

Blunder のアルゴリズム

下山 晃

1 Blunder とは

Blunder は、2008 年度の卒業論文をきっかけに作成しているコンピュータ将棋プログラムである。

名前は、チェス用語で「大悪手」「大失敗」などといった意味の言葉から。

使用言語は C#、規模は学習やその他ツール等を全て含めて 60kstep 程度。

2 主なアルゴリズム

Blunder は、主な既存のアルゴリズムとして、以下を実装している。詳しくは文献^{10, 1, 8, 3, 9)}等を参照されたい。

- 全幅探索 + 静止探索
- 後ろ向き枝刈り
 - PV Search
 - Aspiration Search
 - 置換表
 - 反復深化
 - 再帰的反復深化
 - オーダリング
 - SEE (Static Exchange Evaluation)
 - Killer Move
 - History Heuristics
- 前向き枝刈り
 - Futility Pruning
 - Null Move Pruning
 - Late Move Reductions
- 動的延長 (王手 1 手)
- 一手詰み判定関数
- df-pn+
- 並列探索 (YBWC)
- Bonanza Method による評価関数
- 詰み部分木

また、独自の工夫を行っている箇所としては以下が挙げられる。

- 評価関数の特徴⁶⁾
- オーダリングの学習⁵⁾
- 時間制御の学習⁵⁾
- Futility Pruning のマージンの学習^{5, 14)}
- 独自手法による ponder⁷⁾
- 独自手法による Aspiration Search
- 定跡 DB の作成

以下、これらについて説明する。

3 評価関数の特徴

Blunder の評価関数は、以下の項目から成る 6,662,378 個の要素を持つ特徴ベクトルを用いている。

- 駒割
- 持ち駒
- 持ち駒の種類の数
- 互いの持ち駒の歩の数の組み合わせ
- (歩を持ってないとき, 歩を持っているとき, 歩込みの) の持ち駒の種類の数
- 自駒の総数いろいろ
- 手番
- 手番と持ち駒の種類の数
- 手番と自駒の総数いろいろ
- 「歩・香・桂・角の前方・角の後方・飛車の縦・飛車の横」の効いてる駒の種類を敵の利きの有無毎に
- 「香・角の前方・角の後方・飛車の縦・飛車の横」の、移動出来るマス数と、そのうち敵の効きがある数の組み合わせ
- 飛車と角の移動出来るマスの組み合わせ
- 飛車と敵玉が同じ段ならその段毎に。
- 香・角の前方・角の後方・飛車の縦・飛車の横」の効いてる駒の種類と距離
- 歩・桂 $\times$ 1つ上の駒 $\times$ 2つ上の駒 $\times$ 進行度
- 歩・桂 $\times$ (段 - 2) $\times$ 1つ上の駒 $\times$ 2つ上の駒 $\times$ 進行度
- 段 (+ 畳み込み 1) $\times$ 桂・銀 $\times$ 移動不可理由 $\times$ 移動不可理由 $\times$ 進行度
- 桂馬が二回跳ねた先 3 マスの駒の種類を段毎に
- 駒の種類毎の味方と敵の利きの有無の組み合わせ
- 3~5 段目に歩がある場合その段毎に、その歩への味方の効きの有無と、歩の上への敵の利きの有無
- 歩が打てるマスへの味方と敵の効きの有無と前方の駒
- 段 $\times$ 駒 $\times$ $\min(\text{味方の効きの数}, 3) \times \min(\text{敵の効きの数}, 3)$
- 互いの、3~5 段目に歩がある筋の数の組み合わせ
- 互いの、 $\min(\min(\text{歩の無い筋の数}, \text{持ち駒の歩の枚数}), 7)$
- 敵の玉との距離 (8 段階) $\times$ 駒の種類

- 味方の玉との距離 (8 段階) × 駒の種類
- $\min(\text{敵の玉との距離が近い歩} \cdot \text{香} \cdot \text{桂の数}, 7) \times \min(\text{味方の玉との距離が近い歩} \cdot \text{香} \cdot \text{桂の数}, 7)$
- $\min(\text{敵の玉との距離が近い銀} \cdot \text{金} \cdot \text{と} \cdot \text{成香} \cdot \text{成桂} \cdot \text{成銀の数}, 7) \times \min(\text{味方の玉との距離が近い銀} \cdot \text{金} \cdot \text{と} \cdot \text{成香} \cdot \text{成桂} \cdot \text{成銀の数}, 7)$
- $\min(\text{敵の玉との距離が近い飛車} \cdot \text{角} \cdot \text{馬} \cdot \text{龍の数}, 7) \times \min(\text{味方の玉との距離が近い飛車} \cdot \text{角} \cdot \text{馬} \cdot \text{龍の数}, 7)$
- 互いの、 $\min(\text{敵の玉との距離が近い歩} \cdot \text{香} \cdot \text{桂の数}, 7)$ の組み合わせ
- 互いの、 $\min(\text{敵の玉との距離が近い銀} \cdot \text{金} \cdot \text{と} \cdot \text{成香} \cdot \text{成桂} \cdot \text{成銀の数}, 7)$ の組み合わせ
- 互いの、 $\min(\text{敵の玉との距離が近い飛車} \cdot \text{角} \cdot \text{馬} \cdot \text{龍の数}, 7)$ の組み合わせ
- 互いの、 $\min(\text{味方の玉との距離が近い歩} \cdot \text{香} \cdot \text{桂の数}, 7)$ の組み合わせ
- 互いの、 $\min(\text{味方の玉との距離が近い銀} \cdot \text{金} \cdot \text{と} \cdot \text{成香} \cdot \text{成桂} \cdot \text{成銀の数}, 7)$ の組み合わせ
- 互いの、 $\min(\text{味方の玉との距離が近い角} \cdot \text{飛車} \cdot \text{馬} \cdot \text{龍の数}, 7)$ の組み合わせ
- 互いの、 $\min(\text{持ち駒を含めた歩} \cdot \text{香} \cdot \text{桂の数}, 15)$ の組み合わせ
- 互いの、 $\min(\text{持ち駒を含めた銀} \cdot \text{金} \cdot \text{と} \cdot \text{成香} \cdot \text{成桂} \cdot \text{成銀の数}, 15)$ の組み合わせ
- 互いの、 $\min(\text{持ち駒を含めた角} \cdot \text{飛車} \cdot \text{馬} \cdot \text{龍の数}, 15)$ の組み合わせ
- pin されてる駒の方向と種類
- pin されてる駒の方向と距離と種類
- 味方角有無 × 敵角有無 × 駒の種類 × 数 × 進行度
- 駒の floor(段/3) × 駒の種類 × 玉との相対座標 × 進行度
- 玉の位置の種類 × 駒の種類 × 駒の位置 × 進行度
- 玉の 24 近傍の位置 × 駒の種類
- 玉の 24 近傍の位置 × 玉が四隅か否か × 敵味方の移動利き・縦横の飛び利き・斜めの飛び利きの有無の組み合わせ 2^3bit
- 玉の 24 近傍の位置 × 玉が四隅か否か × 敵味方の利きの有無 × 駒の種類
- 玉が四隅か否か × 玉の 8,24 近傍の (味方の駒の数 + 盤外の数)
- 玉が四隅か否か × 玉の 8 近傍の移動出来るマス数 × 敵の効きがある数
- 玉の位置への味方の、斜めの飛び利きの有無 × 縦横の飛び利きの有無 × $\min(\text{移動利きの数}, 3)$
- 玉の 24 近傍の敵味方の $\min(\text{利きの数の総和}, 31)$ の組み合わせ
- 互いの玉の 24 近傍の $\min(\text{敵の利きの数の総和}, 31)$ の組み合わせ
- 互いの玉の 24 近傍の $\min(\text{味方の利きの数の総和}, 31)$ の組み合わせ
- 玉の 24 近傍のうちの適当な 2 マス (90 通り) の駒の種類の組み合わせ
- 玉の 24 近傍のうちの適当な 2 マス (61 通り) の駒の種類の組み合わせ × 玉の位置 33 種類
- 24 近傍の 2 駒の方向と種類

4 オーダリングの学習

4.1 学習方法

オーダリングの実装方法はいくつか考えられるが、Blunder では、ある局面で読もうとする全ての指し手に対してそれぞれ得点を付け、その値で降順にソートを行うという実装を用いている。この実装の場合、得点を付け

る関数が評価関数と似た形となるため、Bonanza Method が容易に応用できる。以下ではこの関数をオーダリング関数と呼ぶ。

Bonanza Method では、プロなどの棋譜を用い、棋譜中の局面において、棋譜で指された手をそれ以外の手より得点が高くなるように学習を行う。しかし、オーダリングにおいては、棋譜には現れないような、「無駄な手を数手指した局面」が頻繁に出現するため、棋譜をそのまま用いた学習では最善の順序付けは得られない。

また、Killer Move と History Heuristics は、探索中に得られる情報を元に行っているため、棋譜中の局面では機能しない。

そのため、Blunder では、1 手 30 秒の自己対戦を行い、各 30 秒の探索の中の、順序付けの改善により探索効率が向上する全ての局面を学習対象としている。この際、AlphaBeta 法の Beta Cut により、最善の指し手が求まらない局面が存在するが、そこでは Beta Cut した手を最善手と見なし、Beta Cut した手以降の手は用いない。

これらを踏まえた上で、学習方法の設計を行う。基本的には、Bonanza Method の枠組みと同じであるため、詳細については文献¹¹⁾を参照されたい。

この手法では、学習対象となる全局面 P に対応する、より良い順序付けを行う特徴ベクトル v の発見を目指す。

まず、目的関数 $J(P, v)$ を以下のように設計する。

$$J(P, v) = \sum_{l=1}^N L(P_l, v)$$

ここで、 N は学習対象となる局面の数、 $L(P_l, v)$ は、この局面での順序付けと、実際に探索した際の最善手との違いの度合いを表現する関数。

$$L(P_l, v) = \sum_{m=1}^{N_l-1} T[\xi(p_{l,m}, v) - \xi(p_{l,0}, v)]$$

$\xi(p, v)$ は、オーダリング関数。詳細は 4.2 にて後述する。 N_l は局面 P_l に対して可能な指し手の数、または Beta Cut した指し手までの数。 $p_{l,m}$ は各指し手。 $p_{l,0}$ は最善手、または Beta Cut した指し手。

$T(x)$ は、オーダリング関数の値の差を、一致度を表す指標に変換する関数である。シグモイド関数を用いて、

$$T(x) = \frac{1}{1 + \exp\left(\frac{-7x}{256}\right)}$$

としている。

また、特徴ベクトルの要素が必要以上に大きくなる事などを防ぎ、極小点を減らすため、目的関数にペナルティを課す。

$$J'(P, v) = J(P, v) + wM_2(v)$$

ここで、 w はペナルティの強さ (正の実数値)。 $M_2(v)$ は各特徴ベクトル要素の大きさに相当する関数。これは次のように設定した。

$$M_2(v) = \sum_{i=1}^d A_i(v) v_i^2$$

d は v の次元の数。 $A_i(v)$ は、 v_i の、目的関数の勾配への寄与の度合いを表す。

$$A_i(v) = \sum_{l=1}^N \sum_{m=1}^{N_l-1} \left| \frac{dT(x)}{dx} \frac{\partial}{\partial v_i} [\xi(p_{l,m}, v) - \xi(p_{l,0}, v)] \right|$$

そして、最急降下法に基づく以下の式により特徴ベクトルの更新を行う。

$$v_i^{new} = v_i^{old} - h \cdot \text{sign} \left[\frac{\partial J'(P, v)}{\partial v_i} \right]$$

但し、 h は 1 ステップのベクトル要素の変化量 (正の整数または実数値) を示す。

4.2 オーダリング関数の設計

現在の Blunder は、下記の項目からなる 96,428 個の要素を持つ特徴ベクトルを用いている。

- 静止探索中の王手の応手の場合に、打つ手が動かす手か × 動かす駒の種類
- 王手の応手の場合に、打つ手が動かす手か × 動かす駒の種類
- $\text{sign}(\text{SEE}) \times$ 王手の応手かどうか × 静止探索かどうか × 動かす駒の種類 × 進行度・深さ度
- KillerMove の場合に、駒の種類
- HistoryMove の場合に、駒の種類
- 直前の手の駒を取る手の場合に、取った駒 × 取る駒 × 進行度・深さ度
- 直前の手の駒をただ取りする手の場合に、取る駒 × 静止探索かどうか × 王手かどうか × 進行度・深さ度
- 直前の手が取った駒 × 取る駒
- $\text{sign}(\text{SEE}) \times$ 敵玉との相対距離 (8 段階) × 取る駒の種類
- 8 近傍の駒に対して、方向 × 自駒 × 駒 × 進行度
- 移動元の玉との相対座標 × 動かす駒 × 進行度
- 移動先の玉との相対座標 × 動かす駒 × 進行度
- 直前の手の移動先と移動元の相対位置
- 直前の手の移動先と移動先の相対位置
- 直前の手の移動先と移動元の相対位置 (32 段階) × 敵駒の種類 × 自駒の種類
- 直前の手の移動先と移動先の相対位置 (32 段階) × 敵駒の種類 × 自駒の種類

5 思考時間制御の学習

5.1 概要

通常、将棋の対局では、持ち時間が定められ、思考時間の総計がその時間内に収まるように指さなければならない。しかし、コンピュータにとって、長考すべきか否かというのは、明確な判定基準が無く、どの程度時間を使えば十分なのか簡単には分からない。

また、現在のところあまり詳細が公開されてもおらず、情報が入手可能なのは、筆者の知る限り、金沢将棋の手法^{2, pp24-25)}と YSS の手法^{12, pp29-31)}、そして Bonanza の手法⁴⁾ 程度である。また、どれも経験的に設定したルールがベースとなっているため、改善の余地はありと考えられる。

そこで、これについても機械学習を用いて性能の向上を目指す。

5.2 学習方法

学習を行うには、様々な入力と、それに対する正解が必要である。そこで、ある局面において、ある思考時間で探索を行った際に得られる指し手 (最善手) を正解とする事にし、「正解手」と呼ぶ。^{*1}また、その思考時間は「最大思考時間」と呼び、反復深化の過程で最後に最善手が変わった深さを「正解深さ」と呼ぶ事とする。^{*2}

そして、反復深化が 1 回終わる毎に、そこで打ち切るべきか、探索を続行するのか判定する関数を用意し、これのチューニングを学習によって行う事とした。以降ではこの関数を判定関数と呼ぶ。

判定関数は、その値が負なら続行し、正なら打ち切るものとする。また、正解深さより浅い深さを low 状態、正解深さ以上の深さを high 状態、low 状態で判定関数が正の値を返す事を False Negative、high 状態で判定関数が負の値を返す事を False Positive と呼ぶ事にする。

判定関数に求められる性質は、False Negative の回数を出来るだけ少なく、かつ False Positive による時間の損失を出来るだけ抑える、というものである。この 2 つはトレードオフとなるため、目的関数 $J(x, v)$ を以下のように設計する。

$$J(x, v) = (1 - a)L_L(x, v) + aL_H(x, v)$$

ここで、 x は全ての low 状態と high 状態、 v は特徴ベクトル。 a の値は、False Positive と False Negative の重要度のバランスに基づいて調整する。

$L_L(x, v)$ は、low 状態での損失、 $L_H(x, v)$ は、high 状態での損失を表す。

$$L_L(x, v) = \frac{N_H}{N_N} \sum_{l=1}^{N_L} U_l(t_l) T[f(x^{low}_l, v)]$$
$$L_H(x, v) = \sum_{l=1}^{N_H} T[-f(x^{high}_l, v)]$$

N_L は low 状態の数、 N_H は high 状態の数、 x^{low}_l は low 状態、 x^{high}_l は high 状態、 $f(x_l, v)$ は判定関数。詳細は 5.3 にて後述する。

t_l は該当する状態の次の反復深化にかかった時間である。例えば、3 手読み終えた時点で思考開始時から 5 秒、4 手読み終えた時点で 15 秒経っており、3 手読み終えた時点で False Positive していたなら、その状態での t_l は 10 秒とする。また、次の反復深化が終わらず最大思考時間に達した場合は、そこまでの時間を用いる。例えば、8 手読み終えた時点で 35 秒経っており、最大思考時間が 40 秒で、9 手読み終わる前に 40 秒に達したなら、その状態での t_l は 5 秒とする。

$U_l(x)$ は、時間を係数へ変換する関数である。

$$U_l(x) = n \cdot p_l \cdot x$$

n は、 $U_l(x)$ の取る値の平均が 1 程度になるよう補正する定数である。実用上は、 a の値を任意に決定出来るため n の値に精度は必要無く、 a と、 $\frac{N_L}{N_H}$ と、 n の 3 つをまとめて、一つの値で代用が可能である。

p_l は該当する状態のデータを得た PC の、計算能力に比例した定数である。YSS ベンチ¹³⁾ の値を利用している。

^{*1} ゲームの性質上、重要度にはばらつきが存在する可能性はあるが、考慮していない。

^{*2} 実用する際は、残り時間や手数などから適宜最大思考時間を定めて用いる。

$T(x)$ は、順序付けと同様にシグモイド関数を用いた。

$$T(x) = \frac{1}{1 + \exp\left(\frac{-7x}{256}\right)}$$

これに、Bonanza Method と同様に、ペナルティ項を加え、各特徴ベクトルに対する勾配を求め、特徴ベクトルの更新を行う。ここからは順序付けとほぼ同様である。

$$\begin{aligned} J'(x, v) &= J(x, v) + wM_2(v) \\ M_2(v) &= \sum_{i=1}^d A_i(v)v_i^2 \\ A_i(v) &= (1-a)\frac{N_H}{N_N} \sum_{l=1}^{N_L} \left| U_l(t_l) \frac{dT(x)}{dx} \frac{\partial}{\partial v_i} [f(x^{low}_l, v)] \right| + \\ &\quad a \sum_{l=1}^{N_H} \left| \frac{dT(x)}{dx} \frac{\partial}{\partial v_i} [-f(x^{high}_l, v)] \right| \\ v_i^{new} &= v_i^{old} - h \cdot \text{sign} \left[\frac{\partial J'(x, v)}{\partial v_i} \right] \end{aligned}$$

5.3 判定関数の設計

判定関数を設計する上では、経過時間や残り時間を考慮する必要があるが、これらは数値が連続的で扱いにくく、また実行環境に固有のデータとなってしまう恐れがあるため、最大思考時間に対する経過時間の割合の対数スケールの値を利用した。

具体的には、最大思考時間 T_M 、経過時間 T として、経過時間の割合 R を、

$$R = \min(\max(\text{floor}(8 \log_{T_M} T), 0), 7)$$

として、0~7 の整数値に量子化して用いている。

Blunder が現在利用している特徴は下記の通り。(4,180 要素)

- 読んだ深さ × 進行度
- 読んだ深さの偶奇 × 進行度
- 時間 × 進行度
- 深さ × 時間 × 進行度
- 時間 (線形) × 進行度
- 深さ × 時間 (線形) × 進行度
- 偶奇 × 時間 × 進行度
- (最善手が変わらなかった回数、同じ最善手での評価値の増、減、同じ最善手での singular の回数) × 進行度
- (最善手が変わらなかった回数、同じ最善手での評価値の増、減、同じ最善手での singular の回数) × 時間 × 進行度
- 直前の手を取る手の場合に、singular かどうか × 取る駒の種類 × 進行度
- 直前の手を取る手の場合に、singular かどうか × 取る駒の種類 × 時間 × 進行度
- 相手が読み通りの手を指したかどうか × 今の最善手が自分の読み通りの手かどうか × 取られた駒の種類 × 進行度

- 相手が読み通りの手を指したかどうか × 今の最善手が自分の読み通りの手かどうか × 取られた駒の種類 × 時間 × 進行度
- 最善手が深さ 1 の手と同じ場合に、時間 × 進行度
- 最善手が深さ 1 の手と同じ場合に、 $\text{Min}(\text{最善手変更回数}, 3) \times \text{時間} \times \text{進行度}$
- $\text{Min}(\text{最善手回数}-1, 7) \times \text{時間} \times \text{進行度}$
- $\text{Min}(\text{最善手回数} * 8 / \text{深さ}, 7) \times \text{時間} \times \text{進行度}$
- $\text{Min}(\text{最善手変更回数}, 3) \times \text{singular} \times \text{深さ} \times \text{進行度}$
- $\text{Min}(\text{最善手変更回数}, 3) \times \text{時間} \times \text{進行度}$
- $\text{Min}(\text{Max}(\text{一つ前の反復時との評価値の差} / 48 + 7, 0), 15) \times \text{singular} \times \text{深さ} \times \text{進行度}$
- $\text{Min}(\text{Max}(\text{一つ前の反復時との評価値の差} / 48 + 7, 0), 15) \times \text{時間} \times \text{進行度}$
- $\text{Min}(\text{Max}(\text{反復初回時との評価値の差} / 48 + 7, 0), 15) \times \text{時間} \times \text{進行度}$
- 最善手が一つ前の深さでの PV の 3 手目の場合に、 $\text{singular} \times \text{深さ} \times \text{進行度}$
- 最善手が一つ前の深さでの PV の 3 手目の場合に、時間 × 進行度
- PV の 3 手目が一つ前の最善手の場合に、 $\text{singular} \times \text{深さ} \times \text{進行度}$
- PV の 3 手目が一つ前の最善手の場合に、時間 × 進行度
- $\text{Min}(\text{PV と一つ前の PV の一致長}, 7) \times \text{singular} \times \text{深さ} \times \text{進行度}$
- $\text{Min}(\text{PV と一つ前の PV の一致長}, 7) \times \text{時間} \times \text{進行度}$
- 過去に同一局面が存在した場合に、 $(\text{その回数} - 1) \times \text{最善手がその時指した手と同じかどうか} \times \text{進行度}$
- 過去に同一局面が存在した場合に、 $(\text{その回数} - 1) \times \text{最善手がその時指した手と同じかどうか} \times \text{singular} \times \text{深さ} \times \text{進行度}$
- 過去に同一局面が存在した場合に、 $(\text{その回数} - 1) \times \text{最善手がその時指した手と同じかどうか} \times \text{時間} \times \text{進行度}$

5.4 学習データ不足対策

この学習には、最大思考時間まで探索したときのデータが大量に必要となるため、データ収集に時間がかかるという問題がある。

そこで Blunder では、1 つのデータから、最大思考時間を減らして複数のデータを作成している。例えば最大思考時間が 160 秒のデータから、最大思考時間が 80 秒、40 秒、20 秒、10 秒だった場合のデータを作成する。(それ以降の探索結果を切り捨てる)

6 Futility Pruning のマージンの学習

基本的な考え方は時間制御と同様で、「枝刈り出来るのにしなかった回数 (以下 A)」「枝刈り出来ないのにしてしまった回数 (以下 B)」を「重み付けして (後者が重大なので重くして)、足した値」を最小化する、という方針に基づき学習を行う。

教師信号は、1 手 30 秒の自己対戦を行い、そのうち 10 ~ 20 % 程度の割合で^{*3}枝刈りする部分を抽出し、枝刈りせず探索した場合に結果がどうだったかによって A か B として扱う、という形で収集する。

なお、この学習は、オーダリングの学習と同時に行っている。また、Bonanza Method による評価関数の学

^{*3} あまり多いと木の形が変わってしまうため、コストとの兼ね合いで適宜低めにする。

習とも (別 PC で) 同時に行い、それぞれに結果を反映させている。これにより、互いに最適化されていくためか、個別に学習するよりも R50 程度強くなる現象が見られた。

また、将棋では、終盤は、ある程度以上有効な指し手の数は序盤とあまり変わらないが、ルール上可能な指し手の数は大幅に増加する。そのため、「枝刈り出来ないのにしてしまった回数」の重みを、終盤になるほど増加させる、という手法が有効であった。

現在利用している特徴は下記の通り。(2,818 要素)

- 残り深さ
- 手の種類
- 手の種類 × 残り深さ
- 互いの玉との距離 (8 段階) × 残り深さ
- 手の種類 × 互いの玉との距離 (8 段階)
- 手の種類 × 残り深さ × 互いの玉との距離 (8 段階)
- SEE が正の手の場合に、手の種類 × 残り深さ × 互いの玉との距離 (8 段階)
- SEE が負の手の場合に、手の種類 × 残り深さ × 互いの玉との距離 (8 段階)
- 手番側の場合に、残り深さ
- ノード種別 × 深さ
- 手の種類 × 動かす手が打つ手か
- 手の種類 × 動かす駒の種類
- 成る手の場合に、手の種類 × 動かす駒
- 手の種類 × 取る駒
- 手の種類 × 取る駒 × 動かす駒
- 手の種類 × 残り深さ × $\min(\text{オーダリングの順番}, 7)$

7 独自手法による ponder

Blunder では、相手番の時間には、相手番の局面から探索を行い、深さ 2 以上の局面の最善手を、置換表に似た専用のテーブルに保存するという手法を使用している。

性質上、自己対戦では手法の有効性が比較しづらいため、厳密な比較は行っていないが、floodgate でのレーティング測定の結果、一般的な手法^{*4}と同程度の強さは得られていると考えている。

8 独自手法による Aspiration Search

Blunder の Aspiration Search は、反復深化と組み合わせる一般的な実装ではなく、各内部ノードで、PV Search の初手や再探索時に行う実装となっている。評価値の予測には、再帰的反復深化時の結果や、置換表の値を利用する。

これにより、Aspiration Search の再探索発生時にも、大きな遅延が発生しにくい動作となり、時間制御が行いやすくなっていると考えている。

また、ウィンドウ幅は、進行度を用いて、終盤はやや大きめとしている。

^{*4} 相手の手を予測し、それに対する応手を探索する。

9 定跡 DB の作成

Blunder では、プロの棋譜に現れた局面に対し 10 秒程度の探索を行い、評価値が高かった局面を重視する形で定跡 DB の作成を行っている。

Blunder の 10 秒程度の探索では、プロの棋譜に対して優勢・劣勢を評価しきれているとは言い難い。しかし、自己対戦の結果、単純に出現回数などを元に作成した定跡 DB に対して、6 割程度の勝率が得られた。

優勢・劣勢は評価出来なくとも、評価値が高い局面なら、Blunder が得意な局面である可能性がある程度は高い為であると解釈している。

参考文献

- 1) 池泰弘. コンピュータ将棋のアルゴリズム. 工学社, 2005.
- 2) 金沢伸一郎. 金沢将棋のアルゴリズム. コンピュータ将棋の進歩 3, pp. 15–26, 2000.
- 3) 金子知適, 田中哲朗, 山口和紀, 川合慧. 新規節点で深さ優先探索を利用する df-pn アルゴリズム. 第 10 回 ゲーム・プログラミング ワークショップ, pp. 1–8, 2005.
- 4) 金子知適. shogi programming journal bonanza の探索時間の制御方法のメモ. <http://www.sgtpepper.net/kaneko/diary/20090213.html>, 2009.
- 5) 下山晃. aki. の日記 オーダリングと時間制御と枝刈りの学習. <http://d.hatena.ne.jp/ak11/20090507>, 2009.
- 6) 下山晃. aki. の日記 特徴一覧. <http://d.hatena.ne.jp/ak11/20091019>, 2009.
- 7) 下山晃. aki. の日記 ponder のアルゴリズム. <http://d.hatena.ne.jp/ak11/20100211>, 2010.
- 8) 鈴木豪, 乾伸雄, 小谷善行. 将棋におけるゲーム木探索アルゴリズムの比較. 情報処理学会研究報告, Vol. 99, No. 53, pp. 79–84, 1999.
- 9) 脊尾昌宏. 詰将棋を解くアルゴリズムにおける優越関係の効率的な利用について. 第 5 回 ゲーム・プログラミング ワークショップ, pp. 129–136, 1999.
- 10) 保木邦仁. コンピュータ将棋における全幅探索と futility pruning の応用. 情報処理, Vol. 47, No. 8, pp. 884–889, 2006.
- 11) 保木邦仁. 局面評価の学習を目指した探索結果の最適制御. 第 11 回 ゲーム・プログラミング ワークショップ, pp. 78–83, 2006.
- 12) 山下宏. YSS—『コンピュータ将棋の進歩 2 以降の改良点』. コンピュータ将棋の進歩 5, pp. 1–32, 2005.
- 13) 山下宏. かつてにベンチマーク. <http://www32.ocn.ne.jp/~yss/bench.html>, 2009.
- 14) 山本一成. ロジスティック回帰を用いた探索木の前向き枝刈り. 2010.