コンピュータ囲碁におけるブースティングを用いた重み付き多数決合議の提案

眞鍋和子[†] 村松正和[††]

コンピュータ囲碁の重み付き多数決合議において, ブースティングにより重みを決定することを提案する. ブースティングは, 多数用意されたプログラムの重みを自動的に決定することができる. 実験の結果, プロの着手との一致率において, ブースティングによる重み付き多数決の方が単純多数決より良いということがわかった.

Boosting Approach for Consultation by Weighted Majority Vote in Computer Go

KAZUKO MANABE[†] and MASAKAZU MURAMATSU[††]

This paper proposes to use boosting algorithms for consultation by weighted majority vote in Computer Go. Boosting algorithms determine weights of many programs automatically. Experiment results show weighted majority vote by Boosting outperforms simple majority vote in concordance rate with professional players' moves.

1. はじめに

近年, 合議を用いたコンピュータ将棋の棋力向上が著しい. 2010 年には, 4 プログラムの合議によって指し手を決定する「あから 2010」が女流棋士を破り, その力を世界に示した[1]. ゲームにおける合議とは, 疎結合システム上における並列探索のことである. 複数のプレイヤが意見を述べ, それらの意見をまとめて最終的な決定を行う. なぜ合議をすると強くなるのかという理由については, まだはっきりとわかっていない. しかしその有効性は, 実戦において証明されている. 更に近年のハードウェアの進化による計算処理速度向上は, 計算機リソースを多く消費する合議手法の後押しをしている.

コンピュータ囲碁においても, 合議の有効性は示されている. 副島らが行った Root 並列化による合議[2] では, 各プロセスが最善と判断する候補手を一手のみ選択し, 単純多数決により最終的な着手を決定する. この実験では, 従来の探索回数による着手決定に比べて, 良い着手を選出する傾向が確認された. 山下は, 選択された候補手の中から最大の評価値を持つ手を着手とする楽観的合議の有効性を示した[3].

この研究では, 機械学習の手法の一つであるブースティングアルゴリズム（以下, ブースティング）を用いて重み付き多数決合議を実現することを提案し, その効果を検証した.

2. ブースティング

ブースティング[4] とは, 複数の弱仮説を線形結合した統合仮説を作成するアルゴリズムである. 仮説は関数や判別器とも言い換えられ, 入力に対して答えを推測して返す. ブースティングは, より「良い」統合仮説を作るため, 各仮説の意見の重みを学習の中で徐々に調整していく. 本稿では最も代表的なブースティングアルゴリズムである AdaBoost の多値判別版, AdaBoostMlt[4] を用いる.

2.1 事　　例

ブースティングは予め用意された, 入力 $\mathbf{x}$ と判別値 y の組み合わせから成る事例に基づいて学習を行う. 事例集合 S の中の i 番目の事例を $(\mathbf{x}_i, y_i)$ と書くことにする. 入力集合と判別値集合をそれぞれ X, Y で表す.

2.2 仮　　説

仮説が入力 $\mathbf{x}$ を受け取った時, ある判別値 y に与える評価値を $h(\mathbf{x}, y)$ と書き, これを判別評価関数と言う. 弱仮説とはランダムよりは良い答えを返す仮説で, たとえば 2 値判別の場合は, 正解率が 1/2 より大きい仮説は全て弱仮説と呼ぶ.

多値版ブースティングでは, 仮説は次のような集合

† 電気通信大学電気通信学研究科情報工学専攻
Department of Computer Science, The University of Electro-Communications
†† 電気通信大学情報理工学研究科情報・通信工学専攻
Department of Communication engineering and Informatics, The University of Electro-Communications

値関数である.

$$h^{mlt} : X \to 2^Y$$

このような集合値関数のクラスを $\mathscr{W}^{mlt}$ と記述する. このクラスに対応する判別評価関数とクラスを次のように定義する.

$$\mathscr{W} = \Big\{ h : X \times Y \to \mathbf{R} | h(\mathbf{x}, y) = [y \in h^{mlt}(\mathbf{x})], \\ h^{mlt} \in \mathscr{W}^{mlt} \Big\}$$

ここで記号 [] は, 条件文が真のときは 1, 偽の時は 0 である. ブースティングで最終的に得たい統合仮説のクラスは, $\mathscr{W}$ の関数を線形結合して得られる, 次のような判別評価関数のクラスである.

$$\mathscr{L}(\mathscr{W}) = \left\{ f_\lambda | f_\lambda(\mathbf{x}, y) = \sum_{h \in \mathscr{W}} \lambda_h \cdot h(\mathbf{x}, y) \right\}$$

ここで λ_h は, 各仮説 h の重みを表す.

2.3 誤判別率

仮説の「良さ」を表す尺度として,「誤判別率」を定義する. 事例 $(\mathbf{x}_i, y_i)$ と $y' \in Y - \{y_i\}$ の全ての組み合わせにおける誤判別の個数の割合を考えると, 仮説 h^{mlt} の S 上での誤判別率 $err_S(h^{mlt})$ は,

$$err_S(h^{mlt}) = \frac{1}{m} \sum_{1 \le i \le m} \frac{1}{K_i - 1} \sum_{y' \ne y_i} \\ I([y_i \in h^{mlt}(\mathbf{x}_i)] - [y' \in h^{mlt}(\mathbf{x}_i)]) \quad (1)$$

$$I(z) = \begin{cases} 0 & z > 0 \\ 1/2 & z = 0 \\ 1 & z < 0 \end{cases}$$

と表す. ここで, m は事例の個数, K_i は入力 i のときの判別値の個数☆である. この式を用いると, 常に正解のみを推測する仮説の誤判別率が一番低く, 次いで正解と共に間違った答えを返す仮説, 最悪は常に間違った答え全てを返す仮説である. 誤判別率は, 式を見ると 0 以上 1 以下の値を取るように見えるが, 1/2 より大きい値を取るためには, 仮説は敢えて間違えなければならない. つまり, 全てランダムで答えた場合に誤判別率が 1/2 となり, 最悪の値となる.

ところで, 事例は常に正しいとは限らず, 誤差や外れ値が含まれている場合がある. そのような問題に対処するため, 入力例と出力値に対して確率分布を仮定して, 事例の生起確率分布 $P(\mathbf{x}, y)$ を導入する. $P(\mathbf{x}, y)$ は $(\mathbf{x}, y)$ の同時確率分布である. さらに, もう一つの判別値 $y' \in Y - \{y\}$ を含めた拡大同時確率分布 $(\mathbf{x}, y, y')$ を定義する. もし $Y - \{y\}$ の要素間に特に差がなければ,

$$P(\mathbf{x}, y, y') = P(\mathbf{x}, y) \cdot \frac{1}{K - 1}$$

となる. この拡大同時確率分布を使って, 前述の弱仮説の誤判別率 (式 (1)) を一般化すると,

$$err_P(h^{mlt}) = \sum_{1 \le i \le m} \sum_{y' \ne y_i} P(\mathbf{x}, y, y') \\ I(h(\mathbf{x}_i, y_i) - h(\mathbf{x}_i, y')) \quad (2)$$

となる.

弱仮説と同様に, 統合仮説の誤判別率を定義する.

$$err_{P_S}(f_\lambda) = \frac{1}{m} \sum_{1 \le i \le m} \frac{1}{K_i - 1} \sum_{y' \ne y_i} \\ I(f_\lambda(\mathbf{x}_i, y_i) - f_\lambda(\mathbf{x}_i, y')) \quad (3)$$

正解の判別値 y_i に対する評価値が, 間違った判別値 y' に対する評価値より高い場合に誤判別率は低くなる.

2.4 誤判別率の最小化

AdaBoostMlt の目標は, 事例集合 S において最小の誤判別率を持つ統合仮説を得ることである. そのためにどのように係数ベクトル λ を変更すれば良いかを計算したいが, 前章の式 (3) を最小化する λ を計算することは一般には難しい. なぜなら式 (3) は λ に対して階段状の関数になっており, どの方向に進めば誤判別率が減少するのか, 関数の局所的な情報だけで知ることができないからである.

したがって, 不等式

$$I(z) \le e^{-z}$$

を用いて, 式 (3) の上界である次のような期待損失 $R(f_\lambda)$ を考える.

$$R(f_\lambda) = \frac{1}{m} \sum_{1 \le i \le m} \frac{1}{K_i - 1} \sum_{y' \ne y_i} \\ \exp\{-(f_\lambda(\mathbf{x}_i, y_i) - f_\lambda(\mathbf{x}_i, y'))\} \ge err_{P_S}(f_\lambda)$$

この期待損失 $R(f_\lambda)$ を最小化する λ を計算する. λ の更新は $\lambda_{t+1} \leftarrow \lambda_t + (0, ..., 0, \alpha, 0, ..., 0))$ の形で行う.

最小化は座標降下法を用いる. まず降下方向の決定, つまり弱仮説 $h \in \mathscr{W}$ の選択を行う. そのために期待損失 $R(f_\lambda)$ を λ の h 成分 λ_h で微分すると,

$$\frac{\partial}{\partial \lambda_h} R(f_\lambda) = \frac{1}{m} \sum_{1 \le i \le m} \frac{1}{K_i - 1} \sum_{y' \ne y_i} \\ \exp\{f_\lambda(\mathbf{x}_i, y') - f_\lambda(\mathbf{x}_i, y_i)\} \cdot (h(\mathbf{x}_i, y') - h(\mathbf{x}_i, y_i)) \quad (4)$$

となる. この微分値が負で, その絶対値が最大になる h がもとめたい方向である. ここで, 拡大同時確率分布 $P(\mathbf{x}, y, y')$ を, $(\mathbf{x}_i, y_i, y')(1 \le i \le m, y' \ne y_i)$ に対して

$$P(\mathbf{x}_i, y_i, y') = \frac{1}{Z} \cdot \frac{\exp\{f_\lambda(\mathbf{x}_i, y') - f_\lambda(\mathbf{x}_i, y_i)\}}{m} \quad (5)$$

と定める. それ以外の $(\mathbf{x}, y, y')$ に対しては $P(\mathbf{x}, y, y') = 0$ とする. ただし Z は $\sum_{1 \le i \le m} \sum_{y' \ne y_i} = 1$ となるための

☆ 一般には判別値の個数は i に依らず一定だが, 囲碁では入力（盤面）に依って判別値（候補手）の個数が変わる.

正規化定数である. この P を用いて式 (4) を書きかえると,

$$\frac{\partial}{\partial \lambda_h} R(f_\lambda) = -Z \sum_{1 \le i \le m} \frac{1}{K_i - 1} + 2Z \sum_{1 \le i \le m} \sum_{y' \neq y_i} \frac{1}{K_i - 1} P(\mathbf{x}_i, y_i, y') I(h(\mathbf{x}_i, y_i) - h(\mathbf{x}_i, y'))$$

となる. 定数を除いたこの式の最後の項は, 式 (2) の $err_P(h^{mlt})$ にほかならない. したがって,

$$\arg\min_{h \in \mathscr{W}} err_P(h) \tag{6}$$

で得られる仮説 h_{opt} が $\frac{\partial}{\partial \lambda_h} R(f_\lambda)$ を最小にする最適な勾配方向である.

次に, 選択された h_{opt} の係数を求める. これは座標降下法における直線探索の計算に相当する. 直線探索の解は, 極値条件

$$\frac{\partial}{\partial \alpha} R(f_\lambda + \alpha h_{opt}) = 0$$

から求まる. ここで, $p_+(h;P), p_-(h;P), p_0(h;P)$ を以下のように定義する.

$$p_+(h;P) = \sum_{\mathbf{x},y,y'} P(\mathbf{x},y,y')[h(\mathbf{x},y) - h(\mathbf{x},y') > 0]$$
$$p_-(h;P) = \sum_{\mathbf{x},y,y'} P(\mathbf{x},y,y')[h(\mathbf{x},y) - h(\mathbf{x},y') < 0]$$
$$p_0(h;P) = \sum_{\mathbf{x},y,y'} P(\mathbf{x},y,y')[h(\mathbf{x},y) - h(\mathbf{x},y') = 0].$$

すると, 直線探索の解は

$$\alpha_{opt} = \frac{1}{2} \log \frac{p_+(h_{opt};P)}{p_-(h_{opt};P)} \tag{7}$$

となる.

2.5 アルゴリズム

ブースティングは 1 ステップにつき 1 つの仮説の重みを調整し, 徐々に統合仮説の期待損失を下げていく. 期待損失の改善が十分に小さくなったときに終了し, 統合仮説を出力する. 多値判別を行うブースティングの 1 つである AdaBoostMlt のアルゴリズムを, 以下に示す.

ブースティングアルゴリズム概略

t: ステップ数

Initialization:
$P_1 \leftarrow \mathbf{S}$ 上の等確率分布を等確率拡大化したもの;
$\lambda_0 \leftarrow (0, ..., 0)$;
$t \leftarrow 1$;
repeat{
  $h_t \leftarrow P_t$ のもとで最も良い仮説 (式 (6) の解);
  $\alpha_t \leftarrow h_t$ の重要度 (式 (7));
  $\lambda_t \leftarrow \lambda_{t-1} + (0, ..., 0, \alpha_t, 0, ..., 0)$;
  if 期待損失 (式 (4)) が改善されない then break;
  $P_{t+1} \leftarrow$ 確率分布の更新 (式 (5));
  $t \leftarrow t + 1$;
}
統合仮説を出力;

本研究では統合仮説の「良さ」を表す指標として, 一致率 (4 章) を導入している. そのため, 実験では期待損失だけでなく一致率も改善が見られなくなった時に終了するとした.

3. ブースティングの合議への適用

本研究では囲碁プログラムを, 局面を受け取りいくつかの候補手を返す弱仮説と見なすことで, ブースティングと対応させた. 具体的には, 以下のような対応となる.

- 弱仮説 ← 囲碁プログラム
- 事例 $\mathbf{S}$ ← プロ棋譜
- 入力 $\mathbf{x}$ ← 局面
- 判別値 y ← 局面における候補手
- 集合値関数 $h^{mlt}(\mathbf{x})$ ← 評価値が高い 1 つ以上の候補手を出力

この学習では, 集合値関数 $h^{mlt}(\mathbf{x})$ の出力は, 最大の評価値を持つ手と, その最大評価値 $\times 0.7$ 以上の評価値を持つ手全てとし, そのような囲碁プログラムを用意した. 上限数は設定しなかったが, 仮説の返す候補手の数は平均して 8 手程度であった. 学習により得られた統合仮説を持つ合議プログラムは, 入力盤面 $\mathbf{x}$ に対する候補手の中で, 最大の評価値を持つものを着手として決定する. つまり $\arg\max_{y \in Y} f_\lambda(\mathbf{x}, y)$ が合議プログラムの出力となる. ただし, そのような y が複数ある場合は, その中からランダムで選ばれる.

4. 実　　験

2 章で説明した AdaBoostMlt に沿って, ブースティングを行う. 学習には, UCT[5] とプレイアウト中における着手レーティングを実装したプログラムを用いた. レーティングを与えた特徴は以下である.

- 3×3 パターン
- 盤端からの距離
- 一手前の着手からの距離
- 二手前の着手からの距離
- トリ, アタリ, アタリからの逃げ

このレーティング値は, Minorization-maximization[6] を用いて学習した値を用いた. 異なる弱仮説囲碁プログラムを生成するために, 表 1 のような 3 種類の方法を用いる.

表 1　レーティング値に掛けた正規乱数

Type	平均	標準偏差	対象レーティング
Ⅰ	1	0.6	全て
Ⅱ	1	0.6	3×3 パターン
Ⅲ	1	0.9	全て

仮説の「良さ」を表す尺度として, 誤判別率 (2.3 章) がある. しかし誤判別率は, その大小から相対的な良さは分かるものの, 絶対的にどの程度良いのかが直感ではイメージしづらい. そこで, 統合仮説が最大の評価値を与えた手と, 実際のプロの着手が一致している割合を表す「一致率」を以下のように定義する.

$$c_S(f_\lambda) = \frac{1}{m}\sum_{1\le i\le m}\frac{1}{n_i}[y_i = \arg\max_{y\in Y} f_\lambda(\mathbf{x}_i, y)]$$

ただし, $n_i = |\arg\max_{y\in Y} f_\lambda(\mathbf{x}_i, y)|$ である.

4.1　基本実験と新しい提案

TypeI の弱仮説を 32 生成してブースティングを行った. 図 1, 図 2 に, AdaBoost の結果を示す. 横軸がブースティングステップ数, 縦軸がそれぞれ一致率, 誤判別率である. ステップが進むに従って統合仮説の一致率が上がり, 最終的には 0.14 近くになっていることがわかる. しかし, より高い一致率を目指して, 確率分布の更新において次の改良を行った.

$$P(\mathbf{x}_i, y_i, y') = \frac{1}{Z}\cdot\frac{\sqrt[n]{\exp\{f_\lambda(\mathbf{x}_i, y') - f_\lambda(\mathbf{x}_i, y_i)\}}}{m}$$

これは, 式 (5) において, 分子の n 乗根を取ったものである. 図 1, 図 2 に, $n = 2, 10, 16$ の時の結果を重ねて示す. いずれも, 元の AdaBoost に比べて良い値が得られた. 一致率において最大の値を取ったのは 16 乗根で, 約 0.171 である. しかし 10 乗根においても最大値は約 0.170 であり, 16 乗根とほぼ同じ値が得られている. したがって, n を更に大きくしても, これ以上の改善は見込めないと思われる.

n 乗根を取ったものと比べて AdaBoost は値の収束が速く, ステップ数 30 あたりで安定している. 実はこの時, 仮説の重みが正のものは 17 のみであり, 約半数の仮説が使われていない. 対して平方根の仮説数は 27, 10 乗根と 16 乗根は仮説数 30 となっており, より多くの仮説が使われている. これは, n 乗根を取って確率のステップ毎の変化が緩やかになることで, より多くの仮説が選ばれやすくなることにより, 外れ値に強くなったと予想できる. 一方, AdaBoost は一度選ばれた仮説が決定的になりやすい.

以降の実験の結果は, 特に断りがない限り, 一番良い一致率が得られた n を用いてのものとする.

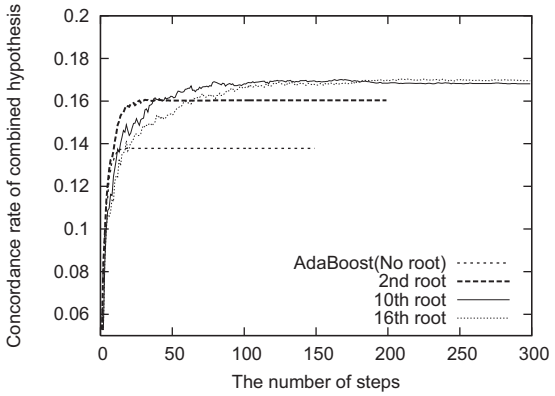

図 1　確率分布の更新が緩和された統合仮説の一致率の遷移

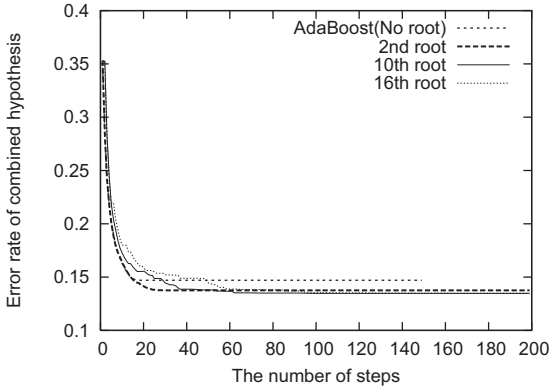

図 2　確率分布の更新が緩和された統合仮説の誤判別率の遷移

4.2　テスト集合に対する性能

本稿では, 総事例数の 7 割を訓練集合, 3 割をテスト集合として用いた. 訓練集合で得た統合仮説を使い, テ

表 2 訓練集合とテスト集合に対する性能の比較

弱仮説	誤判別率		一致率	
	訓練	テスト	訓練	テスト
AdaBoost	0.147	0.140	0.138	0.134
2 乗根	0.138	0.131	0.160	0.161
10 乗根	0.135	0.127	0.168	0.171
16 乗根	0.135	0.127	0.169	0.170

スト集合に対しての一致率と誤判別率を調べたところ, あまり差がないという結果になった. 例として, 4.1 章で実験した統合仮説の一致率と誤判別率について表 2 に示す. なお, 数値はブースティングが終了したステップの時のものである. この結果から, 本研究においてブースティングで作成された統合仮説の汎用性が高いことがわかる.

4.3 仮説数の増加が与える影響

TypeⅡ で生成した弱仮説を使って, ブースティングの効果を仮説数が 8 の時と 32 の時で比較した. また, 仮説数が 32 の時に最も重みが大きい仮説を 8 つ選び, 再度ブースティングを行った. 結果を図 3, 図 4 に示す. 一致率では, 仮説数 8 が 0.110 であるのに対し, 仮説数 32 では 0.154 と良くなっている. 重みの大きい仮説 8 つの場合の一致率は 0.147 であり, 仮説数 32 に比べて悪くはなっているが, 仮説数の大幅な減少に対して一致率の減少幅は小さい. 誤判別率についても, 同様の傾向が読み取れる. この実験から, 統合仮説に用いる弱仮説数は同じでも, 初めにより多くの弱仮説を生成してから選び取った方が良いことがわかる. 生成する仮説数を増やすことにより, より適切なレーティング値を持つ仮説が生成される確率が大きくなることが理由であると考えられる.

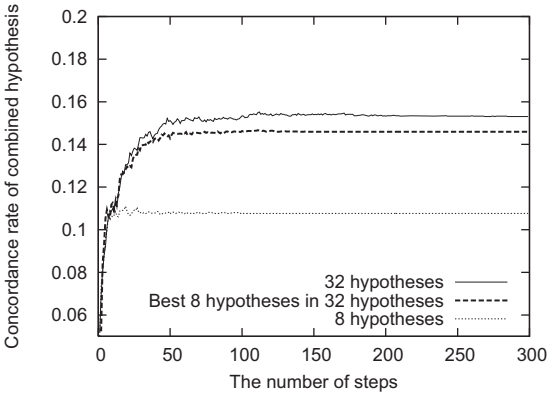

図 3 仮説数の違う TypeⅡ の一致率の遷移

4.4 分散の影響

レーティング値に掛ける乱数の分散が大きければ,

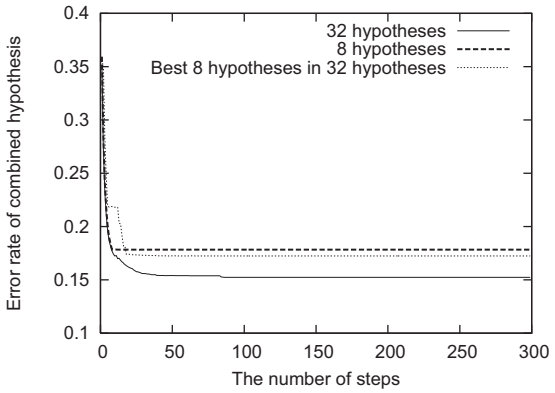

図 4 仮説数の違う TypeⅡ の誤判別率の遷移

生成される弱仮説の性質は大きく異なったものになるであろう. その時のブースティングの効果を見るために, 実験を行った. 具体的には, 表 1 の TypeI, Ⅱ, Ⅲ の仮説をそれぞれ 32 生成し, ブースティングを行った. ステップごとの遷移を図 5, 図 6 に示す. 一致率においては, TypeI が 0.171 で最も大きく, TypeⅡ の 0.155, TypeⅢ の 0.127 と続く. しかし最低の一致率であった TypeⅢ が, 誤判別率では 0.128 で最も良い数値となった. これは, TypeⅢ の統合仮説が, 正解の着手に最大ではないものの良い評価値を与えていると言える.

TypeI と TypeⅡ は, 掛けた乱数の標準偏差は 0.6 で同じだが, 乱数を掛ける対象とした特徴が TypeI の方が多い. Minorization-maximization によって得られた特徴のレーティング値は, 本来なら最適値であるはずである. したがってそれらに乱数を掛ければ当然悪くなるのだが, 統合仮説においてはそれが多様性となり, より分散が大きい TypeI の方が良い結果となっているようだ.

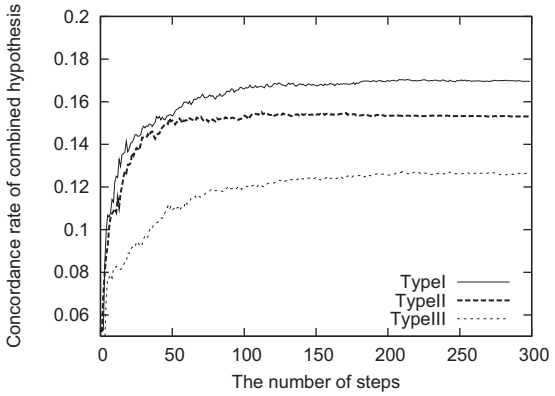

図 5 TypeI, TypeⅡ, TypeⅢ の一致率の遷移

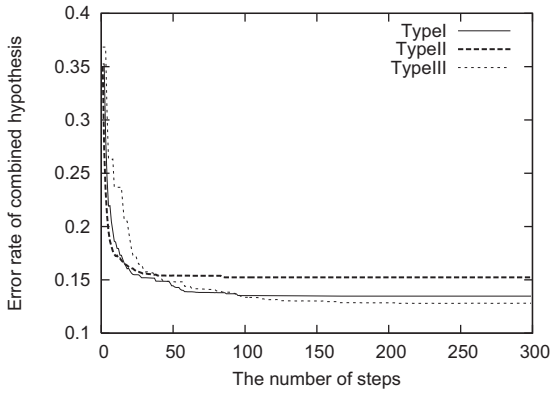

図 6 TypeI, TypeⅡ, TypeⅢ の誤判別率の遷移

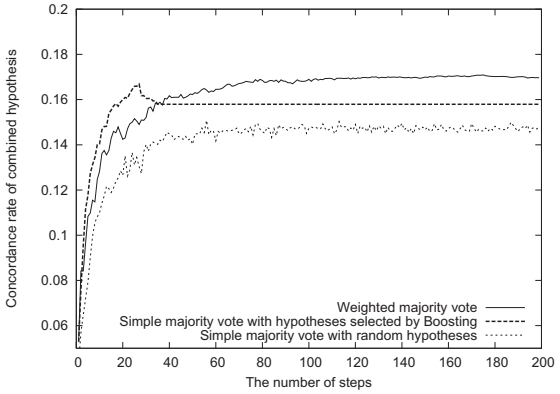

図 7 TypeI の重み付き多数決と単純多数決の一致率の遷移

4.5 単純多数決との比較

ブースティングの統合仮説による重み付き多数決と, ブースティングで選ばれた仮説と同じ個数の仮説をランダムで選んだ単純多数決を比較した. 結果を図 7, 図 8 に示す. この単純多数決の誤判別率と一致率は, 10 回の平均を取っている. 用いた弱仮説は, TypeI の 32 の弱仮説である. その結果, 誤判別率ではほとんど差が出なかったものの, 一致率においては重み付き多数決が単純多数決を上回った.

次に, ブースティングで選ばれた弱仮説の重みを 1, それ以外の重みを 0 とした単純多数決について調べた. 結果は同様に図 7, 図 8 に示す. やはり誤判別率には他との目立った差はなかったが, 一致率はステップ 27 で最大値 0.167 を記録した後, 8 ステップ後には 0.158 に減少し, 収束した. 一方で, 重み付き多数決の一致率は最大値は 0.171 で, 特に目立つ減少はなくそのまま収束している. この単純多数決での急激な一致率の減少は, ステップが進んで初めて選ばれる弱仮説, つまり相対的にあまり良くない仮説に対し, 他の弱仮説と同じ重みを与えていることによるものだと思われる. 参考として, 各弱仮説の重みの遷移を図 9 に示す.

この実験から, ブースティングにより選ばれた弱仮説を用いて単純多数決を行っても, 重み付きの場合とほぼ同じ性能が出ることがわかった.

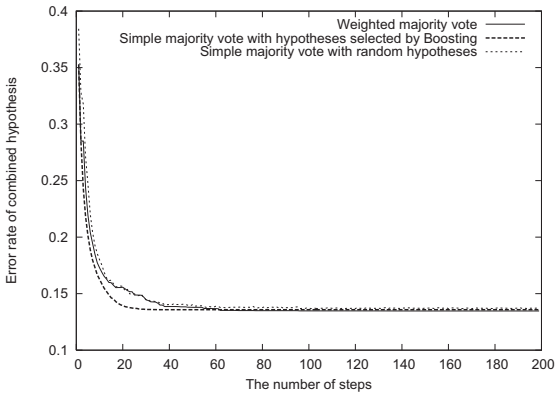

図 8 TypeI の重み付き多数決と単純多数決の誤判別率の遷移

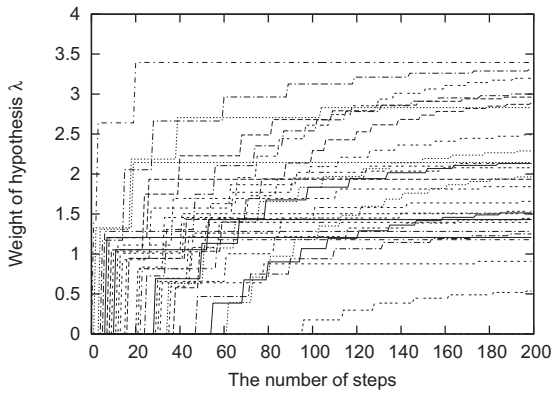

図 9 TypeI のブースティングにおける各弱仮説の重みの遷移

4.6 対　　局

表 1 の TypeI のレーティング値を用いた仮説数 32 の学習の中で, 最も高い重み (3.40) を示した仮説プログラムと, 最も低い重み (0.00) のプログラムを単体で対局させた. 結果は表 3 のようになった. 統合仮説における重みに大きな差があるにも関わらず, 実際の対局では強さにあまり差がないということがわかる.

次に, 合議プログラムと単体プログラムの対局を行った. 合議プログラムは, TypeI の仮説を用いたブースティングにおいて, 最も高い重みを示した仮説 4 つで再度ブースティングを行ったものである. 対局相手としては, 乱数を掛けていないオリジナルのレーティング値による単体囲碁プログラムを用いた. 結果は表 4 のようになった. 合議プログラムが単体プログラムに対し, 有意に勝ち越している.

表 3　重みの異なる仮説プログラムの対局結果

プレイアウト数	低い重みの仮説に対する勝率
8,000	51.14 ± 3.82 %

表 4　合議と単体プログラムの対局結果

プレイアウト数	単体プログラムに対する勝率
8,000	58.95 ± 5.06 %

5. ま と め

本研究では, 特徴のレーティング値に乱数を掛けた囲碁プログラムを弱仮説と見立て, 複数の弱仮説を生成してブースティングを行った. その結果, 誤判別率, 一致率において改善が見られ, 単純多数決よりも良い結果となった. 対局実験においては, 単体のプログラムに合議が勝ち越すことができている.

今後は実験の条件によって学習の効果に変化があるかを確認する必要がある. 本稿では自作のプログラムを用いたが, 他のプログラムにおいてどの程度有効であるかどうかを調べたい. また, 一手ごとのプレイアウト数を変えた時に, 効果がどう変わるのかも興味深い.

謝　　辞

研究を行うにあたり, ご助言いただいた保木邦仁先生, 計算機資源を快く使わせてくださった研究室の学生の皆様に, 深く感謝申し上げます.

参 考 文 献

1) 情報処理学会, **清水市代女流王将** *vs.* **あから** *2010* **速報**, http://www.ipsj.or.jp/50anv/shogi/20101012.html (2011/9/21 アクセス)
2) 副島 佑介, 岸本 章宏, 渡辺 浩, **モンテカルロ木探索の** *Root* **並列化とコンピュータ囲碁での有効性について**, Game Programming Workshop 2009, pp.27-33 (2009)
3) 山下 宏, **囲碁で楽観的合議**, コンピュータ将棋や囲碁の掲示板, http://524.teacup.com/yss/bbs/1827 (2011/9/21 アクセス)
4) 金森 敬文, 畑埜 晃平, 渡辺 治, **ブースティング — 学習アルゴリズムの設計技法—**, 森北出版 (2006)
5) Sylvain Gelly, Yizao Wang, Rémi Munos, Olivier Teytaud, *Modification of UCT with Patterns in Monte-Carlo Go*, Technical Report 6062, INRIA, 2006.
6) Rémi Coulom, *Computing Elo Ratings of Move Patterns in the Game of Go*, In Computer Game Workshop, Amsterdam, The Netherlands, (2007)