RF-003

将棋における評価関数自動生成の高速化
Acceleration Techniques for Acquisition of Evaluation Functions

久保 亮介 †　　六沢 一昭 ‡
KUBO Ryosuke　　ROKUSAWA Kazuaki

1 はじめに

コンピュータによるチェスや将棋の対局では, 形勢判断を行う評価関数の性能が強さを大きく左右する. 評価関数を作るには, 人間が抽象的に理解している様々な評価項目を全て数値化しなければいけない. 例えば, 駒の損得を求める場合, 歩を 100 点, 香車を 300 点などと駒の価値を明確に定める必要がある. 駒の価値以外にも駒の位置関係といった非常に数値化しにくい項目も必要となる.

特に将棋は多くの評価項目が必要で, 評価関数のパラメータは数千から数千万個に及ぶ. これを人手で最適化することは極めて困難である. そのため, 機械による自動調整の方法が早くから研究されてはいたのだが, 大きな成果はなかった.

しかし近年, 初めての実用的な機械学習手法 Bonanza Method の誕生により, 将棋における機械学習が注目を集めている. その有効性は既に周知の事実であり多くのプログラマが採用するに至ったが, 未だに高速化や効率化が模索されている. これは, 評価項目の決定に人間の試行錯誤を必要とし, 機械学習を何度も繰り返さなければいけないためである.

そこで, 本研究では探索窓に着目した新たな高速化手法として, 段階的な探索窓と確率的な復帰の 2 つを提案し, 提案手法を適用しない場合と性能を比較した. このうち, 確率的な復帰では良い結果を得ることができた.

2 将棋における機械学習

本章では, 保木によって確立された Bonanza Method と呼ばれる機械学習の手法について説明する. 今では何人ものプログラマが各々の工夫と共に用いているが, ここでは保木によるオリジナルの手法 [1] について述べる.

2.1 評価関数

将棋におけるトップレベルの対局プログラムでは全幅探索や実現確率探索, ABC 探索などのアルゴリズムが用いられている. これらはどれも Minimax 探索をベースとしたものである. 近年ではモンテカルロ木探索を将棋に適用した例もあるが, トップレベルの強さには至っていない [8].

終局の直前では, Minimax 探索により極めて正確な結果を得ることができる. これは探索木の末端で勝敗が明確に決定するためである. 一方, 終局までを探索しきることが現実的でない場合, 末端局面は何らかの方法で優劣の判断がなされなければいけない. これを行う機構を評価関数と呼ぶ.

表 1: 評価項目の構成 (Bonanza version4.1.2)

評価項目	特徴数
駒割り	13
3 駒の関係	10^8

評価関数 f は与えられた局面 P に対し, その局面の評価値 e を返す. 評価値とは局面の形勢判断を示す値で, 一般に 0 を互角とし大きいほど有利, 小さいほど不利を表す実数値または整数値である. 探索によって得られる動的評価値と区別して, 特に静的評価値と呼ばれる.

評価関数の内部構成は様々考えられるが, 多くの場合は単純な線形和であり一般に次式の形で表される.

$$e = f(P) = \mathbf{x}_P \cdot \mathbf{w} \tag{1}$$

$\mathbf{x}_P$ は局面 P から抽出された特徴ベクトルであり, $\mathbf{w}$ は各特徴の重みを表す重みベクトルである. この $\mathbf{w}$ が評価関数の性能を左右するパラメータであり, 機械学習における知識に相当する.

Bonanza Method が確立される前は, 人間の手によってパラメータの調整が行われていた. しかし, $\mathbf{w}$ の要素数は一般に 10^3 から 10^8 に及び, 最適な解を得ることは困難である. そこで, $\mathbf{w}$ を自動調整する.

局面から抽出する特徴の種類は, 評価関数の評価項目に相当する. 保木が作成した対局システム Bonanza version4.1.2 では, 評価項目の構成を表 1 の通りとしている.

駒割りとは駒の損得のことである. 特徴ベクトル $\mathbf{x}$ の要素中, 駒割りに対応する 13 要素は, 駒の種類毎の先手の枚数と後手の枚数の差を表す.

$$\begin{aligned} x_1 &= \text{先手の歩の数} - \text{後手の歩の数} \\ x_2 &= \text{先手の香の数} - \text{後手の香の数} \\ &\vdots \\ x_{13} &= \text{先手の竜の数} - \text{後手の竜の数} \end{aligned} \tag{2}$$

従って, これに対応する $\mathbf{w}$ の要素は各駒 1 枚当りの価値を意味する.

駒割りはほとんど全ての対局システムで用いられている. また, 駒割りだけでは不十分であるから駒の位置関係に対する評価が必須である.

3 駒の関係では, 盤上の駒と持駒を列挙し, 任意の 3 枚の組み合わせに点数をつける. 但し, 1 枚は玉を含むも

†千葉工業大学 大学院 情報科学研究科 情報科学専攻
‡千葉工業大学 情報科学部 情報工学科

表 2: 駒割り (初期値)

駒の種別	歩	香	桂	銀
点数	100	300	300	400
点数 (成)	400	400	400	500
駒の種別	金	角	飛	
点数	500	600	700	
点数 (成)		800	1000	

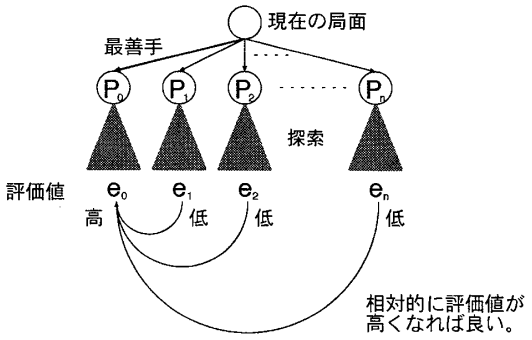

図 1: 兄弟局面の比較

のとする. 1 つの局面に 10^3 程度の組み合わせが存在し, 重みベクトルには, 出現しうる全ての組み合わせの数約 10^8 もの要素が必要となる.

学習による自動調整を行う場合でも, 駒割りは適当な初期値から開始するのが一般的である. Bonanza では駒割りの初期値を表 2 の通りとしている. また, それ以外の要素は全て 0 である.

2.2 Bonanza Method

2.2.1 兄弟局面の比較

Bonanza Method では, 強い対局者の棋譜を用いて最善手 (棋譜の指し手) を真似する方向にパラメータを調整していく. ここで, プロ棋士や高段者の棋譜であっても悪手を含む事はあるが, 全て最善手であると仮定する.

探索を行った結果, 最善手の後の局面 P_0 の動的評価値が最も高くなれば, 機械も棋譜と同じ手を選ぶ. (図 1) そこで, 次式を満たすように $\mathbf{w}$ を更新していく.

$$f(P_0^{\mathrm{leaf}}) > f(P_{i\neq 0}^{\mathrm{leaf}}) \tag{3}$$

但し, P_i^{leaf} は指し手 i に対する PV の葉局面であり, $i = 0$ は最善手を表す.

ここで, PV (Principal Variation) とは Minimax 探索によって最善と判断された経路であり, root から leaf までの評価値が等しくなる [7]. 従って, その葉局面の静的評価値は root 局面での動的評価値に等しい.

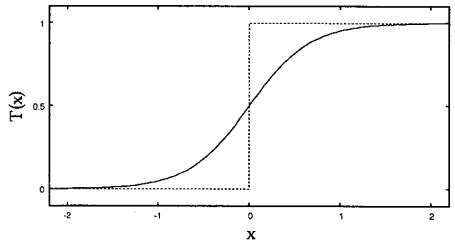

図 2: Sigmoid function.

2.2.2 損失関数

実際には全ての局面かつ全ての指し手 i について (3) 式を満たす解はまず存在しない. そこで, 次式で表される損失関数を定める.

$$L(\mathbf{w}) = \sum_{\text{局面 } P} \sum_{\text{合法手 } i} T(f(P_i^{\mathrm{leaf}}) - f(P_0^{\mathrm{leaf}})) \tag{4}$$

ここで, $T(x)$ はシグモイド関数 (図 2)

$$T(x) = \frac{1}{1+\exp(-ax)} \tag{5}$$

である.

$a \to \infty$ のとき $T(x)$ は

$$T(x) = \begin{cases} 1 & (x > 0) \\ 1/2 & (x = 0) \\ 0 & (x < 0) \end{cases} \tag{6}$$

となる. このとき, 損失 $L(\mathbf{w})$ は $f(P_0^{\mathrm{leaf}}) < f(P_i^{\mathrm{leaf}})$ となった指し手 i の個数に等しい. 従って, 評価関数の最適化問題は損失関数の最小化問題に置き換えられる.

$T(x)$ にはシグモイド関数以外を適用した例もあり,

$$T(x) = \max(x+1, 0) \tag{7}$$

といった直線的な関数を用いてもうまくいくことが示されている [6]. しかし, 最善手の評価 $f(P_0^{\mathrm{leaf}})$ が極めて低い場合もそれを改善する方向に強く働くので, 過学習の危険は高いといえる.

2.2.3 パラメータの更新

損失関数の最小化では 2 次収束のアルゴリズムを適用した例 [2] もあるが, 保木の手法では次式に従ってパラメータを更新する.

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \mathbf{z}^{(k)} \tag{8}$$

ここで, $\mathbf{z}$ の各要素 z_j は w_j の更新値

$$z_j = -h\,\mathrm{sign}(v_j) \tag{9}$$

であり，$\mathbf{v}$ は勾配ベクトル

$$\mathbf{v} = \nabla_{\mathbf{w}} L(\mathbf{w}) \tag{10}$$

とする．また，$\mathrm{sign}(v)$ は v の符号，h は 1 回の更新で各要素を変化させる幅である．但し，駒割り ($j = 1, 2, \cdots, 13$) については (9) 式を用いず，

$$\sum_{j=1}^{13} z_j = 0 \tag{11}$$

となるように調整する．これによって，定数倍の別解や自明な解 $\mathbf{w} = 0$ を除去する．

さらに，$\mathbf{w}$ の要素数が非常に多いため正則化が必要である．そこで，(4) 式に正則化項を加え，

$$L(\mathbf{w}) = \sum_{\text{局面 } P} \sum_{\text{合法手 } i} T(f(P_i^{\text{leaf}}) - f(P_0^{\text{leaf}})) + \lambda \sum_{j=14}^{N} |w_j| \tag{12}$$

とする．これは，3 駒の関係 ($j = 14, 15, \cdots, N$) の大きさを抑えることで，過学習の回避や極小点の減少などの効果がある．

ここで，パラメータの更新を行うと探索結果も変化することが考えられる．従って，実際には単純な最小化問題ではなく，収束に向かう理論上の保証はない．しかし，保木 [1] や金子 [2, 3] により実用上問題のないことが示されている．

2.2.4 探索窓

探索を行う場合に，ある範囲にだけ限って正確な値を知りたい場合がある．その下限値，上限値をそれぞれ α 値，β 値といい，またその範囲を探索窓 (search window) という．探索窓を狭い範囲に限定することで枝刈りが大量に発生し，処理時間を短縮できる．

シグモイド曲線 (図 2) を見ると，$x = 0$ 付近以外では勾配がほぼ 0 であることがわかる．つまり，最善手との評価値の差が十分に大きい局面は扱わなくても影響が少ない．(8) 式は勾配の符号だけを見るため，むしろ除外した方が良い場合もある．そこで，最善手の下を探索して得られた評価値 e_0 を用いて，兄弟局面に対しては $e_0 - M$ から $e_0 + M$ を探索窓とし，その範囲に収まらない局面は除外してしまう．(図 3)

2.2.5 学習手順

学習の流れを図 4 に示す．探索を行い PV を求めることとパラメータを更新することを繰り返す．しかし探索には非常に時間がかかるため，パラメータを複数回 (N 回) 更新する毎に 1 度だけ PV を更新した方が効率が良い [1]．その回数を $N = 50$ とした．

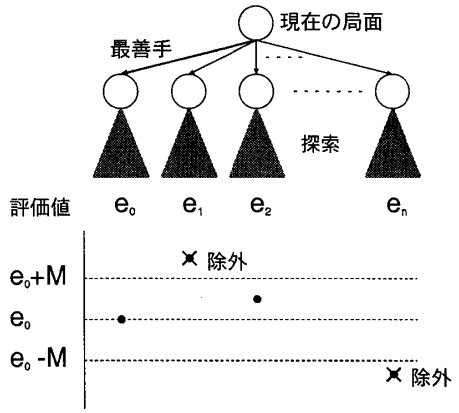

図 3: 探索窓

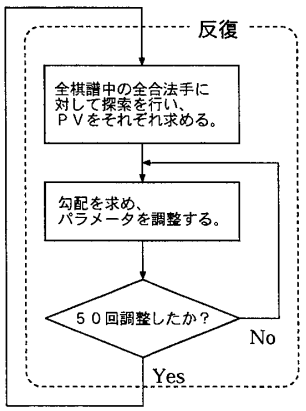

図 4: 学習手順

また，対局時と同等の探索深さでは現実的な時間で全ての探索を終えることができない．そのため極めて浅い探索のみを行う．

3 関連研究

3.1 兄弟節点の比較による評価関数の調整

合法手の中から指し手を絞り込むことによる高速化について，金子による報告がある [2]．人間は与えられた局面で全ての合法手を考えることはない．扱う価値のある手が一部しかないことは機械にとっても同様である．そこで指し手に特徴づけを行い，重要と思われる指し手に限定することで高速化を図る．

金子は ELO レーティングによる指し手の順位付けと絞り込みによって，3 倍程度の高速化を実現している．しかし，絞り込みによる精度の低下が見られるため，絞り込みを行う方が良いかどうかはまだ今後の研究が必要としている．

3.2 既存評価関数のパラメタを活かした適応学習

Bonanza Method は与えられた評価項目に基づいて評価関数の自動調整を行う．つまり，評価項目については人の試行錯誤が必要なのである．評価項目の構成を変更

した場合, 学習を最初からやり直すのが一般的である. しかし, 十分に調整された既存の評価関数を有効に活用できれば, 大きな効率化が期待できる.

そこで, 矢野らはドメイン適応と呼ばれる手法により既存のパラメータを活かして学習を行うことに成功している [4]. 特に, 評価項目が増えただけの場合でも, 既存パラメータをそのまま初期値とするより効率が良い点は興味深い.

4 探索窓に着目した高速化

Bonanza Method により, プログラマの手間は多いに削減された. しかし, 評価項目は未だ人間の試行錯誤が必要であるから, 何度も学習を行うことになる. また, より良いパラメータの獲得には大量の棋譜を用いる必要があり, 処理時間はそれに比例して増加する. そのため, 高速化 (効率化) が求められる.

本稿では探索窓に着目した高速化手法を提案する.

4.1 段階的な探索窓

多くの場合, 探索窓の幅 M とシグモイド関数 (5) 式のスケール a を一定としている. しかし, 調整の初期段階では探索窓の幅を狭めても結果への影響が少ないと考えた.

そこで, 学習が進むにつれて段階的に探索窓の幅を広げた場合, 学習結果や処理時間にどのような影響があるか調べる.

反復回数 j に対して探索窓の幅 M は次式のように直線的に増加するもとのとした.

$$M = \frac{j_{\max} - j}{j_{\max}} \cdot M_{\min} \cdot + \frac{j}{j_{\max}} \cdot M_{\max} \qquad (13)$$

ただし, j は $[0, j_{\max} - 1]$ の範囲で変化するものとする.

$M_{\max} = 256$, $M_{\min} = 256, 64, 32$ について調べる. シグモイド関数のスケール a は探索窓の幅 M に対し,

$$a = \frac{7.0}{M} \qquad (14)$$

とした.

4.2 確率的な復帰

評価値が探索窓に収まらず除外された局面も, パラメータが更新されると次の反復では探索窓に収まることがある. これを復帰と呼ぶことにする.

しかし, 明らかな悪手の後の局面はパラメータが更新されてもいつまでも除外され続ける. 将棋の場合, そのような指し手は非常に多いはずである. 予備実験を行ったところ半分以上の局面が除外され, 除外された局面のうち次の反復で復帰した局面は 5%に満たなかった (探索窓を一定とした場合). さらに, 復帰しても探索窓にぎりぎりで収まった局面は, 勾配が小さく再び除外される確率も高いといえる. このことから, 一度除外された局面は, その後の学習に与える影響が極めて小さいと考えた.

そこで, 前の反復で除外された局面は確率 $q = 1 - p$ で探索することなく除外する. (図 5) ここで p を復帰率と呼ぶことにする. また, 探索窓が狭いほど除外される局面は増える. そこで, 段階的な探索窓と組み合わせることで効率化を図る.

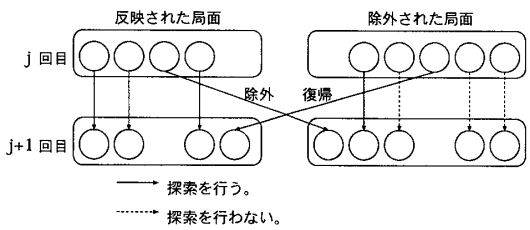

図 5: 確率的復帰

表 3: 復帰率 p - 期待値 E

復帰率 p	期待値 E[回]
1	0
1/2	1
1/4	3
1/8	7
1/16	15
1/32	31
0	∞

一度除外された局面は確率 q^k で k 回連続して探索を行わずに除外される. 従って, k の期待値 E は

$$E = \sum_{k=1}^{\infty} k \cdot q^k \cdot (1 - q) \qquad (15)$$

ここで,

$$E - q \cdot E = \sum_{k=1}^{\infty} q^k \cdot (1 - q) = q \qquad (16)$$

であるから, 式を整理すれば

$$E = \frac{q}{1 - q} = \frac{1 - p}{p} \qquad (17)$$

具体的な p と E の値を表 3 に示す. 処理時間を大きく短縮するには復帰率 p を 1/2 から 1/8 程度まで小さくしなければいけない. しかし, 表 3 からわかるように一度除外された局面はしばらくの間, 探索無しで除外されつづける. そのため, 高速化が期待できる半面, 学習精度を大きく落とす可能性を含んでいる.

5 実験

5.1 実験内容

プロ棋士の公式戦 1000 局を用い, 学習手順 (図 4 の破線部分) を 10 回繰り返す. PV の計算では通常探索の深さを 1 とし, その下で最大深さ 4 の静止探索を行う. ここで, 本研究の高速化手法は学習結果に影響が及ぶため,

表 4: 探索窓 M, 復帰率 p - 処理時間 $t[min]$

$M_{\min}$	p 1	1/2	1/4	1/8	0
256	5494	4684	4182	3918	3596
64	4653	3693	3102	2754	2274
32	4574	3674	3072	2697	2221

表 5: 探索窓 M, 復帰率 p - 不一致度

$M_{\min}$	p 1	1/2	1/4	1/8	0
256	3.7153	3.7906	3.7903	3.8871	4.0361
64	4.2290	4.2781	4.4289	4.5625	5.2586
32	4.3874	4.4680	4.5797	4.8383	5.4878

得られた評価関数の性能を調べる必要がある. そこで, 次式によって棋譜との不一致度を調べる.

$$\text{不一致度} = \frac{L(\mathbf{w})}{\text{局面数}} \tag{18}$$

同じ条件で不一致度を計算するために学習後 $M = 256$ でもう一度 PV を求め直す.

局面の評価項目は表 1 の通りとした. また, 駒割りは適当な初期値 (表 2) を与え, それ以外は全て 0 とした.

探索窓について, 固定 ($M = 256$) と段階的変化 ($M_{\min} = 64, 32$, $M_{\max} = 256$) の計 3 通りを調べる. また, 復帰率について, $p = 1, 1/2, 1/4, 1/8, 0$ の 5 通りを調べる. これらの組み合わせ全 15 通りについて, 処理時間と不一致度を比較する.

5.2 実験結果と評価

探索窓と復帰率を変化させた場合の処理時間と不一致度はそれぞれ表 4, 表 5 の通りである. その分布を図 6 に示す. 図 6 の破線は探索窓が固定 ($M = 256$), かつ復帰

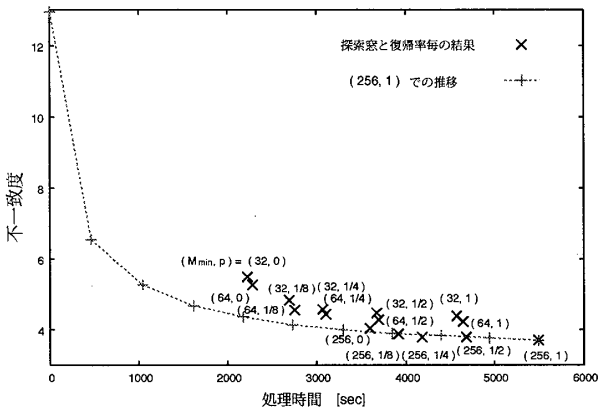

図 6: 各条件における処理時間と不一致度

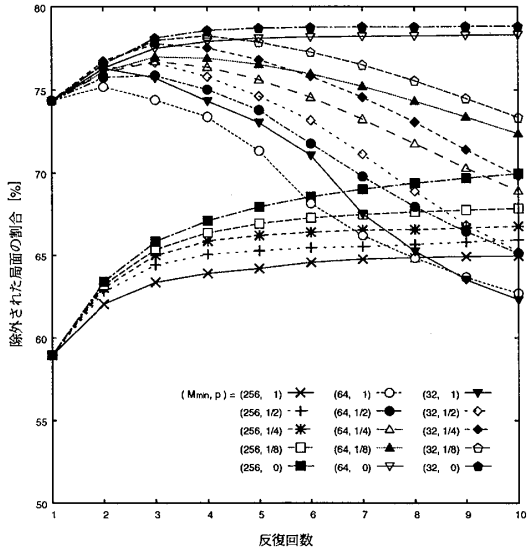

図 7: 反復回数に対する除外された局面の割合

率 $p = 1$ のときの推移である. 尚, 全て 1 回の試行結果である.

処理時間と不一致度について図 6 の破線より下にあれば, 探索窓を固定, 復帰率を 1 とした場合より少ない時間で不一致度を小さくできたことになる. 探索窓の最小幅と復帰率の組 $(M_{min}, p) = (256, 1/4)$ の場合は破線よりも下にあり, 良い結果を得られたといえる.

一方, 段階的な探索窓を用いた場合やそれを確率的な復帰と組み合わせた場合は効果がなかった. また, 確率的な復帰でも復帰率 $p = 1/4$ 以外はほとんど効果が確認できなかった.

ここで注目したいのは, $p = 1/2$ と $p = 1/4$ で不一致度に差が表れなかった事である. 一度除外された局面を無視しても, 学習結果への影響が小さいことを示していると考えられる.

図 7 に, 反復回数に対する除外された局面の割合の変化を示す.

$M_{\min} = 64, 32$ は, 復帰率 p 毎におよそ同じような曲線を描いている. 特に反復回数 1 では差が全く表れなかった. これは初期状態で駒割り以外のパラメータが全て 0 であり, かつ各駒の点数が $M_{\min}$ より大きいためである.

一方, $M_{\min} = 256$ のときだけ初回での除外された局面の数は少ない. また, 図 6 を見ると $M_{\min} \neq 256$ の場合は $M_{\min} = 256$ の場合より相対的に不一致度が大きくなっている. 従って, 学習の初期段階で探索窓を狭めても結果への影響が少ないという仮定は間違っていたといえる.

ここで, $(M_{\min}, p) = (256, 1), (256, 1/4)$ での学習結果の差を見てみたい.

表 6, 表 7 は各駒 1 枚当たりの点数である. 駒割りには大きな差がないようである.

図 8, 図 9 は 3 駒の関係のうち, “8 八玉–XY 金” に対

表 6: 駒割り (256,1)

駒の種別	歩	香	桂	銀
点数	71	263	271	391
点数 (成)	436	492	480	536
駒の種別	金	角	飛	
点数	487	573	667	
点数 (成)		779	954	

表 7: 駒割り (256,1/4)

駒の種別	歩	香	桂	銀
点数	67	274	274	393
点数 (成)	425	487	471	532
駒の種別	金	角	飛	
点数	487	580	667	
点数 (成)		779	964	

-144	-47	-5	0	-40	-228	0	-314	0
0	-4	-2	-1	-2	-120	-12	-58	-28
-2	0	-140	39	-1	-2	-65	-156	-60
-1	-2	-120	0	1	-7	0	1	0
-1	-4	0	-58	0	0	0	1	-1
-2	0	5	12	-33	-49	-48	-72	0
1	52	-11	13	-26	-50	-33	-45	-1
-1	x	46	67	-3	-60	-35	-257	1
-1	0	1	32	-61	-19	-141	0	-1

図 8: 8 八玉と金の位置関係に対する評価 (256,1)

-168	0	-36	-1	-38	-377	-28	-333	-1
47	0	-1	0	0	-108	-3	-99	-41
2	1	-5	49	-1	0	-58	-87	-51
1	-1	-1	-2	1	-52	0	-1	0
0	0	1	-2	1	-2	1	-1	-1
0	-75	-2	16	-2	-47	-12	0	-1
0	46	6	25	-28	-80	-72	-54	0
-2	x	25	20	-23	-78	-59	-161	0
0	29	-1	39	-59	-44	-113	-1	-1

図 9: 8 八玉と金の位置関係に対する評価 (256,1/4)

する評価点である．これは，3 駒の関係のうちほんの一部のパラメータであるが，ところどころ差が認められる．例えば 8 六や 8 九の地点がそうである．ただ，それらはどれも実戦で表れにくい位置関係であり，相対的な影響は小さいと考えられる．

6 まとめ

本研究では探索窓に着目し，段階的な探索窓や確率的な復帰を導入することで将棋における機械学習の高速化を試みた．

直線的に広がる探索窓では効果が確認できなかった．これは，初期段階に選ばれた局面に対する過学習が原因と思われる．

一方，確率的な復帰ではその有効性を示すことができた．この手法は徐々に訓練例を絞り込むものであり，段階的な探索窓とは対称的ともいえる．

Bonanza Method はその実用性が明らかになっているものの，その性質について詳細には明らかにされていない部分が多い．本研究では探索窓について今までにない効率化法を提案し，探索窓と学習結果の関係について新たな側面からの報告を行った．

参考文献

[1] 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, Nov. 2006.

[2] 金子 知適, “兄弟節点の比較に基づく評価関数の調整,” 第 12 回ゲームプログラミングワークショップ, pp.9–16, Nov. 2007.

[3] 金子 知適, 山口 和紀, “将棋の棋譜を利用した, 大規模な評価関数の調整,” 第 13 回ゲームプログラミングワークショップ, pp.152–159, Nov. 2008.

[4] 矢野 友貴, 三輪 誠, 横山 大作, 近山 隆, “既存評価関数のパラメタを活かした適応学習,” 第 14 回ゲームプログラミングワークショップ, pp.1–8, Nov. 2009.

[5] 棚瀬 寧, “棚瀬将棋の技術的背景,” 情報処理 Vol.49, No.8, pp.987–992, Aug. 2008.

[6] 金子 知適, “最近のコンピュータ将棋の技術的背景と GPS 将棋,” 情報処理 Vol.50, No.9, pp.878–879, Sep. 2009.

[7] 篠原 拓嗣, 石田 亨, “N 人ゲームにおける最良優先探索,” 情報処理学会論文誌 Vol.43, No.10, pp.2981–2989, Oct. 2002.

[8] 佐藤 佳州, 高橋 大介, “モンテカルロ木探索によるコンピュータ将棋,” 第 13 回ゲームプログラミングワークショップ, pp.1–8, Nov. 2008.