修　　士　　論　　文

ゲーム構成要素を組み合わせた特徴の最適化

Optimizing conjunctive features of game components

指導教員　　近山 隆 教授

東京大学工学系研究科
電気系工学専攻

氏 名　　37-096511 矢野 友貴

提 出 日　　平成 23 年 2 月 9 日

概 要

近年の計算資源の充足に伴い, 基礎的な特徴の組み合わせを用いた識別モデルが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, 組み合わせ特徴が利用される分野の一つであるコンピュータゲームプレイヤを題材に, 基礎特徴間の関連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, 従来よりも高次の組み合わせ特徴を活用する手法を提案する. 将棋を例に既存手法と比較した結果, 機械的に組み合わせを抽出した提案手法によって, 既存手法と同程度の精度を維持しつつ, 総特徴数を削減することに成功した.

目 次

図 目 次

表 目 次

第1章　序論

1.1　背景と目的

与えられた問題に対して, コンピュータになんらかの判断を行わせる場合, その問題を表現するモデルを構築する必要がある. 良いモデルを構築するためには, 注目すべき問題の特徴を適切に選ぶことが重要となるが, この選別は問題に対する深い知識が要求されるため困難である. 問題に対する知識に依らず, 精度のよいモデルを構築することは人工知能の分野において大きな課題の一つである.

このようなモデル構成の問題に対し, 近年, 問題に関する基礎的な特徴を組み合わせることでモデルを構築する手法が提案されている. このような単純なモデルが利用されるようになった背景には, 計算資源の充足や機械学習手法の発展に伴い, 従来では扱うことが難しかった膨大な特徴量を扱えるようになったことが挙げられる [1,2].

組み合わせ特徴は, 問題の基礎的な特徴の組み合わせをすべて数え上げるだけで構成することが可能なため, 対象とする問題に対する熟練した知識なしでも簡単にモデルの設計ができるという利点がある. 組み合わせ特徴はその汎用性から非常に幅広い分野で利用されており, 経験的に高い精度が得られることが示されている [3,4]. 一方で, 組み合わせ特徴には 2 つの問題点がある. 一つは, 特徴を組み合わせる過程で実際には意味のないような組み合わせも多く生成してしまう冗長性である. もう一つは, 組み合わせる要素の個数が増えると特徴の総数が爆発的に増加してしまう点である. このような問題点のため, 特に基礎的な特徴の総数が多い問題では, 組み合わせ特徴は限られた組み合わせでの利用にとどまっている.

組み合わせ特徴が利用される分野の一つとして, コンピュータゲームプレイヤがある. コンピュータゲームプレイヤでは, 局面の優劣を判断する評価関数の構成に駒などのゲーム構成要素の組み合わせ特徴が用いられるため, 得られた特徴の組み合わせの意味が比較的評価しやすい問題である. 本研究では, ゲーム構成要素の種類が多く, 高次の組み合わせ特徴の利用が難しいゲームである将棋を題材に, 組み合わせ特徴を利用した評価関数の新しい構成法の提案を行う.

本研究では, 前述のような組み合わせ特徴の問題に対し, 具体的には以下の 2 つのアプローチを組み合わせた手法を提案する.

1. ゲーム構成要素間の結びつきに着目した組み合わせ特徴の絞り込み
2. ハッシュ関数を用いた特徴数の削減

アプローチ 1 では, 事前に各ゲーム構成要素の関連性を調べ, それに応じて組み合わせる要素を厳選し, 局面評価に有効そうな組み合わせ特徴の選択を行う. これにより, 組み合わせ特徴の冗長性の問題を軽減すると共に, 高次の組み合わせ特徴の展開の効率化を実現する. アプローチ 2 では, ハッシュ

関数を利用して特徴数をハッシュサイズに落とし込むことで組み合わせ爆発の問題を回避しつつ, 高次の組み合わせ特徴と重みベクトルとのインデックスの対応計算の簡略化を図る. 2 つのアプローチを用いることにより, 従来よりも高次の組み合わせ特徴の効率的な活用が可能となる. 高次の組み合わせ特徴は局面をより具体的に特徴付けることができるため, 精度の向上が期待される.

1.2　本研究の位置づけ

本研究は, 組み合わせ特徴を用いた識別モデル, とりわけ将棋のコンピュータゲームプレイヤにおける評価関数を対象としている. 1.1 節でも述べたように, 組み合わせ特徴には冗長性と組み合わせ爆発の 2 つの問題点があり, これは評価関数において高次の組み合わせ特徴の利用の妨げとなっている. 本研究で扱う将棋は, 訓練データが膨大で, また学習方法が特殊であるために, 高次の組み合わせ特徴の利用でよく用いられるカーネル法 [5] や特徴選択 [6] を適用することが難しいゲームである. そのため, 現在は最大でも玉を含む 3 駒間の組み合わせ特徴の利用にとどまっている. 本研究では, 将棋における高次組み合わせ特徴が利用しづらいという問題点を, 要素間の関連性に着目することで解決させることを目的としている. 本研究の内容は将棋のコンピュータゲームプレイヤを対象としたものであるが, 提案手法の概念は組み合わせ特徴を利用する一般的な問題にも適用可能である.

1.3　本研究の貢献

本研究の貢献としては, 提案手法によって冗長性の緩和, 組み合わせ爆発の回避を実現したこと, 将棋の評価関数に対するハッシュ利用の有効性を示したこと, 将棋における高次組み合わせ特徴の有効性を評価したことが挙げられる. 本研究は, 将棋における高次組み合わせ特徴の利用の可能性を広げるものであり, 今後のコンピュータゲームプレイヤの発展に貢献したといえる. また, カーネル法や従来の特徴選択とは異なるアプローチを提案したことは, これらの手法が適用しにくい組み合わせ特徴の問題への解決の糸口を与えたといえる.

1.4　本論文の構成

本論文では以降, 2 章において提案手法で用いる要素技術に関する研究, また組み合わせ特徴の最適化に関する先行研究について述べたのち, 3 章にて提案手法を, 4 章にて各種評価実験とその結果について述べ, 最後に 5 章にて本論文のまとめと今後の課題について述べる.

第2章　関連研究

2.1　評価関数と組み合わせ特徴

評価関数とは, ゲーム中の各局面について有利不利の度合いを数値として計算する関数である. 評価関数は多くの場合, 局面 s から抽出される特徴ベクトル $\boldsymbol{\phi}(s)$ と各特徴の重要度を表す重みベクトル $\boldsymbol{w}$ を用いて (2.1) 式のように構成される.

$$f(\boldsymbol{w}, s) = \boldsymbol{w} \cdot \boldsymbol{\phi}(s) \tag{2.1}$$

精度の高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探し出す必要があるが, 近年では組み合わせ特徴を評価要素として用いることが広く行われるようになっている. 組み合わせ特徴は, 基礎的な要素を複数組み合わせた非常に単純な特徴であるが, 将棋では図 2.1 のように駒の種類, 位置の情報の組み合わせを用いることが一般的である. 以降, 本稿で対象とするゲームである将棋の組み合わせ特徴に関する研究について述べる.

2.1.1　駒の 2 項関係による評価関数

将棋の評価関数において駒による組み合わせ特徴を初めて利用した事例として, 金子等の研究が挙げられる [4]. 金子等は駒の 2 項関係を評価するモデル, すなわち図 2.1 のような 2 駒間の組み合わせ特徴を利用する評価関数の提案を行っている. $R_{p,q}$ を駒 p, q の 2 項関係, $value(R_{p,q})$ を 2 項関係 $R_{p,q}$ の価値と定義したとき, 以下のように既存の評価要素の多くが駒の 2 項関係によって表現可能であることを示している.

- 駒の価値 $= value(R_{p,p})$
- 玉への迫り方 $= value(R_{p,q})$　where $(p = \text{玉}) \vee (q = \text{玉})$
- 敵陣の飛車 $= value(R_{p,p})$　where $(p = \text{飛車}) \wedge (p_{\text{位置}} = \text{敵陣})$

金子等はさらに, 評価関数のパラメタを調整する方法として親子モデル, 兄弟モデルの 2 つについて比較を行っている. 親子モデルとは, 棋譜中の指し手前後の局面の評価値を比較し, 指し手の後の局面が前の局面よりも指し手の手番にとって有利と判断されるように評価関数を調整するモデルである. また, 兄弟モデルとは, 各局面のすべての合法手に対し, 棋譜の手の後の局面が他の合法手の後の局面よりも有利と判断されるように評価関数を調整するモデルである. 金子等は 2 つのモデルによって調整されたパラメタを用いて, 定跡の評価, 悪形の回避の 2 点について比較を行い, 兄弟モデ

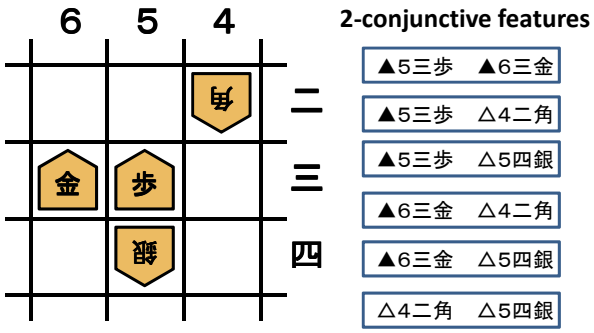


図 2.1: 2 要素間の組み合わせ特徴の例

ルが親子モデルを精度で上回る結果を得たと述べている．また, 全合法手中での棋譜の指し手の評価順位を示した表 2.1 のように, 提案した評価関数を利用することで人間の手をうまく模倣する結果を得ることに成功しており, 組み合わせ特徴の有効性を示す結果となっている．

一方, 金子等は 2 駒間だけでは表現できない特徴についても言及している．例えば, 大駒などの遠くまで届く利きは, 利きを遮る駒が存在する場合に 2 駒間の位置関係だけではその情報を抽出することができない．金子等は, 将来的に 3 駒間を用いることで精度の向上が期待されるが, 3 駒間の組み合わせ特徴をすべて用いることは組み合わせ爆発のために提案が行われた時点では難しいと述べている．

2.1.2　詰み評価関数の自動生成

非常に高次の組み合わせ特徴を利用したものとして, 三輪等による詰み評価関数の自動生成に関する研究がある [7]．三輪等は, 詰将棋において局面の詰み・不詰を判断する詰み評価関数を, 組み合わ

表 2.1: 一手読みによる指し手の順位付け [T. Kaneko, et al., 2003]

手数	候補手の数		順位	
	合法手	候補手	親子	兄弟
0-20	35.0	30.5	12.2	1.7
20-40	44.6	27.2	8.3	2.3
40-60	68.9	17.9	5.7	2.0
60-80	109.3	12.9	4.1	1.8
80-100	133.2	10.3	3.3	1.7
100-	161.3	9.1	2.9	1.9
全体	75.7	21.1	7.3	1.9

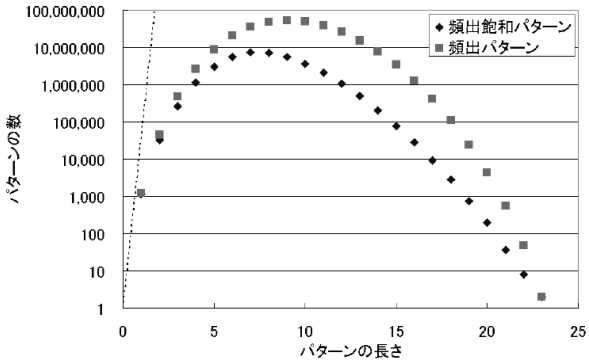


図 2.2: 頻出飽和パターンの長さごとの分布 [M. Miwa, et al., 2005]

せ特徴を利用して設計している. 三輪等の研究では, 組み合わせ特徴の要素として以下に示す 2 つを用いている.

- 駒の位置 (ex. 盤面上の駒:[1 一, 香, 後手], 持ち駒:[歩 1, 歩, 先手])
- 利き (ex. [1 一, 後手] → [1 二, 空])

評価要素の総数は 41,224 個となり, これらの全組み合わせ特徴を用いることは空間コストの面から非現実的である. そこで, 三輪等の研究では, 棋譜中の頻出飽和パターンを抽出することで, 組み合わせ特徴の絞り込みを行っている. また, 三輪等は得られた頻出飽和パターンに対して, 実際に詰み評価において有効であるかどうかを, ベイズ学習に基づいた確率の推定値, カイ二乗値の 2 つの基準を用いて判別し, 組み合わせ特徴の選別を行っている.

最小サポートを 2%としたときの頻出飽和パターンの長さの分布は図 2.2 のようになったと報告されている. 図 2.2 を見ると, 最大で 23 要素間というとても長い組み合わせ特徴が抽出されていることが分かる. 三輪等は, 得られた組み合わせ特徴を前述の 2 つの指標によりさらに選別することで評価関数を構成し, そのパラメタをパーセプトロンにより学習した結果, 最も良いもので 76.62%の正解率を得たと述べている. この数値は, 人手によって評価要素を抽出し学習したものに比べ 5%ほど低い結果であったものの, 組み合わせ特徴を利用することで精度の高い詰み評価関数の設計に成功したといえる.

2.1.3　将棋プログラム「ボナンザ」と比較学習

組み合わせ特徴を利用した将棋プログラムの代表例として保木によって作成された「ボナンザ」がある [8]. ボナンザは第 16 回世界コンピュータ将棋選手権で優勝を収め, また 2010 年には合議を利用した将棋プログラム「あから 2010」のメンバーとして利用されるなど, トップレベルの強さを誇る将棋プログラムである. ボナンザでは評価関数の評価要素として 2 玉と 1 駒の位置関係 (King King

Piece, KKP), 1 玉と 2 駒の位置関係 (King Piece Piece, KPP) の 2 つを用いており, その総特徴数は約 1 億個と非常に膨大な数となっている. 保木はこのような膨大なパラメタ群を適切に調整させる手法の提案を行っている [9].

ゲームの評価関数のパラメタを調整する場合, 一般には棋譜を訓練データとして用いて損失関数を設計し, 勾配法等によって損失関数を最小化するようにパラメタの調整を行う. オセロなどのような最終的にスコアが確定するゲームの場合, 棋譜中の各局面における最終的なスコアと評価関数が推定したスコアの差が小さくなるように学習を行えば良いため, (2.2) 式のような二乗誤差が損失関数として用いられる [10]. ここで, S は訓練データの集合, s は訓練データの局面, y は訓練データのラベル (ここでは盤面の最終的なスコア) を表す.

$$L(\boldsymbol{w}, S) = \sum_{(s,y)\in S} (y - f(\boldsymbol{w}, s))^2 \tag{2.2}$$

チェスや将棋の場合, オセロ等と違い最終的な勝敗はスコアとして得られず, また勝敗がいつ決まるのかがわからないため, (2.2) 式のようなアプローチではラベル y を適切に設定することが困難となる. そこで, チェスや将棋では評価関数の値をゲームのスコアなどの絶対的な基準で調整するのではなく, 棋譜中の人間の指し手の評価値が相対的に高くなるような評価関数の調整 (比較学習) が行われる.

将棋において, 比較学習は 2.1.1 節で述べた兄弟モデルのようにボナンザの登場以前から提案はされていたが, 保木は比較学習の際に局面の探索を導入することで, 学習の精度を飛躍的に向上させることに成功している. このような探索を用いる比較学習は, チェスのコンピュータゲームプレイヤである Deep Blue の評価関数の調整にも利用されている [11]. コンピュータ将棋の分野では, 保木によって提案された比較学習手法はボナンザメソッドという名前で知られており, 現在の多くの将棋プログラムで利用されている.

ボナンザメソッドにおけるパラメタ調整の具体的な手順は図 2.3 及び以下の通りである.

1. 棋譜中の局面 s_{rt} から棋譜の手の後の局面 s_1 を展開し, 探索によって末端局面 s'_1 を得る
2. 棋譜の手以外の手に対しても同様に局面 s_k を展開し, 探索によって末端局面 s'_k を得る
3. 得られた末端局面に対し, 次の (2.3) 式を満たすようにパラメタを調整する

$$\forall k(k \neq 1), f(\boldsymbol{w}, s'_1) > f(\boldsymbol{w}, s'_k) \tag{2.3}$$

1 手先の局面 s_k ではなく, 探索した後の局面 s'_k をパラメタ調整で用いるのは, 駒の取り合いが起きているような評価値の変動の大きい局面を学習に用いることを避けるためである. (2.3) 式を満たすようにパラメタを調整することによって, 探索の際に棋譜の手が選ばれるような評価関数を得ることができる. ボナンザメソッドでの具体的な損失関数は (2.4) 式のような形となる. (2.4) 式中の y は (2.5) 式のように s_{rt} の手番によって決まるラベルである. ラベルを手番に応じて ± 1 にするのは, 評価関数が先手有利ならば大きな値を, 後手有利ならば小さい値を返すために, 手番によって評価値の

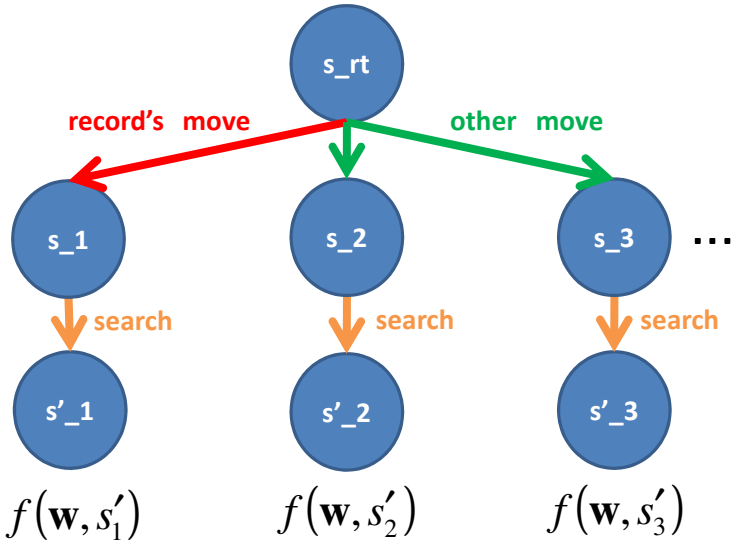


図 2.3: ボナンザメソッドにおける局面の展開

差の符号が逆転するようにしたためである.

$$L(\boldsymbol{w}, S) = \sum_{(s_{rt}, y) \in S} \sum_{k=2}^{n} T\left(y\left(f_1 - f_k\right)\right) \quad \text{where } f_k = f(\boldsymbol{w}, s'_k) \tag{2.4}$$

$$y = \begin{cases} +1 & (\text{turn}(s_{rt}) = \text{先手}) \\ -1 & (\text{turn}(s_{rt}) = \text{後手}) \end{cases} \tag{2.5}$$

(2.4) 式中の $T(z)$ は評価値の差を棋譜の手との一致度に変換する関数であり, z が小さいほど損失が大きくなる関数が用いられる. 具体的な $T(z)$ の形式には (2.6) 式のようなものがあり, シグモイド関数は [9] で, ロジスティック回帰モデルは [12] で, ヒンジ損失関数は [13] でそれぞれ用いられている.

$$T(z) = \begin{cases} 1/\left(1 + \exp(az)\right) & (\text{シグモイド関数}) \\ \log(1 + \exp(-z)) & (\text{ロジスティック回帰モデル}) \\ \max(0, m - z) & (\text{ヒンジ損失関数}) \end{cases} \tag{2.6}$$

2.2　ハッシュ関数と次元圧縮

配置パターン, 文字列, 部分グラフといった要素の組み合わせを用いた特徴は, 構成する要素数が増えると次元数が膨大なものとなる. 例えば配置パターンの場合, 各配置での状態が m 通り, 配置する位置が n 通りあれば, その次元数は m^n と n が増えるに従い指数関数的に増大する. このため, これら膨大な次元数を持つ特徴群をそのままメモリ上に展開することは現実的でない. このような問題に対し, ハッシュ関数を用いることで次元圧縮を行う手法が提案されている.

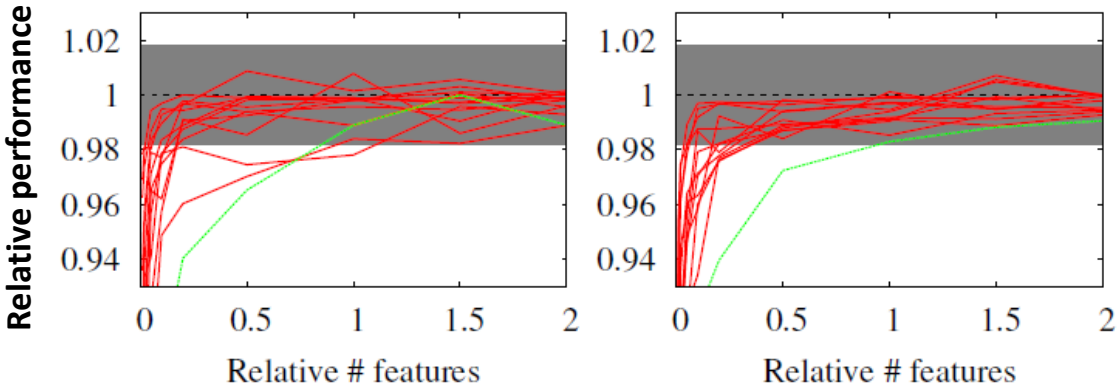


図 2.4: ハッシュサイズと相対的な性能 (左:SVM (Support Vector Machine), 右:MIRA (Margin Infused Relaxed Algorithms)) [K. Ganchev, et al., 2008]

2.2.1　Random Feature Mixing

Ganchev 等は自然言語処理で用いられる判別関数に対して, それを構成する特徴群をハッシュ関数によって圧縮する手法の提案を行った [14]. 自然言語処理では文章中から得られる非常に多くの特徴を判別関数の評価要素として用いるため, 特徴の展開のために膨大なメモリが要求される. 例えば, 判別関数の評価要素として文章中に出現する各単語を用いた場合, 特徴の次元数は単語数に比例して大きくなり, 複数の単語を組み合わせれば次元数は指数的に増加する. また, 各文字列と判別関数上のインデックスとの対応表も保持する必要がある. Ganchev 等は, この空間コストの問題が特にモバイル端末などリソースの制約が大きい環境で問題となると述べている.

Gauchev 等は, ハッシュ関数を用いる利点として以下の 2 点を挙げている.

1. 特徴数をあらかじめ決められたサイズに固定でき, 空間コストを削減できる
2. ハッシュ関数を利用することで, 文字列からインデックスへの変換が容易になる

一方で, ハッシュ関数を用いることによって特徴のインデックスの衝突が起こるために性能の低下が懸念されるが, 以降で述べる Gauchev 等の実験では, 特徴数を大幅に削減しても性能はそれほど劣化しないことが示されている.

Gauchev 等は実際に自然言語処理における 4 つのデータセットに対してハッシュ関数を用いた場合の精度について実験を行っている. 学習アルゴリズムとして SVM (Support Vector Machine), MIRA (Margin Infused Relaxed Algorithms) [15] を用い, すべてのデータセットに対してハッシュを用いなかった場合との相対的な性能の比較を行った結果は図 2.4 のように示されている. 図 2.4 の縦軸はハッシュを用いなかった場合 (オリジナル) に対する相対的な精度を表し, グレー部分はオリジナルでの精度の標準偏差を表す. また, グラフ中の赤, 緑の線は各データセットでの結果を表す. 図 2.4 より, 特徴数を 10 分の 1 程度まで圧縮してもオリジナルの 96.7%-97.4%程度の性能を得ることに成功

していることが分かる. 一方で, 図 2.4 の緑線で示されるように性能が大きく落ちたデータセットがあるが, これについて Gauchev 等は, ある特定の特徴が性能に大きく寄与していた可能性を指摘している.

2.2.2　ハッシュカーネル

Shi 等はハッシュ関数による写像を利用したカーネル関数の提案を行った [16]. カーネル関数とは, インスタンスの特徴を別の特徴空間に写像し, そこでの内積を計算する関数である. ハッシュカーネルではこの写像としてハッシュ関数を用い, インスタンスの特徴を低次元に圧縮し, そこでの内積を計算する. 元の特徴空間を $\mathcal{J}$, ハッシュ関数を $h : \mathcal{J} \to \{1, \ldots, n\}$ とすると, 2 つのインスタンス $\boldsymbol{x}$, $\boldsymbol{x}'$ での Hash Kernel の具体的な計算は (2.7) 式のように行われる.

$$\bar{k}(\boldsymbol{x}, \boldsymbol{x}') = \langle \bar{\boldsymbol{\phi}}(\boldsymbol{x}), \bar{\boldsymbol{\phi}}(\boldsymbol{x}') \rangle \quad \text{with} \quad \bar{\boldsymbol{\phi}}_j(\boldsymbol{x}) = \sum_{i \in \mathcal{J}; h(i)=j} \boldsymbol{\phi}_j(\boldsymbol{x}) \tag{2.7}$$

ハッシュカーネルではハッシュ関数を用いることにより,

1. データの型 (数値, 文字列, 構造データなど) にとらわれない高い汎用性
2. 次元圧縮による密な特徴行列の生成
3. 逐次データ処理への適用

を実現している. 2. に示した密な特徴行列の生成は, CPU や GPU 上での効率的な計算を考慮した特性である. 2.2.1 節でも述べたように, ハッシュを用いた場合は衝突による性能劣化が懸念されるが, Shi 等はデータセットとして RCV1 (Reuters Corpus Volume 1) [1]を用いた文書分類問題の実験において, 表 2.2 のように衝突率が 40%程度あっても精度が落ちないことを示している. 3. に示した特性は, データの集合から各データが順次得られるような場合に, それらデータを一つ一つ逐次に処理で

表 2.2: ハッシュの衝突率とエラー率 [Q. Shi, et al., 2009]

Dim	#Unique	Collision %	Error %
2^{24}	285614	0.82	5.586
2^{22}	278238	3.38	5.655
2^{20}	251910	12.52	5.594
2^{18}	174776	39.31	5.655
2^{16}	64758	77.51	5.763
2^{14}	16383	94.31	6.096

[1] http://trec.nist.gov/data/reuters/reuters.html

きることを意味する. 例えば, MCMC (Markov-Chain Monte Carlo 法) 等のサンプリングを用いて, 巨大なデータからサンプルを抽出し, それらをハッシュ関数によりまとめることで, データを保存するコストをなくしつつ, 得られたハッシュ値からデータ群の統計情報を得ることが可能となる. Shi 等は, グラフ分類問題において, 部分グラフを MCMC でサンプリングし, そのデータを用いてハッシュカーネルを計算する手法を提案しており, 実験の結果, 既存のグラフカーネル手法に比べて高い精度を得ることに成功している.

2.2.3　碁石パターンへのハッシュ利用

コンピュータゲームプレイヤの評価関数にハッシュ関数を利用した例として, Silver 等の囲碁プレイヤに関する研究がある [17]. Silver 等は囲碁の評価関数の特徴として 5×5 までの盤面の部分局面上の碁石の配置パターンを利用し, それを強化学習を用いて調整を行っている. 5×5 までの配置パターンを特徴として利用する場合, その特徴数は膨大となり, 例えば 5×5 の配置パターンの場合, 総特徴数は $3^{25} \approx 8.47 \times 10^{11}$ とメモリにのせることができないサイズとなる. この問題に対して Silver 等は表 2.3 のように 4×3 以上のパターンの特徴数をハッシュサイズ h に圧縮し, 重みを共有することで解決を図っている. なお, 表 2.3 の n_i は 5×5 内で各パターンが配置可能な位置の総数を, N_i は特徴数を表し, Location independent は碁石の絶対位置に依存しない場合, Local dependent は依存する場合を表す.

Shi 等は $h = 100,000$ として学習した各パターンの有効性の評価を行っており, 2×2 の配置パターンが最も効果的であるという結果を得ている. 他のパターンについては, 2×2 よりも小さいパターンでは表現力が足りずに効果が薄く, また大きいパターンでは特徴数が多く, パターンが細かすぎるために学習時間内に十分に学習することができなかったと述べている.

表 2.3: 配置パターンと特徴数 [D. Silver, et al., 2007]

Template size	Location independent		Location dependent	
	n_i	N_i	n_i	N_i
1×1	25	3	25	27
2×1	40	9	40	54
2×2	16	81	16	324
3×2	24	729	24	2916
3×3	9	19683	9	78732
4×3	12	h	12	h
4×4	4	h	4	h
5×4	4	h	4	h
5×5	1	h	1	h

2.3　有効パターンの選別

組み合わせ特徴では考えられる要素間の組み合わせをすべて展開するため, 本来は意味のないような特徴も多数生成する冗長性の問題がある. そのため, 組み合わせ特徴を用いた場合, 無駄な評価を行うことによる速度低下が生じる. 前述の通り, 組み合わせ特徴では組み合わせ爆発により膨大な特徴が生成されるため, この計算コストの問題は実用面で大きな課題となる. 以降では, 有効パターンを選別することにより, 計算コストの削減や精度の向上を図った研究について述べる.

2.3.1　特徴選択

与えられた特徴空間において, 実際に有効な特徴群を選ぶ手法は特徴選択として広く研究されている [6]. 特徴選択の手法は対象とする分類器を基準として用いるか否かで 2 つに大別される.

分類器を特徴選択の基準として用いる手法は wrapper method と embedded method の 2 つがある. 前者は分類器をブラックボックスとして扱うのに対し, 後者は分類器の学習に特徴選択を組み込む方法で, 現在は後者の方が一般的によく用いられる. 学習に特徴選択を組み込む方法としては, 特徴の追加, 削除による損失関数の値や勾配の変化に応じて特徴選択を行う方法, 損失関数に特徴数に応じたペナルティを直接加える方法がある. 前者の例としては Grafting [18], 後者の例としては L1 正則化 [2] があげられる.

分類器を特徴選択に用いない手法は filter method と呼ばれる. filter method では, 分類器の学習の前処理として特徴選択が行われる. 分類器を用いずに特徴選択を行う方法としては, 訓練データから得られる統計情報を特徴選択の指標とする方法がある. 例えばカイ二乗値を指標として用いる場合, 2 値分類問題におけるクラス c と特徴 x の出現数の関係を表 2.4 のように定義すると, 特徴 x のカイ二乗値 $\chi^2(x)$ は (2.8) 式のように計算される. カイ二乗値はクラス c と特徴 x の独立性を計る指標であり, 特徴選択ではクラスとの独立性が低い特徴, つまりカイ二乗値が高い特徴を選択すればよい. 統計情報の利用方法としては他にも様々な指標が提案されている [19].

$$\chi^2(x) = \mathrm{chi}(O_x, O_{xc}) = \sum_{i\in\{x,\bar{x}\},j\in\{c,\bar{c}\}} \frac{(O_{ij} - E_{ij})^2}{E_{ij}} \tag{2.8}$$

$$\text{where } E_{ij} = N \cdot \frac{O_i}{N} \cdot \frac{O_j}{N}$$

表 2.4: 2 値分類問題におけるクラス c と特徴 x の出現数の定義

	c	$\bar{c}$	$\sum$ row
x	O_{xc}	$O_{x\bar{c}}$	O_x
$\bar{x}$	$O_{\bar{x}c}$	$O_{\bar{x}\bar{c}}$	$O_{\bar{x}}$
$\sum$ column	O_c	$O_{\bar{c}}$	N

統計情報を利用する以外には, filter method の前処理として embedded method を利用する方法がある. この方法では, 対象とするモデルよりも簡単なモデルにおける特徴選択を filter として用いる. 例えば [20] では, あらかじめ L1 正則化によって線形モデルでの特徴セットを選別し, その結果を利用して非線形モデルを構築する手法が提案されている.

2.3.2　多項式カーネルの高速化

自然言語処理などの分野では, 組み合わせ特徴は一般的に多項式カーネル [5] を介して利用される. 多項式カーネルとは, 2 つの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 $\boldsymbol{x}$, 組み合わせの最大数を d としたとき, 多項式カーネルは (2.9) 式のように計算される.

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = (\boldsymbol{x}_1 \cdot \boldsymbol{x}_2 + 1)^d \tag{2.9}$$

多項式カーネルは非常に汎用性の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る. 速度がデータ数に依存する理由は, 多項式カーネルが一般的に (2.10) 式のように SVM と組み合わせて用いられるため, 計算速度が SVM におけるサポートベクター (Support Vector; SV) の数に依存するためである.

$$y(\boldsymbol{x}) = \mathrm{sgn}\left(\sum_{j \in SV} y_j \alpha_j k(\boldsymbol{x}_j, \boldsymbol{x}) + b\right) \tag{2.10}$$

工藤等は, 多項式カーネルの速度問題に対し, 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 [3]. 工藤等の手法では, 多項式カーネルを線形和に展開することで, 計算コストをサポートベクター数と独立にする. 線形和に展開した場合, 組み合わせ特徴の重みをすべて計算する必要があるため, 高次の組み合わせ特徴を扱うことが困難となるが, 工藤等はこの問題に対し, PrefixSpan [21] を利用した組み合わせ特徴の選別を行っている. PrefixSpan とは, 要素のパターンの集合から頻出のサブパターンを列挙する手法で, 各パターンを prefix と postfix の 2 つに分解し, 短い prefix から徐々に長い prefix へと再帰的にパターンを延長することで深さ優先探索により頻出パターンを見つけ出す. 工藤等は, PrefixSpan を次のように拡張している.

1. パターンの長さに上限値を設ける. 具体的には, 多項式カーネルの次数 d
2. PrefixSpan の頻度として, 正例・負例の頻度から計算される重みを利用
3. 正例・負例で別々にパターンを展開
4. PrefixSpan の最小サポートとして重みの閾値を利用

このような拡張により, 組み合わせ特徴のうち重みが閾値以下のパターン, つまり, 分類への寄与の小さいパターンを排除することができ, 精度を低下を抑えつつ, 総パターン数の削減により速度の向

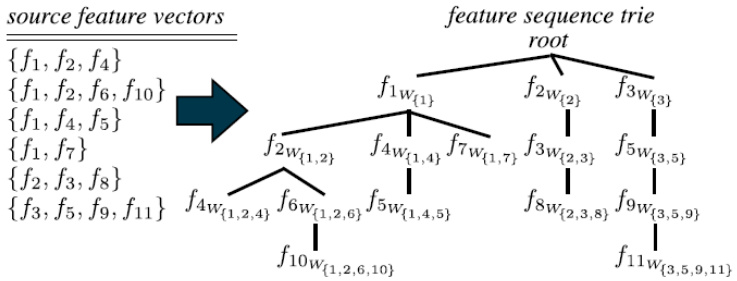


図 2.5: feature sequence trie [N. Yoshinaga, et al, 2009]

上が実現される. 工藤等は提案手法を用いることにより, 従来手法に比べほぼ同程度の精度を維持しつつ, 全体で 30-300 倍程度の高速化に成功したと述べている.

パターンの削減をするのではなく, 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が吉永等によって提案されている [22]. 吉永等の手法では, 線形和に展開された多項式カーネルに対し, (2.11) 式のように事前に特徴ベクトル $\boldsymbol{x}$ の一部 $\boldsymbol{x}_c$ について重みの総和を計算することで高速化を行う. なお, (2.11) 式中の $\boldsymbol{x}$ は非ゼロの 2 値特徴の集合 $\{f_i | f_i(\boldsymbol{x}) = 1\}$ を表す.

$$g(\boldsymbol{x}) = \sum_{f_i \in \boldsymbol{x}^d} w_i = W_{\boldsymbol{x}_c} + \sum_{f_i \in \boldsymbol{x}^d - \boldsymbol{x}_c^d} w_i \tag{2.11}$$

(2.11) 式を用いた場合, 各 $\boldsymbol{x}$ での最適な $\boldsymbol{x}_c$ をいかに効率よく探索するか, 事前に計算する $\boldsymbol{x}$ の集合をどのように選別するか, の 2 点が問題となる. 吉永等はこれらの問題に対し, パターンを効率的に格納するデータ構造として feature sequence trie (fstrie) を提案している. fstrie の具体的な構築手順は以下の通りである. また, fstrie の例を図 2.5 に示す.

1. 訓練データ中に出現するパターン (source feature vectors) を列挙
2. 各要素 f_i の訓練データ中の出現頻度によって各パターンの要素を降順にソート
3. 要素のインデックスを基準にトライ木を作成し, 各 prefix (パターン) の重みを保存

fstrie を用いることで, 最適な $\boldsymbol{x}_c$ を $O(|\boldsymbol{x}_c|)$ で得ることが可能となる. fstrie を用いてすべての出現するパターンを展開すると, そのサイズは非常に大きくなるため, 吉永等はさらに下記の 2 つの指標を組み合わせて fstrie の枝刈りを行っている.

- データ中の各パターンの出現頻度
- 各パターンを用いることによる計算コストの削減量

吉永等の手法は (2.11) 式中の $\boldsymbol{x}^d - \boldsymbol{x}_c^d$, つまり相対的に頻度の小さい特徴の集合の数え上げが計算速度に大きく影響する. そのため, L1 正則化などのスパースな解が得られる手法との相性が良い. 実験では, 重みの選別を行う工藤等の手法 [3] や L1 正則化を用いた対数線形モデルと組み合わせた場合に最大で 10 倍程度の高速化に成功している.

2.3.3　木カーネルにおける部分木の選別

木カーネルとは, 木構造のデータを入力とするカーネル関数のことである. 木カーネルでは, 2 つのデータ中の共通部分木を数え上げることで 2 つのデータの類似度を計算する. 木 T での i 番目の部分木 t_i の出現回数を $h_i(T)$ とした時, 木カーネルは (2.12) 式のように計算される. (2.12) 式中の $d(t_i)$ は部分木 t_i の深さ, λ は減衰定数を表す. 減衰定数を導入するのは, 部分木の総数が増えることによる値の発散を防ぐためである.

$$k(T_1, T_2) = \sum_i \lambda^{d(t_i)} h_i(T_1) h_i(T_2) \tag{2.12}$$

木カーネルの計算は, 対象とする木のノードを再帰的に展開することで $O(|N_1||N_2|)$ で計算可能であることが示されている [23]. なお, $|N_1|$, $|N_2|$ はそれぞれ木 T_1, T_2 のノードの総数を表す. 木カーネルを効率的に計算するためには, いかに無駄な部分木を展開しないかが重要となる.

鈴木等は, データ中の部分木の統計情報を利用することで部分木の選別を行う手法の提案を行った [24]. 鈴木等の手法では, 選別の基準としてカイ二乗値を利用し, PrefixSpan [21] と組み合わせることで有効な部分木の選別を行っている. カイ二乗値を用いる場合に問題となるのは, カイ二乗値がパターンの包含関係と相関がない点である. 例えば, 出現頻度を基準とした場合, 2 つのパターン P, Q が $P \subseteq Q$ という包含関係にあれば, 2 つの頻度には $freq(P) \geq freq(Q)$ という関係が成り立つが, カイ二乗値ではこれが必ずしも成立しない. この問題に対し鈴木等は, AprioriSMP [25] で提案されている包含関係にあるパターンでのカイ二乗値の上限を利用して枝刈りを実現している. 包含関係にあるパターン $u \in uw$ に対して, カイ二乗値 $\chi^2(uw)$ の上限 $\hat{\chi}^2(u)$ は (2.13) 式のように定義される.

$$\chi^2(uw) \leq \hat{\chi}^2(u) = \max\left(\mathrm{chi}(O_{uc}, O_{uc}), \mathrm{chi}(O_u - O_{uc}, 0)\right) \tag{2.13}$$

テキスト分類問題を用いた鈴木等の実験では, 速度面での比較結果については言及されていないものの, 提案手法により, 従来手法では過学習を起こすような大きな部分木を利用した場合でも安定した精度が得られることが示されている.

Rieck 等は, 部分木の根のラベルを制限することで木カーネルを近似する手法の提案を行った [26]. Rieck 等は, 木構造のデータの多くが冗長な構造を持つ点に着目し, 冗長な部分の評価を省くことで高速化, 及びメモリ消費量の削減を図っている. 例えば, 木構造で表現された HTML データからスパムサイトを分類するような場合, header タグや meta タグはスパムを分類する指標として有効であるが, 一方で多くの formatting タグはスパムとの関連が薄く, 時として分類精度を悪化させる原因ともなりうる. Rieck 等の手法では, 木の各ノードのラベル (HTML の例ならタグの種類) の選択問題を訓練データを用いた最適化問題として定式化し, それを事前に解くことによって計算すべきラベルの集合を得る. Web サイトのスパム分類問題を用いた比較実験では, 従来手法に比べ高速かつ小メモリで同等の精度を得ることに成功している. 従来手法との速度, メモリ消費量の差は特に木のサイズが大きい場合に顕著で, 実行時間, メモリ消費量ともに最大で 1000 分の 1, 平均でも 100 分の 1 に改善できたと報告されている.

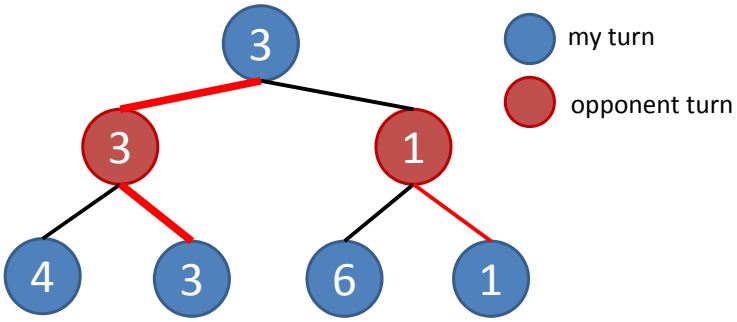


図 2.6: ミニマックス法

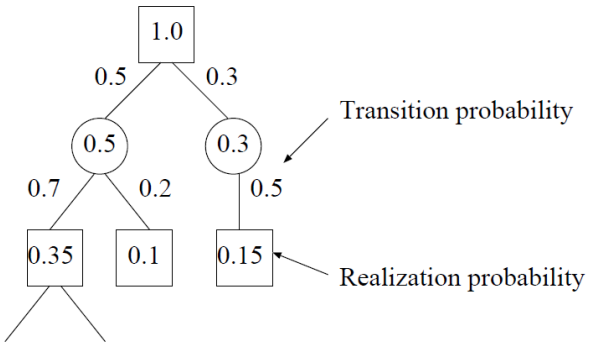


図 2.7: 実現確率探索 [Y. Tsuruoka, et al., 2002]

2.3.4　実現確率探索

コンピュータゲームプレイヤの重要な要素の一つとして, ゲーム木探索がある. ゲーム木探索は, 人間における「先読み」に相当する概念であり, 限られた時間でどれだけ深く, 効率的にゲーム木を探索するかが強さの鍵を握る. ゲーム木探索の最も基本的な手法はミニマックス法と呼ばれ, 図 2.6 のように自分の手番では自分にとって最も有利な手を, 相手の手番では逆に自分にとって最も不利な手を選択することで最適な手の探索を行う. ゲーム木探索にはミニマックス法を拡張した様々な手法が存在するが, その一つに鶴岡等によって提案された実現確率探索がある [27].

一般的なゲーム木探索では, 探索時の深さを探索の閾値として用いる. これは, 各指し手に対して深さ 1 のコストを設定することに等しく, すべての合法手を等価に扱うことを意味する. しかし, ゲーム中の合法手には人間から見て必然の一手や明らかな悪手といった指し手が混在しており, すべての合法手はそれぞれ異なる価値を持っていると考えられる. 実現確率探索では, これら各指し手の重要度の差異に注目し, より重要度の高い指し手をより深く探索することでゲーム木探索の効率化を行っている. 具体的には, 各指し手に対して遷移確率を定義し, それを用いて応手手順後の局面の重要度

(実現確率) を計算し, 得られた実現確率を閾値として探索を行う. 局面 x の実現確率 P_x は (2.14) 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. 具体的な動作例は図 2.7 の通りである.

$$P_x = P_m \cdot P_{x'} \tag{2.14}$$

各指し手の遷移確率は, 指し手の種類や移動後のマス周辺の情報など局面から得られる情報を利用して計算される. 例えば, 将棋プログラム「激指」 [28] 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. 鶴岡等は実験の結果, 深さを閾値としたゲーム木探索に比べ, 約 2 倍程度効率よく探索を行えたと述べている.

第3章　提案手法

3.1　ゲームにおける組み合わせ特徴の現状

現在, 組み合わせ特徴は多くのコンピュータゲームプレイヤで利用され, 高い成果を上げている. 既存のコンピュータゲームプレイヤでは, 単純にすべての組み合わせ特徴を展開することで組み合わせ特徴を利用している. 例えば将棋の場合, 2.1 節でも述べたように駒の種類, 位置情報を組み合わせることが一般的である. 組み合わせ特徴はその実装が非常に容易である一方で, 以下のような問題点を持つ.

1. 特徴の冗長性

   組み合わせ特徴は, 考えられるゲーム構成要素の組み合わせをすべて展開するため, 意味のない特徴も多く生成する. 例えば, 2.1.3 節で述べた将棋プログラム「Bonanza」の重み[1]の絶対値の累積値をプロットすると図 3.1 のようになる. 図 3.1 を見ると分かるように, 実際に評価に大きく寄与する特徴は全体の約 5%ほどと, 非常に少ないことが分かる. 特徴の冗長性は, 組み合わせ特徴の数が少ないうちは大きな問題とならないが, より高次の組み合わせ特徴を利用する場合には, 無駄な演算やメモリ消費量が増えるために実行が困難となる.

2. 組み合わせ爆発

   組み合わせ特徴の総数は, それを構成する要素の種類の指数オーダで増大する. 例えば, 将棋においてゲーム構成要素として駒の情報を用いた場合, 組み合わせる要素数と総組み合わせ数の関係は表 3.1 のようになる[2]. 表 3.1 を見ると分かるように, 要素数が増えるに従い総組み合わせ数は爆発的に増大しており, 3 駒以上ではメモリ上にすべての特徴を展開するには非現実的なサイズとなっていることが分かる.

表 3.1: 将棋における要素数と総特徴数

要素数	総組み合わせ数
2	2,570,778
3	1,941,794,316
4	1,099,541,031,435

[1] BonanzaFeliz ver0.0 の重みを使用
[2] 駒の種類が 14 通り, 駒の位置が 81 通り, 駒の手番が 2 通りより, 総組み合わせ数 $= {}_{(14\cdot81\cdot2)}C_{\text{要素数}}$

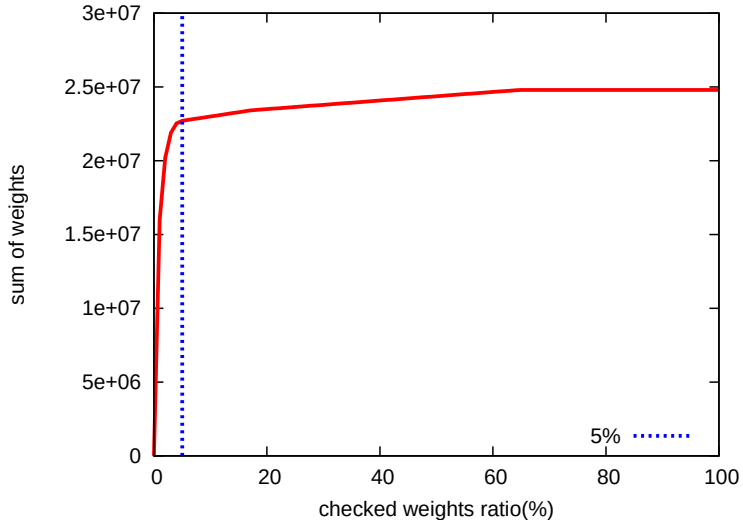


図 3.1: ボナンザの重みの累積値

このような制約のため, 特に構成要素の種類が多いゲームでは組み合わせ特徴の利用が限定的になっている. 将棋の場合, ほとんどのコンピュータプレイヤは 2 駒の組み合わせ特徴までしか用いておらず, 3 駒以上の利用もボナンザのように限られた範囲に限定されている. 一方で, 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利用する必要がある. また, 2.1.1 節でも述べたように, 駒に遮られた長距離利きなど 2 駒間では表現できない特徴も多数存在する. 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は高いと考えられる.

3.2　組み合わせ特徴の最適化

本稿では, 3.1 節で述べた問題に対し, 評価に用いる組み合わせ特徴を絞り込むことによって, 従来よりも高次の組み合わせ特徴を利用する評価関数の提案を行う. 本節では将棋を例に, 提案手法の詳細及び実装について述べていく.

3.2.1　盤面のグラフ表現とパスの展開

提案手法では, ゲームの盤面を図 3.2 のようにゲーム構成要素間を結んだ重み付きグラフで表現する. グラフ中の各ノードにはゲーム構成要素の状態に応じたラベルを付与する. 例えば, 将棋の駒をグラフのノードとした場合, 各ノードには図 3.2 のように駒の種類, 位置に応じたラベルが付与される. また, 各ノード間の関連性の度合いを表す指標として結合度を定義し, グラフ中の各エッジに対して結合度に応じた重み付けを行う. 盤面をゲーム構成要素を用いたグラフとして定義した場合, 組

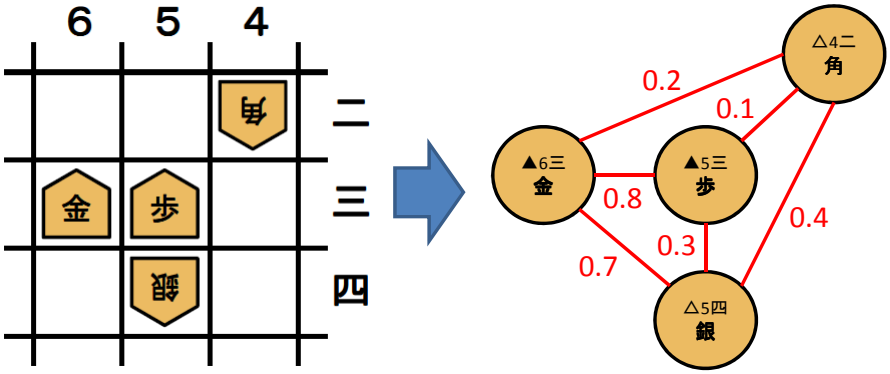


図 3.2: 盤面のグラフ表現

み合わせ特徴はグラフ上のパスとして表現される．そのため, 提案手法ではグラフ上の各頂点からパスを展開することによって組み合わせ特徴の生成を行う．

3.2.2　パスの重要度と枝刈り

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, グラフのエッジの総数を $|E|$, 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は $O(|E|^{d-1})$ と非常に巨大なものとなるため, 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．そこで, 提案手法では各ノード間の結合度を用いてパスの枝刈りを行う．

結合度とは, 各ノードのつながりが局面評価においてどれほど重要であるかを表す指標である．提案手法では, 結合度の高いノードの組み合わせ, つまりつながりの強いパスを選別して展開することによって, 従来の組み合わせ特徴の持つ冗長性の問題の解決を図る．具体的には, 各パスに対して結合度から計算される重要度を定義し, それにしたがってパスの展開, 枝刈りをしていく．グラフ中のノードを n_i, ノード n_i と n_j をつなぐエッジを $e_{i,j}$, パスを $p = \{n_1, n_2, \cdots, n_{|p|}\}$, パスの長さを $|p|$ としたとき, パス p の重要度 $imp(p)$ は各エッジの結合度 $con(e_{i,j})$ を用いて (3.1) 式のように定義される．なお, 結合度は $0.0 \leq con(e_{i,j}) \leq 1.0$ の範囲の値をとるものとする．

$$imp(p) = \begin{cases} 1.0 & \text{if } |p| = 1 \\ \prod_{i=1}^{|p|-1} con(e_{i,i+1}) & \text{otherwise} \end{cases} \tag{3.1}$$

$$\text{where} \quad 0.0 \leq con(e_{i,i+1}) \leq 1.0$$

パスの展開, 枝刈りの具体的な動作を図 3.3 に示す．図 3.3 は図 3.2 のグラフを展開したもので, root と明記されたノードはパスの頂点を統一するための仮想ノードである．また, 下線が引かれた数値は

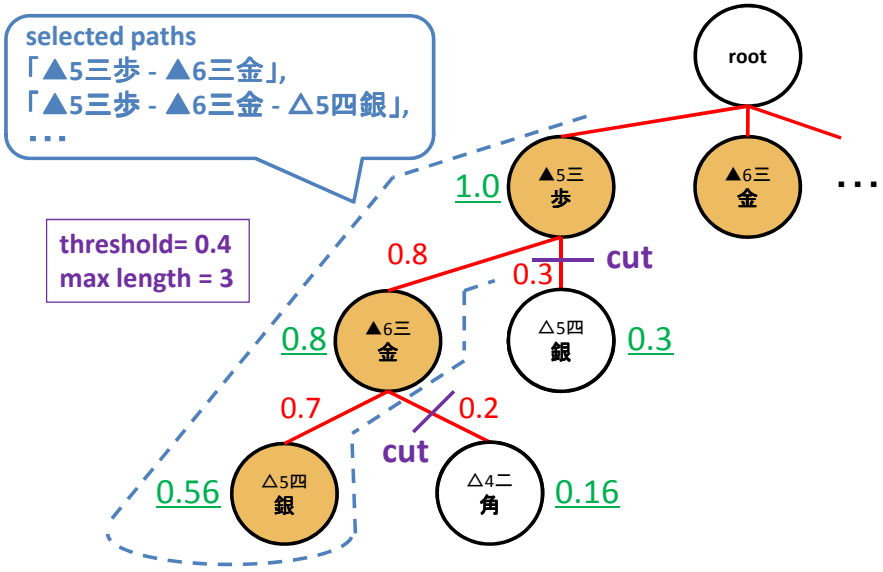


図 3.3: パスの展開と枝刈り

ルートからそのノードまでのパスの重要度を, エッジの横に書かれた数字はそのエッジの結合度を表す. パスの展開は各ノードから伸びるエッジを結合度の大きい順にソートすることで, 深さ優先探索によって効率的に行うことが可能である. N をノードの集合, E_n をノード n からのエッジの集合, P を選択されたパスの集合としたとき, パス展開の具体的な手順は以下の通りである.

1. 選択されたパスの集合 P 及び現在のパス p を初期化
2. すべてのノード $n_i \in N$ に対して以下を繰り返す
   (a) パス p へノード n_i を追加
   (b) ノード n_i から伸びるすべてのエッジ $e_{i,j} \in E_{n_i}$ に対して以下を繰り返す
      i. エッジの先のノード n_j をパス p に追加
      ii. パスの重要度 $imp(p)$ が閾値未満ならば p から n_j を削除して (b) の繰り返しを終了
      iii. パス p を P に追加
      iv. エッジの集合 E_{n_j} に対して (b) の処理を再帰
      v. パス p からノード n_j を削除
   (c) パス p からノード n_i を削除

前述のパス展開の手順では, パス展開の終了条件として重要度の閾値のみを用いているが, 実際の計算では再帰的に (b) の処理を呼ぶ際に, 同一パスでのノードもしくはエッジの重複回避処理, パスの

長さによる展開制限を行う．なお, 図 3.3 および前述の展開手順では省略されているが, 実装では反転させた場合に同じとなるパスが出現しないようにパスの展開を行う．ここで, 反転させて同じになるパスとは, $p_{\{A,B,C\}}$ と $p_{\{C,B,A\}}$ のようなパスのペアのことを意味し, 有向グラフを用いる場合はエッジの向きも考慮する．

提案手法を実現するにあたり問題となるのが, 結合度をいかにして計算するかである．結合度の計算方法としては以下のような方法が考えられる．

1. ゲームの事前知識に基づく結合度
2. 棋譜から学習した結合度

方法 1 は, ゲームのルールや経験から重要と考えられる関係を絞り込む方法である．本稿では, 将棋における利きに注目した結合度を方法 1 として用いた．方法 2 は, ゲームの記録である棋譜を用いて重要と考えられる関係を統計的に見積もる方法である．本稿では, 方法 2 の結合度の計算方法としてロジスティック回帰を用いた．ロジスティック回帰を用いた結合度計算の詳細については 4.2 節にて述べる．

3.2.3　ハッシュ関数を利用したインデックス計算

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, 前述のような枝刈りに伴い得られる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な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て, 提案手法では Zobrist Hash [29] を利用したインデックスの計算を行う．ハッシュ関数を用いることにより, 特徴数をあらかじめハッシュサイズに固定することができ, さらに可変長のパスのインデックス計算を容易に行うことが可能となる．

Zobrist Hash とは, ハッシュ値を求めたいオブジェクトを構成する要素に対してあらかじめ乱数を振り, それら乱数の XOR をとることによってハッシュ値を計算するハッシュ関数であり, コンピュータゲームプレイヤで広く用いられている．XOR には (3.2) 式のように, 同じ値を XOR することによって値を変化させる, 元に戻すといった操作が容易に行えるという性質がある．例えばゲームの盤面のハッシュ値を計算する場合, Zobrist Hash を用いることで手を進める, 戻すといった操作をハッシュで表現することが可能となる．

$$A \oplus B = C$$

$$C \oplus B = A, \quad C \oplus A = B \tag{3.2}$$

提案手法ではパスのハッシュ値を Zobrist Hash を用いて計算するが, このとき, 乱数の振り方として

- グラフ上のノードに乱数を振る
- グラフ上のエッジに乱数を振る

の 2 通りが考えられる．ここで, パス p に対してノードに乱数を振る場合のハッシュ値を $hash_n(p)$, エッジに乱数を振る場合のハッシュ値を $hash_e(p)$ と定義する．ノードに乱数を振る場合, パスのハッシュ値はパス上のノードの組み合わせのみに依存し, パスの経路はハッシュ値には考慮されない．一方, エッジに乱数を振る場合, パスのハッシュ値はパス上のエッジに依存するため, パスの経路によってハッシュ値が変化する．$p_{\{A,B,C\}}$, $p_{\{B,C,A\}}$, $p_{\{C,A,B\}}$ の 3 つのパスを例にとると, ノードに乱数を振る場合は $hash_n(p_{\{A,B,C\}}) = hash_n(p_{\{B,C,A\}}) = hash_n(p_{\{C,A,B\}})$ とすべてのハッシュ値が等しくなり, エッジに乱数を振る場合は $hash_n(p_{\{A,B,C\}}) \neq hash_n(p_{\{B,C,A\}}) \neq hash_n(p_{\{C,A,B\}})$ と 3 つのハッシュ値は異なる値となる．$hash_n(p)$, $hash_e(p)$ の具体的な計算方法は (3.3) 式, (3.4) 式のようになる．

$$hash_n(p) = rand(n_1) \oplus rand(n_2) \oplus \cdots \oplus rand(n_{|p|}) \tag{3.3}$$

$$hash_e(p) = rand(e_{1,2}) \oplus rand(e_{2,3}) \oplus \cdots \oplus rand(e_{|p|-1,|p|}) \tag{3.4}$$

なお, エッジに乱数を振る場合は $rand(e_{i,j}) \neq rand(e_{j,i})$ とした．また, 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, エッジに乱数を振る場合にはエッジが重複しないようにパスの展開を行うこととした．

3.2.4　具体的な評価関数

評価関数の重みベクトルを $\boldsymbol{w}$, 局面 s での盤面グラフを $G(s)$ としたとき, 提案手法の評価関数 $f(\boldsymbol{w}, s)$ は (3.5) 式のようになる．

$$f(\boldsymbol{w}, s) = \boldsymbol{w} \cdot \boldsymbol{\phi}(s) = \sum_i w_i \phi_i(s) \tag{3.5}$$

$$\text{where} \quad \phi_i(s) = \sum_{\substack{p \in G(s), 1 < |p| \leq d, hash(p) = i \\ imp(p) \geq threshold}} \lambda^{|p|-1}$$

ここで, d はパスの最大長, つまり組み合わせ数の最大値を表し, λ は高次組み合わせ特徴の影響度を制御する減衰定数である．また, $hash(p)$ は 3.2.3 節で定義した $hash_n(p)$, $hash_e(p)$ のいずれかである．(3.5) 式中でパスの長さ $|p|$ の範囲を $1 < |p| \leq d$ と制限しているのは, 本手法では 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．(3.5) 式を見ると分かるように, 提案手法ではハッシュ値が同じ特徴については重みを共有して評価値の計算を行うことで総特徴数をハッシュサイズに抑え込んでいる．乱数をノードに振った場合の具体的な評価関数計算の擬似コードはアルゴリズム 3.1 のようになる．

Algorithm 3.1 calc_eval(*eval*, *graph*, *path*, *hash*, *imp*)

```
1: // check path length
2: if len(path) ≥ max_length then
3:    return
4: end if
5:
6: // check all edges from the last node
7: sorted_edge_list ← get_sorted_edge_list(graph, tail(path))
8: for all edge ∈ sorted_edge_list do
9:
10:   // get next_node and check next_node has been already checked
11:   next_node ← next_node(edge)
12:   if is_checked(next_node) then
13:      continue
14:   end if
15:
16:   // update importance of path and compare with the threshold
17:   next_imp ← imp × con(edge)
18:   if next_imp < threshold then
19:      break
20:   end if
21:
22:   // check next_node, update path and hash
23:   check(next_node)
24:   push_back(path, next_node)
25:   hash ← hash ⊕ rand(next_node)
26:
27:   // update evaluate value and call calc_eval recursively
28:   if len(path) > 1 then
29:      eval ← eval + λ^(len(path)−1) × weight(hash)
30:   end if
31:   calc_eval(eval, graph, path, hash, new_imp)
32:
33:   // restore path and hash, uncheck next_node
34:   hash ← hash ⊕ rand(next_node)
35:   pop_back(path)
36:   uncheck(next_node)
37: end for
```

第4章　実験と評価

4.1　実験方法

本稿では, ゲームとして将棋を用いて提案手法の評価を行った. 実験では, 将棋プログラム「激指」[28] をベースとして用いた. 激指は, 2010 年に開催された第 20 回世界コンピュータ将棋選手権[1]で優勝を収めるなど, トップレベルの強さを誇る将棋プログラムである. 実験では, 激指の評価関数部分に変更を加え, 評価関数のパラメタを機械学習によって調整することで手法の精度評価を行った. 以降, 評価関数の詳細, 具体的な学習手法, 評価基準について述べる.

4.1.1　評価関数

本実験では, 評価関数として (4.1) 式のように, 盤面の駒割に提案手法や比較手法の評価関数を追加したものを用いた.

$$eval(s) = \text{駒割}\ (s) + f(\boldsymbol{w}, s) \tag{4.1}$$

駒割とは, 「盤面上に歩が一枚あれば 100 点」のように各駒に対して割り振られた点数のことで, 将棋では最も基本的な評価要素である. 実験では駒割を固定の値として扱い, パラメタの調整は追加した評価関数部分に制限した.

3.2.2 節において, 提案手法の結合度の計算方法として 2 つの方法を言及したが, 本実験では各手法に対応する結合度を用いた以下のような 2 つの盤面グラフを実装した.

- 利きグラフ (kiki)

  駒の利き (間接利き[2]を含む) のあるエッジの結合度を 1.0, その他のエッジの結合度を 0.0 と設定した盤面グラフ. これは, 利きをエッジとするグラフを意味し, 3.2.2 節で述べた「ゲームの事前知識に基づく結合度」に相当する. 利きグラフでは, 利きのついている空升も盤面グラフのノードとしてカウントし, また, ハッシュの計算では各エッジに対して乱数を振る方法を用いた. エッジに対して乱数を振るのは, 利きグラフではどのように利きを辿ってきたかという経路情報が局面評価において重要になると考えたためである. なお, 利きグラフは利きの向きを考慮した有向グラフとして実装し, パス展開では向きの違うエッジを区別して扱った.

[1] http://www.computer-shogi.org/wcsc20/
[2] 歩の後方の飛車から歩の前方への利きなど, 実際は利いていないが駒交換が進むと利いてくる利き

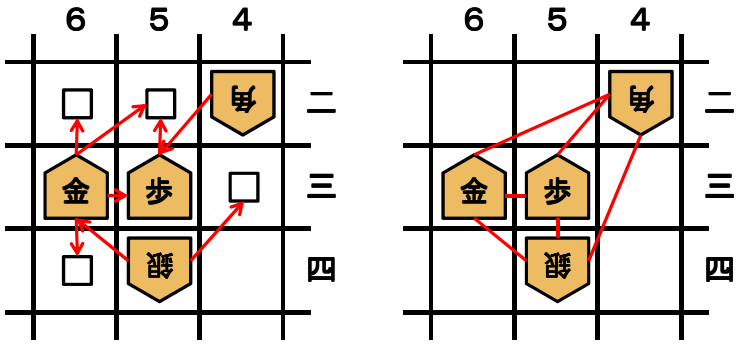


図 4.1: 左：利きグラフ, 右：駒グラフ

- 駒グラフ (koma)

  盤面上の駒を全対全で結び, 結合度として棋譜から学習した値を用いる盤面グラフ．これは, 3.2.2 節で述べた「棋譜から学習した結合度」に相当する．駒グラフでは, グラフ上のノードを盤面上の駒に限定し, 空升は除外した．また, 駒グラフではパス上の駒の組み合わせが重要な情報と考えられるため, ハッシュの計算として各ノードに対して乱数を振る方法を用いた．

利きグラフ, 駒グラフの例を図 4.1 に示す．

4.1.2　学習方法

本実験では, 評価関数のパラメタの学習方法として, 激指の方法 [13] を用いた．激指では, 2.1.3 節で述べたボナンザメソッド [9] 及び Averaged Perceptron [30] をベースとした学習手法が用いられている．ここで, Averaged Perceptron とはオンライン学習の一種で, (4.2) 式のように学習途中の各ステップ i でのパラメタ $\boldsymbol{w}^i$ の総和を保持し, それを総ステップ数 N で割った平均 $\bar{\boldsymbol{w}}$ 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．

$$\bar{\boldsymbol{w}} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{w}^i \tag{4.2}$$

学習は棋譜中の局面毎にパラメタの更新を行うオンライン学習の方式で行った．各局面では棋譜の手と棋譜の手を除くすべての合法手の中からランダムに 16 手を選択し, 各指し手 m_j の後の局面 s_j から現在のパラメタを用いて深さ 5 の探索を行うことで末端局面 s'_j を展開した．すべての合法手を選択せずに一部をランダムに選択しているのは, 学習時間の短縮のためである．学習ステップ t でのパラメタ $\boldsymbol{w}^t$ の更新は, 得られた最善応手手順後の局面 s'_j を用いて (4.3) 式のように行った．

$$\boldsymbol{w}^{t+1} \leftarrow \boldsymbol{w}^t + \frac{1}{|M|} \sum_{m_j \in M} y \left(\boldsymbol{\phi}(s'_1) - \boldsymbol{\phi}(s'_j) \right) \tag{4.3}$$

ここで, 指し手 m_1 は棋譜中で指された手を表す. また, y は (2.5) 式で表される局面の手番に応じたラベルであり, (4.3) 式中の M は (4.4) 式を満たすような合法手 $m_j(j \neq 1)$ の集合である.

$$y\left(\boldsymbol{w}^t \cdot \boldsymbol{\phi}(s'_1) - \boldsymbol{w}^t \cdot \boldsymbol{\phi}(s'_j)\right) < margin \tag{4.4}$$

$$\text{where} \quad margin = 10 + 2 \cdot bt \ (0 \leq bt \leq 127) \tag{4.5}$$

(4.4) 式は (2.6) 式で示したヒンジ損失関数で m を $margin$ としたものに相当する. (4.5) 式中の bt は進行度を表す変数である. 激指では, 進行度は盤面上の駒の位置や持ち駒の数から計算され, 相手の陣地に攻め込むほど, つまりゲームが終局に進むほど大きな値をとるように設計されている. $margin$ を (4.5) 式のように設定することにより, 指し手の善し悪しが明確になってくる終盤ほど棋譜の手を相対的により高く評価するようなパラメタを得ることが可能となる.

学習の具体的な流れは以下の通りである.

1. $\boldsymbol{w} \leftarrow \boldsymbol{0}$ に初期化
2. 棋譜 S 中の局面とラベルのペア $(s_{rt}, y) \in S$ をランダムに取り出し, (s_{rt}, y) を S から除外.
3. (s_{rt}, y) に対して以下の操作を行う
   (a) 局面 s_{rt} の全合法手 M_{all} から棋譜の手 m_1 及び棋譜の手以外からランダムに 16 手 $M = \{m_2, \cdots, m_{17}\}$ を取得
   (b) 選択した指し手 $m_j \in \{M \cup m_1\}$ に対して以下の操作を行う
      i. 指し手 m_j の後の局面 s_j を展開
      ii. 局面 s_j から深さ 5 で探索を行い末端局面 s'_j を取得
   (c) 得られた末端局面のうち, $m_j \in M$ に対して以下の操作を行う
      i. 末端局面 s'_j が (4.4) 式を満たさなければ M から m_j を除外
   (d) (4.3) 式を用いてパラメタ $\boldsymbol{w}$ を更新
4. S が空でなければ 2 に戻る
5. (4.2) 式を用いてパラメタを平滑化し, $\bar{\boldsymbol{w}}$ を最終的なパラメタとして得る

学習にはプロ棋士による対局 30,000 棋譜を用い, 更新を行う総局面数は 3,346,647 局面となった. また, 一部の実験では学習の高速化のために近似データを利用した. ここで近似データとは, 学習用の 30,000 棋譜を激指の評価関数を用いて各局面の合法手であらかじめ深さ 5 の探索を行い, 各合法手での最善応手手順を記録したものである. 近似データを用いた場合, 学習及び次に述べるテストでは探索を行わない. なお, 学習は Intel Xeon X5560 (2.80GHz) の環境にて行った.

4.1.3　評価基準

本実験では精度の評価基準として, 以下に示すような 3 つの指標を用いた.

- 一致率 (%) [accuracy]

  棋譜中の各局面を探索し, その結果得られた手が棋譜の手と一致した割合を表す指標である. 一致率は (4.6) 式のように定義される.

$$\text{一致率 (\%)} = \frac{\{\text{ 探索した手 = 棋譜の手 }\}\text{ となる局面数}}{\text{棋譜中の全局面数}} \times 100 \tag{4.6}$$

  なお, 本実験では各局面の全合法手後の局面にて探索を行い, 最も評価値が高かった局面の指し手が棋譜の手と一致したかを調べることで一致率の計測を行った.

- 不一致度 [j prime]

  棋譜中の各局面を探索し, 棋譜の手以外の指し手を間違って評価した度合いを表す指標である. 本実験では不一致度を (4.7) 式のように定義した.

$$\text{不一致度} = \frac{1}{|S|} \sum_{(s_{rt},y)\in S} \sum_{j\in M_{all}, j\neq 1} T\left(y(\boldsymbol{w}\cdot\boldsymbol{\phi}(s'_1) - \boldsymbol{w}\cdot\boldsymbol{\phi}(s'_j))\right) \tag{4.7}$$

$$\text{where} \quad T(x) = \frac{1}{1+\exp(3x/100)} \tag{4.8}$$

  (4.7) 式では, 棋譜の手の評価値とその他の手の評価値の差分を計算し, それを (4.8) 式で表されるシグモイド関数によって $(0.0, 1.0)$ の範囲に正規化している. 不一致度は, 直感的には各局面において棋譜の手よりも相対的に良く評価した手の総数の平均を表す指標ともいうことができる. なお, 不一致度は値が小さいほど精度が良いことを表す.

- 対戦

  本実験では, 探索でのノード数を固定して対戦を行う. ノード数を制限しているのは, 探索速度等の要因を排除し, 純粋な評価関数の強さの比較を行うことを目的としているためである. 具体的には探索ノード数を 50 万ノードと制限し, 双方はじめの 30 手を固定とし, 各初期局面を先手後手入れ替えて対戦を行う.

評価は, 棋譜をそのまま用いる場合には学習で用いなかった 250 棋譜を, 近似データを用いる場合には 500 棋譜を用いて行った. 一致率, 不一致度は学習での方法を合法手の数を制限せずに行うことで計測した. また, 対戦で用いる初期局面は学習で用いなかった棋譜のうち, 激指の評価関数で先手後手の優劣の差が 50 以下[3]となる局面を利用した. 対戦での対局数は 500 試合とした. なお, 対戦結果を示す際, 有意水準 5%の 2 項検定で有意に勝ち越した結果[4]には o の印を, 負け越した結果には x の印をつけている.

[3] 評価値が ±50 以内, この数値は歩 1 枚の点数である 100 点から決定した.
[4] 具体的には, 500 戦中 273 勝以上

4.2　結合度の学習

本節では, 駒グラフで用いる結合度の学習方法の詳細について述べる. 本実験では, 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により学習することで結合度を導出した. 具体的な特徴は 2 駒の位置関係, 計 2,570,778 特徴量とした.

はじめに, パラメタ調整で用いる 30,000 棋譜から結合度学習のための訓練データを生成した. 訓練データの生成は, ボナンザメソッドで利用される比較学習のように棋譜の手とそれ以外の手での特徴の差分を抽出することで行った. 具体的な手順は以下の通りである.

1. 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 $\{s_1, s_j\}$ をランダムに一つずつ抽出する
2. 各ペアについて, それぞれの局面の特徴ベクトル (=エッジ) の差 $\boldsymbol{x} = \boldsymbol{\phi}(s_1) - \boldsymbol{\phi}(s_j)$ を計算し, 訓練データとする
3. 各訓練データのラベル y については, 元の局面が先手なら $y = +1$ (正例), 後手なら $y = -1$ (負例) とする

1. 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すぎるのを防ぐためである. データの展開には Intel Xeon X5560 (2.80GHz) の環境で約 2 時間を要し, 得られたデータファイルのサイズは約 7.5GB となった.

次に, 得られた訓練データを用いてロジスティック回帰による結合度の学習を行った. 本実験では, ロジスティック回帰として LIBLINEAR [31,32] を利用した. ロジスティック回帰は L2 正則化, L1 正則化の 2 つの方法を用い, 具体的には L2 正則化では (4.9) 式, L1 正則化では (4.10) 式の最適化を行う.

$$\min_{\boldsymbol{w}_e} \quad \frac{1}{2}\|\boldsymbol{w}_e\|_2^2 + C \sum_{(\boldsymbol{x}_i, y_i) \in S} \log\left(1 + e^{-y_i \boldsymbol{w}_e \cdot \boldsymbol{x}_i}\right) \tag{4.9}$$

$$\min_{\boldsymbol{w}_e} \quad \|\boldsymbol{w}_e\|_1 + C \sum_{(\boldsymbol{x}_i, y_i) \in S} \log\left(1 + e^{-y_i \boldsymbol{w}_e \cdot \boldsymbol{x}_i}\right) \tag{4.10}$$

ここで, (4.9) 式及び (4.10) 式中の C は 2 つの項の影響度を制御する正則化項である. 本実験では LIBLINEAR のデフォルト値である $C = 1$ とした. ロジスティック回帰によって得られた各エッジの重み $\boldsymbol{w}_e$ は (4.11) 式を用いて結合度に変換した.

$$con(e_{i,j}) = \frac{1}{1 + \exp(-w_{e_{i,j}})} \tag{4.11}$$

なお, (4.11) 式中の $w_{e_{i,j}}$ は $\boldsymbol{w}_e$ の中でエッジ $e_{i,j}$ に対応する重みを表す. (4.11) 式より, 結合度は $0.0 < con(e_{i,j}) < 1.0$ の範囲の値となる.

ロジスティック回帰を用いて (4.11) 式のように結合度を計算した場合, 結合度は以下のように 2 通りのとらえ方ができる.

- 正例 (先手) 側から見た結合度 $con_{pos}(e_{i,j}) = con(e_{i,j})$
- 負例 (後手) 側から見た結合度 $con_{neg}(e_{i,j}) = 1 - con(e_{i,j})$

このため, 本実験では各パスの重要度を正例側, 負例側の 2 つについて計算し, 値の大きい方をそのパスの重要度として採用することとした. 具体的には, 2 つの重要度は (4.12) 式のように計算される.

$$imp(p) = \max(imp_{pos}(p), imp_{neg}(p)) \tag{4.12}$$

$$\text{where} \quad \begin{cases} imp_{pos}(p) = \prod_{i=1}^{|p|-1} con_{pos}(e_{i,i+1}) = \prod_{i=1}^{|p|-1} con(e_{i,i+1}) \\ \\ imp_{neg}(p) = \prod_{i=1}^{|p|-1} con_{neg}(e_{i,i+1}) = \prod_{i=1}^{|p|-1} (1 - con(e_{i,i+1})) \end{cases}$$

L2 正則化, L1 正則化で得られた結合度について, $con(e_{i,j}) > 0.5$ となるものを正例, $con(e_{i,j}) < 0.5$ となるものを負例とし, $con(e_{i,j}) = 0.5$ (i.e. $w_{e_{i,j}} = 0$) となるエッジを除外した場合の結合度の分布は図 4.2, 具体的な割合は表 4.1 のようになった. 結合度の学習は Intel Xeon X5680 (3.33GHz) の環境において, L2 正則化が 62 分, L1 正則化が 18 分の時間を要した. 図 4.2 を比較すると, L2 正則化では小さい値の結合度に分布が偏っているのに対し, L1 正則化では結合度が広く分布していることが分かる. これは (4.9) 式及び (4.10) 式のように, 2 つの正則化では重みの大きさによるペナルティのかかり方が異なるためである. また, 表 4.1 より L1 正則化が L2 正則化に比べ最終的な非零の重みのついたエッジの割合が 4 分の 1 程度と少ないことが分かる.

4.2.1　ボナンザとの整合性比較

本節では, 得られた結合度によって重要パターンがどの程度取れるのかを調べるために, ボナンザのパラメタとの整合性の評価を行った. 実験では, ボナンザの非零のパラメタを重みの絶対値が大きい順にソートし, 各パラメタに相当する組み合わせ特徴がどの程度結合度を用いてカバーできるかを評価した. なお, 本節では持ち駒を組み合わせ特徴の要素とするパラメタについては考えない. パス展開時の閾値の値は, 探索速度が同程度となる値として L2 正則化が 0.4、L1 正則化が 0.45 とした.

表 4.1: 結合度の全体に対する割合 (左：L2 正則化, 右：L1 正則化)

$con(e_{i,j})$	pos (%)	neg (%)	total
0.9 - 1.0	0.01	0.01	0.02
0.8 - 0.9	0.04	0.04	0.08
0.7 - 0.8	0.15	0.15	0.30
0.6 - 0.7	1.69	1.68	3.37
0.5 - 0.6	25.55	25.65	51.20
all	27.44	27.53	54.97

$con(e_{i,j})$	pos (%)	neg (%)	total
0.9 - 1.0	0.12	0.12	0.24
0.8 - 0.9	0.31	0.31	0.62
0.7 - 0.8	0.72	0.71	1.43
0.6 - 0.7	1.73	1.73	3.46
0.5 - 0.6	4.24	4.26	8.50
all	7.12	7.13	14.25

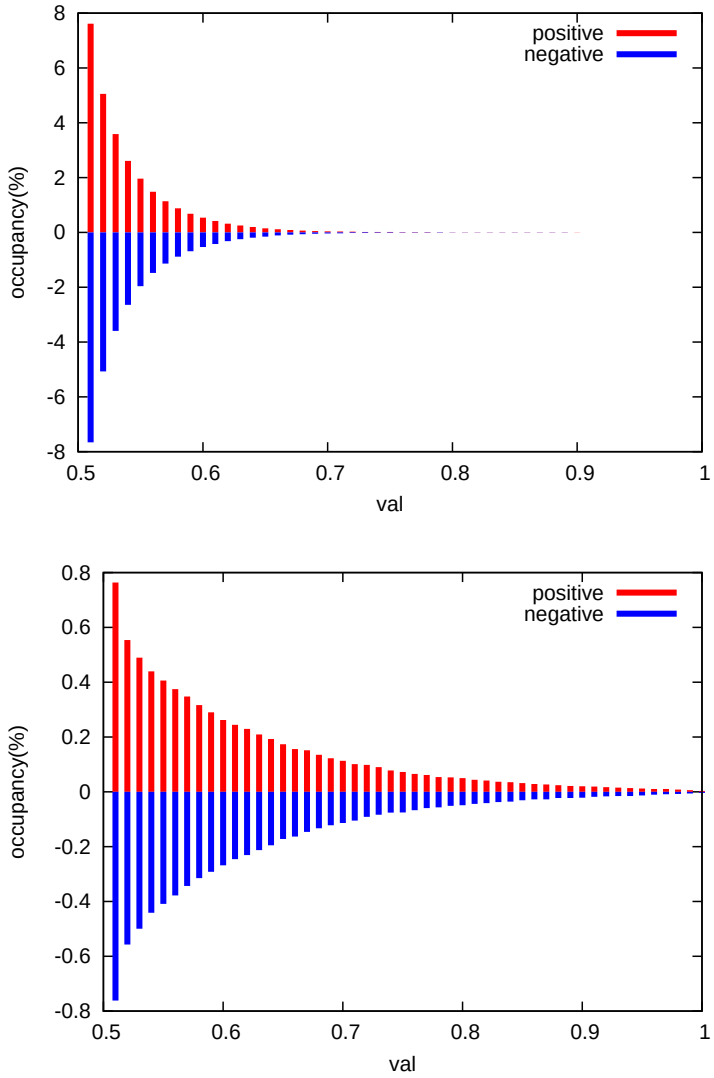


図 4.2: 結合度の分布 (上：L2 正則化, 下：L1 正則化)

各手法でのカバー率は図 4.3 のようになった. 図 4.3 を見ると, 2 つの手法とも上位のパターンで高いカバー率を示したのち, パターンが増えるに従いカバー率が下がる傾向があることが分かる. これは, 重みの絶対値が大きい, つまり重要度の高いパターンを取り出し, 逆に重要度の低いパターンを枝刈りしていることを示しており, 提案手法によってうまく有効パターンを抽出できているといえる. 他方, ボナンザのパターンのカバー率の最大値は 25% 程度にとどまっているが, これは提案手法において 2 駒間の結合度から 3 駒間のパターンの重要度を近似していることが影響していると考えられる.

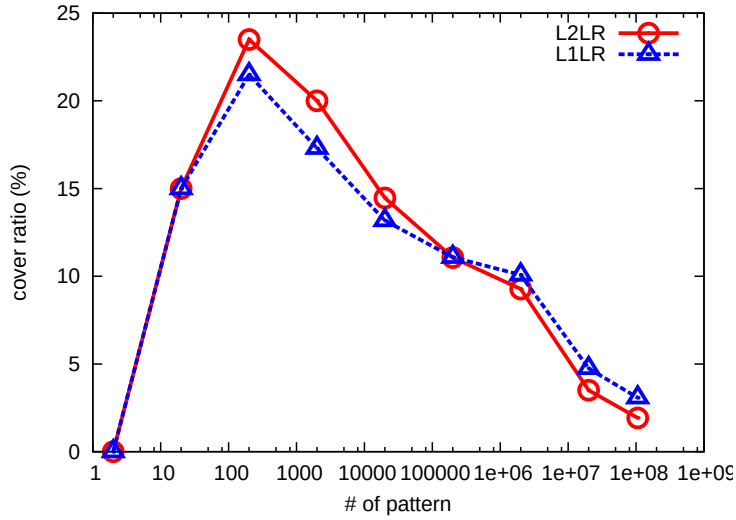


図 4.3: ボナンザのパターンのカバー率

4.2.2　結合度と精度

4.2 節の初めに述べたように, 結合度の学習には L2 正則化ロジスティック回帰と L1 正則化ロジスティック回帰の 2 つを用いている. 本節では, この 2 つの結合度を駒グラフに実際に適用し, それぞれの手法での結合度の比較を行った. 実験では, (3.5) 式中の減衰定数 λ を 0.8, パスの最大長 d を 4, ハッシュサイズを 2^{24} とし, パス展開での閾値は 4.2.1 節での数値を用いた. 学習の結果得られた精度及び相互対戦結果を表 4.2 に示す. 表 4.2 の有効特徴数は最終的に得られたパラメタ中の非零のパラメタの総数を, 勝率はもう一つの手法に対する勝率を表す. 表 4.2 を見ると, 精度, 勝率共に 2 手法間で差がなく, 2 つの手法は同程度の強さであるといえる. 有効特徴数に注目すると, L2 正則化が L1 正則化に比べ特徴数が少ないことが分かる. これは, 図 4.2 で示したように, 結合度の分布が L1 正則化の方が広く薄く分布しているため, L2 正則化に比べ実際に展開された高次組み合わせ特徴の種類が多かったからと推測される. 実際に各局面評価で展開した組み合わせ特徴の平均数 (平均評価特徴数) は表 4.3 のようになり, L1 正則化を用いた場合は L2 正則化に比べ, 高次組み合わせ特徴をより多く展開するような尖った分布となっていることが分かる. 提案手法では, 3.2.3 節で述べたようにパス上の要素の乱数からハッシュ値を計算するため, 要素数が多い, つまり高次の組み合わせ特徴ほどハッシュ値のばらつきが大きくなり, その結果 L1 正則化の方が有効特徴数で上回る結果となったといえる.

4.2.3　抽出されるパターン

本節では, 得られた結合度を用いて実際に展開される具体的なパターンについて述べる. パターンはテスト用の棋譜を含む 500 棋譜のうち, 30 手から 70 手までの局面から 4 駒間のパターンで重要度の高い上位 1000 パターンを抽出した.

抽出したパターンのうち, 重要度が最も大きかったパターンを図 4.4 に, サブパターンも含めた総合の評価値が最も大きかったパターンを図 4.5 に示す. 表中の重要度の符号は, 正の数は正例側から見た重要度を, 負の数は負例側から見た重要度をそれぞれ表す. 重要度が最も大きいパターンに注目すると, △１三角と△３四飛, △９一玉と△７一金と部分的に意味のありそうなパターンは取れているものの, 両者ともに全体では意味のあるパターンとは言い難い形となっている. これは, 提案手法が 2 駒間の関係性から学習した結合度を用いて 4 駒間のパターンの重要度を近似しているために, 単純に結合度の高い駒だけを取り出した場合に必ずしも意味のある高次パターンが取れるとは限らないためである. 3 駒間, 4 駒間といったまとまった単位でも整合性の取れる結合度の導入は今後の課題の一つといえる. 一方, 評価値が最も大きいパターンでは, L2 正則化では穴熊囲いのパターンが, L1 正則化では矢倉囲いのパターンが抽出されていることが分かる. また, 各パターンの評価値に注目すると, 提案手法では抽象度の高い 2 駒間でおおまかな評価をしつつ, 3 駒以上の高次パターンによって評価値を補正する形となっていることが読み取れる. 各パターンの重要度に注目すると, 先手の駒によるパターンでは負例側の重要度が, 後手の駒によるパターンでは正例側の重要度が用いられており, 訓練データ作成時のラベルと逆になっていることが分かる. この原因としては, 棋譜の手とは別の手をランダムに抽出したことが考えられる. これは, ラベルの符号と逆の重みがつく可能性があるのが棋譜以外の手に対応する特徴であるため, また本来正例・負例に広く分布するはずの特徴がランダムに手を選択したことでどちらか片方に偏り, その結果大きな重みがつく可能性があるためである.

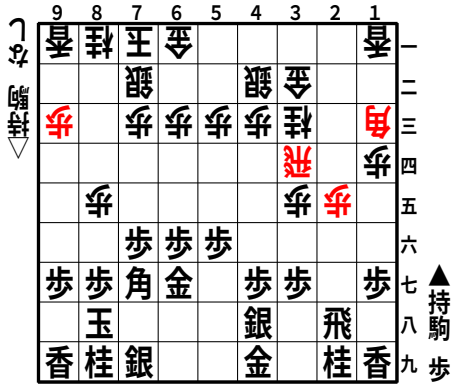
本稿では, 以降の実験において結合度として L2 正則化により学習したものを用いることとした.

表 4.2: 各結合度での学習結果

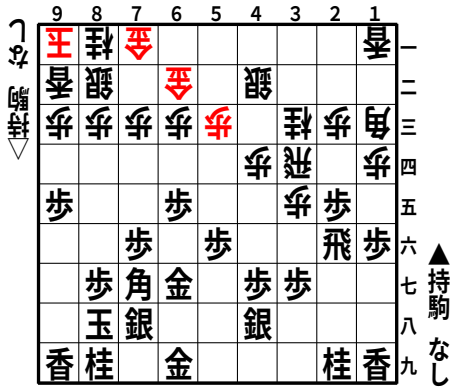
手法	学習時間	有効特徴数	一致率 (%)	不一致度	勝率 (%)
L2 正則化	18h 54m	6,194,174	40.8342	4.38366	51.8
L1 正則化	18h 43m	9,483,675	40.6971	4.38233	48.2

表 4.3: 各結合度での平均評価特徴数

手法	2 駒	3 駒	4 駒	計
L2 正則化	519	444	54	1017
L1 正則化	519	300	82	901

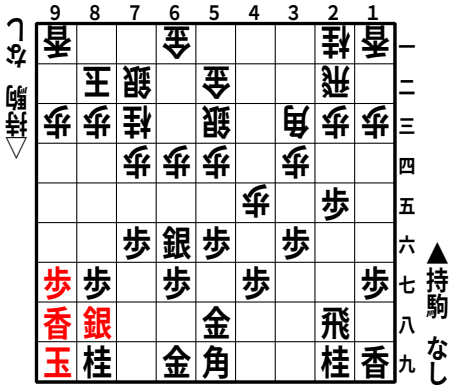


サブパターン	重み	減衰
△９三歩, △１三角	8.047	0.8
△１三角, △３四飛	2.348	0.8
△３四飛, △２五歩	-0.292	0.8
△９三歩, △１三角, △３四飛	0.779	0.8^2
△１三角, △３四飛, △２五歩	-0.426	0.8^2
△９三歩, △１三角, △３四飛, △２五歩	7.058	0.8^3
パターンの重要度	+0.757	
パターンの評価値	11.921	

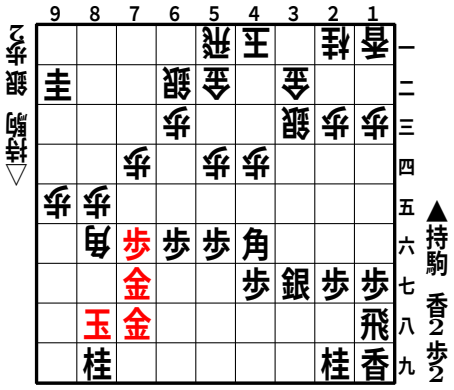


サブパターン	重み	減衰
△５三歩, △６二金	-5.548	0.8
△６二金, △９一玉	3.737	0.8
△９一玉, △７一金	-5.008	0.8
△５三歩, △６二金, △９一玉	0.612	0.8^2
△６二金, △９一玉, △７一金	-1.415	0.8^2
△５三歩, △６二金, △９一玉, △７一金	-0.537	0.8^3
パターンの重要度	+0.902	
パターンの評価値	-6.244	

図 4.4: 重要度が最も大きいパターン (上：L2 正則化, 下：L1 正則化)



サブパターン	重み	減衰
▲９七歩, ▲９八香	-25.744	0.8
▲９八香, ▲９九玉	96.787	0.8
▲９九玉, ▲８八銀	75.880	0.8
▲９七歩, ▲９八香, ▲９九玉	28.516	0.8^2
▲９八香, ▲９九玉, ▲８八銀	71.114	0.8^2
▲９七歩, ▲９八香, ▲９九玉, ▲８八銀	24.514	0.8^3
パターンの重要度	-0.587	
パターンの評価値	193.853	



サブパターン	重み	減衰
▲７六歩, ▲７七金	16.659	0.8
▲７七金, ▲８八玉	62.843	0.8
▲８八玉, ▲７八金	129.304	0.8
▲７六歩, ▲７七金, ▲８八玉	26.024	0.8^2
▲７七金, ▲８八玉, ▲７八金	13.293	0.8^2
▲７六歩, ▲７七金, ▲８八玉, ▲７八金	21.821	0.8^3
パターンの重要度	-0.702	
パターンの評価値	203.380	

図 4.5: 評価値が最も大きいパターン (上：L2 正則化, 下：L1 正則化)

4.3　ハッシュサイズと精度

3.2.3 節で示したように, 提案手法ではインデックスの計算にハッシュ関数を利用するため, ハッシュの衝突の精度への影響が懸念される. 本節では, 複数のハッシュサイズを用いて精度比較を行うことで, 衝突率と精度の関係について調べる.

実験では駒グラフの結合度として 4.2 節で述べた L2 正則化ロジスティック回帰によって学習されたものを用い, 評価関数のパラメタは 4.2.2 節と同様に減衰定数として 0.8, パス展開の閾値として 0.4 を用いた. 実験ではハッシュサイズとして 2^{16}, 2^{18}, 2^{20}, 2^{22}, 2^{24}, 2^{26} の 6 種類を用意した. また, テスト用の 500 棋譜の近似データを用いて各ハッシュサイズでの衝突率及び平均重複数の計測を行った. 衝突率, 平均重複数は具体的には (4.13), (4.14) 式のように計算した.

$$\text{衝突率 (\%)} = \frac{(\text{総インスタンス数} - \text{有効ハッシュ数}) \times 100}{\text{総インスタンス数}} \tag{4.13}$$

$$\text{平均重複数} = \frac{\text{総インスタンス数}}{\text{有効ハッシュ数}} \tag{4.14}$$

ここで, 総インスタンス数は棋譜中で出現したインデックスの総数, 有効ハッシュ数は棋譜中で出現したハッシュ値の総数を表す. (4.14) 式からも分かるように, 平均重複数は衝突率を重みが共有されている特徴の平均数に言い換えた指標である.

それぞれのハッシュサイズでの精度は表 4.4, 一致率, 不一致度の学習曲線は図 4.6 のようになった. 表 4.4 及び図 4.6 を見ると, 一致率, 不一致度はハッシュサイズが 2^{20} になるまで大きく向上し, その後も 2^{26} まで少しずつ向上していることが分かる. 2^{20} までの向上率が大きい要因としては, ハッシュの衝突率及び平均重複数が考えられる. 2^{20} 以下のハッシュサイズではハッシュの衝突率は 80% 程度かそれ以上と高い数値となっており, それが性能劣化の原因となっている. 特に平均重複数に注目すると, 2^{20} 以下ではハッシュサイズの変化に伴う平均重複数の増減が非常に大きいことがわかり, この結果として向上率が高い. 表 4.4 のうち, 衝突率と不一致度の推移をプロットしたグラフは図 4.7 のようになる. 図 4.7 を見ると, 本実験の条件では衝突率が 80% 近くあっても精度をある程度維持できていることが読み取れる.

表 4.4: 各ハッシュサイズでの学習結果

ハッシュサイズ	学習時間	有効特徴数	衝突率 (%)	平均重複数	一致率 (%)	不一致度
2^{26}	19h 50m	7,298,994	3.6740	1.0381	40.8166	4.36953
2^{24}	18h 54m	6,194,174	13.6231	1.1577	40.8342	4.38366
2^{22}	18h 4m	3,530,284	41.8259	1.7190	40.6690	4.41196
2^{20}	18h 30m	1,047,920	79.3790	4.8494	40.1279	4.44674
2^{18}	18h 43m	262,144	94.8025	19.240	39.6325	4.58743
2^{16}	19h 2m	65,536	98.7006	76.960	38.2656	4.99655

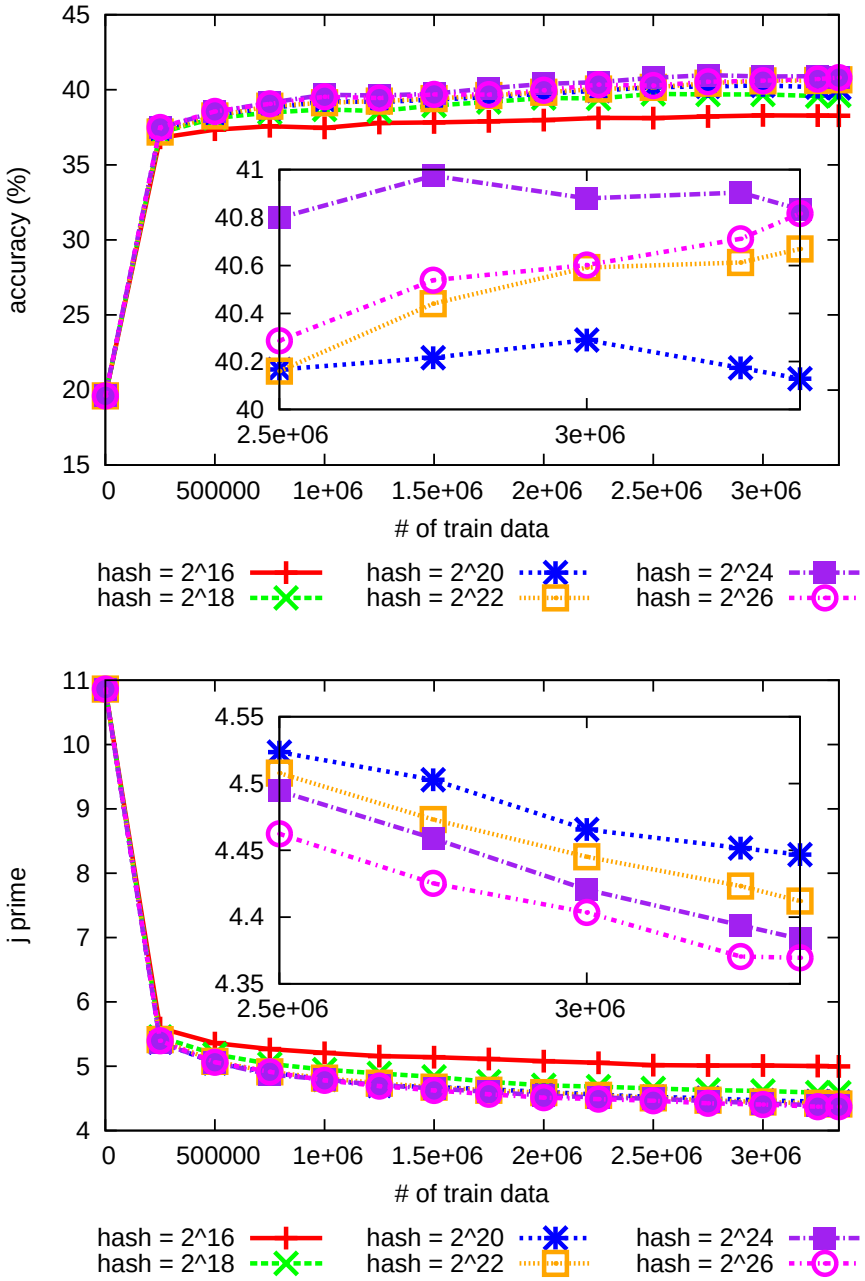


図 4.6: 各ハッシュサイズでの学習曲線 (上：一致率, 下：不一致度)

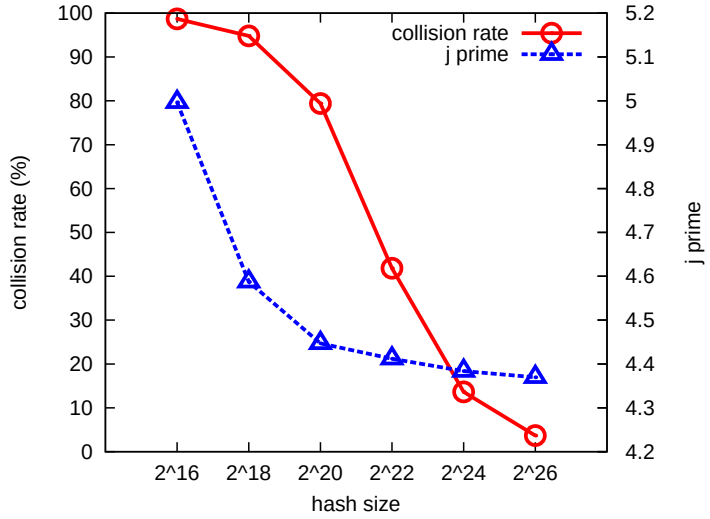


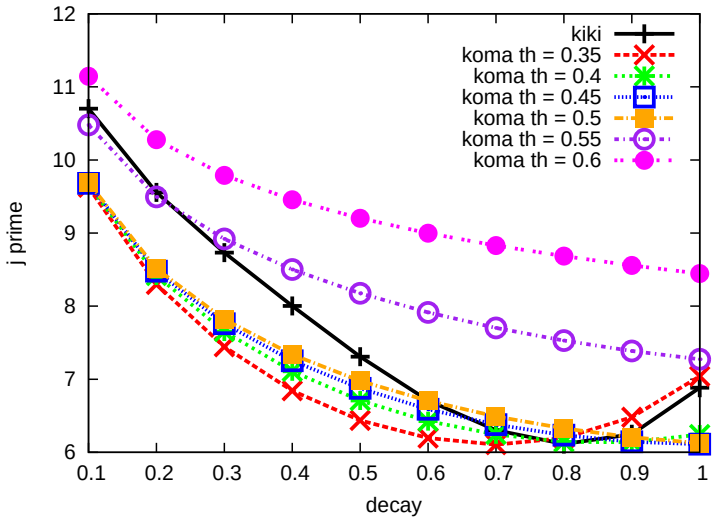
図 4.7: 衝突率と不一致度の関係

各ハッシュサイズでの相互対戦の結果を表 4.5 に示す. 表 4.5 を見ると, 図 4.7 で精度が相対的に悪い 2^{16}, 2^{18} の 2 つが他のハッシュサイズに比べて有意に負け越していることが分かる. 一方で, 2^{20}, 2^{22}, 2^{24}, 2^{26} の 4 つについてはハッシュサイズが大きくなることによる多少の勝率の改善はみられるものの, 有意な差はつかなかった. この結果から, 衝突率がある閾値を下回っていれば, 強さの面でも大きく悪化することを防止できるといえる.

以上の結果より, ハッシュを用いた場合に衝突率が多少高くとも精度, 強さを維持できることが示された. 特に, 本実験では衝突率が 80% 近くある場合でも対戦で互角に近い成果を挙げることに成功した. なお, 本節では提案手法での結果のみを示しているが, 4.5 節で述べる既存手法においても同様の結果が得られている (詳細は付録 A を参照されたい). 本稿では以降, ハッシュサイズとして衝突率と精度を考慮し 2^{24} を用いることとした.

表 4.5: 各ハッシュサイズでの相互対戦結果

自分 \ 相手	2^{16}	2^{18}	2^{20}	2^{22}	2^{24}	2^{26}
2^{16}		**x36.0**	**x28.2**	**x27.6**	**x25.2**	**x22.0**
2^{18}	**o64.0**		**x42.6**	**x44.2**	**x38.6**	**x41.8**
2^{20}	**o71.8**	**o57.4**		46.6	45.6	49.2
2^{22}	**o72.4**	**o55.8**	53.4		47.4	49.6
2^{24}	**o74.8**	**o61.4**	54.4	52.6		49.2
2^{26}	**o78.0**	**o58.2**	50.8	50.4	50.8	



手法	減衰定数
kiki	0.8
0.35	0.7
0.4	0.8
0.45	1.0
0.5	1.0
0.55	1.0
0.6	1.0

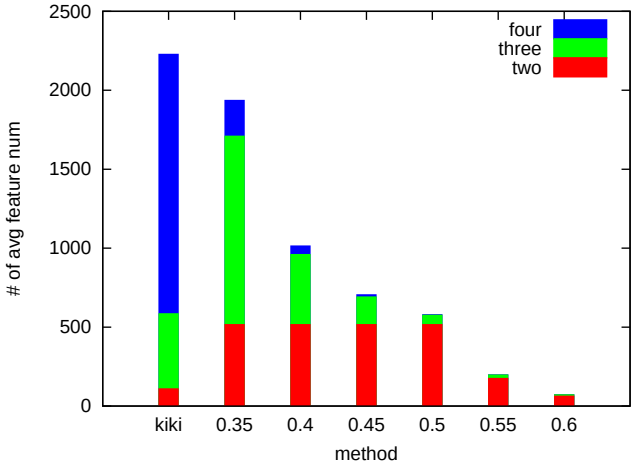
図 4.8: 各閾値での減衰定数と不一致度の関係 (左) と最適値 (右)

4.4　枝刈り閾値と精度

本節では, 提案手法の一つである駒グラフにおけるパス展開の閾値について, 閾値の変化に伴い最適な減衰定数がどのように変化するか, また精度がどのように変化するかの 2 点について調べる. なお, 減衰定数の最適値推定では, 駒グラフに加え利きグラフについても議論する. 利きグラフの精度については 4.5 節において述べる.

4.4.1　減衰定数の最適値推定

まず初めに, 各閾値での減衰定数の最適値の推定を行った. 実験では近似データを用い, 各閾値毎に 0.1 から 1.0 まで 0.1 刻みで 10 通りの減衰定数を用いて学習することで精度の推移を調べた. なお, 前述の通りここでは利きグラフについても減衰定数の最適値推定を行った. 近似データを用いた場合の減衰定数と不一致度の推移と最適値を図 4.8 に示す. ここで, 減衰定数の最適値は不一致度のグラフの極小値を元に推定したが, 閾値が 0.4 のものに関しては極小値となる減衰定数 0.9 と 0.8 での不一致度の値が非常に隣接していたため, 一致率についても比較を行い 0.8 とした. 図 4.8 を見ると, 最適な減衰定数の値は閾値が小さくなるにつれて小さい値に変化していくことが読み取れる. この要因としては, 閾値が下がることによって高次の組み合わせ特徴の割合が増え, 各要素数の組み合わせ特徴でバランスをとる必要が生じるためと考えられる. 各局面評価での平均評価特徴数を調べると図 4.9 のようになる. 図 4.9 を見ると, 実際に閾値が増えることによって 3 次以上の組み合わせ特徴の割合が増加していることがわかり, また, 減衰定数の最適値が 0.8 であった利きグラフにおいても高次組み合わせ特徴の割合が大きいことが分かる. 利きグラフと閾値 0.35 の駒グラフを比較すると, 特徴の総数では利きグラフが勝っているものの, 減衰定数は閾値 0.35 の方が小さくなっている.



手法	2 駒	3 駒	4 駒	計
kiki	112	475	1,643	2,230
0.35	519	1,193	227	1,939
0.4	519	444	53	1,016
0.45	519	175	13	707
0.5	519	59	3	581
0.55	178	20	1	199
0.6	66	8	0	74

図 4.9: 各閾値での平均評価特徴数

これは, (3.5) 式で示したように, 次数 d の組み合わせ特徴の減衰が λ^{d-1} となるため, 4 次よりも 3 次の組み合わせ特徴の割合が多い閾値 0.35 の駒グラフでは減衰定数 λ をより小さくする必要があったためと考えられる.

4.4.2　精度比較

4.4.1 節で得られた減衰定数の最適値を用いて実際に学習を行った. 各閾値での精度を表 4.6 に, 学習曲線を図 4.10 に示す. ここで, 表 4.6 の速度は実行時の探索速度を表し, 単位は nodes/sec (nps) である. 前述のように, 利きグラフの精度については 4.5 節で述べるためここでは割愛する. 表 4.6 及び図 4.10 で特徴的なのは, 閾値 0.5 を境に精度に大きな開きがある点である. これは, 図 4.9 を見ると分かるように, 閾値が 0.5 以下では 2 駒間の組み合わせ特徴をすべて使うようになるためであり, この結果は将棋プログラムで一般的に用いられる 2 駒間の関係が局面評価に対して大きな影響力が

表 4.6: 各閾値での学習結果

閾値	学習時間	速度 (nps)	有効特徴数	一致率 (%)	不一致度
0.35	28h 16m	78,052	13,075,101	40.6234	4.39987
0.4	18h 54m	114,871	6,194,174	40.8342	4.38366
0.45	16h 13m	133,915	3,111,783	40.2368	4.39327
0.5	15h 14m	143,623	1,986,039	40.3001	4.37635
0.55	12h 15m	172,874	415,343	39.0773	5.09832
0.6	11h 17m	187,315	125,478	36.7722	5.93794

あることを示している．一方で, 閾値が 0.5 以下では精度の改善が非常に鈍く, 特に不一致度では図 4.10 のように改善が見られないことが分かる．閾値が小さい場合に精度が伸びない原因の一つとしては, 減衰定数による学習速度の低下が考えられる．減衰定数により, 学習時の更新幅は減衰分だけ減少することとなるため, 学習率を小さくした場合と近い状態になると推測される．また, 図 4.10 の不一致度において, 300 万局面以降の閾値 0.35, 0.4 の学習曲線の傾斜が他に比べ若干急であり, より棋譜数を増やすことで改善がみられる可能性がある．一方で, 提案手法において高次組み合わせ特徴をうまく活用できていない可能性も考えられ, 結合度といった組み合わせ特徴の選別方法の改善が必要であるといえる．

各閾値での相互対戦の結果を表 4.7 に示す．対戦結果の傾向は表 4.6 の精度と相関があり, 精度の悪かった閾値 0.55, 0.6 の 2 つは他の閾値に対して有意に負け越している．一方, 他の閾値については有意な優劣をつけるほどの差が付いていないものの, 表 4.6 の不一致度が他に比べわずかに小さかった閾値 0.4 と 0.5 が勝ち越しを記録している．以降の比較実験では, 精度の結果を考慮して, 閾値として 0.4 を用いることとした．

表 4.7: 各閾値での相互対戦結果

自分 \ 相手	0.35	0.4	0.45	0.5	0.55	0.6
0.35		**x43.0**	50.6	47.2	**o59.4**	**o77.6**
0.4	**o57.0**		53.6	46.4	**o64.6**	**o74.0**
0.45	49.4	46.4		**x44.2**	**o71.0**	**o81.6**
0.5	52.8	53.6	**o55.8**		**o70.0**	**o77.2**
0.55	**x40.6**	**x35.4**	**x29.0**	**x30.0**		**o71.0**
0.6	**x22.4**	**x26.0**	**x18.4**	**x22.8**	**x29.0**	

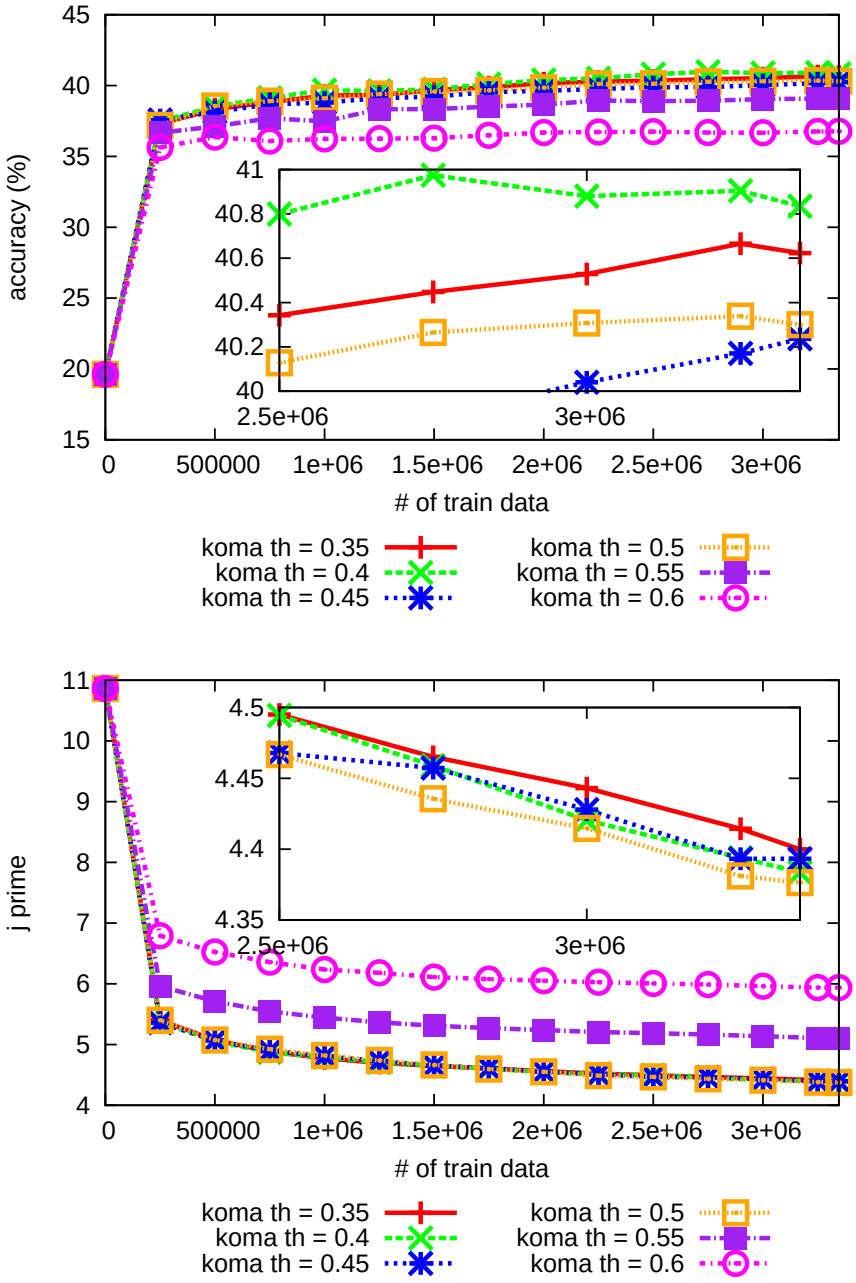


図 4.10: 各閾値での学習曲線 (上：一致率, 下：不一致度)

4.5　既存手法との比較

本節では, 提案手法の有効性を評価するために, 既存の組み合わせ特徴を利用した手法との比較実験を行った. 既存手法としては, 次の 3 つを用意した.

- 2 駒の組み合わせ特徴 (all-two)
- 玉を含む 3 駒の組み合わせ特徴 (kkp-kpp)
- 2 駒の組み合わせ特徴 + 玉を含む 3 駒の組み合わせ特徴 (atkk)

本節の実験では, 既存手法は提案手法と同様に盤面上の駒のみを組み合わせ特徴の要素として用い, 持ち駒は用いないこととした. また, 既存手法についても減衰定数を導入し, 4.4.1 節の方法によって減衰定数の推定を行った. 減衰定数の適用方法は (3.5) 式のように, 2 要素間の組み合わせ特徴には λ の減衰を, 3 要素間の組み合わせ特徴には λ^2 の減衰を与えるものとした. なお, 各手法の平均評価特徴数は表 4.8 のようになる. さらに, 3 要素間の組み合わせ特徴を利用する kkp-kpp, atkk については提案手法と同様にハッシュを利用し, ハッシュサイズは 2^{24} とした.

各手法での学習結果を表 4.9 に, 学習曲線を図 4.11 に示す. 表 4.9, 図 4.11 を見ると, 2 要素の組み合わせ特徴のみを用いる all-two に比べ, 3 要素以上の組み合わせ特徴も併用して用いる $atkk$, $koma$ の 2 つの手法が一致率, 不一致度ともに上回っていることが分かり, 高次組み合わせ特徴を利用することの有効性が伺える. 一方で, 3 要素の組み合わせ特徴のみを用いた kkp-kpp は他の手法に比べて悪い結果となっている. この原因としては, 3 要素というスパースな特徴のみを用いて局面評価を行うため, 学習時に出現しなかった特徴が含まれる局面評価がうまくいっていない可能性が考えられる. また, kiki についても不一致度で他に比べ劣る結果となっている. この原因としては, kiki では利きがついていない要素間の関係を把握することができず, 特に最も抽象度の高い 2 駒間の組み合わせ特徴を捉え切れていないことが精度に影響していると考えられる. これらの結果から, 高次組み合わせ特徴を利用する際には抽象度の高い低次の組み合わせ特徴をうまく活用する必要があるといえる. atkk, koma の 2 つに注目すると, 精度面では同程度となっているものの, 最終的な特徴数は koma が atkk の半分以下にとどまっており, 提案手法によってより少ない特徴で既存手法と同程度の精度が得られていることが分かる. これは, 提案手法での組み合わせ特徴の選別がうまく働いていることを示

表 4.8: 各手法の平均評価特徴数

手法	2 駒	3 駒	4 駒	計
all-two	519	0	0	519
kkp-kpp	0	944	0	944
atkk	519	944	0	1,463
kiki	112	475	1,643	2,230
koma	519	444	53	1,016

す結果である. 他方, 速度では提案手法は既存手法に比べ劣る結果となっている. これは, 提案手法ではグラフの構築, パス展開に大きな計算コストがかかるためで, グラフ関連の処理の改善による速度の向上は今後の課題の一つであるといえる.

各手法での相互対戦結果を表 4.10 に示す. ここで, 対戦結果の順位は, 有意に勝ち越した数と負け越した数の差を基準に決めており, 差が同じ手法については直接対決の勝率を元に順位付けを行った. 対戦結果を見ると, 既存手法については精度と相関のある結果となっているが, 提案手法では kiki, koma については精度に比べやや悪い結果となっており, 特に kiki の勝率が表 4.9 の精度に対して悪いことが分かる. kiki の結果が悪い原因としては, 前述の利きがない関係を取れない性質が関連していると考えられる. 将棋では 1 手の悪手が形勢を大きく変える可能性があり, すべての組み合わせ特徴が利きの有無に左右される kiki は実際の対戦では非常に不利であるといえる. また, 提案手法ではパスの枝刈りのために既存手法に比べ局面毎の精度のばらつきが大きいと推測され, それが精度と対戦結果がかみ合わない原因となっている可能性がある. しかしながら, 提案手法である koma は他の手法に対して有意に勝ち越している atkk と互角の結果を残しており, 既存の各手法に比べ同程度以上の強さを得ていると考えられる. 特徴数で提案手法が勝っている点を考慮すると, 本節の実験結果から, 機械的に組み合わせ特徴の選別を行った提案手法によって, 既存手法の組み合わせ特徴と互角以上の成果を上げることに成功したといえる.

表 4.9: 各手法での学習結果

手法	減衰定数	学習時間	速度 (nps)	有効特徴数	一致率 (%)	不一致度
all-two	1.0	7h 54m	299,871	1,569,828	40.2193	4.51190
kkp-kpp	1.0	14h 55m	181,851	15,607,513	39.0773	4.69772
atkk	0.8	19h 21m	147,143	15,642,643	40.7674	4.38599
kiki	0.8	25h 00m	93,685	16,777,216	40.7428	4.55356
koma	0.8	18h 54m	114,871	6,194,174	40.8342	4.38366

表 4.10: 各手法での相互対戦結果

自分 \ 相手	all-two	kkp-kpp	atkk	kiki	koma	順位
all-two		**o59.6**	**x44.6**	**o70.6**	50.4	3
kkp-kpp	**x40.4**		**x34.4**	54.2	**x36.2**	4
atkk	**o55.4**	**o65.6**		**o72.0**	51.8	1
kiki	**x29.4**	45.8	**x28.0**		**x34.8**	5
koma	49.6	**o63.8**	48.2	**o65.2**		2

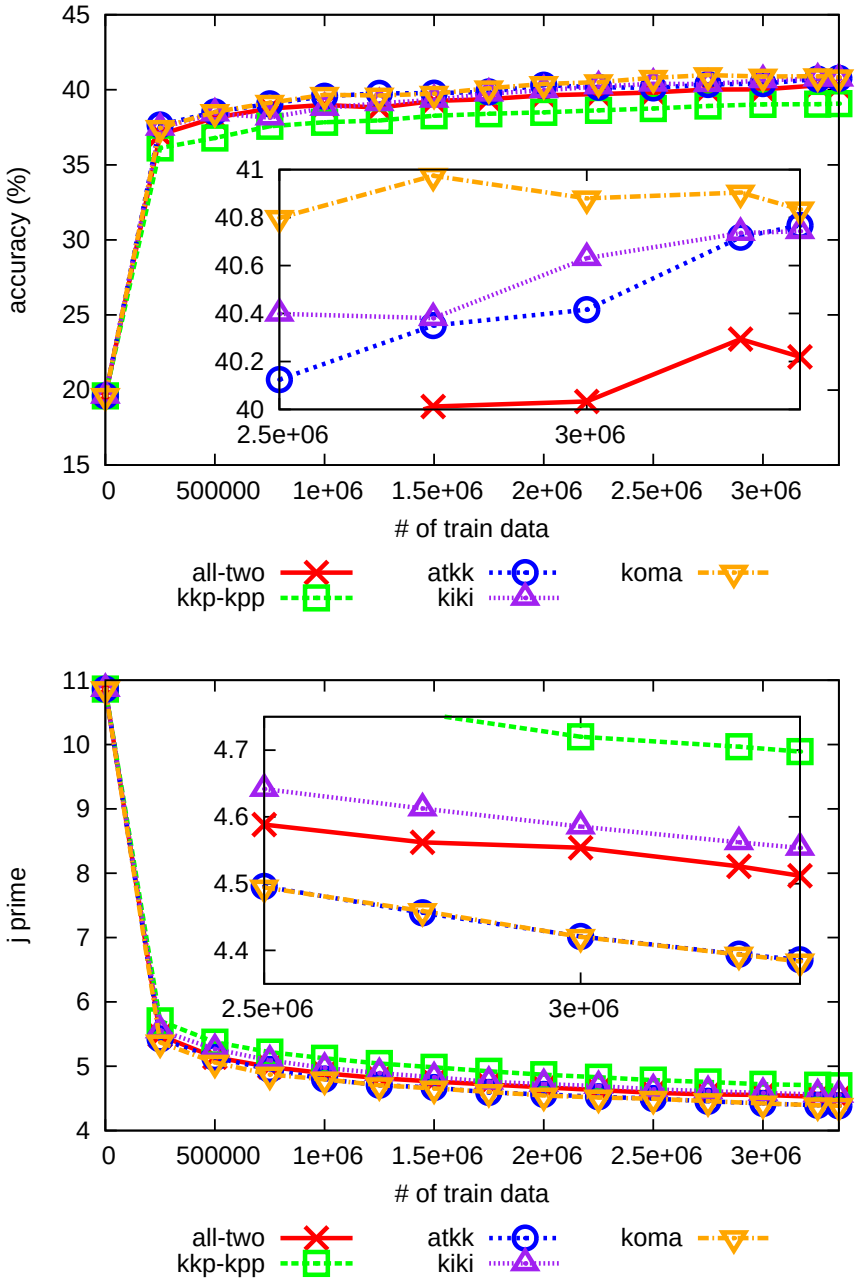


図 4.11: 各手法での学習曲線 (上：一致率, 下：不一致度)

4.5.1　各手法の局面毎の精度のばらつきに関する考察

相互対戦結果でも示したように, 各手法の比較実験では提案手法において精度と勝率がかみ合わない結果となった. 提案手法で精度と勝率がかみ合わない原因の一つとして, 提案手法におけるパスの枝刈りの影響が考えられる. 本節では, 各手法に対して局面毎の精度の分布について調べることで, パスの枝刈りの影響について考察する.

本節では, 具体的に次の 2 つの指標について局面毎の分布を調べた.

- 不一致度
- 合法手の順位付け

このうち, 合法手の順位付けは基準となる順位が必要となるが, 本実験では激指の探索によって得られた合法手の順位を基準として用いた. テストデータとしては学習で用いた棋譜以外から 1,000 棋譜を用意した. 不一致度を指標として用いる場合は, 相手の詰みが確定した局面, 棋譜で指された手が詰んでいる局面の 2 つは結果が極端になるため本実験では対象外とした. 順位付けを指標として用いる場合は, 基準の順位との類似度の指標として NDCG (normalized discounted cumulative gain) [33] を用いた. NDCG とは, 得られた順位の各要素の関連度 (例えば, 検索エンジンにおける検索クエリとの関連の度合) を, 順位に応じて減衰させながら足しこむことで順位付けの良さを計算する指標である. 本実験では, 検索クエリが棋譜中の局面に, 要素列が順位付けされた指し手列に相当する. 検索クエリ q_i から得られた要素列の j 位の要素の関連度を $r_i(j)$ と定義したとき, NDCG は (4.15) 式のように計算される.

$$NDCG_i = \frac{DCG_i}{DCG_{i,max}} \tag{4.15}$$

$$\text{where} \quad DCG_i = \sum_{j=1}^{n} \frac{2^{r_i(j)} - 1}{\log(1+j)}$$

ここで, $DCG_{i,max}$ はクエリ q_i での DCG_i の最大値[5]である. 関連度 $r_i(j)$ は, 激指での評価値を用いて (4.16) 式のように定義した.

$$r_i(j) = R \cdot \exp(-(eval_{max} - eval_j)/100) \tag{4.16}$$

(4.16) 中の R は関連度の最大値を表し, 本実験では $R = 3$ とした.

各指標の局面毎での累積分布を図 4.12 に示す. ここで, 累積分布は各指標で得られる分布を図 4.12 の x 軸の左から右に向かって足していった場合の累積値の変化をプロットしたものである. また, 不一致度の累積分布については全体の累積分布の x 軸を対数としている. 不一致度に注目すると, 不一致度が広範囲に分布していることが分かる. これは, 将棋におけるデータそのもののばらつきが大きいことを表す結果であり, 精度と対戦結果での整合性が欠ける一つの原因であると考えられる. 各手法の結果に注目すると, 非常に小さな差ではあるものの, 表 4.9 において不一致度が相対的に悪かっ

[5]要素列が関連度に対して降順に並んだ場合に最大値となる.

た kkp-kpp, kiki が他の手法に比べ, 累積分布がわずかに x 軸右方向にずれていることが分かり, 不一致度が大きくなる局面が相対的に多かったことが精度低下の原因であったといえる. 一方, all-two, atkk, koma の 3 手法については, 悪手を指す要因となりうる不一致度の大きい領域ではほぼ同等の分布となっており, 精度のばらつきで差異は見られなかった. 不一致度が 1 以下, つまりほとんどの指し手を正確に判別できている領域に注目すると, all-two が他の手法に比べて精度で劣っていることが分かる. これは, 2 要素の組み合わせ特徴しか用いない場合に比べ, 3 要素以上の組み合わせ特徴を用いた手法の方が正しい指し手とその他の指し手をより明確に分類できることを示す結果であるといえる.

NDCG の累積分布を見ると, 不一致度に比べ各手法での差異がはっきりと表れている. 5 手法を比較すると, all-two が最も NDCG が高く, 続いて koma と atkk, kkp-kpp, kiki と続いていることが分かる. 表 4.8 に示した各手法の平均評価特徴数に注目すると, この順位は全体での高次の特徴の比率の高さと相関があるといえる. 高次の特徴に大きな比重をおいて局面評価をする手法では, 特徴が具体的過ぎるために特定の局面に偏った評価関数を構成している可能性があり, その影響で指し手の順位付けにずれが生じたと推測される. 特に, 抽象度の高い 2 駒間の組み合わせ特徴をあまり用いず, 抽象度の低い 4 駒間の組み合わせ特徴を多用する kiki は NDCG が最も悪く, 表 4.10 の対戦結果で述べた不完全な 2 駒間の利用による悪手選択の可能性と合わせて対戦結果が悪かった要因の一つとなっていると考えられる. 一方, koma は不一致度と同様に既存手法の一つである atkk の分布とほぼ一致しており, NDCG においても分布の崩れは見られていない.

以上の結果より, 2 つの指標について以下のことがいえる.

- 不一致度は強さのための必要条件であるが, 不一致度だけでは強さを完全には把握できない.
- NDCG は評価関数のモデルの安定性を見ており, 評価に偏りがないかを示している.

本節の初めに述べた精度と勝率がかみ合わない問題は, 一致率, 不一致度といった精度が全体での平均のみ注目するために, 実際の局面毎の評価での偏りを検知できないことに起因していると推測される. 一方, NDCG では各指し手の相対的な位置関係を見ることで, 局面評価の偏りを調べることができていると考えられる. 表 4.10 の対戦結果と図 4.12 の結果を比較すると, 強い手法の特徴は不一致度が小さく, さらに NDCG の値がある程度大きいことであるといえる. しかしながら, NDCG は精度と異なり全指し手の順位付けが必要であり, この情報は棋譜から得ることができない. 本実験では激指の指し手列を基準として用いたが, 正確な評価を行うためにはより一般的な指し手列の基準が必要である. また, 強さを維持するためにはどの程度まで NDCG の悪化が許容されるのかを知ることも今後の課題といえる.

提案手法について細かく見ていくと, 本節の結果から次のことがいえる.

- koma では分布に大きな崩れはなく, 目立った精度のばらつきは見られない
- kiki では局面評価が高次の組み合わせ特徴に偏るために指し手の順位付けが不安定になり, 対戦結果が悪くなる

提案手法の一つである koma に関しては精度のばらつきが見られなかったものの, atkk と分布がほぼ一致することから提案手法の安定性が示されていると捉えることができる.

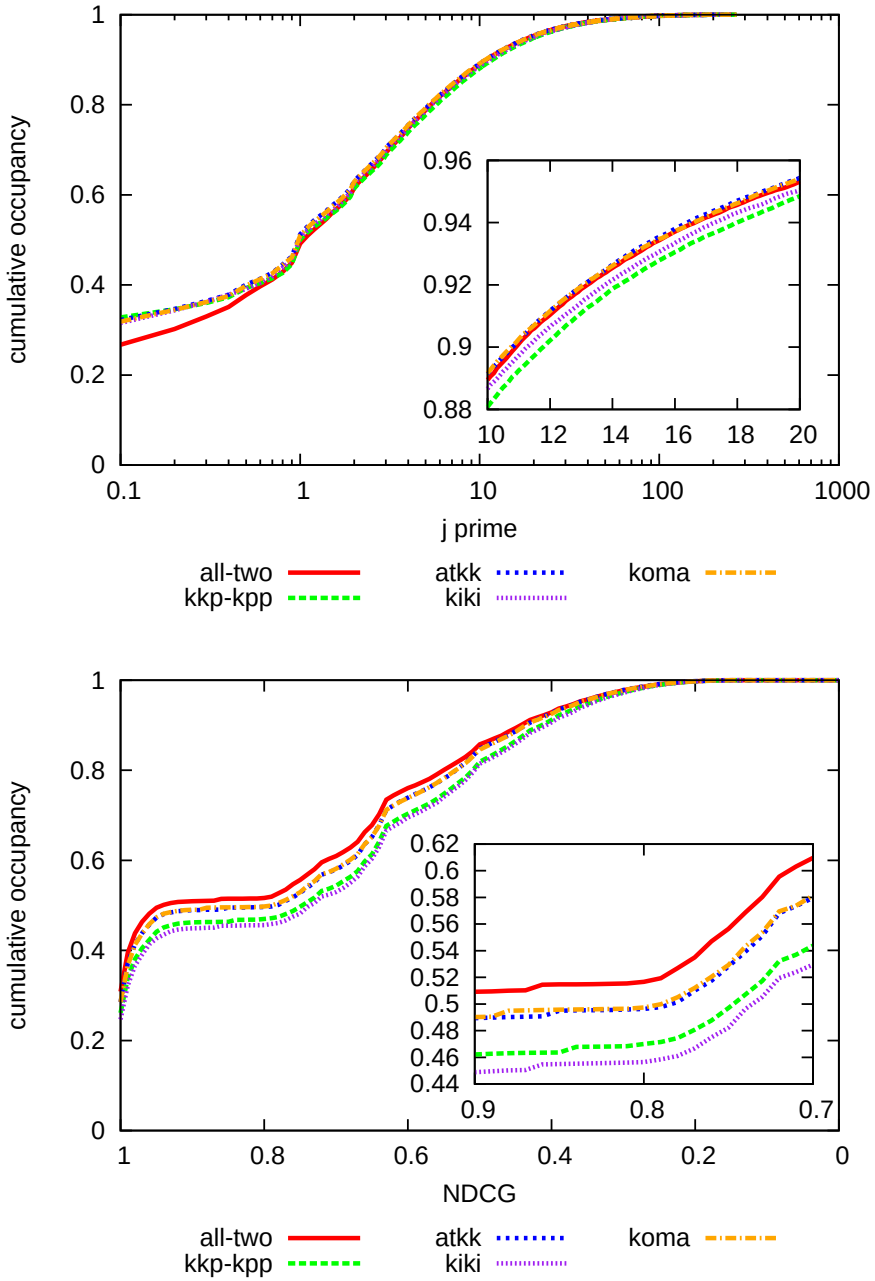


図 4.12: 各指標の局面毎での累積分布 (上：不一致度, 下：NDCG)

4.6　提案手法の拡張

4.6.1　既存評価関数への組み込み

4.1 節でも述べたように, 本実験では将棋プログラムのベースとして「激指」を利用している. 本節では, この激指の評価関数を 4.5 節で述べた各手法と組み合わせた場合の精度について述べる. 激指は現在トップレベルの将棋プログラムであり, 非常に洗練された評価関数をもつ. 激指の評価関数では, 各駒に関する人の知識に基づく評価特徴に加え, 玉や大駒との位置関係などの 2 駒の組み合わせ特徴の一部が利用されている. 3 駒以上の組み合わせ特徴は用いられていないため, 提案手法のような高次の組み合わせ特徴を利用した手法を組み込むことによって精度向上が得られることが期待される.

実験では, 評価関数として (4.17) 式のように激指の評価関数 GEKI($\boldsymbol{w}, s$) に各手法の評価関数を加えたものを用いた.

$$eval(s) = \mathrm{GEKI}(\boldsymbol{w}, s) + f(\boldsymbol{w}, s) \tag{4.17}$$

なお, 駒割を用いた (4.1) 式の場合とは異なり, 激指の評価関数に対応するパラメタも調整の対象とし, 激指の評価関数のパラメタの初期値として激指で用いられているデフォルトのパラメタを利用した. また, 各手法の減衰定数は 4.4 節の方法によって決定した.

激指の評価関数を利用した場合の各手法での学習結果を表 4.11, 学習曲線を図 4.13 に示す. ここで, 表 4.11 及び図 4.13 中の none は激指の評価関数のみを用いて学習した場合の結果を表す. 駒割のみを用いた 4.5 節での結果と比較して, 激指の評価関数を用いた場合には全体的に精度の向上が見られることが分かる. これは, 激指の評価関数を用いたことにより形勢判断の表現力が向上したこと, また図 4.13 を見ると分かるように, 十分に学習された激指のパラメタを初期値として用いたことにより, パラメタ調整の初期から精度が高かったことが影響している. 個々の手法に注目すると, 評価関数を拡張したすべての手法が, 激指の評価関数をそのまま学習した none に対して精度で勝る結果となっていることが分かる. 4.5 節において結果の悪かった kkp-kpp, kiki も大きく精度を伸ばしており, 特に kiki の精度向上は顕著で, 6 つの手法の中で最もよい精度をあげることに成功している. この要因としては, kiki が従来とは異なる利きのつながりに注目した組み合わせ特徴であるため激指の特徴との関連が薄く, そのために最も効果を挙げていると推測される.

激指の評価関数を利用した場合の各手法での相互対戦結果を表 4.12 に示す. 駒割だけを用いた表 4.10 の結果に比べ, 表 4.12 では全体的に勝率の差が小さいことが分かる. これは, ベースとして十分に調整された激指のパラメタを用いたため, 激指の評価関数の精度が対戦に大きく寄与したためと考えられる. それぞれの結果に注目すると, 精度で大きく躍進した kkp-kpp が勝率でも他よりわずかに良い結果を残していることが分かる. 一方で, 他の手法については多少の優劣の差は見られるものの, 激指の評価関数をそのまま学習した none に対してどの手法も有意な差がついておらず, 精度の向上に反して強さでは向上が見られない. kkp-kpp とその他の 4 手法の違いは, 2 駒間の特徴の利用の有無であり, これは激指の評価関数の特徴との重複の有無と見ることができる. 表 4.11 で示した精度では平均をとる過程で重複の影響が相殺されていたものの, 一手一手の精度が直接影響する対戦実験においては重複の影響が結果を大きく作用したと推測される.

表 4.11: 激指の評価関数を利用した場合の各手法での学習結果

手法	減衰定数	学習時間	速度 (nps)	有効特徴数	一致率 (%)	不一致度
none	1.0	6h 27m	282,572	345,865	41.6845	3.55983
all-two	0.7	10h 42m	181,151	1,846,575	43.0057	3.33234
kkp-kpp	0.6	18h 0m	126,755	15,365,009	43.6031	3.29693
atkk	0.6	20h 23m	107,243	15,479,250	43.5855	3.24840
kiki	0.6	26h 10m	70,910	17,123,138	44.9313	3.22037
koma	0.6	21h 22m	86,570	6,046,571	43.7296	3.22968

表 4.12: 激指の評価関数を利用した場合の各手法での相互対戦結果

自分 \ 相手	none	all-two	kkp-kpp	atkk	kiki	koma	順位
none		49.2	50.2	**x44.8**	53.4	48.0	5
all-two	50.8		49.6	50.6	51.4	**o55.2**	2
kkp-kpp	49.8	50.4		**o55.0**	**o54.8**	47.0	1
atkk	**o55.2**	49.4	**x45.0**		49.0	51.2	3
kiki	46.6	48.6	**x45.2**	51.0		48.6	6
koma	52.0	**x44.8**	53.0	48.8	51.4		4

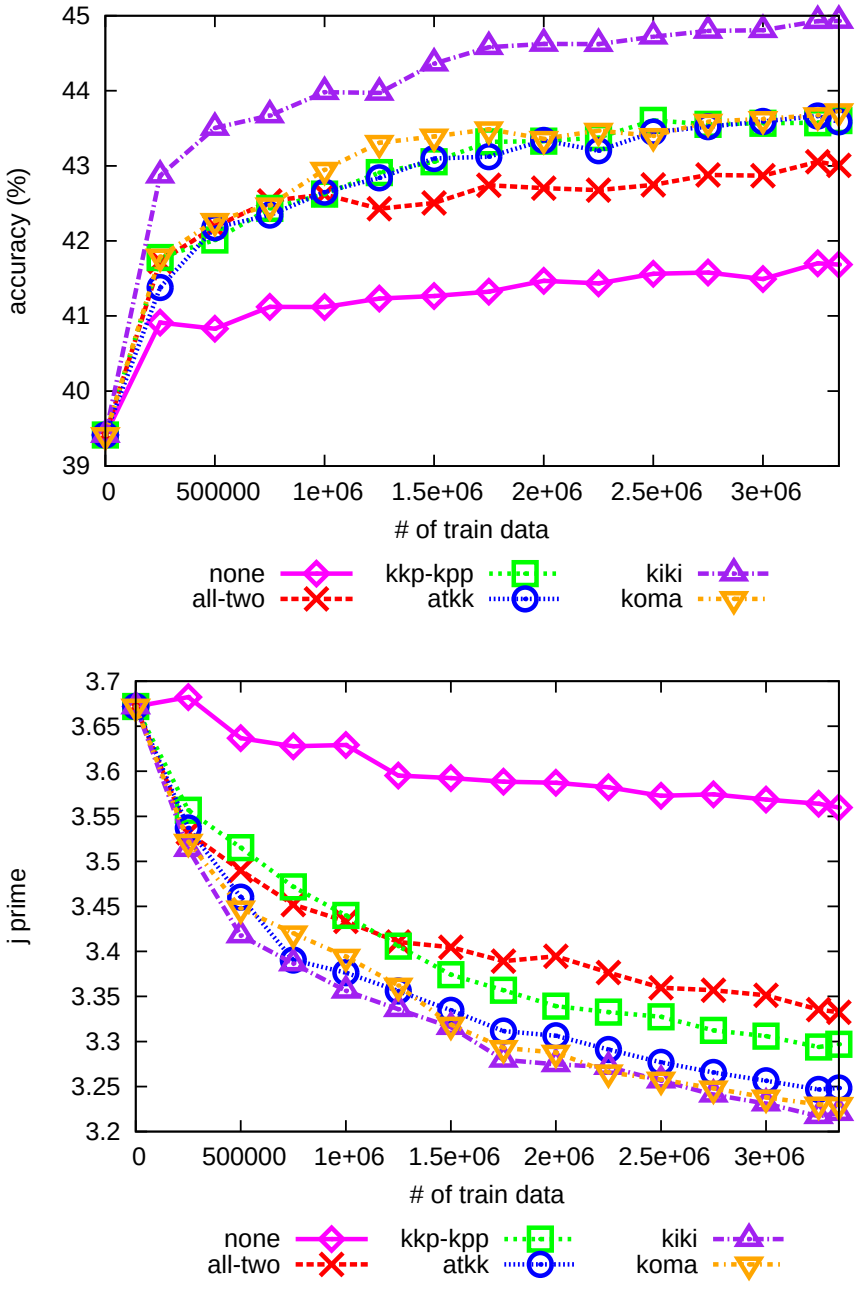


図 4.13: 激指の評価関数を利用した場合の各手法での学習曲線 (上：一致率, 下：不一致度)

4.6.2　持ち駒の利用

将棋において, 持ち駒は将棋のゲームプレイヤの作成を難しくする要素の一つであり, これを特徴として用いることは評価関数の精度向上のために重要であるといえる. 本節では, 4.5 節での比較実験を組み合わせ特徴の要素として持ち駒を追加して行った場合について述べる.

まず初めに, 持ち駒を利用した場合の結合度について述べる. 結合度の学習は 4.2 節で述べた方法を, 2 駒間の組み合わせの要素として持ち駒を追加することで行った. 持ち駒は, 持ち駒の種類, 数の組み合わせを特徴の要素として用いた. なお, 持ち駒の数が 0 個の場合も特徴の一つとして用いることとした. 訓練データの展開には持ち駒を用いない場合と同程度の時間を要し, 得られたデータファイルのサイズは約 13.7GB となった. 学習に LIBLINEAR を利用し, L2 正則化ロジスティック回帰の正則化項を $C = 1$ とした場合の結合度の分布は図 4.14, 表 4.13 のように, 枝刈り閾値を 0.4 とした場合の持ち駒も含めたボナンザのパターンのカバー率は図 4.15 のようになった. なお, 結合度の学習は Intel Xeon X5680 (3.33GHz) の環境で約 146 分の時間を要した. 図 4.14, 表 4.13 を見ると, 結合度の分布は持ち駒を用いずに L2 正則化を用いた場合と同じような形状をしていることが分かる. また, 図 4.15 に関しても持ち駒を用いなかった場合と同様に, パラメタの絶対値が大きい重要なパターンを重点的に展開することに成功していることが分かる. さらに, 4.2.3 節と同様の方法で抽出したパターンのうち, 持ち駒を含むものは図 4.16 のようになった. 図 4.4, 4.5 で示した持ち駒を含まないパターンに比べ, 持ち駒を含むパターンでは重要度の最大パターン, 評価値の最大パターンともに人間から見て意味を見出すのは難しいパターンとなっている. 一方で, 相対的に大きな重みの付いたサブパターンが存在することから, 機械的には意味のあるパターンが含まれている可能性が考えられる.

次に, 各手法に持ち駒を追加した場合の比較実験について述べる. 本節の比較実験では, 4.5 節で用いた手法のうち kiki を除いた 4 手法を比較対象として用いた. kiki を対象から外した理由は, 持ち駒と利きのつながりを定義することができないためである. 実験は各手法の組み合わせ特徴の要素として持ち駒を追加することで行い, 減衰定数については 4.4.1 節の方法により最適値の推定を行った. なお, 提案手法での枝刈り閾値は 0.4 とし, 持ち駒を用いた場合の各手法の平均評価特徴数は表 4.14 のようになった.

持ち駒を用いた場合の各手法の精度を表 4.15 に, 学習曲線を図 4.17 に示す. 表 4.15 を見ると, 持

表 4.13: 持ち駒を利用した場合の結合度の全体に対する割合

$con(e_{i,j})$	pos (%)	neg (%)	total
0.9 - 1.0	0.02	0.03	0.05
0.8 - 0.9	0.05	0.06	0.11
0.7 - 0.8	0.19	0.19	0.38
0.6 - 0.7	2.69	2.65	5.34
0.5 - 0.6	33.29	32.64	65.93
all	36.24	35.57	71.81

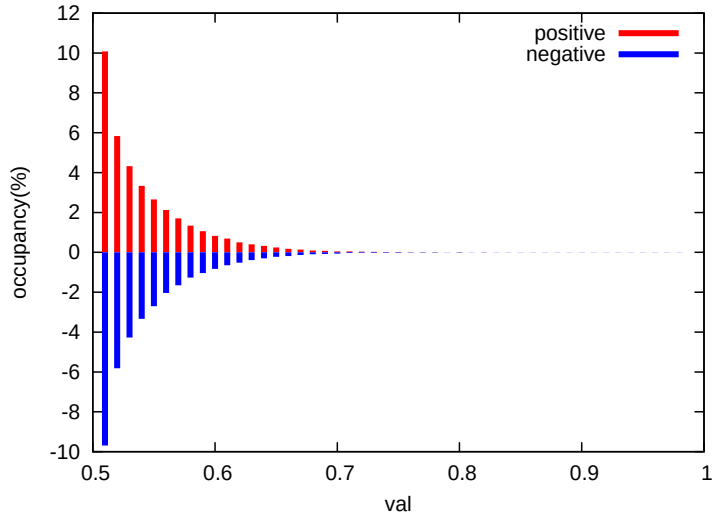


図 4.14: 持ち駒を利用した場合の結合度の分布

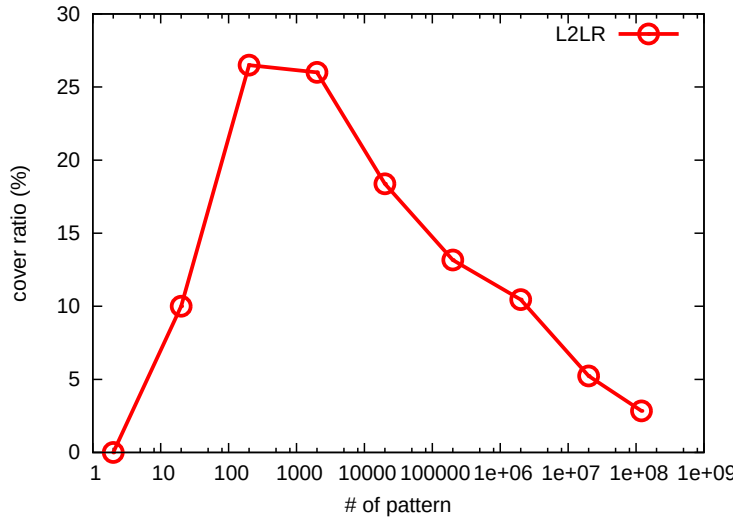


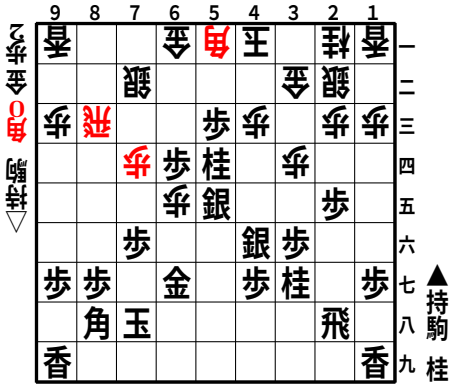
図 4.15: 持ち駒を利用した場合のボナンザのパターンのカバー率

ち駒を用いない場合の精度である表 4.9 に比べ精度の向上が見られ, 特に不一致度は大きく向上していることが分かる. これは, 持ち駒を用いることが局面評価に有効であることを示す結果である. 個々の手法の精度を比較すると, 各手法の優劣の傾向は持ち駒を用いなかった表 4.9 と同じようになっており, 持ち駒を用いた場合についても 4.5 節と同様の考察が成り立つと考えられる. 図 4.17 に注目す

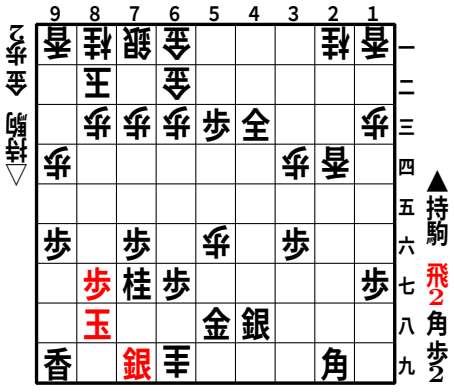
ると, 持ち駒を使わなかった図 4.11 に比べ, atkk が koma に対してわずかに勝っていることが読み取れる. この要因の一つとしては, 持ち駒利用による平均評価特徴数の変化が考えられる. 持ち駒を用いなかった表 4.8 に比べ, 表 4.14 では総計の特徴数の比率こそ変わらないものの, 総数の増加に伴い平均評価特徴数の差は 2 倍程度に広がっており, その差の広がりが atkk の相対的な精度を良くした可能性がある. しかしながら, 提案手法は持ち駒を利用した場合においても, より少ない特徴数で提案手法である koma が既存手法で最も精度の高かった atkk に近い精度を得ることに成功している.

持ち駒を利用した場合の持ち駒なしとの対戦結果を表 4.16 に, 各手法の相互対戦結果を表 4.17 に示す. まず表 4.16 を見ると, すべての手法が持ち駒を用いない場合に対して勝ち越しており, うち 3 要素以上の組み合わせ特徴を利用する 3 手法は有意に強くなっていることが分かる. この結果より, 持ち駒の利用は特に 3 要素以上の高次の組み合わせ特徴に対して有効性が高いといえる. 一方, 各手法の相互対戦結果である表 4.17 に注目すると, 全体的に持ち駒を用いない表 4.10 と似た傾向となっている. しかし, 有意な差まではいかないものの, all-two, atkk, koma の 3 手法の強さの差に開きが生じていることが読み取れる. all-two の勝率が相対的に下がった原因は, 表 4.16 から分かるように, 他の手法に比べて持ち駒を利用した場合の強さの伸びが小さいためである. 一方, atkk の勝率が相対的に上がった原因としては, 表 4.15 で述べた精度の向上が関連していると考えられる.

持ち駒を用いなかった場合に比べ, 持ち駒を用いた場合には既存手法の一つである atkk に対して相対的に差が広がる結果となった. このような結果となった要因としては, 図 4.16 でも示したように提案手法では意味のあるパターンがうまく取れていない点が挙げられる. より意味のあるパターンが取れるように持ち駒を含む結合度の学習を改善することが, 強さ向上のための課題の一つといえる.



サブパターン	重み	減衰
△7四歩, △8三飛	-3.027	0.8
△8三飛, △5一角	-1.926	0.8
△5一角, △角 0	16.811	0.8
△7四歩, △8三飛, △5一角	3.097	0.8^2
△8三飛, △5一角, △角 0	-1.779	0.8^2
△7四歩, △8三飛, △5一角, △角 0	0.839	0.8^3
パターンの重要度	+0.681	
パターンの評価値	10.760	



サブパターン	重み	減衰
▲8七歩, ▲8八玉	12.503	0.8
▲8八玉, ▲7九銀	9.006	0.8
▲7九銀, ▲飛 2	11.247	0.8
▲8七歩, ▲8八玉, ▲7九銀	5.118	0.8^2
▲8八玉, ▲7九銀, ▲飛 2	1.606	0.8^2
▲8七歩, ▲8八玉, ▲7九銀, ▲飛 2	2.850	0.8^3
パターンの重要度	-0.620	
パターンの評価値	31.967	

図 4.16: 持ち駒を含む最大パターン (上：重要度, 下：評価値)

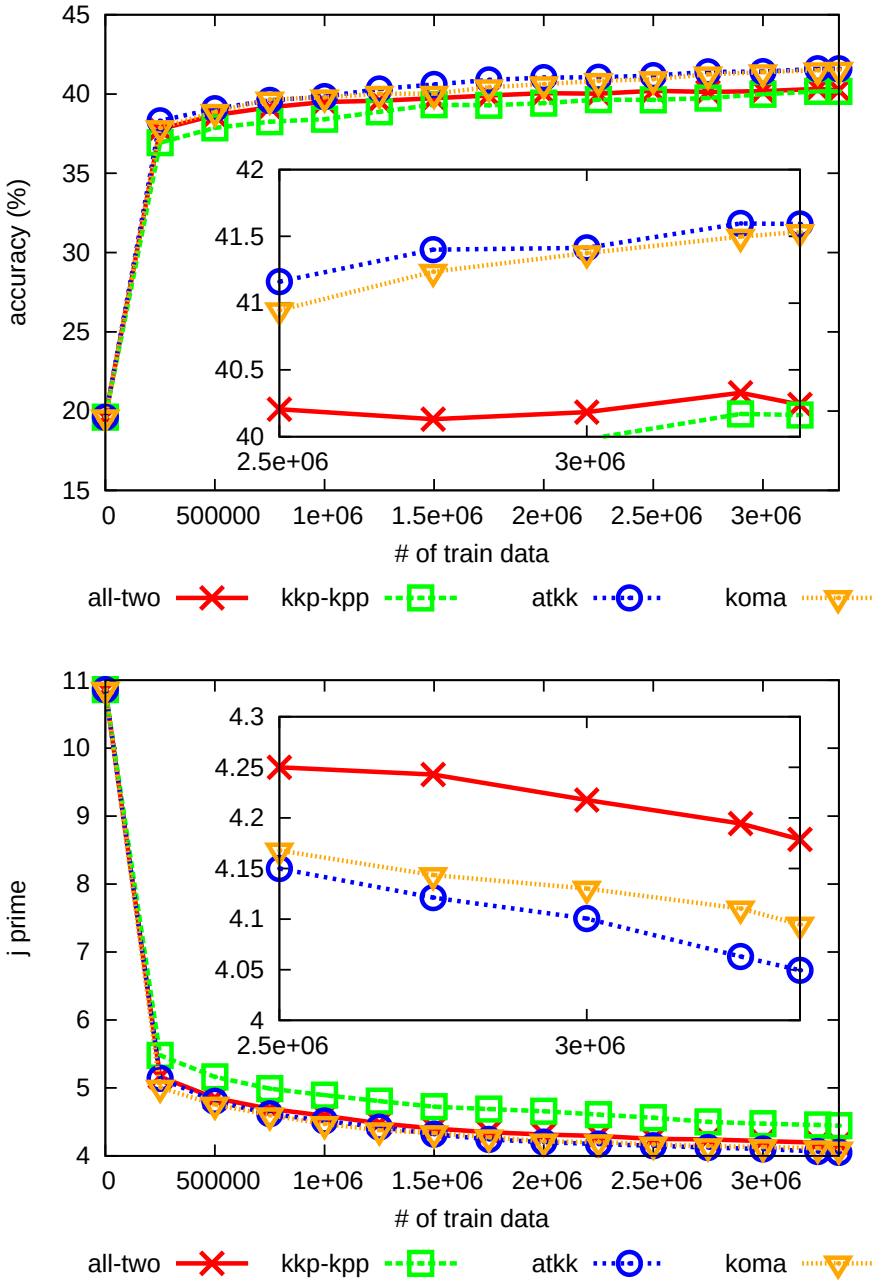


図 4.17: 持ち駒を利用した場合の各手法での学習曲線 (上：一致率, 下：不一致度)

表 4.14: 持ち駒を利用した場合の各手法の平均評価特徴数

手法	2 駒	3 駒	4 駒	計
all-two	1,063	0	0	1,063
kkp-kpp	0	1,992	0	1,992
atkk	1,063	1,992	0	3,055
koma	1,063	830	147	2,040

表 4.15: 持ち駒を利用した場合の各手法での学習結果

手法	減衰定数	学習時間	速度 (nps)	有効特徴数	一致率 (%)	不一致度
all-two	1.0	12h 23m	166,317	1,685,710	40.2403	4.17877
kkp-kpp	0.9	29h 16m	84,293	16,076,917	40.1630	4.44477
atkk	0.7	37h 23m	64,907	16,099,893	41.5932	4.04948
koma	0.8	41h 44m	44.436	9,389,655	41.5334	4.09465

表 4.16: 持ち駒なしに対する勝率

手法	all-two	kkp-kpp	atkk	koma
勝率 (%)	53.8	**o58.0**	**o57.8**	**o56.0**

表 4.17: 持ち駒を利用した場合の各手法での相互対戦結果

自分 \ 相手	all-two	kkp-kpp	atkk	koma	順位
all-two		**o60.6**	45.6	46.8	3
kkp-kpp	**x39.4**		**x37.6**	**x42.4**	4
atkk	54.4	**o62.4**		53.2	1
koma	53.2	**o57.6**	46.8		2

第5章　結論

5.1　まとめ

本研究では, 将棋のコンピュータゲームプレイヤを対象に, 従来は扱うことが困難であった高次の組み合わせ特徴に対し, 3 章で述べたゲーム構成要素間に関連性に基づいて特徴を絞り込むことで高次の組み合わせ特徴を最適化する手法の提案, 高次組み合わせ特徴を効率的に活用する方法としてハッシュの将棋の評価関数への導入, また 4.2 節で述べたゲーム構成要素間の結合度の学習方法の提案を行った.

本論文では, 将棋プログラム「激指」の上に提案手法及び既存手法を実装することで評価を行った. 4 章で述べた実験の結果, 次のような結論を得るに至った.

1. 提案手法による組み合わせ特徴の冗長性の緩和

   4.2.1 節の実験より, 提案手法の結合度を用いることで重要なパターンを重点的に選別でき, 組み合わせ特徴の冗長性が緩和できることを示した. また, 4.5 節の比較実験より, 従来用いられてきた単純な組み合わせ特徴の展開に比べ, 精度を維持しつつ特徴数の削減できており, 実際に冗長な組み合わせ特徴を削減に成功した.

2. 提案手法による組み合わせ爆発の回避

   4.4 節の実験より, 提案手法を用いることで将棋の評価関数で従来用いられてこなかった 4 駒間の組み合わせ特徴を, 枝刈り及びハッシュを利用することで展開することに成功した. また, 4.3 節の実験より, 将棋の評価関数においてもハッシュを用いた特徴数の削減が有効であることを示した.

3. 将棋における高次組み合わせ特徴の評価

   4.2.3 節及び 4.4 節の実験より, 将棋の評価関数では抽象度の高い 2 駒間の組み合わせ特徴が最も評価に影響を及ぼし, 3 駒間以上の組み合わせ特徴はそれを補正する意味合いが強いことを示した. 他方, 本稿の実験では, 提案手法において 3 駒以上の高次組み合わせ特徴の数を増やした場合に, 有意な精度の向上を得るには至らなかった. また, 4.6.2 節の実験より, 持ち駒を用いた組み合わせ特徴は特に 3 駒間以上の組み合わせ特徴で有効性が高いことを示した.

将棋のコンピュータゲームプレイヤにおいて, 全 3 駒間や 4 駒間といった高次の組み合わせ特徴を対象とする評価関数は今までにほとんど用いられていない. 本研究の成果は, これら将棋における高次の組み合わせ特徴の利用の可能性を広げるものであるといえる. また, 提案手法はゲームに限らず

より一般的な組み合わせ特徴にも適用可能である．本実験の結果は要素間のつながりによる組み合わせ特徴の生成手法の有効性を示す結果でもあり, 今後の高次組み合わせ特徴の利用の発展が期待される．

5.2　今後の課題

提案手法の今後の課題としては, 以下のようなことが挙げられる．

1. 実行速度の向上

   4.5 節及び 4.6.2 節の実験結果からも分かるように, 現状の提案手法は実行速度が既存手法に比べ遅い．これは, 盤面のグラフへの変換, 局面評価時のパスの展開の計算コストが大きいためである．この問題の解決策の一つとして, 2.3.2 節で述べた吉永等の手法の適用がある．提案手法では, 総特徴数が少ないことから重要度による枝刈りによって局面評価で出てくる特徴集合に偏りが生じていると考えられる．そのため, これら特徴集合のうち高頻度で出現するものをあらかじめデータベースとして保持し, 展開の省略を行うことで高速化が実現される可能性が高いと考えられる．

2. 結合度学習の改善

   本研究では結合度を単純な 2 駒の関係として学習しているため, かなり荒い関連度を求めているといえる．関連度の精度を上げる方法としては, 駒周辺の局所特徴の利用が考えられる．また, 現状では 2 要素間の関係から高次パターンの重要度を近似しているため, 4.2.3 節や 4.6.2 節で述べたように重要度の高い高次パターンが必ずしも意味を持っているとは言い難く, 今後は複数要素間でも整合性が取れるような結合度の学習を考える必要がある．さらに, 抽出パターンの際に述べたように, 現状の結合度ではパターンを構成する駒の手番と重要度の正例, 負例の対応が直感とは異なるものになっている．そのため, 訓練データの作成方法等についても改良の余地があると考えられる．

3. さらなる要素の追加

   提案手法においてほとんど用いなかったゲーム構成要素として, 空升の情報がある．空升は, 駒の自由度を表現する上で重要な要素であり, これを組み合わせ特徴の要素として加えることで精度の向上が期待される．一方で, 単純に空升を要素として加えた場合, 計算コストが非常に大きくなることが容易に予想できるため, どのようにして利用を絞るかを検討する必要がある．

付 録A　既存手法におけるハッシュと精度

4.3 節において, 提案手法ではハッシュの衝突率が 80% 近くある場合でもある程度の精度を維持することができることを示した. 本節では, 4.5 節でハッシュを用いた kkp-kpp, atkk の 2 つの既存手法について, ハッシュサイズと精度の関係について述べる.

実験では, 4.3 節と同様にハッシュサイズとして 2^{16}, 2^{18}, 2^{20}, 2^{22}, 2^{24}, 2^{26} の 6 種類を用意した. また, ハッシュサイズ以外の kkp-kpp, atkk の条件は 4.5 節と同じ数値を用いた.

各ハッシュサイズでの精度を表 A.1 に示す. 表 A.1 を見ると, kkp-kpp, atkk 共にハッシュサイズが 2^{20} になるまで精度が大きく向上し, それ以上では向上幅が小さくなっていることが分かる. これは, 4.3 節で示した結果と同様に, 衝突率が一定の値を下回ると精度が安定することを示している. 各手法での衝突率と不一致度の関係をプロットすると図 A.1 のようになる. 図 A.1 より, 既存手法においても提案手法の場合と同様に, 衝突率が 80% 程度までは精度をある程度維持できていることが分かり, ハッシュが既存手法に対しても有効に働くことが示されている.

表 A.1: 各ハッシュサイズでの学習結果 (上：kkp-kpp, 下：atkk)

ハッシュサイズ	学習時間	有効特徴数	衝突率 (%)	平均重複数	一致率 (%)	不一致度
2^{26}	16h 2m	32,573,937	14.8057	1.1738	39.0913	4.70749
2^{24}	14h 55m	15,607,513	44.4405	1.7999	39.0773	4.69772
2^{22}	12h 48m	4,194,210	81.1324	5.3001	38.9156	4.75315
2^{20}	11h 23m	1,048,576	95.2589	22.117	38.5748	4.88971
2^{18}	10h 22m	262,144	98.8147	84.369	37.1693	5.33729
2^{16}	10h 14m	65,536	99.7037	337.48	35.5002	6.06320

ハッシュサイズ	学習時間	有効特徴数	衝突率 (%)	平均重複数	一致率 (%)	不一致度
2^{26}	20h 17m	32,573,937	15.5786	1.1845	40.8834	4.37184
2^{24}	19h 21m	15,607,513	46.0874	1.8549	40.7674	4.38599
2^{22}	16h 1m	4,194,210	82.1506	5.6024	40.2368	4.41004
2^{20}	13h 13m	1,048,576	95.5208	22.325	39.8363	4.53761
2^{18}	11h 33m	262,144	98.8802	89.302	39.1405	4.74966
2^{16}	10h 59m	65,536	99.7201	357.21	37.7947	5.30138

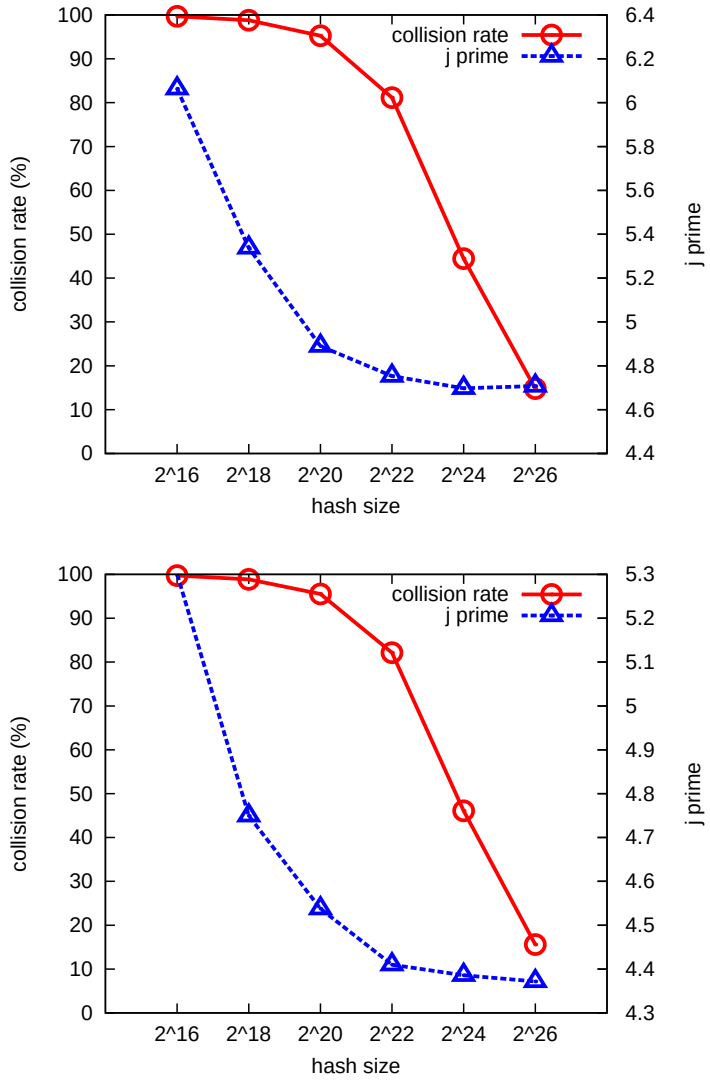


図 A.1: 衝突率と不一致度の関係 (上：kkp-kpp, 下：atkk)

参考文献

[1] J. Gao, G. Andrew, M. Johnson, and K. Toutanova. A comparative study of parameter estimation methods for statistical natural language processing. In *ANNUAL MEETING-ASSOCIATION FOR COMPUTATIONAL LINGUISTICS*, Vol. 45, p. 824, 2007.

[2] Guo-Xun Yuan, Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. A comparison of optimization methods and software for large-scale l1-regularized linear classification. *J. Mach. Learn. Res.*, Vol. 11, pp. 3183–3234, 2010.

[3] Taku Kudo and Yuji Matsumoto. Fast methods for kernel-based text analysis. In *ACL '03*, pp. 24–31. Association for Computational Linguistics, 2003.

[4] 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.

[5] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, Vol. 20, No. 3, pp. 273–297, 1995.

[6] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, Vol. 3, pp. 1157–1182, 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] Bonanza - The Computer Shogi Program. http://www.geocities.jp/bonanza_shogi/.

[9] 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.

[10] Michael Buro. Experiments with multi-probcut and a new high-quality evaluation function for othello. In *Games in AI Research*, pp. 77–96, 1997.

[11] G. Tesauro. Comparison training of chess evaluation functions. *Advances In Computation: Theory And Practice*, pp. 117–130, 2001.

[12] 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.

[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] K. Ganchev and M. Dredze. Small Statistical Models by Random Feature Mixing. In *Workshop on Mobile Language Processing*, p. 19, 2008.

[15] Koby Crammer and Yoram Singer. Ultraconservative online algorithms for multiclass problems. *J. Mach. Learn. Res.*, Vol. 3, pp. 951–991, March 2003.

[16] 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.

[17] David Silver, Richard Sutton, and Martin Müller. Reinforcement learning of local shape in the game of go. In *IJCAI'07: Proceedings of the 20th international joint conference on Artifical intelligence*, pp. 1053–1058, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[18] Simon Perkins, Kevin Lacker, and James Theiler. Grafting: fast, incremental feature selection by gradient descent in function space. *J. Mach. Learn. Res.*, Vol. 3, pp. 1333–1356, 2003.

[19] George Forman. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.*, Vol. 3, pp. 1289–1305, 2003.

[20] Jinbo Bi, Kristin Bennett, Mark Embrechts, Curt Breneman, and Minghu Song. Dimensionality reduction via sparse support vector machines. *J. Mach. Learn. Res.*, Vol. 3, pp. 1229–1243, March 2003.

[21] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. *Data Engineering, International Conference on*, Vol. 0, p. 0215, 2001.

[22] Naoki Yoshinaga and Masaru Kitsuregawa. Polynomial to linear: efficient classification with conjunctive features. In *EMNLP '09*, pp. 1542–1551. Association for Computational Linguistics, 2009.

[23] M. Collins and N. Duffy. Convolution kernels for natural language. *Advances in neural information processing systems*, Vol. 1, pp. 625–632, 2002.

[24] Jun Suzuki and Hideki Isozaki. Sequence and tree kernels with statistical feature mining. In *Advances in Neural Information Processing Systems*, pp. 1321–1328. MIT Press, 2005.

[25] Shinichi Morishita and Jun Sese. Transversing itemset lattices with statistical metric pruning. In *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '00, pp. 226–236, New York, NY, USA, 2000. ACM.

[26] K. Rieck, T. Krueger, U. Brefeld, and K.R. M
"uller. Approximate tree kernels. *The Journal of Machine Learning Research*, Vol. 11, pp. 555–580, 2010.

[27] 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.

[28] 将棋プログラム「激指」のページ.
http://www.logos.ic.i.u-tokyo.ac.jp/ gekisashi/.

[29] A.L. Zobrist. A new hashing method with application for game playing. Technical Report 88, Univ. of Wisconsin, 1970.

[30] 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.

[31] LIBLINEAR – A Library for Large Linear Classification .
http://www.csie.ntu.edu.tw/ cjlin/liblinear/.

[32] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, Vol. 9, pp. 1871–1874, 2008.

[33] Kalervo Järvelin and Jaana Kekäläinen. Ir evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '00, pp. 41–48, New York, NY, USA, 2000. ACM.

発表文献

1. 矢野友貴, 三輪誠, 横山大作, 近山隆. ゲーム構成要素を組み合わせた特徴の最適化. 第 15 回ゲーム・プログラミング ワークショップ, pp 15-22, 2010
2. 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, 2009

謝辞

本研究を進めるにあたって多くの方々にお世話になりました.

近山隆教授ならびに田浦健次朗准教授には, 卒論生として研究室に配属されて以来, 研究や発表に関して数多くのご指導を賜りました. 喜連川・豊田研究室の横山大作助教には, 勉強会などで研究に関するアドバイスをいただき, また激指のコード全般に関してご教授いただきました. 辻井研究室の三輪誠さんには, 研究の方針を決める際に非常にお世話になり, また論文の構成など数多くの点でアドバイスをいただきました. 北陸先端科学技術大学院大学の鶴岡慶雅准教授には, 実験で用いる棋譜を提供していただき, また激指で行われている学習の詳細についてご教授いただきました.

研究室の方々にも感謝しております. 同期の浦晃君には, 激指のコードや研究内容について様々な意見をいただき, 研究の質を高めることができました. 同じく同期の原健太朗君ならびに中島潤君には, プログラムの実行や計算機の使い方など実験環境面で大変お世話になりました. 修士 1 年の加辺友也君ならびに卒論生の鈴木洋平君には, 将棋に関して色々とアドバイスをいただきました. 元卒論生で現山口研究室修士 1 年の山本一成君には, 将棋プログラムに関して様々な知見をいただきました.

ここには書ききれませんが, その他にも多くの方々に支えられこれまでの研究生活を送ることができました. この場をお借りして厚くお礼申し上げます.

最後に, これまでの研究生活を支えてくださった両親に感謝いたします. 本当にありがとうございました.

平成 23 年 2 月 9 日