ゲームプログラムのためのパターン型評価関数の自動生成法

金子知適[†] 山口和紀[††] 川合慧[†]

本稿では，ゲームプログラムのために必要となる評価関数を，ゲームのルールから自動的に作りだす手法を提案する．評価関数とは局面のよしあしを数量化する関数で，評価関数を自動生成するためには局面の様々な特徴を数量化するための特徴関数の自動獲得が必要となる．機械学習の分野では，汎用性のある述語論理で記述した特徴関数を対象に自動獲得の手法が研究されてきた．しかしながら述語論理は，効率化を行わないと局面評価の効率が悪いという問題点がある．一方，図形的パターンを用いて特徴関数を表現すると局面評価効率が良いことが知られているが，これまでパターンを自動的に生成する汎用性のある方法はなかった．本稿では，述語論理で記述した特徴関数から機械的な操作でパターンを抽出し，評価関数に用いる手法を提案する．本手法で生成された評価関数は，パターンを用いているため局面評価効率が良く，また本手法は生成の際にゲームのルール以外の知識を必要としないため汎用性が高い．オセロおよびはさみ将棋を対象に実際に評価関数を作成し，提案した手法の有効性を確認した．

Automatic Construction of Pattern-based Evaluation Functions for Game Programming

Tomoyuki Kaneko,[†] Kazunori Yamaguchi[††] and Satoru Kawai[†]

In this paper, we describe a method that automatically constructs evaluation functions without any assistance by expert players of a target game. Such automated construction of evaluation functions is crucial to developing a general game player that can learn to play an arbitrary game. As for the automatic generation, features written in logic programs (called logical features) are useful, because they can be generated from the rules of a game. As for the evaluation, conjunctive formulas (called patterns) can be evaluated more efficiently than logical features, and the accuracy of the evaluation functions of patterns can be better, because they can employ a larger number of patterns, which means a larger number of adjustable parameters in the linear combination. Our approach is to generate logical features and then extract patterns from the logical features. By this two step process, we can get the generality of the logical features and the practicality of patterns. Experiments on Othello and Hasami-Shogi showed significant improvements on efficiency and accuracy of evaluation functions, which are now approaching to that of the strongest program in Othello.

1. はじめに

ゲームプログラミングの研究目標の 1 つは，様々な思考ゲームをプレイできる汎用ゲームプレイヤを開発することである．囲碁や将棋などの思考ゲームはルールは簡単でも状態空間が大きく，単純な探索で最善手を求めることはできないため，ゲーム特有の知識を探索に組み込む必要がある．そのため，そのような知識を自動獲得する手法が重要な研究課題となっている．

† 東京大学総合文化研究科広域科学専攻
Graduate School of Arts and Sciences, The University of Tokyo
†† 東京大学情報基盤センター
Information Technology Center, The University of Tokyo

1.1 評価関数と特徴関数

思考ゲームの探索では，局面の「勝ちやすさ」を数値で測定する「評価関数」と呼ばれる関数が用いられる．強いゲームプログラムを作るためには，正確でかつ局面評価効率の良い評価関数が必要となる．評価関数を作る有力な方法は，局面を数量化する特徴関数を複数用意し，それらの重み付きの和（線形結合やニューラルネットワークなど）とするものである．特徴関数には，オセロの石の数や将棋の駒の数のように簡単な関数が使われる．重みを自動調整する研究は進んでいるが，特徴関数は人間が与えた研究が多い[10),13)]．

1.2 特徴関数の自動生成

汎用ゲームプレイヤを作るには，プレイするゲームの評価関数を，特徴関数を含めて自動生成する手法が必要である．我々の大きな目標は，特定のゲーム専用

の手法で作られた評価関数と同等の性能の評価関数を汎用の手法で自動生成することである．本稿ではゲームのルールから，特徴関数を生成する手法を提案し，実際に評価関数を生成できることを示す．

次章では関連研究の問題点と解決のアイデアを述べる．3 章ではゲームのルールを論理式で記述する方法を説明する．続いて 4 章で特徴関数としてパターンを生成する方法を，5 章で効率的な局面評価方法を提案する．6 章で実験結果を示し，7 章で本稿を結ぶ．

2. 関連研究と問題点

2.1 パターンを用いることの問題点

図形的パターンは特徴関数の表現として広く使われている．特徴関数として出力する値は，パターンが局面にマッチすれば 1，そうでなければ 0 とする[2]．パターンの生成・選択はある程度の自動化が提案されているが，多くの場合，対象のゲームにおける重要な形を人間が指定することでパターンの候補の数を抑えている．したがって，このアプローチを任意のゲームに適用可能な汎用のものにすることは難しい．生態学的アプローチ[9] により囲碁の様々な形のパターンを獲得した研究もあるが，この手法は隣の石が重要であることに間接的に依存するため，たとえば将棋など駒の効きが重要なゲームへの応用は難しいと考えられる．

2.2 論理特徴を用いることの問題点

汎用ゲームプレイヤを志向した研究[4),8),12)] では，述語論理（ホーン節）で記述した特徴関数（以下本稿では論理特徴と呼ぶ）を採用している．特に Zenith システム[3),4)] では，述語論理で記述されたゲームのルールから，論理特徴を自動的に生成する方法を提案している．この方法は，ゲームのルールとして与えた論理式から，条件の分解や 1 手前の状況の生成といった汎用の変形ルールを用いて論理特徴を生成する．個々のゲームの性質に依存しないため，ゲームのルールを述語論理で記述するだけで異なるゲームに適用できる．

一方，論理特徴には局面評価がきわめて遅いという問題点がある．特徴関数の値を計算する際に論理式の評価を行う必要があるためで，この計算はパターンのマッチングよりも際立って遅い．局面評価の遅さは，探索で可能なノード数が減り，また使用できる特徴関数の数が減るため評価関数の正確さも損なう．

2.3 解決のアイデアと枠組み

論理特徴をまず生成し，それらをパターンに変換することができれば，汎用性と評価効率を両立させることが可能になる．本稿では，そのための，論理特徴からパターンを抽出する手法を提案する．具体的には以下の手順で評価関数を自動生成する．

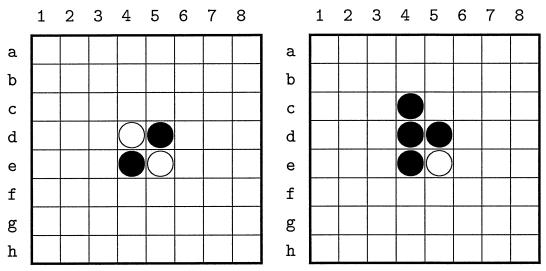

図 1 オセロの初期局面（左）とその 1 手後（右）の局面と事実
Fig. 1 Othello initial position (left) and a position after black played c4 (right). Facts under each board define the position.

(1) ゲームのルールから論理特徴を自動生成[4)]
(2) 論理特徴をパターンに変換
(3) 有用なパターンを選択
(4) （線形結合などの）重みを調整

実際に，近年研究されている論理特徴の効率的な評価[7)] の結果と比較して，パターンへの変換により効率と正確さを大幅に向上させることができた．

3. 論理式によるゲームの記述と論理特徴

本章では，論理特徴の定義と，論理式によりゲームのルールを記述する方法をオセロを題材に説明する．

3.1 局面の記述

局面は事実（頭部のみのホーン節）の集合で定義される．本稿の例ではオセロの局面を，石のあるマスを表す`owns` と空白のマスを表す`blank` という述語により定義する．**図 1** に局面とその定義の例を示す．以後これらの局面を表現する事実をさして単に局面と呼ぶ．

3.2 ルールの記述

ゲームのルールとして，盤面の静的な形状，合法手や勝利条件を記述する．例として 4×4 のサイズのオセロのルールを**図 2** に掲載する．この例では`square` と`neighbor` によって盤面上のマスの配置を，また`legal_move` によって，着手可能なマスを定義した．

3.3 論理特徴関数の記述

論理特徴は以下の例のようにホーン節で表現する．

```
f(A):-owns(x,A).   % DISCS FOR BLACK☆
```

☆ 文献 4) では，`f2(Num):-count([A],(owns(black,A)),Num).` となっているが，この形式の論理特徴では組み込み述語 “count” が必須であるので，本稿では簡単に count の中のみを表記する．

```
%rules
legal_move(Square, Player):-
  square(Square), bs(Square,_FlipEnd,Player).
bs(S1,S3,P):-blank(S1), opponent(P,Opp),
  neighbor(S1,Dir,S2), span(S2,S3,Dir,Opp),
  neighbor(S3,Dir,S4), owns(P,S4).
span(S1,S2,Dir,Owner):-square(S1), square(S2),
  player(Owner), owns(Owner, S1),
  neighbor(S1,Dir,S3), span(S3,S2,Dir,Owner).
span(S,S,Dir,Owner):-
  square(S), player(Owner), owns(Owner,S),
  direction(Dir).
line(S,S,Dir):-square(S),direction(Dir).
line(From,To,Dir):-
  neighbor(From, Dir, Next), line(Next,To,Dir).
%board topology
opponent(x, o). opponent(o, x).
direction(n).direction(ne).direction(e).
direction(se).direction(s).direction(sw).
direction(w).direction(nw).
square(a1). square(a2). square(a3). square(a4).
（中略）
square(d1). square(d2). square(d3). square(d4).
neighbor(a1, s, a2). neighbor(a2, n, a1).
neighbor(a2, s, a3). neighbor(a3, n, a2).
（中略）
neighbor(c4, ne, d3). neighbor(d3, sw, c4).
```

図 2 オセロのルール（4 × 4）
Fig. 2 A sample domain theory of Othello (4 × 4).

論理特徴が出力する「値」は，解の個数とする．ただし，ホーン節に対してその節を真にするような（頭部の変数に対する）束縛を「解」と呼ぶ．

上記の例では`A`が変数であり，`owns`は第 1 引数のプレイヤ（黒を表す定数`x`）の石が第 2 引数のマス（変数`A`）にあるという意味を表す述語である．したがって，この論理特徴`f(A)`の値は（その局面においての）黒石の数に相当する．評価関数が局面を評価する際には各特徴関数の値を用いる．局面が与えられたときにこの値を求めることを論理特徴の「評価」と呼ぶ．

3.4 局面の更新と論理特徴の評価

ゲームのルールを表す述語の中で，定義が局面に依存するもの（たとえば着手可能なマスを表す`legal_move`）は，局面に応じて解が異なる．論理特徴にはそのような述語が使われるので，論理特徴を評価する際には，局面に応じた解を求める必要がある．この計算の効率化が課題である．

4. パターンの自動生成

本章では論理特徴に含まれるパターンを抽出する手法を提案する．まず論理特徴を等価な基底（変数を持たない）節に変換し，続いてパターンを抽出する．

```
ic1(S) :- blank(S), owns(_Player,S).
ic2(S) :- owns(x,S), owns(o,S).
```

図 3 オセロの一貫性制約
Fig. 3 Integrity constraints of Othello.

```
legal_move(a1,o):-blank(a1),owns(x,a2),owns(o,a3).
legal_move(a1,o):-blank(a1),owns(x,b1),owns(o,c1).
legal_move(a1,o):-blank(a1),owns(x,b2),owns(o,c3).
```

図 4 `legal_move`を変換した結果の一部
Fig. 4 Partial result of unfolding `legal_move`.

4.1 部分計算による基底節への変換

論理特徴を基底節に変換するには部分計算の要素技術である *unfolding* と *pruning* を用いる．それらは論理プログラムの意味（解）を変えない変換である[1)]．

4.1.1 Unfolding

Unfoldingは節 A:-$A_1, ..., A_{i-1}, A_i, A_{i+1}, .., A_n.$ を $(A$:-$A_1, ..., A_{i-1}, B_1, ..., B_h, A_{i+1}, ..., A_n.)\theta_j$ となるような節すべてで置き換える．ただし B:-$B_1, ..., B_h.$ は代入 θ_j に対して $B\theta_j = A_i\theta_j$ となるような節とする．

4.1.2 Pruning

Pruningはつねに偽となるような節を取り除く．次のような場合が相当する．

(1) つねに偽となるような項を含んでいる．
(2) 一貫性制約に違反する．

この pruning を効率的に行うために，演繹データベースで使われる一貫性制約（integrity constraint）を導入した．例として**図 3** に，図 2 のゲームのルールに対応する一貫性制約を掲載した．1 行目の`ic1`は空白のマスに石はないという制約を表す．次の`ic2`は黒白両方の石があるマスはないという制約を表す．

4.1.3 基底節への変換

上記の unfolding と pruning を節に変数がなくなるまで繰り返し適用することで，ホーン節を基底節に変換することができる．途中，局面でない基底項は静的に真偽を計算可能であり，部分計算によって取り除かれるため，最終的に得られる基底節の本体部は局面のみから構成される．例として，図 2 の述語`legal_move`を変換して得られる節の一部☆を**図 4** 示す．なお，この例ではルールを変換したが，論理特徴も同様に変換可能である．

4.2 パターンの抽出

以上の部分計算により得られた基底節の本体部をパターンとする．部分計算により得られた基底節は，頭部の引数は解に対応し，本体部は局面のみで構成され

☆ ほかに 3 つ節があるが，スペースの都合上省略した．

- blank(a1) ∧ owns(x,a2) ∧ owns(o,a3)
- blank(a1) ∧ owns(x,b1) ∧ owns(o,c1)
- blank(a1) ∧ owns(x,b2) ∧ owns(o,c3)

図 5 `legal_move` から抽出されたパターンの一部
Fig. 5 Sample patterns extracted from `legal_move(a1,o)`.

る．したがって抽出されるパターンは論理特徴の解を局面の積和標準形式で表現したときの積の部分である．

例として，図 4 の解`legal_move(a1,o)` から抽出されたパターンを**図 5** に示す．3.1 節で述べたように`owns` や`blank` は 1 つのマスの状態を表すため，パターンは図形的特徴の連言となっている．通常，1 つの論理特徴関数から複数のパターンが抽出される（オセロの実験では平均 20 個程度であった）．

4.3 パターンの選択

実行時の効率向上と過学習の回避のために，生成したパターンから有用なもののみを選択して用いる必要がある．一般に，特徴選択は難しい問題であり[6]，正確な選択を行うためには莫大な計算を必要とする．ゲームプログラミングでは，訓練例に対しマッチした局面の数が閾値以上のものを選ぶという簡便な方法が提案されており[2]，本稿の実験でもそれを採用した．

5. パターンによる局面評価

5.1 パターンの包含関係と差分計算

パターンを用いた評価関数で局面評価を行う際には，対象となる局面に対して大量のパターンのマッチングを行う必要がある．本章ではそのための効率的なアルゴリズムを提案する．基本的なアイデアは，(1) 探索の際には差分の小さい局面を次々に評価することが多いので，更新が必要なパターンのみを計算する差分計算を行うことと，(2) 抽出したパターンには包含関係を持つものが多いため，差分計算の際に包含関係を利用することである．例として，6 つのパターン（$P = \{a\text{-}b\text{-}c,\ a\text{-}b,\ b\text{-}c,\ a, b, c\}$）を持つ評価関数を考える．ここで各 a，b，c は局面の定義に用いられる項（たとえば`owns(o,a1)`）とする．また a-b は a と b の連言を表すとする．局面の変化の際に，項 a の真偽値が変わったとすると，すべてのパターンのマッチングをやり直すのではなく，a を含むパターン a-b-c, a-b, a のみ考える．さらにパターン a-b がマッチしない限り，パターン a-b-c はマッチしないという依存関係があるので，これを利用して不要なマッチングを省く．

5.2 カバーリスト

まず，パターン間の依存関係を，ハッセ図で表現する．**図 6** は，先の例のパターン P に関するハッセ図[5]である．各パターンは論理式（項の連言）であり，パターンが局面にマッチする（しない）とはその論理式の真偽値が真（偽）であることである．論理式 p が q を含意することを半順序 $p \geq q$ で記述する．またパターン p に含まれる項 $\{e \mid e$ は項, $p \geq e\}$ を $base(p)$ で表す．$(base(p) \supseteq base(q)) \Leftrightarrow (p \geq q)$ である．

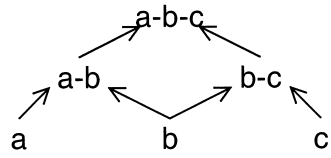

図 6 パターンの包含関係に関するハッセ図
Fig. 6 A Hasse diagram of sample patterns.

ハッセ図では，ノードにパターンを対応させ，p が q を「カバー」する場合にノード q から p へ伸びる辺を描く．ただし，$p \neq q$ かつ $p \geq q$ かつ $p \geq r \geq q$ であれば $p = r$ または $r = q$ となる場合を，p が q をカバーするという．ノード p をカバーするノードを $upd(p)$，ノード p がカバーするノードを $dep(p)$ で表す．$upd(p)$ を p のカバーリストと呼び，パターンの真偽値を更新する際に用いる．

5.3 カウンタによる真偽値の表現

続いて，各ノードにカウンタを用意する．カウンタの値と真偽値を対応させ，カウンタを操作することで各ノードの真偽値を高速に更新するためである．

まず，ノード x に対応する論理式の真偽値を $val(x)$ とする．そして，ノード x の子ノード $dep(p)$ のうち真であるものの数を表すカウンタ $cur(x)$ を次の式によって定義する．

$$cur(x) = |\{c \in dep(x) | val'(c) = T\}|^{\star} \qquad (1)$$

ここで $val'(x)$ は，カウンタから計算される真偽値で次のように再帰的に定義されるものとする．

$$val'(x) = \begin{cases} val(x) & (x \text{ が葉}) \\ cur(x) = |dep(x)| & (\text{それ以外}) \end{cases} \qquad (2)$$

（議論を簡単にするために，項 1 つのみからなるパターンはすべて存在し葉であるとする．）

性質 1 すべてのノード x で，

$$val(x) = val'(x) \qquad (3)$$

が成り立つ．

証明 1 ノードの高さに関する数学的帰納法により証明する．ここでノード x の高さ $h(x)$ を葉までのパ

☆ $|A|$ を集合 A の濃度の意味で用いる．また真偽値をそれぞれ T, F で表し，論理式 f の値が真であることを $f = T$ と表す．

```
W ← {p_f}. // p_f は真から偽になった事実
while W ≠ ∅ do // W は更新すべきノード
    Pick q ∈ W.
    W ← W − {q}.
    for each r ∈ upd(q) do
        cur(r) ← cur(r) − 1 // q の変更をカウント
        if cur(r) < dep(r) // どれかの子ノードが偽
            W ← W ∪ {r} // r の値は偽に更新
W ← {p_t}. // p_t は偽から真になった事実
while W ≠ ∅ do // W は更新すべきノード
    Pick q ∈ W.
    W ← W − {q}.
    for each r ∈ upd(q) do
        cur(r) ← cur(r) + 1 // q の変更をカウント
        if cur(r) = dep(r) // 子ノードがすべて真
            W ← W ∪ {r} // r の値は真に更新
```

図 7 カバープロパゲーションのアルゴリズム
Fig. 7 The cover propagation algorithm.

スの最大の長さと定義する：

$$h(x) = \begin{cases} 0 & (\text{葉}) \\ 1 + \max_{c \in dep(x)} h(c) & (\text{それ以外}). \end{cases}$$

高さ 0 では定義により式 (3) が成立する．また高さ n まで式 (3) が成立する仮定をおくと，高さ $n+1$ のノード x について $val(x) = T \leftrightarrow (\forall_{c \in dep(x)} val(c) = T)$ である．高さの定義から $\forall_{c \in dep(x)} h(c) \leq n$ であり，帰納法の仮定から $val(x) = T \leftrightarrow (\forall_{c \in dep(x)} val(c) = T) \leftrightarrow (\forall_{c \in dep(x)} val'(c) = T) \leftrightarrow (cur(x) = |dep(x)|) \leftrightarrow val'(x) = T$．以上，帰納法により式 (3) が成立する．

5.4 カバーリストによる真偽値の更新

図 7 に提案するアルゴリズム，カバープロパゲーションを示す．これは局面の真偽値の変更を入力としてネットワークのカウンタを更新するアルゴリズムである．性質 1 から，このアルゴリズムでカウンタを更新後，式 (2) を用いてパターンの真偽値を求められる．

アルゴリズムの動作を以下の例により説明する．図 6 の 5 つのパターン $P = \{a\text{-}b\text{-}c, a\text{-}b, b\text{-}c, a, b, c\}$ を考える．**表 1** に upd と $|dep|$ を示す．そして，$val(a) = F$, $val(b) = T$, $val(c) = T$, $cur(a\text{-}b) = 1$, $cur(b\text{-}c) = 2$, $cur(a\text{-}b\text{-}c) = 1$ という状況で，局面の変化にともない項 a が真になったとする．以下，ある事実が偽から真になった場合の変更（アルゴリズムの後半）を説明する．真から偽になった場合の変更も本質的に同じである．

表 1 カバーリスト
Table 1 Cover lists of sample patterns.

	upd	$\|dep\|$
a	$a\text{-}b$	0
b	$a\text{-}b, b\text{-}c$	0
c	$b\text{-}c$	0
$a\text{-}b$	$a\text{-}b\text{-}c$	2
$b\text{-}c$	$a\text{-}b\text{-}c$	2
$a\text{-}b\text{-}c$	$\emptyset$	2

まず，W が $\{a\}$ となる．次の行で a が W から選択され W は $\emptyset$ となる．$upd(a) = \{a\text{-}b\}$ であるから，r として $a\text{-}b$ が選ばれる．$cur(a\text{-}b)$ が $1+1 = 2 = dep(a\text{-}b)$ となったので W は $\emptyset \cup \{a\text{-}b\} = \{a\text{-}b\}$ となる．次のループで，W の中から $a\text{-}b$ が q として選ばれ W は $\emptyset$ に戻る．続いて，$upd(a\text{-}b) = \{a\text{-}b\text{-}c\}$ であるから，$a\text{-}b\text{-}c$ が r として選ばれる．$cur(a\text{-}b\text{-}c)$ は $1 + 1 = 2$ であり $cur(a\text{-}b\text{-}c) = dep(a\text{-}b\text{-}c)$ であるが，$upd(a\text{-}b\text{-}c) = \emptyset$ なのでアルゴリズムはここで止まる．

その結果，$val'(a\text{-}b) = (cur(a\text{-}b) = dep(a\text{-}b)) = T$, $val'(a\text{-}b\text{-}c) = (cur(a\text{-}b\text{-}c) = dep(a\text{-}b\text{-}c)) = T$ であるが，実際 $val(a\text{-}b) = (val(a) \wedge val(b)) = (T \wedge T) = T$, $val(a\text{-}b\text{-}c) = (val(a) \wedge val(b) \wedge val(c)) = (T \wedge T \wedge T) = T$ であり，アルゴリズムは適切に真偽値を更新していることが確認できる．

5.5 冗長な辺の削除による最適化

最適化として $upd(p)$ を減らすことで，空間効率が向上し，またアルゴリズムがカウンタを更新する回数が減るために時間効率も向上する．実際，このアルゴリズムは $upd(q)$ がすべての p について $(\bigcup_{q \in dep(q)} base(q)) = base(p)$ を満たす限り正しく動く．たとえば，4 つのパターン $\{a\text{-}b\text{-}c, a\text{-}b, b\text{-}c, c\text{-}a\}$ を考えると，$a\text{-}b\text{-}c \in upd(a\text{-}b)$, $a\text{-}b\text{-}c \in upd(b\text{-}c)$, $a\text{-}b\text{-}c \in upd(c\text{-}a)$ である．この場合 $a\text{-}b\text{-}c$ を $upd(c\text{-}a)$（また他の 2 つのどちらか）から抜くことができる．これはもしパターン $a\text{-}b$ と $b\text{-}c$ が真であれば事実 a，b，c は真であり，パターン $a\text{-}b\text{-}c$ は真であるからである．

6. 実　　験

本稿で提案した内容を実装し，オセロとはさみ将棋を対象に実際に評価関数を作成しその性能を評価した．

6.1 オ セ ロ

正確さおよび効率の比較のために 3 種類の評価関数を訓練した．それぞれ以下の特徴関数を用いている．

A. Zenith システム[4] が作った 18 の論理特徴
B. A の論理特徴を本稿の手法で変換した約 4 千のパターン
C. Buro が用いた約 27 万のパターン[2]

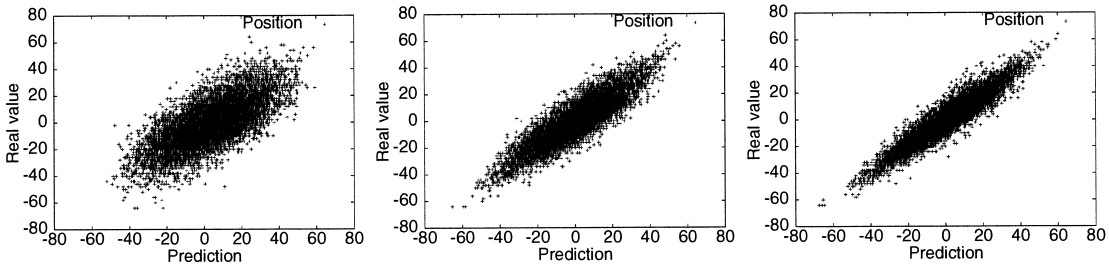

図 8 評価関数の予測値と実際値（オセロ，石数 60）
Fig. 8 Predictions by evaluation functions vs. real values (Othello, 60 discs).

表 2 評価関数の正確さ
Table 2 Accuracy of evaluation functions.

discs		A	B	C
60	r	0.67	0.88	0.94
	$\sqrt{V_e}$	12.9	8.17	5.77
55	r	0.74	0.81	0.89
	$\sqrt{V_e}$	12.5	10.7	8.39

表 3 局面の選好に関する評価関数の正確さ（オセロ，石数 60）
Table 3 Accuracy of move selection (Othello, 60 discs).

A	B	C
0.75	0.84	0.89

すべての評価関数は，簡単のため線形結合を用いた．訓練例としてベストプレイ後の終局時の石の数の差を全探索により求め，それを用いて線形結合の重みを線形回帰で調整した．訓練例の局面としては，IOS[☆]の棋譜のなかで最後まで指された約 30 万局面を用いた．

評価関数の正確さ

表 2 および**図 8** に結果を示す．テストには LOGISTELLO と KITTY の棋譜[☆☆]から約 5 万局面を用いた．表の r は相関係数を，V_e は誤差の分散を表す．図は散布図で横軸が予測値，縦軸が正しい値である．また，2 つの局面を与えたときにどちらが良い局面かを正しく判定する確率について行った実験結果を**表 3** に示す．本稿で提案した手法により生成した評価関数 B の正確さは，元の論理特徴を用いた場合のもの A と比べて大きく改善され，オセロの強いプログラムの評価関数 C に近い性能を実現できたといえる．この理由は，元の評価関数 A では 18 の重みしか用いないのに対して，パターンに変換した B では調整する重みは約 4000 となり，より細かな局面評価が可能になった効果と考えられる．

表 4 オセロの局面評価の効率（1000 局面/sec.）
Table 4 Efficiency of evaluation functions (Othello, 1000 positions/sec.).

A	B
3.18	86.6

局面評価効率

続いて局面評価効率について測定した，A の局面評価には文献 8) のアルゴリズムを，B には本稿で提案したアルゴリズムを用いた．結果を**表 4** に示す．

実験に使用したプログラムは GNU C++でコンパイルし Pentium III 933-MHz CPU の PC（FreeBSD）上で動かした．使用した局面は，IOS の記譜から無作為に 23 記譜を選び，その 49 手目の局面から df-pn$^+$ 探索[11] を用いて勝ち負けを読み切った際に訪れた約 300 万局面[☆☆☆]である．局面評価効率においても，パターンへの変換により性能を大きく改善したといえる．

総合して，評価関数の正確さ・効率とも向上させることができたことで，より強いプログラムが作成可能になったといえる．

正確さ向上のための課題

今後，さらに正確さを向上させるためには，多数のパターンを生成し，より良いパターンを探すことが有効と考えられる．パターンの生成手法は，現状で十分多数のパターンを扱える．文献 4) の手法により生成した約 1 万の論理特徴について，それらを基底節に変換した際のコストを**図 9** と**図 10** に示す．論理特徴によりばらつきがあるものの，ほとんどのものは所要時間，大きさともに十分扱える範囲である．

一方，パターンの選択方法については新たに研究が必要である．今回の実験ではマッチする局面数により

☆ ftp://external.nj.nec.com/pub/igord/othello/ios/
☆☆ ftp://external.nj.nec.com/pub/igord/IOS/misc/
☆☆☆ パラメータは $h_\phi = h_\delta = cost_\delta = 0$, $cost_\phi = -2$ を用いた．

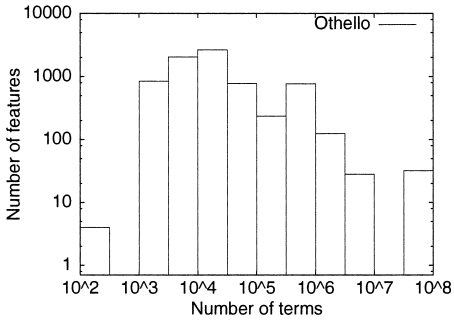

図 9 基底節の項の数の分布（オセロ）
Fig. 9 Number of terms in unfolded features (Othello).

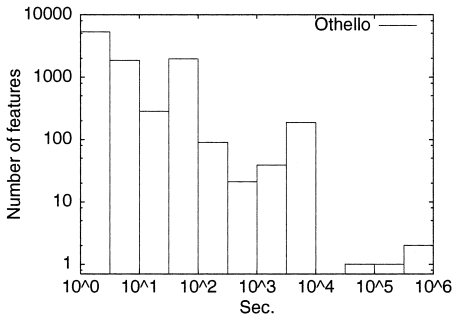

図 10 基底節への変換時間（オセロ）
Fig. 10 Time required for unfolding (Othello).

選択を行ったが，多数の局面にマッチするパターンが必ずしも有用というわけではないため，この手法は多数の候補から少数に絞るには向かないからである．

6.2 はさみ将棋

続いて，駒を動かすタイプのゲームとして「はさみ将棋」を題材に選び，特徴関数の自動生成とパターンへの変換，局面評価効率について実験を行った．重みを調整するために必要な適切な訓練例を作ることが難しいため，正確さの評価は行わなかった．

まず，ゲームのルールを記述し☆，文献 4) の手法を用いて，自動生成した論理特徴約 2 千個をパターンに変換した．変換のコストを**図 11** および**図 12** に，オセロでの実験同様にヒストグラムで示す．

続いて，局面評価効率を測定した．1 回の比較について論理特徴 30 個を無作為に選び，それをそのまま用いた評価関数と，パターンに変換して用いた評価関数の 2 つを作り，それらの局面評価効率を比較した．乱数で駒を配置した局面に対し，30 回比較を行った．結果の散布図を**図 13** に示す．横軸が論理特徴を用いた場合の評価速度，縦軸がパターンを用いた場合の評価速度である．全体的に約 10 倍以上の効率向上を達成している．

☆ ただし，実験時間の都合で盤の大きさを 6×6 で行った．

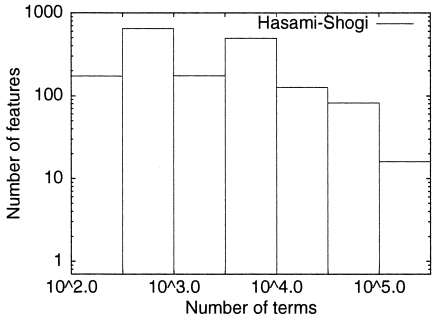

図 11 基底節の項の数の分布（はさみ将棋）
Fig. 11 Number of terms in unfolded features (Hasami-Shogi).

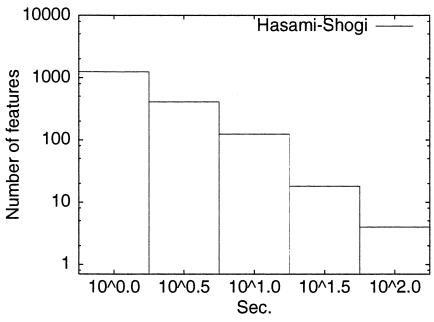

図 12 基底節への変換時間（はさみ将棋）
Fig. 12 Time required for unfolding (Hasami-Shogi).

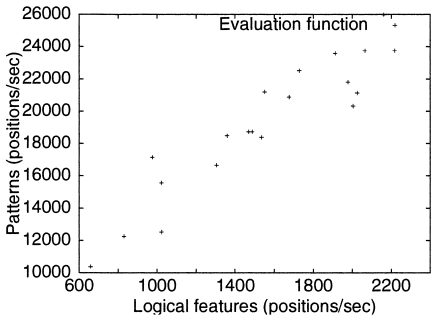

図 13 局面評価効率の比較（はさみ将棋）
Fig. 13 Efficiency of evaluation functions (Hasami-Shogi).

7. ま と め

本稿では，パターンを用いた評価関数の自動生成手法を提案した．論理特徴から機械的にパターンを抽出する手法と，変換されたパターンの効率の良い局面評価手法を開発し組み合わせたことで，実用的な評価関数を汎用な手法で自動生成することに成功した．オセ

ロの実験では，従来手法で自動生成した評価関数を，正確さ・局面評価効率とも大きく上回った．効率の向上は，はさみ将棋の実験でも同様に確認され，本手法の汎用性が示された．

より正確な評価関数を生成するためには，大量の候補から少数のパターンを効率良く選択する研究が重要である．最終的に残すパターンの数に関しても，正確さと効率のバランスのとり方についての研究が必要である．また，汎用ゲームプレイヤの実現のためには，パターンの使用が難しいといわれているゲームへの応用や，良い手だけを選択的に生成する候補手生成器の自動生成への応用などが今後の課題として残っている．

参 考 文 献

1) Bossi, A., Cocco, N. and Dulli, S.: A Method for Specializing Logic Programs, *ACM Trans. Programming Languages and Systems*, Vol.12, No.2, pp.253–302 (1990).
2) Buro, M.: Improving heuristic mini-max search by supervised learning, *Artificial Intelligence*, Vol.134, No.1–2, pp.85–99 (2002).
3) Fawcett, T.E. and Utgoff, P.E.: Automatic Feature Generation for Problem Solving Systems, *Proc. 9th International Conference on Machine Learning*, pp.144–153, Morgan Kaufmann (1992).
4) Fawcett, T.E.: Feature Discovery for Problem Solving Systems, Ph.D. Thesis, Department of Computer Science, University of Massachusetts, Amherst (1993).
5) Gries, D. and Schneider, F.B.: *A Logical Approach to Discrete Math*, Springer-Verlag, New York (1993).
6) Jain, A., Duin, P. and Mao, J.: Statistical pattern recognition: a review, *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol.22, No.1, pp.4–37 (2000).
7) Kaneko, T., Yamaguchi, K. and Kawai, S.: Compiling Logical Features into Specialized State-Evaluators by Partial Evaluation, Boolean Tables and Incremental Calculation, *PRICAI 2000 Topics in Artificial Intelligence*, Melbourne, Australia, pp.72–82 (2000).
8) Kaneko, T., Yamaguchi, K. and Kawai, S.: Automatic Feature Construction and Optimization for General Game Player, *The 6th Game Programming Workshop*, IPSJ Symposium Series 2001, No.14, pp.25–32 (2001).
9) Kojima, T., Ueda, K. and Nagano, S.: An Evolutionary Algorithm Extended by Ecological Analogy and its Application to the Game of Go, *Proc. 15th International Joint Conference on Artificial Intelligence*, Nagoya, Japan (1997).
10) Levinson, R. and Weber, R.: Chess Neighborhoods, Function Combination, and Reinforcement Learning, *Computer and Games*, Marsland, T.A. and Frank, I. (Eds.), LNCS, No.2063, pp.133–150, Springer-Verlag (2001).
11) Nagai, A. and Imai, H.: Application of df-pn+ to Othello Endgames, *Game Programming Workshop in Japan '99*, pp.16–23 (1999).
12) Pell, B.D.: Strategy Generation and Evaluation for Meta-Game Playing, Ph.D.Thesis, University of Cambridge (1993).
13) Tesauro, G.: Practical issues in temporal difference learning, *Machine Learning*, Vol.8, pp.257–278 (1992).

(平成 14 年 2 月 22 日受付)
(平成 14 年 9 月 5 日採録)

金子 知適（正会員）

1997 年東京大学教養学部卒業．2002 年同大学大学院総合文化研究科博士課程修了．博士（学術）．2002 年東京大学院総合文化研究科助手．思考ゲーム，知識処理に興味を持つ．

山口 和紀（正会員）

1978 年東京大学理学部数学科卒業．1981 年東京大学理学部助手．1985 年理学博士．1989 年筑波大学電子情報工学系講師．1992 年東京大学教養学部助教授．1999 年東京大学情報基盤センター教授．モデリング全般に興味を持つ．ACM，IEEE CS 各会員．

川合 慧（正会員）

1967 年東京大学理学部物理学科卒業．1981 年東京大学理学部情報科学科助教授．1984 年東京大学教育用計算機センター助教授．1988 年東京大学教養学部教授．1996 年東京大学総合文化研究科教授，理学博士．研究分野グラフィクス，プログラミング，ユーザインタフェース．電子情報通信学会，ソフトウェア科学会，ACM 各会員．