減が図れる．

式 (4) を用いた枝刈りは，Vmax の値が小さいほど効率的に働く．チェスの場合，終盤において盤上の駒の数が変化する等の特殊な条件を除いて，Vmax の値は将棋と比較して小さい．だいたいの目安として，Vmax をポーン 3 個分程度の値に設定し，安全な枝刈りが行われる．

この枝刈りの手法の考えを推し進め，frontier node のさらに親の局面，pre-frontier node においても枝刈りを行う extended futility pruning の手法が Heinz らにより提唱された [4]．

$$M(P) + Mcap(m) + Fmgn \leqq alpha \quad (5)$$

ここで，Fmgn は探索残り深さが 2 の時，手番側のプレイヤが回復可能な最大評価値に対応する．この推論は，残り深さが少ない場合での，チェスにおけるゲーム木の性質に基づく．王手や，終盤において駒を取る手以外の指し手により，評価値を大きく回復することはまれである．だいたいの目安として，Fmgn をポーン 5 個分程度の値に設定し，安全な枝刈りが行われる．

futility pruning の将棋への応用

式 (4) を用いた枝刈りの手法を将棋に応用した場合，評価値 E(P) への駒の位置関係の寄与 Vmax が大きく，十分な枝刈りを行うことはできない．そこで，指し手による評価値の変化 E(P) + Mcap(m) – -E(p) の絶対値が Vmax よりも小さいことに着目し，frontier node に対する枝刈りの条件 (4) を以下のように書き換える．

$$E(P) + Vdiff(m) \leqq alpha \quad (6)$$

ただし，Vdiff(m) は指し手 m から決定される評価値の変化の最大値であり，

$$-E(p) \leqq E(P) + Vdiff(m) \quad (7)$$

を満たす．本稿の実験では，Vdiff (m) を以下のように設定した．

$$Vdiff(m)=Mcap(m) + Mpiece\text{-}king(m) + Vmax(m) \quad (8)$$

ここで，Mpiece-king(m) は，王以外の駒が移動した場合，位置が変わった駒と玉の位置を考慮した評価値の変化値，Vmax(m) は駒の移動の種類（王の移動・王以外の移動）

に対応する定数を表す．この定数に，王の移動の場合歩の交換値 12 個，王以外の移動の場合歩の交換値 4 個分程度の値を設定し，安全な枝刈りが行われる．

同様に，pre-frontier node に対する枝刈りの条件 (5) は以下のように変更される．

$$E(P) + Vdiff(m) + Fmgn \leqq alpha \quad (9)$$

Fmgn には，歩の交換値 2 個分程度の値を設定し，安全な枝刈りが行われる．

評価値の変化の最大値 Vdiff(m) が小さいほど，(9) 式による枝刈りの効率がよくなる．この条件が偽の場合には実際に局面を動かし，さらに精度よく評価値を求める．式 (9) と (7) を用いて pre-frontier node の子局面における枝刈りの条件を得る．

$$beta \leqq E(p) - Fmgn \quad (10)$$

図 -5 に，この futility pruning の実践例を示す．式 (6) による枝刈りは図 -5 (a) 15 〜 17 行と，図 -5 (b) 12, 13 行に行われる．また，式 (9) による枝刈りは図 -5 (a) 19 〜 21 行，式 (10) による枝刈りは図 -5 (a) 7 〜 9 行に行われる．

この手法では，静止探索の場合に加え，通常の探索部においても frontiernode と pre-frontier node で静的評価関数 `evaluate()` を呼ぶ必要がある．しかし，この追加された計算による探索時間増加は，探索局面数の大部分は静止探索部が占めることから，それほど多くならない．また，式 (4), (5) による，駒割の値のみからなる枝刈りもあわせて使用することが可能である．この場合，Vmax の値に，歩の交換値 15 枚程度に設定し，安全な枝刈りが行われる．また，Vmax, Vmax(m) の値は，静的評価関数を計算するつど妥当性をチェックし，探索中動的に増加させると効率が良い．

表 -1 は，図 -4 で示される 2 つの局面を探索するのに必要としたノード数を示す．プログラムに用いたアルゴリズムは付録に示される．局面により futility pruning の効率は異なるが，平均して，序盤 50%，終盤 90% 程度の局面が安全に枝刈りされる．この枝刈りにより短縮される計算時間は，序盤 45%，終盤 85% 程度となる．また，主に中・終盤の局面から構成される次の一手問題集 2,000 問 [6] を一問 10 秒の時間制限の下で解かせた

```
(a)
 1: int alpha_beta( int depth, int alpha, int
    beta )
 2: {
 3:   if ( depth == 0 ) return quies( alpha, beta
    );
 4:
 5:*  stand_pat = evaluate();
 6:
 7:*  if ( ! in_check
 8:*       && depth + 1 <= EXT_F_DEPTH
 9:*       && beta <= stand_pat - EXT_F_MGN )
    return beta;
10:
11:   generate_all_moves();
12:
13:   while ( move = next_move() )
14:     {
15:*      if ( ! move_is_check( move )
16:*           && depth == 1
17:*           && stand_pat + Vdiff( move ) <=
    alpha ) continue;
18:
19:*  if ( ! move_is_check( move )
20:*           && depth <= EXT_F_DEPTH
21:*           && stand_pat + Vdiff( move ) +
    EXT_F_MGN <= alpha ) continue;
22:
23:       make_move( move );
24:
25:       value = -alph_beta( depth - 1, -beta,
    -alpha );
26:
27:       unmake_move( move );
28:
29:       if ( beta <= value )  return beta;
30:       if ( alpha < value )  alpha = value;
31:     }
32:
33:   return alpha;
34: }

(b)
 1: int quies( int alpha, int beta )
 2: {
 3:   stand_pat = evaluate();
 4:
 5:   if ( beta <= stand_pat ) return beta;
 6:   if ( alpha < stand_pat ) alpha =
    stand_pat;
 7:
 8:   generate_tactical_moves();
 9:
10:   while ( move = next_move() )
11:    {
12:*      if ( ! move_is_check( move )
13:*           && stand_pat + Vdiff( move ) <=
    alpha ) continue;
14:
15:       make_move( move );
16:
17:       value = -quies( -beta, -alpha );
18:
19:       unmake_move( move );
20:
21:       if ( beta <= value )  return beta;
22:       if ( alpha < value )  alpha = value;
23:    }
24:
25:   return alpha;
26: }
```

◘ 図 -5　将棋への futility Pruning 応用例．(a) は alpha-beta 法に基づいた探索，(b) は静止探索を行う．行頭の * マークは，図 -3 から変更された部分を示す．`in_check` は王手がかかっている局面，`move_is_check(move)` は指し手 `move` が王手の場合，真になる．

	futility pruning なし	futility pruning あり
図 -4(a)，基準深さ 10	10,600,000・△同銀　・　0	5,490,000・△同銀　・　0
図 -4(b)，基準深さ 6	11,570,000・▲７四歩・ 969	1,270,000・▲７四歩・ 969

◘ 表 - 1　図 -4 の局面を探索するのに必要としたノード数・指し手・評価値

（使用マシンは AMD Opteron 252, 2.6GHz）．正答数は futility pruning なしの場合 867 問，ありの場合 928 問であった．

謝辞　　本研究を行うにあたり，有用な助言をしていただいた山下宏様と金子知適博士に謝意を表します．

参考文献

1) Iida, H., Sakuta, M. and Rollason, J.: Computer Shogi, Artificial Intell, Vol.134, pp.121-144 (2002); 松原　仁 編著：コンピュータ将棋の進歩 Vol.1-5，共立出版 (1996-2005).
2) Knuth, D. E. and Moore, R. W.: An Analysis of Alpha-beta Pruning, Artificial Intell, Vol.6, pp.293-326 (1975).
3) Beal, D. F. : Experiments with the Null Move, Advances in Computer Chess 5 (Ed. D. F. Beal), Elsevier Science, pp.65-79 (1989).
4) Heinz, E. A. : Extended Futility Pruning, ICCA Journal, Vol.21, No.2, pp.75-83 (1998), http://supertech.lcs.mit.edu/~heinz/dt/node18.html
5) Seo, M., Iida, H. and Uiterwijk J. W. H. M. : Artificial Intell, Vol.129, pp.253-277 (2001); 長井　歩，今井　浩：df-pn アルゴリズムと詰将棋を解くプログラムへの応用，情報処理学会論文誌，Vol.43, No.6, pp.1769-1777 (2002); Kishimoto, A. and Müller, M. : A Solution to the GHI Problem for Depth-First Proof-Number Search (Long version), Information Sciences Vol.175, Issue 4, pp.296-314 (2005); Kishimoto, A. and Müller, M.: Df-pn in Go: An Application to the One-Eye Problem, Advances in Computer Games 10, pp.125-141(2003).
6) 月刊「近代将棋」別冊付録，ナイタイ出版，昭和 49 年 2 月号～平成 18 年 1 月号．

（平成 18 年 7 月 11 日受付）

付　　録

テスト計算を行ったプログラムに用いたアルゴリズム概要を，以下に列挙する．

通常の探索部
- 2, 8 段目の香の不成り，飛角歩の不成りの指し手は探索しない
- 反復深化
- aspiration search．ウインドウの幅は歩の交換値 2 つ分
- pv search
- 延長 : 王手 (1 手)，one reply (0.5 手)，リキャプチャ (0.5 手)

指し手の順序並び替え
- transposition table．静止探索中は局面の参照，登録を行わない．
- 再帰的反復深化（pv node で hash move が手に入らない場合のみ）
- static exchange evaluation (SEE)
- killer moves
- history heuristics

枝刈り
- beta cut
- null move pruning (残り深さ depth が 1 の場合は行わない．depth が 7 以上の場合 R = 3，その他 R = 2)
- futility pruning．depth = 2 の場合の extended futility pruning においては，図 -5 (a)，9 行目の `stand_pat` の代わりに `quies()` を用いた．また，図 -5(a)，16, 20 行目の変数 `depth` は，指し手 `move` による延長込みの深さ．`EXT_F_DEPTH` は 3 とした．さらに，式 (4), (5) による，駒割の値のみを使用した futility pruning もあわせて使用した．

静止探索
- 静止探索を開始してから深さ 7 段目まで SEE の下で駒損しない取る手，成る手，王の移動，一手詰の王手を生成
- 8 段目手目以降は，歩を取る成らない手を除いた，駒損しない駒を取る手を生成
- SEE による指し手の順序並び替え

静的評価関数で考慮する局面の特徴
- 駒割（歩の交換値は 100 点程度）
- 玉と他の駒 2 つの位置
- 玉と，玉に隣接した（周囲 8 枡の）味方の駒と，他の味方の駒 3 つの位置
- 隣接しあった駒 2 つの位置関係
- 竜馬飛角桂香の利き上にいる駒の種類
- 竜馬飛角香が動ける枡の数
- pin analysis
- 角と同じ色の枡にいる味方の歩の数
- 歩，桂が前進，銀が後退できるか
- 竜飛香の前・後の歩
- 王の周囲 25 枡の利きの配置