

平成 20 年度 卒業論文

コンピュータ将棋における  
機械学習を用いた  
指し手の順序付けと思考時間制御

指導教員 河上肇 准教授

秋田大学 工学資源学部 情報工学科

7505331 下山 晃

## 概要

本研究では、機械学習を用いて将棋プログラムの性能を向上させる2つの手法の構築を行った。1つは、指し手の順序付けである。学習により約3000の要素を持つ特徴ベクトルを生成することで、従来より精緻な順序付けを可能とし、それによるゲーム木の探索効率の改善を図り、一般的な手法の一つと比較して、5%程度の効率の改善に成功した。もう1つは、思考時間制御である。こちらも、学習により約800の要素を持つ特徴ベクトルを生成し、統計的データに基づく時間制御を可能とすることで、制御を行わない場合に比べ、30~60%程度効率的な時間運用を可能とした。

# 目次

第 1 章	はじめに	1
第 2 章	本研究で用いるプログラム	2
第 3 章	指し手の順序付けの学習	4
3.1	概要 . . . . .	4
3.2	学習方法 . . . . .	4
3.3	オーダリング関数の設計 . . . . .	6
3.4	結果・考察 . . . . .	6
第 4 章	思考時間制御の学習	9
4.1	概要 . . . . .	9
4.2	学習方法 . . . . .	9
4.3	判定関数の設計 . . . . .	11
4.4	結果・考察 . . . . .	12
第 5 章	おわりに	14
	参考文献・URL	16

# 第 1 章

## はじめに

2006 年に行われた第 16 回世界コンピュータ将棋選手権<sup>16)</sup>において、Bonanza というプログラムが、初出場で優勝を納めた。このプログラムは、従来の将棋プログラムからすると珍しい、以下のような特徴を持っている。

- 作者が将棋についてほとんど知らない<sup>13, p53)</sup>
- コンピュータチェスの手法を数多く応用
- 機械学習のみによる評価関数

従来は、強いプログラムの作者は、大抵ある程度の棋力を持っていたが、Bonanza の登場により、将棋の素人でも強い将棋プログラムの開発が可能である事が初めて示された。

これは、将棋では可能な手の数がチェスより多いために、ゲームの知識に基づいた選択的なゲーム木探索<sup>\*1</sup>が必須であると考えられていたことや、駒が再利用可能なルールなどにより、チェスに比べ性能の良い評価関数の設計が困難であることに由来する。

Bonanza は、前者の問題を、チェスのアルゴリズムの応用により、将棋の知識にあまり依存しない全幅探索を可能にすることで解決<sup>11)</sup>し、後者の問題を、このプログラムの作者が自ら開発した、機械学習の手法により解決<sup>12)</sup>した。

また、この機械学習の手法は、チェスやその変種のゲームにおいてほぼ初めてと思われる、「実用に耐え、役に立つ」手法として、大きく話題となり、後に Bonanza Method と名付けられた。この手法は、その後多くのプログラムが利用し始め<sup>\*2</sup>、近年の最も大きなブレイクスルーの 1 つと言っても過言では無い。

本研究では、この Bonanza Method を参考に、機械学習を用いて将棋プログラムの効率を向上させる 2 つの手法の構築を行った。3 章ではその 1 つ目である指し手の順序付け、4 章では 2 つ目の思考時間の制御について述べる。

---

\*1 ルール上可能な全ての指し手を考慮した探索 (先読み) ではなく、いくつか絞った指し手のみの探索。

\*2 2008 年の選手権では、決勝に出場した 8 チームのうち、5 チームが学習を使っていたようだ。<sup>8)</sup>

## 第 2 章

# 本研究で用いるプログラム

本研究では、検証用に開発した、Blunder と名付けたプログラムを用いた。

このプログラムは、Bonanzaなどを参考に、現在将棋プログラムを作る上で一般的と思われる、以下のアルゴリズムを実装してある。詳しくは文献<sup>11, 2, 5, 4, 6)</sup>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
  - 反復深化法<sup>\*1</sup>
  - 多重反復深化法
  - 指し手の順序付け
    - 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、recapture・one reply・pawn attack 0.5 手)
- 一手詰み判定関数
- df-pn
- 並列探索
- Bonanza Method による評価関数

Blunder は、実装の手間を省くため、計算速度の点で少し不利なプログラミング言語、C#を用い、現時点では高速化にもあまり力を入れていないため、NPS<sup>\*2</sup>で比較した場合、他のプログラム

---

<sup>\*1</sup> 最初は 1 手、次は 2 手と、順次読む深さを増やす方法。本稿で「 $n$  手読む」と書いた場合、これが  $n$  手になるまで繰り返す事を示す。

<sup>\*2</sup> 1 秒あたりにプログラムが調べる事の出来る局面の数。

の  $\frac{1}{10}$  以下の速度しか出ていない。しかし、floodgate<sup>15)</sup> でのレーティング\*<sup>3</sup>を見ると、Bonanza などのトップクラスのプログラムと 500 点差程度であるので、仮に 10 倍程度高速化すれば、それらに近い強さとなる可能性が高い。\*<sup>4</sup>

従って、アルゴリズムはトップクラスのプログラムに近いものを実現していると考えられ、特殊なアルゴリズムも用いていないため、このプログラムを用いた検証の結果は、他のプログラムを用いた場合と近い結果が出るケースの多い事が期待される。

---

\*<sup>3</sup> 強さを表す指標の一つ。各自の持ち点を勝ち負けによりやりとりする。

\*<sup>4</sup> 一般に、3~4 倍高速化すれば一手多く読め、一手多く読めば 200 点増えると言われている。

## 第3章

# 指し手の順序付けの学習

### 3.1 概要

AlphaBeta 法をベースとした、現在主流のゲーム木探索の手法の元では、評価値の良い手から順に展開することで探索の効率が向上する。

そこで一般的に行われるのは、例えば以下のような評価項目を用いた指し手の順序付けである。

- SEE
- Killer Move
- History Heuristics
- その他、ゲームの特性に応じた、有力な可能性の高い手への加点<sup>7)</sup>

また、順序付けを正確に行う事は不可能であるので、出来るだけ正確な場合の効率に近づけるためのアルゴリズムとして、

- PV Search または MTD
- Aspiration Search
- 置換表
- 反復深化法
- 多重反復深化法

などが提案されており、多くの有力なプログラムは、これらのほぼ全てを組み合わせで利用している。

本研究では、この指し手の順序付けに着目し、Bonanza Method の応用を行った。

順序付けの実装方法はいくつか考えられるが、今回は、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、容易に応用が可能となる。以下ではこの関数をオーダリング関数と呼ぶ。

### 3.2 学習方法

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、指し手の順序付けにおいては、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。例えば、プロの棋譜中に「飛車をただで取れる局面」が出て来ても、そのうちのいくつかでは、取れば形勢が悪化するために、取らない手を指す。しかし、探索中に出てくる「飛車をただで取れる局面」のかなり多くは、無意味に飛車を捨てる手であるので、取らない手が最善である割合は、プロの棋譜で取らない割合に比べ、かなり低いと考えられる。

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、本研究では、入力された局面から 3 手読み、反復深化 3 回目の、深さ 1 の局面 (与えられた局面から 1 手進めた局面) と、その時の最善手を用いて学習する事とした。

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、ここでは 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。ここでは、その局面から 2 手読み直す事で真の最善手を得る事も可能ではあるが、実験したところ、得られたオーダリング関数の性能は、読み直さなかった場合に比べ劣っていた。これは、多くの局面で Beta Cut を起こす手であれば、優先すべき理由にはなり得る反面、オーダリング関数で完全に真の最善手を推定する事は精度不足で不可能であるため、真の最善手は学習対象には向かないためと考えられる。

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<sup>12)</sup>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 3.3 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、

$$T(x) = \frac{1}{1 + \exp\left(\frac{-3x}{\text{歩の交換値}}\right)}$$

とした。

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。

### 3.3 オーダリング関数の設計

本研究では、周知の手法である SEE、Killer Move、History Heuristics に加え、

- 直前に指された駒を取る手の場合、その双方の駒の種類
- 移動元・移動先に隣接した駒の種類と方向
- 移動元・移動先と、直前の相手の手の移動先との相対位置
- 白玉・敵玉との相対位置 (駒の種類毎)

という項目からなる約 3000 の要素を持つ特徴ベクトルを用いた。<sup>\*1</sup>

### 3.4 結果・考察

プロの棋譜、約 3 万局のうち、1 ステップ毎にランダムに 1000 局程度を抽出し、学習を行った。

得られた特徴ベクトルの一部を表 3.1 に示す。段 +0、筋 ±0 の欄が王の位置、段 + $n$  (resp. 段 - $n$ ) の行が王より下 (resp. 上) の段の駒に対する評価を表す。また、ここでは歩の交換値を 100 としている。王に近いところで正の値、離れる程大きな負の値、という傾向は、評価関数で同様の項目を Bonanza Method で学習した場合と似てはいるが、値の範囲を見ると、例えば Bonanza の場合は、文献<sup>12)</sup>によると精々 +500 から -100 程度であるのに対し、こちらは +100 から -1200 程度と大きく、特に、王より下の段の駒に対するマイナスの評価が目立つ。

また、性能の検証として、Blunder を用い、プロの棋譜よりランダムに選んだ 300 の局面で、各深さまで探索した際の、ノード数の合計と時間の合計を表 3.2 に示す。*learn* の列が、今回学習を行ったオーダリング関数を用いたもの。比較対象として、乱数のみを用いたもの (*rand*) と、SEE、Killer Move、History Heuristics の 3 つのみを用いたもの (*simple*) を掲載した。末尾に [秒] と付けた列は、かかった時間の合計である。

*rand* の列の結果から、例え順序付けがでたらめであっても、Blunder で実装されている順序付け以外の各種のアルゴリズムにより、かなりの効率化がなされている事が分かる。仮に完

<sup>\*1</sup> プログラム上は、各特徴を 1 次元から 3 次元程度の配列で持ち、学習時には 1 次元配列にコピーして扱った。

表 3.1 王と金駒の相対位置評価

段	筋 ±0	筋 ±1	筋 ±2	筋 ±3	筋 ±4	筋 ±5	筋 ±6	筋 ±7	筋 ±8
-8	-195	-177	-105	-74	-154	-135	-224	-249	-185
-7	-120	-112	-82	-102	-96	-148	-145	-131	-290
-6	-98	-89	-74	-64	-90	-103	-163	-183	-306
-5	-80	-111	-87	-69	-101	-158	-172	-153	-400
-4	-125	-98	-126	-109	-134	-166	-277	-297	-554
-3	-57	-86	-72	-101	-142	-203	-287	-280	-656
-2	62	31	-20	-102	-175	-261	-311	-351	-831
-1	88	33	41	-135	-232	-337	-419	-367	-536
+0	0	81	42	-125	-319	-356	-437	-387	-882
+1	58	-4	42	-181	-287	-455	-499	-613	-1062
+2	113	10	-74	-280	-353	-479	-550	-780	-1244
+3	-201	-184	-252	-372	-555	-639	-875	-1162	-1301
+4	-359	-385	-381	-503	-622	-755	-859	-1261	-1131
+5	-656	-554	-664	-615	-737	-868	-1124	-1374	-1276
+6	-845	-767	-682	-821	-1039	-1079	-1043	-1343	-1236
+7	-948	-972	-1076	-928	-1136	-1173	-1279	-1285	-1099
+8	-1064	-1205	-1166	-1220	-1150	-1285	-1173	-861	-621

表 3.2 ノード数の合計と時間

深さ	<i>rand</i>	<i>simple</i>	<i>learn</i>	<i>rand</i> [秒]	<i>simple</i> [秒]	<i>learn</i> [秒]
2	169,975	167,527	152,705	16.51	16.71	19.42
3	661,503	560,294	528,364	32.05	27.71	28.72
4	1,487,612	1,228,145	1,181,615	55.39	46.71	46.17
5	4,069,672	2,891,737	2,796,762	120.49	85.87	90.05
6	9,110,883	6,253,296	6,033,513	255.02	181.99	179.04
7	32,609,498	16,236,270	15,586,143	1025.32	535.71	530.41
8	86,205,710	38,648,963	35,382,217	2982.01	1392.19	1283.99

全な総当たりを行えば、平均分岐数を 80 として、深さ 8 のとき、86,205,710 に相当する値は、 $80^8 \times 300 = 503,316,480,000,000,000$  となるはずである。

また、*simple* は、チェスプログラム Crafty の実装とほぼ同じである。<sup>\*2</sup> これは簡単な実装である割に、非常に効果が高く、8 手の時点で *rand* の半分以下のノード数となっている。

これに対し、本研究で作成したオーダリング関数による性能向上 (ノード数の削減) は、*simple* と比較すると 5% 前後である。これは、各種のアルゴリズムの組み合わせと *simple* の順序付けにより、既にオーダリングでの性能向上の限界に近い効率が得られているためと推測される。順序付

<sup>\*2</sup> Bonanza も同様の方法を用いているようだ。<sup>9)</sup>

けを含む全ての後ろ向き枝刈りのアルゴリズムは、順序付けが不完全な場合の効率を、最善に近付けるものであるため、それらはどれほど上手く機能したところで、AlphaBeta の最善の効率\*<sup>3</sup>が理論上の上限となってしまう。従って、*simple* と似たアルゴリズムを用いた Crafty などのプログラムが、十分に性能が出ている事から考えても、性能向上の余地があまり無い可能性が高いと言える。

時間の列に関しては、実行環境などに大きく左右される事や、Blunder の実装があまり速度を重視していないために、十分に高速化を行ったプログラムとは違う特性を示す可能性もあるが、今回の実験に限って言えば、こちらも *simple* に比べ 5% 近い性能向上は見られた。<sup>\*4</sup>

また、*simple* と *learn* のノード数と時間に対して符号検定を行った。ノード数は、300 局中、*simple* の方が *learn* より少ないノード数だった局面は 104 局面、*learn* の方が少ないノード数だった局面は 195 局面。従って、有意水準 5% の下で、*learn* の方が統計的に有意に多くの局面で、ノード数の減少が行えたと言える。同様に、時間についても、*simple* の方が短い局面の数が 130、*learn* の方が短い局面の数が 170 であり、有意水準 5% の下で、*learn* の方が有意に多くの局面で時間の短縮が行えたと言える。

---

\*<sup>3</sup> 探索の最もベースとなる AlphaBeta 法は、順序付けが最悪の場合、完全な総当たりとなり、最善の場合、総当たりのノード数の  $\frac{1}{2}$  乗となる事が証明されている。

\*<sup>4</sup> 但し、ノード数が減ろうとも、時間が減らなくてはプログラムの強さの向上には結びつかない事に注意されたい。

## 第 4 章

# 思考時間制御の学習

### 4.1 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<sup>3, pp24-25</sup>) と YSS の手法<sup>14, pp29-31</sup>) のみである。また、どちらも経験的に設定したルールがベースとなっているため、改善の余地はあると考えられる。

そこで、本研究ではこれについても機械学習を用いて性能の向上を目指す。これは評価関数や指し手の順序付けとはあまり似た問題ではないため、SVM やニューラルネットワークなどのより一般的な学習手法の利用も可能ではあるが、学習結果に対する手動チューニングの容易性や、ソースコードやノウハウの共有による開発の省力化を見込み、本研究では Bonanza Method を参考にした学習手法の設計を行った。

### 4.2 学習方法

Bonanza Method は教師あり学習の一種であるので、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、本研究では、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。<sup>\*1</sup>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。<sup>\*2</sup>

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 fail-high、high 状態で判定関数が負の値を返す事を fail-low と呼ぶ事にする。

判定関数に求められる性質は、fail-high の回数を出来るだけ少なく、かつ fail-low による時間

<sup>\*1</sup> ゲームの性質上、重要度にはばつきが存在する可能性はあるが、本研究では考慮しなかった。

<sup>\*2</sup> 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。

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、fail-low と fail-high 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 4.3 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 fail-low 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 手読み終える前に 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。本研究では NPS 値の平均を用いた。並列探索を行う場合は、スレッド数  $t$  に対して  $\sqrt{t}$  倍の効率が出ると仮定すれば  $p = \frac{\text{NPS 値}}{\sqrt{t}}$  と出来るが、非並列探索を  $t$  個同時に行った方が同じ時間でより多くのデータが得られると考え、本研究では並列探索は行っていない。

$T(x)$  は、順序付けと同様にシグモイド関数を用いた。今回は値の大小には意味がないため、本研究では  $v$  の値を 1 バイト変数に納められるよう、解像度を高めに設定した。

$$T(x) = \frac{1}{1 + \exp\left(\frac{-3x}{30}\right)}$$

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

表 4.1 深さ予測の平均値

	N=1	N=2	N=3	N=4	N=5	N=6	N=7	N=8	N=9
平均	5.08	6.43	7.38	8.16	8.78	9.12	9.38	9.63	9.86

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。

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

### 4.3 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、深さに変換する事を考える。

一般に、読みの深さ  $D$  に対する必要時間  $T_D$  は、大まかには  $3^D \sim 5^D$  に比例すると言われている。<sup>10)</sup> そこで、既に探索を終えた深さ  $N$  とその時の経過時間  $T_N$  より、

$$D = N \log_{T_N} T_D$$

として  $D$  の推定を行った際の、妥当性について調べた。

プロの棋譜よりランダムに選んだ 300 の局面から、深さ 10 まで探索を行い、各深さを読み終えた時間を記録し、深さ 10 を読み終えた時間を  $T_D$  として、 $D$  の値をそれぞれ求めた。平均値を表 4.1 に示す。

浅い時の予測ほど悲観的という明確な傾向<sup>\*3</sup>があるため、そこを補正してよりよい近似式を作る事は可能ではあるが、マジックナンバーが必要となる可能性や、 $N$  が小さい時は偏差も大きい事、多少の不正確さは特徴ベクトルが吸収出来る事などを踏まえ、本研究ではこの式による予測をそのまま用いる事とした。

これにより、ある程度の推定は可能となったため、本研究では以下の情報からなる約 800 の要素を持つ特徴ベクトルを用いた。

- 読んだ深さ
- 読めると推定される残り深さ
- 読めると推定される深さに対する読んだ深さの割合

<sup>\*3</sup> 恐らく、初期化などのオーバーヘッドや、Blunder では浅い深さでしか行っていない千日手などの処理による。

表 4.2 連続回数の評価

	0回	1回	2回	3回	4回	5回	6回	7回以上
最善手変更無し	-17	-13	-4	0	8	-4	10	10
評価値連続増加	-17	0	8	8	8	10	7	2
評価値連続減少	-17	-4	-4	-4	-4	-1	7	4
singular	-17	13	35	40	46	66	66	70

- 最善手が singular かどうか<sup>1)</sup>
- 取る手かどうか。取る手なら取る駒の種類
- 打つ・動かす駒の種類
- 前回の探索の最善手順と、相手の指し手や今回の最善手が一致しているかどうか
- 最善手の変わった回数、最善手が過去最善手だった回数や割合
- 今の最善手が前回、前々回と連続で同じだった回数
- 最善手と同じで、評価値が連続で上がっている場合にその回数
- 最善手と同じで、評価値が連続で下がっている場合にその回数
- 最善手と同じで singular でもある状態が連続した場合にその回数

## 4.4 結果・考察

探索データは、自己対戦を行い、各局面毎に 10, 20, 40, 80, 160 秒の中からランダムに思考時間を定め、64 万局面ほど収集し、学習を行った。

得られた特徴ベクトルの一部を表 4.2 に示す。

singular は、連続して現れた際に、その後最善手が変わらない可能性が高い事を示すよい指標となっている事が分かる。それ以外はあまり大きな値にはなっておらず、結果に対する影響はほぼ誤差程度である。

YSS の文献<sup>14, p31)</sup>では、評価値が下がった場合などに 2 倍から 3 倍程度時間を延長する手法が紹介されているが、ここでは、評価値が連続して下がった事単体では、singular などに比べると、あまり重要な指標とは見なされなかったようである。<sup>\*4</sup>

学習のために集めた探索データを利用して、各最大思考時間に対する思考時間の平均を求めたところ、最大思考時間の約 40% 程度であり、その時、正解深さ以上の深さを読んだ局面は、全体の 90% から 95% 程度であった。この値を参考に、 $a$  の値の調整や、実用時の最大思考時間の設定を行うと良い。

なお、あまり極端に思考時間が短い時点で打ち切る事は、あまり時間の節約にはならず、弊害だけが大きいと考えられるため、経過時間が最大思考時間の  $\frac{1}{16}$  未満の場合は打ち切りの判定を行わないようにしたところ、思考時間の平均はほとんど変わらず、正解深さ以上の深さを読んだ局面の割合は少し上昇した。従って、自己対戦や実用の際には、同様のやり方を行っている。

また、強さへの影響を調べるため、この時間制御を、最大思考時間を 40 秒に固定して用いたものと、常に 20 秒、25 秒思考するものとの、自己対戦を行ったところ、20 秒固定との対戦では、

<sup>\*4</sup> 但し、5 回目までは心なし負の値、それ以降は正の値ではあるため、それらしい傾向はあると言えなくはない。

394 戦して 218 勝 176 敗の勝率 55%、25 秒固定との対戦では、348 戦して 165 勝 183 敗の勝率 47% となり、学習側の思考時間の平均はどちらも 15.1 秒だった。

なお、この結果に対して二項検定を行うと、有意水準 5% の下で、20 秒固定より時間制御が統計的に有意に強いとは言えるが、時間制御より 25 秒固定が有意に強いとは言えない。

よって、この手法により、全く制御を行わなかった場合に比べ、30~60% 程度の高速化に相当する程度に、効率的に時間を使う事が可能となったと言える。

## 第 5 章

# おわりに

本研究で構築した 2 つの手法は、それぞれ 2008 年 10 月中旬と 2008 年 11 月中旬から、floodgate で対戦しているプログラムにも組み込んでおり、それ以前と比べて、2 つ合わせてレーティングで 30 程度の上昇が見られた。

今後の課題として、順序付けに関しては、前述の通りあまり改善の余地がない可能性があるため、前向き枝刈りへの応用などが考えられる。

思考時間制御に関しては、データ収集に時間がかかることが挙げられる。時間を深さに変換して用いた事により、ある程度スペックの違うコンピュータも用いる事は可能であるが、理想的には、最終的に用いる環境で、最も長考した場合の計算量に匹敵するデータまでを集める必要があるため、より効率的に多様な局面のデータを収集する方法などに改善の余地があると思われる。

# 謝辞

本研究を進めるにあたり、多くの方々にお世話になりました。河上准教授には、指導教官として、多くのご指導、貴重なご意見を賜りました。やねうら様には、Blunderの実装等に関して有益なアドバイスを頂きました。河上研究室・小林研究室の皆様にも、日頃からお世話になりました。協力していただいた皆様へ心から感謝申し上げます。

## 参考文献・URL

- 1) M.S.Campbell T.Anantharaman and F.H.Hsu. Singular extensions: adding selectivity to brute-force searching. *Artificial Intelligence*, Vol. 43, No. 1, pp. 99–109, 1990.
- 2) 池泰弘. コンピュータ将棋のアルゴリズム. 工学社, 2005.
- 3) 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.
- 4) 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.
- 5) 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.
- 6) 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.
- 7) 竹歳正史, 橋本剛, 作田誠, 飯田弘之. コンピュータ将棋における指し手の順序付けによる探索効率化. 情報処理学会論文誌, Vol. 43, No. 10, pp. 3074–3077, 2002.
- 8) 奈良和文. 第 18 回世界コンピュータ将棋選手権 参加記. <http://www2.odn.ne.jp/~aao62090/WCSC18/index.html>.
- 9) 西村則久. マイムーブ ブログ記事 no.406. <http://chocobo.yasuda-u.ac.jp/~nisimura/mymove/index.cgi?no=406>, 2006.
- 10) 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. [http://www.geocities.jp/bonanza\\_shogi/gpw2006.pdf](http://www.geocities.jp/bonanza_shogi/gpw2006.pdf).
- 11) 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.
- 12) 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.
- 13) 保木邦仁, 渡辺明. ボナンザ VS 勝負脳. 角川書店, 2007.
- 14) 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.
- 15) floodgate. <http://wdoor.c.u-tokyo.ac.jp/shogi/floodgate.html>.
- 16) 第 16 回世界コンピュータ将棋選手権. <http://www.computer-shogi.org/wcsc16/>.