ゲーム構成要素を組み合わせた特徴の最適化

矢野友貴[†1] 三輪誠[†2]
横山大作[†3] 近山隆[†1]

近年の計算資源の充足に伴い, ゲーム構成要素を単純に組み合わせた特徴がコンピュータゲームプレイヤの評価関数に広く用いられるようになった. 組み合わせ特徴は対象とする問題の知識に依らず簡単に設計可能であるが, 組み合わせ爆発を起こすために特徴数が膨大となり, 高次の組み合わせ特徴を扱うことは現在の計算機では困難となっている. 本稿では, ゲーム構成要素間の関係性に注目して有効な組み合わせ特徴の絞り込みを行うことにより, 従来扱うことが困難であった高次の組み合わせ特徴を活用する手法を提案する. 将棋の評価関数を例に既存手法と比較した結果, 精度面で一定の向上を得ることに成功した.

Optimizing conjunctive features of game components

Yuki YANO,[†1] Makoto MIWA,[†2] Daisaku YOKOYAMA[†3]
and Takashi CHIKAYAMA[†1]

With abundant computing resources that have become available with recent information processing apparatus, expressing game positions with features based on all the possible combinations of primitive features has become worth considering. Constructing the set of all possible combinations is easy without deep knowledge of the subject, however, combinatorial explosion results in a huge set of features, which can hardly be handled efficiently even with massive computational resources. In this paper, we propose a new method to skim off the most effective features from the set of high dimensional conjunctive features. Applying this method to the game of shogi, the evaluation function constructed shows better accuracy than those obtained using conventional methods.

1. はじめに

コンピュータゲームプレイヤにおいて, 局面の有利不利を正確に判断する評価関数は強いプレイヤを作成する上で重要な要素の一つである. 精度のよい評価関数を作成するためには, 形勢判断に有効と考えられる特徴を見つけ出すことが必要となる. 有効な特徴の発見は, 対象とするゲームに関する熟練した知識が要求され, また, 特徴の吟味のために膨大な時間が必要となるなど, 非常に手間のかかる困難な作業である.

近年では, 単純にゲームの構成要素のすべての組み合わせを特徴とした評価関数が広く用いられている. この背景には, 巨大な特徴群を扱えるだけの計算資源が容易に確保できるようになったこと, また機械学習技術の発展に伴い, 今まで人手で扱うことが難しかった膨大な量の特徴を調整できるようになったことが挙げられる[1),2)]. 例えば, 将棋では駒の種類, 位置, 手番を用いた組み合わせ特徴が広く用いられており, 将棋プログラム Bonanza[3)] では 2 玉と 1 駒, 1 玉と 2 駒のすべての組み合わせ, 約 1 億パラメタを機械学習によって調整することで, 高い精度の評価関数の設計に成功している.

組み合わせ特徴は, 従来の特徴の発見とは異なり, ゲームに対する知識に依らずに簡単に設計が可能であり, また比較的高い精度を得ることができる. 組み合わせ特徴はゲームの分野に限らず, 例えば自然言語処理の分野[4),5)] などでもその有用性が示されている. 一方で, 組み合わせ特徴には 2 つの問題がある. 一つは組み合わせ数が増えると簡単に組み合わせ爆発を起こす点である. 将棋の駒の位置関係を用いた組み合わせ特徴の場合, 2 駒間で約 500 万, 3 駒間で約 117 億, 4

†1 東京大学大学院工学系研究科
Graduate School of Engineering, The University of Tokyo
†2 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
†3 東京大学生産技術研究所
Institute of Industrial Science, The University of Tokyo

駒間で約 26 兆と, 3 駒間以上ではメモリ上に展開するには非現実的なサイズとなる. もう一つはすべての組み合わせが必ずしも有効なわけではなく, 表現が冗長となる点である. これは特に組み合わせ数が多い場合に問題となり, 過学習等の原因になる. このように, 組み合わせ特徴は計算コスト, 空間コストの両面で大きな問題を抱えており, 実際の問題では限られた組み合わせ数しか用いられていないのが現状である.

本研究では, ゲーム構成要素間の関連性の強さに注目することで, 組み合わせ特徴を選択する手法を提案する. 具体的には, ゲームに対する知識や棋譜から得られる情報を元に組み合わせ特徴を絞りこむことによって, 組み合わせ爆発, 冗長性の問題を軽減し, 従来よりも高次の組み合わせ特徴を用いる評価関数の設計を行う. 将棋を例に提案手法を既存手法と比較実験したところ, 対戦実験では有意な差を得られなかったものの, 精度面で一定の向上を得ることに成功した.

本論文では以降, 2 章にて関連研究について述べたのち, 3 章にて提案手法を, 4 章にて提案手法に関する実験と結果について述べ, 最後に 5 章にてまとめと今後の課題を述べる.

2. 関 連 研 究

2.1 組み合わせ特徴と評価関数

コンピュータゲームプレイヤでは, 組み合わせ特徴としては盤面上のゲーム構成要素の配置パターンがよく用いられる. 将棋では, ゲーム構成要素として駒, 升が主に用いられ, それらの組み合わせ特徴を用いた評価関数に関する研究が行われている.

金子等は, 将棋の評価関数として駒の 2 項関係を評価するモデルの提案を行った[6]. 金子等の研究では, 2 駒の関係を用いることによって既存の評価要素の多くを表現可能であることを示しており, 兄弟モデルによる学習を行うことで棋譜に近い指し手の生成に成功している. 一方で, 金子等は大駒の長距離利きなど 2 駒間だけでは表現できない評価要素についても言及しており, 精度向上の方法としてより多くの駒関係の利用の可能性について述べている.

三輪等は, 棋譜から抽出したゲーム構成要素を組み合わせて詰将棋の評価関数の自動生成を行った[7]. 三輪等は, 棋譜中の各局面を駒位置と利きの 2 つの単純な特徴要素の集合とし, それら特徴要素の論理積の飽和頻出パターンの抽出, カイ 2 乗値を用いた有効なパターンの選別をすることで評価関数の生成を行っている. 得られた評価関数を学習した結果, 従来の人手による評価関数に近い精度を得ることに成功している.

2.2 多項式カーネルと計算の効率化

組み合わせ特徴は特徴数が非常に膨大となるため, 多くの場合は多項式カーネルを介して利用される. 多項式カーネルは, 入力ベクトルの組み合わせ特徴の内積を暗に計算するカーネル関数である. 入力ベクトルを $\mathbf{x}$, 組み合わせの最大数を d とすると, 多項式カーネルは (1) 式のように計算される.

$$k(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 \cdot \mathbf{x}_2 + l)^d \qquad (1)$$

多項式カーネルは非常に汎用性が高いカーネル関数であるが, 膨大なデータを扱う場合に計算速度の遅さが問題となり, 高速化に関する研究がなされている.

工藤等は, 多項式カーネルを線形和に展開し, 各特徴の重みを近似することで高速化をする手法を提案した[4]. 工藤等の手法では, PrefixSpan を利用して組み合わせ特徴の重みを計算し, 事前に設定した閾値以下の重みの特徴を削ることで計算コストの削減を行っている. 実験の結果, 従来の手法に比べて精度を落とさずに最大で 300 倍の高速化に成功している.

吉永等は, 高頻度で出現するパターンを事前に計算することで高速化をする手法を提案した[5]. 吉永等の手法では, 頻度に加え, 事前に計算することで短縮できる計算コストの見積もりも行い, 2 つが共に高い要素群の全組み合わせの重みの総和を事前に計算し, 実際の分類で展開の省略を行う. 実験では最大 10 倍程度の高速化を実現しており, 特に次数の高い組み合わせ特徴で効果が高いことが示されている.

2.3 ハッシュ関数と次元削減

部分木, 部分グラフ, 配置パターンといった特徴は次元数が非常に大きく, すべてをそのままメモリ上に展開することは現実的ではない. このような空間コストの問題に対して, ハッシュ関数を用いて次元削減を行う試みがなされている.

Silver 等は囲碁において, 碁石の配置パターンのハッシュ値を利用した評価関数の学習を行っている[8]. Silver 等の研究では, 5×5 までのパターンに対して, 4×3 以上のパターンを Zobrist Hash[9] でハッシュ値に変換することで膨大なパターンの利用を可能にしている. 9 路盤において強化学習を用いて評価関数を調整し, CGOS にて対戦をした結果, Elo rating で+1210 [⋆1]を得ることに成功したと述べられている.

Shi 等はハッシュ関数を用いたカーネル関数の提案を行っている[10]. ハッシュカーネルは (2) 式のように, 特徴のインデックス集合 $\mathcal{J}$ に対してハッシュ関数

⋆1 実験時ではランダムプレイヤが-170, 最も強いプレイヤが+1860

$h(i)$ を適用し, ハッシュ値が同じ要素の個数の内積を計算することで 2 つのインスタンスの類似度を求める.

$$\bar{k}(x, x') = \langle \bar{\phi}(x), \bar{\phi}(x') \rangle \quad (2)$$
$$\text{where} \quad \bar{\phi}_j(x) = \sum_{i \in \mathcal{J}; h(i)=j} \phi_i(x)$$

なお, (2) 式中の $\phi_i(x)$ はインスタンス x の特徴ベクトルの i 番目の要素を表す. ハッシュカーネルは, 非常にスパースな特徴を扱う場合や, サンプリングによって一部の特徴から全体の傾向を見るような場合に有効性が高い. Shi 等の研究ではグラフ分類問題に対して, ハッシュカーネルと MCMC を組み合わせることで従来手法よりも高い精度が得られることが示されており, 構造データといった複雑な問題においても適用可能な高い汎用性が示されている.

2.4 実現確率探索

実現確率探索は鶴岡等によって提案されたゲーム木探索手法である[11]. 一般的なゲーム木探索では深さを閾値として探索を行うが, 実現確率探索では各局面の実現確率を閾値として用いる. 局面 x の実現確率 P_x は (3) 式のように, 直前の局面 x' の実現確率 $P_{x'}$ と指し手 m の遷移確率 P_m の積によって再帰的に計算される.

$$P_x = P_m \cdot P_{x'} \quad (3)$$

各指し手の遷移確率は局面から得られる情報を利用して計算される. 例えば, 激指[12] ではロジスティック回帰によって各指し手の遷移確率の学習を行っている. 実現確率を用いることによって, より実現されやすい局面を重点的に探索することが可能となる.

3. 提 案 手 法

既存のコンピュータゲームプレイヤの評価関数では, 単純にすべての組み合わせを展開することで組み合わせ特徴を利用している. そのため, 計算機の制約から利用可能な組み合わせ数が大きく制限される. 将棋における駒情報を用いた組み合わせ特徴の場合, ほとんどのコンピュータゲームプレイヤは 2 駒間までしか用いておらず, 3 駒以上の利用も限られた範囲に限定されている. より精度の高い評価関数を作成するためには, 局面のより詳細な特徴を活用する必要がある. そのため, 従来よりも高次の組み合わせ特徴を有効に活用できれば, 精度向上が得られる可能性が高い.

本研究では, 棋譜から得られる情報を元に組み合わせ特徴の絞りこみを行うことで, 組み合わせ特徴の選別を行う手法を提案する. 具体的には, 盤面を駒や升

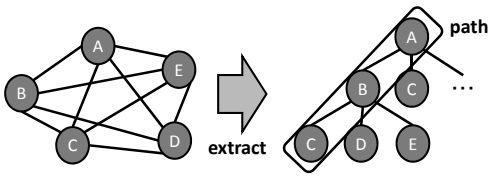

図 1 グラフの展開とパス生成

によって構成されたグラフであると仮定し, 各エッジに結合度を定義することで組み合わせの絞り込みを行う. 本手法では, グラフの各ノードに種類, 絶対位置, 手番によるラベル[⋆2]を付け, 組み合わせ特徴をグラフ中のパスとして定義し, 図 1 のように各頂点からパスを展開することで組み合わせ特徴を生成する.

盤面をグラフと考えた場合, 従来の組み合わせ特徴は盤面をゲーム構成要素を全対全でつないだ完全グラフと考えてすべてのパスを生成することに相当する. しかし, グラフのエッジの数を $|E|$, パスの最大サイズを d とすると, グラフ中のパス, つまり組み合わせ特徴の総数のオーダーは $\mathcal{O}\left(|E|^{d-1}\right)$ と非常に巨大なものになるため, すべてのパスを展開することは困難である. これは, 従来の組み合わせ特徴が各ゲーム構成要素間の関係を等価に扱うために, 無駄な組み合わせ特徴を生成していることが原因であると考えられる. 本手法ではこの問題に対し, 各パスの重要度を用いたパスの絞り込みを行う. 具体的には, 図 2 のように各要素間に結合度を定義し, 各パスの重要度をそれを構成するエッジの結合度の積として計算する. パスの重要度があらかじめ設定した閾値を下回った場合, パスは枝刈りされ, それ以降の展開は行われない. 結合度としては以下のようなものが考えられる.

(i) ゲームの事前知識に基づく結合度
(ii) 棋譜から学習した結合度

本研究では, (i) として利きに基づく結合度を, (ii) としてロジスティック回帰により学習した結合度を用いた.

高次の組み合わせ特徴を用いる場合, 最終的に出現する組み合わせ数がメモリに収まるサイズに収束しない可能性がある. また, 上記の枝刈りに伴いパスの長さが不均一になるため, 各組み合わせ特徴にどのようにインデックスを振るかが問題となる. これらの問題に対し, 提案手法では Zobrist Hash[9] を用いたインデックス付けを行う. 具体的には, 盤面グラフ上の各ノードまたは各エッジに対して乱数を振り, (4) 式のようにパス p 上の各要素 n (ここではノード) の乱数 $r(n)$ の XOR を計算することでハッシュ値を求める.

⋆2 駒のない升は種類を空升とする

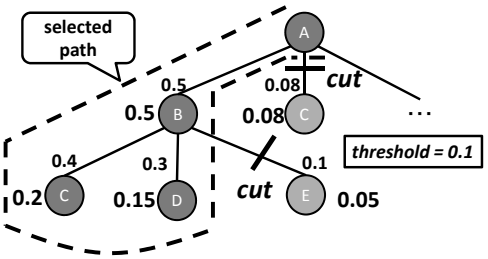

図 2 結合度を用いたパスの絞り込み

なお, エッジを用いる場合にはパスが経由した方向を区別するために, 各エッジに対して方向に応じた 2 つの乱数を設定した. ハッシュを用いることにより, あらかじめ特徴の総数をハッシュのサイズに決定することができ, また (4) 式を用いることによって各パスのインデックスを容易に計算可能となる.

$$h(p) = r(n_1) \oplus r(n_2) \oplus \cdots \oplus r(n_{|p|}) \qquad (4)$$

なお, ノードに乱数を振る場合にはノードが重複しないように, エッジに振る場合にはエッジが重複しないようにパスの生成を行うこととした.

評価関数の重みベクトルを $\mathbf{w}$, 盤面のグラフを G としたとき, 提案手法での評価関数 $f(\mathbf{w}, G)$ は (5) 式のように定義される.

$$f(\mathbf{w}, G) = \sum_i w_i \phi_i(G) \qquad (5)$$

$$\text{where} \quad \phi_i(G) = \sum_{p \in G, 1<|p|\leq d, h(p)=i} \lambda^{|p|-1}$$

ここで, d は最大組み合わせ数, λ は高次組み合わせ特徴の影響を制御する減衰定数である. (5) 式において, パスの長さ $|p|$ の範囲を $1 < |p| \leq d$ と制限しているのは, 2 要素以上の組み合わせ特徴を対象としているためである.

4. 実　　験

4.1 実験方法

本手法の有用性を評価するために, 将棋プログラム「激指[12]」を用いて実験を行った. 激指は, 第 20 回世界コンピュータ将棋選手権において優勝を納めるなど, トップレベルの強さを誇る将棋プログラムである. 本実験では, 第 20 回世界コンピュータ将棋選手権でのバージョンを利用した. 評価関数として単純な駒割に提案手法や比較手法の特徴を追加したものを用意し, それぞれに対して学習を行うことで性能評価を行った.

提案手法では結合度として次の 2 つを用いた.

- 利き (間接利きを含む) のあるエッジを 1.0, その他のエッジを 0.0 とする結合度 (利きグラフ, kiki)
- ロジスティック回帰によって棋譜から学習した各駒間の結合度 (駒グラフ, ko)

なお, 利きグラフではゲーム構成要素として空升も考慮し, パス生成時の起点には駒のみを用いた. また, 駒グラフでは乱数をノードに, 利きグラフではエッジにそれぞれ振った. これは, 駒グラフではパス上の駒の組み合わせのみを考慮すればいいのに対し, 利きグラフではパス上のエッジの向き情報が重要となると考えたためである.

ハッシュ関数を用いた場合, 特徴数の圧縮による精度の低下が懸念されるが, KKP+KPP を 4.4 節で述べる近似データで学習した予備実験では, 表 1 のようにハッシュサイズが 2^{22} までは精度をある程度維持しつつ特徴数を最大で 9%程度まで削減できる結果が得られている. 実験ではハッシュサイズを 2^{22}, 最大組み合わせ数を 4 とした.

学習手法としては激指の方法[13] を用いた. 激指では, ボナンザメソッド[1] 及び Averaged Perceptron[14] をベースにした学習手法が用いられている. ボナンザメソッドとは, 棋譜の手筋に沿うような評価関数を得る学習手法で, 具体的には各局面において棋譜の手が他の手に比べて相対的に高く評価されるようパラメタの調整を行う. Averaged Perceptron はオンライン学習の一種で, 学習途中の各ステップでのパラメタの総和を保持し, それを総ステップ数で割った値を最終的なパラメタとする手法である. Averaged Perceptron ではパラメタの平滑化を行うことで, ノイズに強い学習を実現している. 各ステップでの具体的なパラメタ $\mathbf{w}$ の更新は (6) 式の通りである.

$$\mathbf{w} \leftarrow \mathbf{w} + \frac{1}{|M|} \sum_{j \in M} (\boldsymbol{\phi}(t_1) - \boldsymbol{\phi}(t_j)) \qquad (6)$$

ここで, t_j は j 番目の合法手の後からの最善応手手順後の局面 (1 番目の合法手は棋譜の手), $\boldsymbol{\phi}(t_j)$ は局面 t_j の特徴ベクトル, M は (7) 式を満たす合法手 j の集合である.

$$\mathbf{w}^T \boldsymbol{\phi}(t_1) - \mathbf{w}^T \boldsymbol{\phi}(t_j) < margin \qquad (7)$$

(7) 式は合法手 j に比べて棋譜の手を相対的に悪く評

表 1 KKP+KPP を用いた場合のハッシュサイズと精度の関係

ハッシュサイズ	一致率	不一致度	特徴数比
org	32.8934	6.55760	1.000
2^{26}	32.7896	6.56698	1.381
2^{24}	32.8713	6.61887	0.355
2^{22}	32.5056	6.73625	0.086
2^{20}	31.9001	7.10760	0.022
2^{18}	30.5684	7.94648	0.005
2^{16}	28.5836	8.90990	0.001

価したことを表す．なお, $margin$ は棋譜の手を相対的にどれだけよく評価すべきかを決める閾値で, 激指では終盤になるほど大きな値が設定される．

実験ではプロとアマ混合の棋譜を用い, 学習用に 30,000 棋譜, テスト用に 500 棋譜を用意した．また, 速度向上のために各局面ではすべての合法手を見るのではなく, 棋譜の手以外はランダムに 16 手のみ選び学習を行った．

4.2 評価方法

実験では評価基準として以下の 3 つを用いた．

(i) テスト用の 500 棋譜に対する一致率 (%)
(ii) テスト用の 500 棋譜に対する不一致度
(iii) ノード数制限による対戦結果

ここで, 不一致度は (8) 式のように定義した．

$$\text{不一致度} = \frac{1}{N}\sum_{i=0}^{N}\sum_{j\in M_i} T(\mathbf{w}^T\boldsymbol{\phi}(t_j) - \mathbf{w}^T\boldsymbol{\phi}(t_1)) \quad (8)$$

(8) 式の N は棋譜中の全局面数, M_i は局面 i での棋譜の手以外の合法手の集合, $T(x)$ は (9) 式で定義されるシグモイド関数である．

$$T(x) = \frac{1}{1+\exp(-3x/100)} \quad (9)$$

一致率が大きいほど精度がよいことを表すのに対し, 不一致度は小さいほど精度がよいことを表す．

対戦は双方はじめの 30 手を固定とし, その後は 1 手当たりの探索ノード数を最大 50 万ノードに制限して最善手の探索を行わせた．対戦で用いる 30 手後の初期局面はテスト棋譜から用意し, 各初期局面に対して先手後手を入れ替えて対戦をして勝率を求めた．

4.3 結合度の計算

駒グラフについて, 各エッジの結合度を計算した．具体的には, 駒グラフのエッジを特徴とする線形分類器を考え, それをロジスティック回帰で学習することで各エッジの結合度を導出した．特徴は 2 駒の位置関係, 計 5,143,824 特徴量とし, 学習後に対称なエッジへの値のコピーを行った．学習で用いる訓練データは学習用の 30,000 棋譜から以下の手順によって作成した．

(i) 棋譜中の各局面について, 棋譜の手とそれ以外の合法手の直後の局面のペア $\{t_1, t_j\}$ をランダムに一つずつ抽出する．
(ii) 各ペアについて, それぞれ局面の特徴ベクトル (=エッジ) の差 $\mathbf{x} = \boldsymbol{\phi}(t_1) - \boldsymbol{\phi}(t_j)$ を計算し, 訓練データとする．
(iii) 各訓練データのラベル y について, 元の局面が先手なら $y = +1$ (正例), 後手なら $y = -1$ (負例) とする．

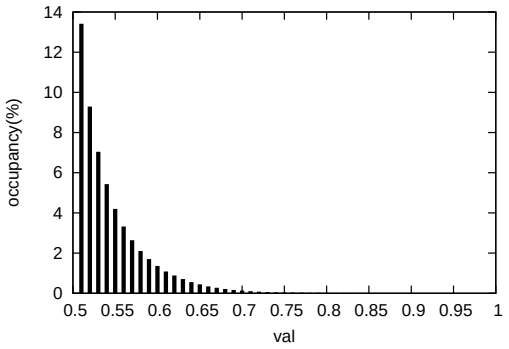

図 3 駒グラフでの結合度の分布

(i) において, 各局面から抽出する手のペアを一つに絞っているのは, 訓練データが大きくなり過ぎることを防ぐためである．30,000 棋譜から得られた訓練データ数は 3,436,819 個であり, データファイルのサイズは 7.5GB となった．

ロジスティック回帰による学習には LIBLINEAR[15] を用い, 具体的には (10) 式の最適化を行った．

$$\min_{\mathbf{w}_e} \frac{1}{2}\mathbf{w}_e^T\mathbf{w}_e + C\sum_{(\mathbf{x}_i,y_i)\in S}\log\left(1+e^{-y_i\mathbf{w}_e^T\mathbf{x}_i}\right) \quad (10)$$

S は訓練データの集合, C は 2 つの項の影響度を制御するパラメタである．本実験では $C = 1$ とした．AMD Opteron 2216 (4 コア, 2.4GHz), メモリ 32GB のマシンを用いて学習をした結果, 165 分程度の時間を要し．全体の 55.9%の特徴に重みがついた．

最終的な各エッジの結合度は, 得られた重み $\mathbf{w}_e$ を用いて (11) 式のように計算した．

$$prob_{e,i} = \frac{1}{1+\exp(-|w_{e,i}|)} \quad (11)$$

(11) 式中の $prob_{e,i}$ 及び $w_{e,i}$ はそれぞれ i 番目のエッジの結合度及び重みを表す．なお, 結合度の範囲は $0.5 \leq prob_{e,i} \leq 1.0$ となる．駒グラフでの $prob_{e,i} = 0.5$ ($i.e.$ $|w_{e,i}| = 0$) を除いた結合度の分布を図 3 に示す．

得られた結合度によって高い重要度のついた 4 駒の関係の例及び各エッジの結合度を図 4 に示す．図 4 中の駒間を結ぶ線は提案手法で展開されたパスの経路を表す．図 4 を見ると, ▲8 五飛, ▲8 四歩, △8 二飛という先手が有利な形となっている．また, △7 三歩は△8 一桂がはねることを妨げる位置にあると見ることができ, 図 4 のパターンは形勢判断に有効な特徴と言える．なお, 後に述べる学習実験ではこのパターンには正の重みが付いた．

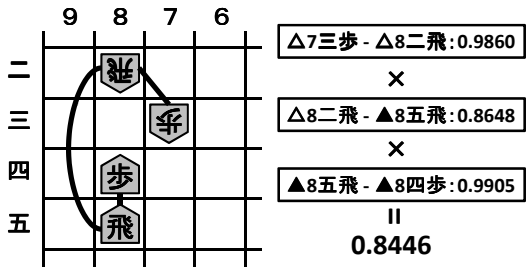

図 4 4 駒の関係の例と各エッジの結合度

4.4 各手法に対する減衰定数の推定

利きグラフ及び 4.3 節で求めた結合度を利用した駒グラフについて, 最適な減衰定数の推定を行った. 駒グラフについては局面評価で出現する特徴数の平均 (平均評価特徴数) を元に, 0.4, 0.45, 0.5, 0.55, 0.6 の 5 種類の閾値を用意した. 各種手法での平均評価特徴数を表 2 に示す. なお, 提案手法では起点の異なる同じパスを重複してカウントするため, 平均評価特徴数が既存手法に比べて多くなっている.

実験では, 減衰定数の推定を高速に行うために, 学習用の 30,000 棋譜及びテスト用の 500 棋譜の各合法手の最善応手手順をあらかじめ激指を用いて展開したデータ (PV データ) を用いた. PV データ作成時の激指の探索深さは 6 とし, 学習及びテストは探索を行う代わりに PV データの最善応手手順後の局面を展開して行った. なお, 学習時は駒割を固定の値とした.

減衰定数と不一致度の関係を図 5 に示す. 図 5 の駒グラフでの結果を見ると, 0.5 の閾値を境にグラフの形状が大きく変わっていることがわかる. これは, 本実験で用いている結合度の範囲が $0.5 \leq prob_{e,i} \leq 1.0$ であるため, 閾値が 0.5 以下になると 2 駒間までのすべての特徴を使うようになり, 特徴が大幅に増えるためである. また, 0.5 以下の閾値では, 閾値が小さくなるにつれて減衰定数に対する不一致度のカーブが急になっていることがわかる. これは, 表 2 を見るとわかるように 0.5 以下では 3 駒間以上の特徴の影響度が大きくなるためであり, 多くの高次組み合わせ特徴を扱う場合に減衰定数を適切に設定しなければ精度が大きく下がることがうかがえる. 同様の傾向は, 高次組み合わせ特徴を多く用いる利きグラフでも見られる.

各手法で最も不一致度が良くなった減衰定数を表 3 に示す. 以降, 減衰定数として表 3 の値を用いる.

4.5 各種手法との比較実験

実際に探索を行いながら学習を行い, 駒グラフでの閾値の選択及び提案手法と既存手法との比較実験を行った. 今回, 比較対象として以下の 3 手法を用いた.

- 2 駒間の関係 (all_two)

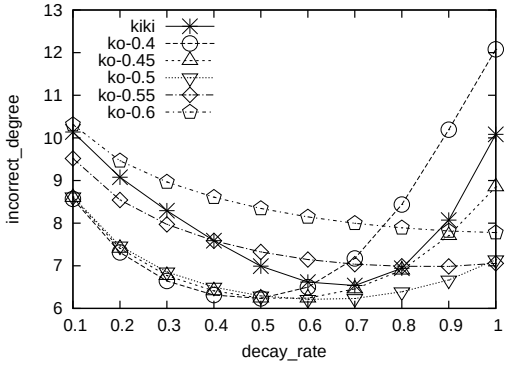

図 5 減衰定数と不一致度の関係

- 1 玉と 2 駒の関係, 2 玉と 1 駒の関係 (kkp_kpp)
- all_two と kkp_kpp の組み合わせ (atkk)

なお, kkp_kpp, atkk では提案手法と同じように減衰定数を設定した[⋆3]. 減衰定数は 4.4 節と同様の方法を用いて推定し, kkp_kpp は 0.9, atkk は 0.8 とした. なお, kkp_kpp, atkk では提案手法と同じ条件でハッシュ関数によるインデックス計算を行った.

まずはじめに, 駒グラフでの各閾値に対して学習を行い, 不一致度を比較することで最適な閾値の選択を行った. 学習過程での不一致度の推移を図 6, 最終的な不一致度を表 4 に示す. 図 6 を見ると, 閾値が 0.5 になるまでは大きく不一致度が向上していることが分かる. これは, 閾値 0.5 までは影響度の大きい 2 駒間の特徴が増加する傾向があるためと考えられ, 2 駒間の組み合わせ特徴の有用性がうかがえる. 閾値 0.5 以下では不一致度の向上がないように見えるが, 表 4 を見ると不一致度が若干向上していることが分かる. しかし, 閾値 0.5 以下で不一致度の向上が鈍いことは, 高

表 2 各種手法での平均評価特徴数 (小数点以下切り捨て)

手法	2 駒間	3 駒間	4 駒間	計
kiki	150	507	1,951	2,608
ko-0.4	1,031	1,669	534	3,234
ko-0.45	1,031	732	141	1,904
ko-0.5	1,031	273	38	1,342
ko-0.55	383	97	11	491
ko-0.6	150	37	3	190
激指の特徴				472
all_two	515			515
kkp_kpp		938		938

表 3 各手法に対する最適な減衰定数

手法	減衰定数
kiki	0.7
ko-0.4	0.5
ko-0.45	0.6

手法	減衰定数
ko-0.5	0.6
ko-0.55	0.9
ko-0.6	1.0

⋆3 kkp_kpp では減衰定数を 3 駒間の特徴量として用いた

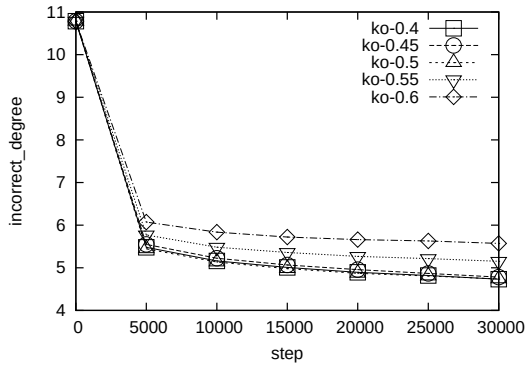

図 6 駒グラフにおける各閾値での不一致度の推移

次の組み合わせ特徴をうまく活用できていない可能性が考えられ, 減衰定数など手法の吟味が必要であるといえる. ko-0.45 だけは精度の悪化が見られるが, これは近似的に減衰定数を粗く推定したことが原因ではないかと推測される. 表 4 より, 駒グラフでは閾値が 0.4 のものが最もよい結果となっているため, 以降の実験では閾値として 0.4 を用いることとした.

次に, 各種手法での比較実験を行った. テスト結果を表 5 に示す. ここで, 表中の特徴数は最終的に得られたパラメタの非零の要素数を表す. 表 5 を見ると, 駒グラフを用いた提案手法が不一致度で最もよい結果を得ていることが分かる. 一方, 利きグラフを用いたものは all_two, atkk, ko-0.4 に比べ精度で劣る結果となっている. これは, 利きグラフでは利きのない駒間の情報が得られないことが影響していると推測される. 実験では kkp_kpp の結果が他の駒関係を用いた手法に比べ悪くなっているが, この原因としては 3 駒間のみを特徴として用いたために, 十分に学習が収束していない可能性が考えられる. 計算速度に注目すると, 提案手法は既存手法に比べ長い学習時間を要していることがわかる. Xeon X5560 (2.8GHz) を用いた場合の具体的な計算速度を表 6 に示す. 提案手法では盤面をグラフに変換するオーバーヘッドが大きく, さらに高次の組み合わせ特徴をチェックするために計算速度が既存手法に比べて遅くなっている. 計算速度の向上は今後の課題の一つといえる.

各種手法との 200 試合での対戦結果を表 7 に示す. 表中の o と x はそれぞれ有意水準 5%の 2 項検定で有意に勝ち越した, 負け越した結果を表す. 表 7 を見ると, 利きグラフは駒グラフに比べ結果が悪く, atkk, ko-0.4 に対しては有意に負け越していることがわかる. このことから, 単純に利きを辿るだけでは十分に形勢判断ができないと推測される. 一方, 駒グラフでは既存手法と比べ有意な差を得られなかった. この原因としては, 提案手法がパス生成時に重要な特徴を十分に選択しきれていない可能性が考えられる.

表 4 駒グラフにおける各閾値での不一致度

ko-0.4	ko-0.45	ko-0.5	ko-0.55	ko-0.6
4.73297	4.78093	4.73848	5.15124	5.57154

4.6 激指の評価関数を用いた実験

利きグラフ及び閾値 0.4 の駒グラフを用いた提案手法について, 激指の評価関数に追加した場合の精度について実験を行った. 実験方法は 4.5 節と同様に行い, 比較対象は 4.5 節で用いた 3 手法に加え, 激指の評価関数をそのまま学習したもの (none) を用意した. 実験では激指のパラメタを初期値として与え, 激指の評価関数も学習対象とした. なお, 各手法の減衰定数は激指の評価関数を追加した状態で実験を行い, kkp_kpp は 0.4, atkk は 0.6, kiki は 0.5, ko-0.4 は 0.3 とした.

テスト結果を表 8, 対戦結果を表 9 に示す. 表 8 を見ると, 表 5 に比べ all_two の結果が相対的に悪くなっていることがわかる. これは, 激指の評価関数の中で 2 駒関係の一部をすでに評価しているためと考えられる. 一方で, 激指で用いられていない 3 駒以上の関係を見る手法は全体的に良い結果を納めている. 特に, 利きグラフでの結果は表 5 に比べて大きな向上が見られる. これは, 空升も含めた従来とは異なる組み合わせ特徴であるためと推測され, 利きグラフと既存評価要素との組み合わせの有用性がうかがえる. テスト結果では駒グラフを用いたものが一致率, 不一致度の両方で最も良い結果を得ることに成功している. 一方, 対戦結果では有意な差を得ることができなかったが, これは探索結果が激指の評価関数部分に大きく影響さ

表 5 各種手法のテスト結果

手法	学習時間	特徴数	一致率	不一致度
all_two	9h 48m	1,585,694	37.8370	4.80734
kkp_kpp	19h 46m	4,194,240	36.8526	5.08549
atkk	27h 35m	4,194,240	**37.9171**	4.74396
kiki	84h 09m	4,194,304	36.4218	5.06288
ko-0.4	95h 44m	4,167,469	37.7035	**4.73297**

表 6 各種手法と計算速度

手法	nodes/sec
all_two	207,856
kkp_kpp	109,376
atkk	76,598

手法	nodes/sec
kiki	33,388
ko-0.4	23,183

表 7 各種手法との対戦結果 (%)

\	at	kp	ak	ki	ko
kiki	45.5	47.5	**x31.0**		**x35.0**
ko-0.4	47.5	56.5	44.0	**o65.0**	

れているためと考えられる.

5. おわりに

本稿では, ゲーム構成要素間の結合度を定義し, それを用いて組み合わせ特徴を絞り込むことで高次の組み合わせ特徴を利用する手法の提案を行った. 実験では, 単純な特徴である駒割, 既存の特徴である激指の評価関数の 2 つに提案手法と従来手法を組み込み学習を行うことで比較を行った. 実験の結果, 対戦結果では十分に有意な結果を得るに至らなかったものの, 提案手法では高次組み合わせ特徴を特徴数の爆発を抑えつつ活用し, 精度面で既存手法と同程度以上の結果を得ることに成功した. 本稿の結果は, 高次の組み合わせ特徴の評価関数への適用の可能性を示唆するものと言え, 今後, 高次の組み合わせ特徴を用いた評価関数の発展が期待される.

今後の課題としては, まず計算速度の向上が挙げられる. 高次の組み合わせ特徴を毎回計算することは計算コスト面で非常に不利であり, 効率的な差分計算の実装や, 高頻度の組み合わせの事前計算等の方法が解決策として考えられる. 次に, 組み合わせ特徴の選択方法の吟味が必要である. 本稿では 2 駒間のみに注目し, そこから高次の組み合わせ特徴を展開する方法を取ったが, 既存研究の多くでは, 用いるすべての組み合わせ特徴を事前に調べ上げることで精度を維持しつつ速度の向上を行っている. ゲームにおいても, すべての組み合わせ特徴を効率的に調べることで性能が向上する可能性があると推測される. 最後に, 盤面をどのようにグラフにするかという問題が挙げられる. 例えば, 駒グラフにおいて升の情報も有効に活用できれば, 評価関数の表現力の向上につながると考えられる.

謝辞 本研究の一部は文部科学省科学研究費補助金特定領域研究「情報爆発に対応する高度にスケーラブルなソフトウェア構成基盤」の助成を得て行われた.

表 8 激指の評価関数を用いた場合の各種手法のテスト結果

手法	学習時間	特徴数	一致率	不一致度
none	8h 41m	350,730	39.1640	3.77321
all_two	12h 59m	1,862,596	40.1511	3.58608
kkp_kpp	23h 51m	4,544,510	41.0785	3.56295
atkk	31h 05m	4,544,459	41.1267	3.51394
kiki	85h 52m	4,544,876	40.9798	3.52035
ko-0.4	94h 43m	4,496,565	**41.2659**	**3.49969**

表 9 激指の評価関数を用いた場合の各種手法との対戦結果 (%)

\	no	at	kp	ak	ki	ko
kiki	54.0	50.0	50.0	54.5		47.0
ko-0.4	56.5	52.5	52.0	45.0	53.0	

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, 2006.
2) 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, 2008.
3) Bonanza - The Computer Shogi Program. http://www.geocities.jp/bonanza_shogi/.
4) Taku Kudo and Yuji Matsumoto. Fast methods for kernel-based text analysis. In *ACL '03*, pp. 24–31. Association for Computational Linguistics, 2003.
5) Naoki Yoshinaga and Masaru Kitsuregawa. Polynomial to linear: efficient classification with conjunctive features. In *EMNLP '09*, pp. 1542–1551. Association for Computational Linguistics, 2009.
6) 金子知適, 田中哲朗, 山口和紀, 川合慧. 駒の関係を利用した将棋の評価関数. 第 8 回ゲームプログラミング ワークショップ, pp. 14–21, 2003.
7) 三輪誠, 横山大作, 近山隆. 駒位置と効き関係に注目した詰み評価関数の自動生成. 第 10 回ゲームプログラミング ワークショップ, pp. 48–55, 2005.
8) David Silver, Richard Sutton, and Martin Müller. Reinforcement learning of local shape in the game of go. In *IJCAI'07*, pp. 1053–1058. Morgan Kaufmann Publishers Inc., 2007.
9) A.L. Zobrist. A new hashing method with application for game playing. Technical Report88, Univ. of Wisconsin, 1970.
10) Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, and S.V.N. Vishwanathan. Hash kernels for structured data. *J. Mach. Learn. Res.*, Vol.10, pp. 2615–2637, 2009.
11) Y.Tsuruoka, D.Yokoyama, and T.Chikayama. Game-tree search algorithm based on realization probability. *ICGA Journal*, Vol.25, No.3, pp. 132–144, 2002.
12) 将棋プログラム「激指」のページ. http://www.logos.ic.i.u-tokyo.ac.jp/gekisashi/.
13) 鶴岡慶雅. 選手権優勝記 -激指の技術的改良の解説-. 情報処理, Vol.51, No.8, pp. 1001–1007, 2010.
14) Michael Collins. Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In *EMNLP '02*, pp. 1–8. Association for Computational Linguistics, 2002.
15) LIBLINEAR – A Library for Large Linear Classification . http://www.csie.ntu.edu.tw/cjlin/liblinear/.