LinUCBの１人麻雀への適用

中張　遼太郎[1,a)]　水上　直紀[2]　浦　晃[2]　三輪　誠[3]　鶴岡　慶雅[2]　近山　隆[2]

概要：麻雀は広く遊ばれているが, 多人数不完全情報ゲームであるためにコンピュータ麻雀プレイヤに関する研究はあまり進んでいない. 本稿では問題を簡単にするために, 多人数ゲームであるという点を排除した 1 人麻雀に対して Linear UCB を適用し, 人間のプレイヤ及び UCB との比較を行った. いずれの手法も人間のプレイヤには及ばないという結果となったが, UCB の結果との比較から, LinUCB で用いる, 局面の特徴表現の不完全情報ゲームに対する有効性をある程度示すことができた.

Applying LinUCB to Single-Player Mahjong

Ryotaro Nakahari[1,a)]　Naoki Mizukami[2]　Akira Ura[2]　Makoto Miwa[3]　Yoshimasa Tsuruoka[2]
Takashi Chikayama[2]

Abstract: Mahjong is one of the most popular table games in Japan, but there are not many studies on algorithms for mahjong programs because of its complexity – it is a multi-player game with imperfect information. In this paper, we apply the Linear UCB algorithm to single-player mahjong, which is a significantly simplified version of standard mahjong, and evaluate its effectiveness by comparing it with human players and the standard UCB algorithm. In our evaluation, although the mahjong player built with the Linear UCB algorithm was not as strong as human players, the experimental results indicate that the feature-based representation used in the Linear UCB algorithm is probably effective in imperfect information games.

1. はじめに

麻雀は囲碁, 将棋などとは違い, 多人数不完全情報ゲームであるために研究はあまり進んでいない. 麻雀に関する先行研究としては北川らの 3 層ニューラルネットワークを用いた打ち手評価関数の学習 [1] や, 三木らの SVM による打ち手の順位学習 [2], 根本らの CRF を用いた手牌推定 [3] などがあるが人間の上級者に匹敵するプレイヤを実現するには至っていない.

麻雀には多くの難しさがあるため, 様々な要因を一度に扱うことは難しい. そこで, 本研究では問題を限定し, 簡単にするために, 1 人麻雀を対象とする. ここで, 1 人麻雀とは, 1 人のプレイヤが牌を引いて切ることを, プレイヤがあがるか, 牌を一定数引くまで繰り返す麻雀である. 麻雀を難しくしている要因の 1 つである多人数ゲームであるという点を排除したものであるといえる.

1 人麻雀においてある局面からゲーム木を展開しようとした場合, 次にどの牌をツモるかは未知であるため, 無作為に決定しノードを展開していくことになる. これを繰り返し行うと探索空間が大きくなり手の選択が難しくなる. このような場合に木を展開せずに有望な手を選択する手法として, Upper Confidence Bound (UCB) [4] がある. UCB ではある局面から考えられる全ての手に対して, よい結果を返しそうな手を重視しつつ, 何度もプレイアウトと呼ばれるランダムシミュレーションを行うことで, 最善の手を決定する手法である. UCB では牌譜などの学習データを用いて教師あり学習を行うことはできない.

UCB は局面ごとにこの探索を実行するが, 各局面を完全に別の局面として扱うため他の局面の探索において得た情報を利用することができない. そのため, 状態空間が大き

[1] 東京大学工学部電気電子工学科
Engineering Department of Electrical and Electronic Engineering, The University of Tokyo
[2] 東京大学大学院工学系研究科
Graduate School of Engineering, The University of Tokyo
[3] マンチェスター大学コンピュータ科学科
School of Computer Science, The University of Manchester
[a)] nakahari@logos.t.u-tokyo.ac.jp

い麻雀では, 単純な UCB では高い性能を達成することは難しい考えられる. そこで本研究では, 局面を特徴で表すことで, 対象とする局面が異なっても, それまで対象とした局面の情報を利用できる Linear UCB (LinUCB) [5] を適用することで性能向上を図る.

2. 関連研究

2.1 UCB1

UCB1 は式 (1) で計算される UCB1 値が最大となるノードに対してプレイアウトを行い, その結果により UCB1 値を更新するという手順を一定回数繰り返し, 最も平均報酬が高い選択肢を選ぶアルゴリズムである.

$$UCB1 = \overline{x_j} + \alpha\sqrt{\frac{2\log n}{n_j}} \qquad (1)$$

ただし x_j は子ノード j の平均報酬, α は定数, n は親ノードの探索回数, n_j は子ノード j の探索回数である. 式 (1) 右辺の第 1 項は平均報酬を, 第 2 項は信頼度を示している. 信頼度はそのノードのプレイアウト回数が少ないと大きく, 多いと小さくなる. UCB1 値を用いることで UCB1 ではより有望そうな手に対して多くのシミュレーションを行う事ができる. UCB1 は各局面ごとにこの手順を実行するが, 各局面を完全に異なる局面として扱うため, 他の局面の探索において得られた情報を共有することができないという欠点を持つ. この欠点を補うアルゴリズムとして期待されているのが LinUCB である.

2.2 LinUCB

LinUCB [5] は, UCB を局面を特徴で表すことができるように拡張したものであり, 牌譜の局面からの教師あり学習や異なる探索の結果の共有ができる手法である. LinUCB はプレイアウトを行う子ノードを選択する評価値の計算に, 重みベクトル, 特徴ベクトル, 特徴の頻度を表す相関行列を用いる. 計算により求めた評価値が最大となる子ノードに対してプレイアウトを行い, 共通で保持する重みベクトルを更新する. 重みベクトルは, 選択したノードの特徴ベクトルの各項にプレイアウト結果の報酬を乗じた値を, 重みベクトルの対応する項に足し込んで更新する. この更新により, 高い報酬を得た特徴は大きな重みを, 低い報酬の特徴は小さな重みを持つようになるため, 当該ノードだけでなく, 同様の特徴を持つ他のノードの評価値も更新される. これにより異なる局面で得られた情報を共有し, 利用することができる. また LinUCB は探索を行った結果を重みベクトルとして記録しておくことで, 事前学習の結果を用いる探索として利用することも可能である.

LinUCB のアルゴリズムを Algorithm 1 に示す. ただし $\boldsymbol{x}_{t,a_t}$ はプレイアウト回数が t 回目の時の選択肢 a の特徴ベクトル, $\boldsymbol{A}$ は特徴の共起を含めた頻度を表す相関行列, $\boldsymbol{b}$ は

Algorithm 1 Linear UCB (LinUCB) アルゴリズム

1: $\boldsymbol{A} \leftarrow \boldsymbol{I}_d$ ▷ A は d 次元の正方行列
2: $\boldsymbol{b} \leftarrow \boldsymbol{0}_d$ ▷ b は d 次元のベクトル
3: **for** $t = 1, 2, ..., T$ **do**
4: $\quad\boldsymbol{\theta}_t \leftarrow \boldsymbol{A}^{-1}\boldsymbol{b}$
5: $\quad$Observe K features, $\boldsymbol{x}_{t,1}, \boldsymbol{x}_{t,2}, ..., \boldsymbol{x}_{t,K} \in \mathbb{R}^d$
6: $\quad$**for** $a = 1, 2, ..., K$ **do**
7: $\quad\quad p_{t,a} \leftarrow \boldsymbol{\theta}_t^{\mathrm{T}}\boldsymbol{x}_{t,a} + \alpha\sqrt{\boldsymbol{x}_{t,a}^{\mathrm{T}}\boldsymbol{A}^{-1}\boldsymbol{x}_{t,a}}$
8: $\quad$**end for**
9: $\quad$Choose action $a_t = argmax_a\ p_{t,a}$
10: $\quad$Observe payoff $r_t \in \{0, 1\}$
11: $\quad\boldsymbol{A} \leftarrow \boldsymbol{A} + \boldsymbol{x}_{t,a_t}\boldsymbol{x}_{t,a_t}^{\mathrm{T}}$
12: $\quad\boldsymbol{b} \leftarrow \boldsymbol{b} + \boldsymbol{x}_{t,a_t}r_t$
13: **end for**

各ノードごとの報酬の総和を表すベクトル, $\boldsymbol{\theta}_t$ は重みベクトル, $p_{t,a}$ は選択肢 a の評価値, α はバランスパラメータ, r_t は報酬を表している. r_t はプレイアウトにより与えるか, 学習データにより与えるものとする. LinUCB は 4 行目で前回のプレイアウト結果を反映して重みベクトル $\boldsymbol{\theta}_t$ を更新する. 7 行目で重みベクトル, 特徴ベクトル, 相関行列を用いて各ノードの評価値を計算し, 9 行目で評価値が最大となるノードを選択する. 10 行目でプレイアウトを行い報酬を受け取り, 11 行目, 12 行目で $\boldsymbol{A}$ と $\boldsymbol{b}$ の更新を行う.

LinUCB は Algorithm 1 の 7 行目に示したように, 式 (2) により各ノードの評価値を求める. 式 (2) の第 1 項は各ノードの平均報酬を計算しており, 第 2 項で信頼度を計算している.

$$p_{t,a} = \boldsymbol{\theta}_t^{\mathrm{T}}\boldsymbol{x}_{t,a} + \alpha\sqrt{\boldsymbol{x}_{t,a}^{\mathrm{T}}\boldsymbol{A}^{-1}\boldsymbol{x}_{t,a}} \qquad (2)$$

また, UCB で用いている評価値も, 平均報酬と信頼度の和で計算される. つまり LinUCB と UCB は本質的に等しい計算をしていることが分かる. また, LinUCB の第 2 項はデータ数の増加により十分速く小さくなることが保証されている [5]. 以上より, LinUCB は UCB と近い運用を行うことができると考えられる.

3. 提案手法

本稿では不完全情報ゲームである 1 人麻雀へ LinUCB を適用することを提案する. LinUCB は局面を特徴で表すことでそれまでに対象としてきた局面の情報を利用できるため, 他の局面の情報を利用できない UCB よりも良い性能を示すと期待される. また, LinUCB は事前学習の結果を用いることで UCB より少ないプレイアウト回数で同等の性能を示すこともできると考えられる. 本稿では以下に示す 3 種類の LinUCB を提案する.

- **独立 LinUCB**
  通常の LinUCB では特徴ベクトルを用いて局面の抽象化を行う. この LinUCB は UCB1 との比較を行うために, 局面の抽象化を行わないよう, 局面そのものを特徴として持つように設計をした. 局面の抽象化を行わ

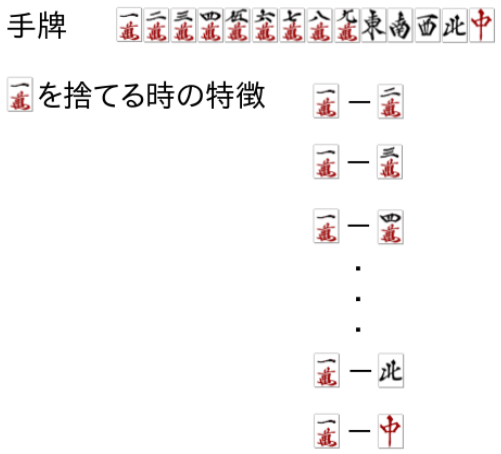

図 1 捨てる牌と残る牌 1 牌の組合せ

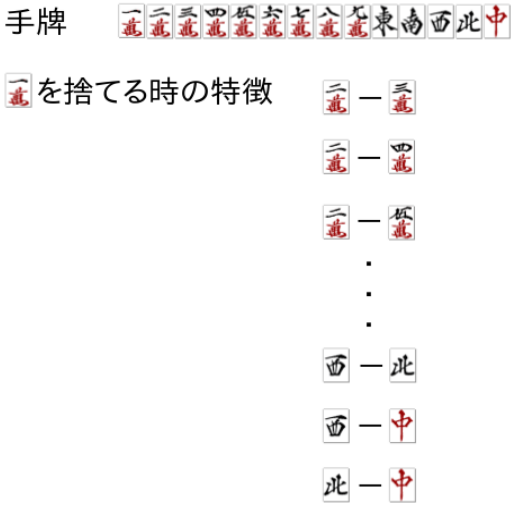

図 2 残る牌の内 2 牌の組合せ

ないため, UCB1 と近い結果になると考えられる.

- **Plain LinUCB**
  以下に示す 3 種類の特徴を合わせて用い, プレイアウトの結果を基に捨て牌の選択を行う.
  - **捨てる牌と残る牌 1 牌の組合せ**
    捨てる牌とその他 13 牌の内 1 牌の組合せを特徴として用いた. 例えば 14 牌の手牌を 1, 2, 3, …, 14 とすると 1 を捨てるという動作の特徴ベクトルは 1-2, 1-3, 1-4, …, 1-14 という 13 個の特徴量を持つことになる. 実際の手牌での特徴量の例を図 1 に示す. 牌は全部で 34 種類あるため特徴空間の次元は 1,156 となる.
  - **残る牌の内 2 牌の組合せ**
    捨てる牌を除いた 13 牌の内 2 牌の組合せを特徴として用いた. 1 を捨てるという動作の特徴ベクトルは 2-3, 2-4, 2-5, …, 12-13, 12-14, 13-14 という 78 個の特徴量を持つことになる. 実際の手牌での特徴量の例を図 2 に示す. 特徴空間の次元は 595 となる.
  - **シャンテン数**
    その捨て牌があがりに近づく牌であるかどうかを特徴として用いた. 麻雀においてあがりまでの距離をシャンテン数と呼ぶ. 特徴空間の次元は 1 となり, シャンテン数が下がる牌は 1 を, そうでない牌は 0 を持つ.
- **事前学習+LinUCB**
  Plain LinUCB で用いた特徴を用い, 牌譜による学習をさせた後の重みベクトルを初期値として, プレイアウトを行い, 捨て牌の選択を行う.

提案手法では報酬と逆行列の計算について変更を加えている. W.Chu らは報酬 r_t を 0 または 1 としている [5] が, 提案手法では-1 または 1 とした. また, 行列 $\boldsymbol{A}$ については対角成分のみを保存している. Algorithm 1 を見ると, このアルゴリズムは $\boldsymbol{A}$ の逆行列を用いている事が分かる. $\boldsymbol{A}$ を n 次正方行列としたとき, $\boldsymbol{A}$ の逆行列を求める計算量は $O(n^3)$ となる. n は特徴空間の次元と一致するため特徴空間が大きくなると計算量が急増する. しかし $\boldsymbol{A}$ の対角成分のみを保存するだけでも精度に大きな影響がないことが経験的に知られており [6], これにより逆行列を $O(n)$ の計算量で求めることができるので大きな特徴空間も扱うことが可能である. また, バランスパラメータ α については適宜実験を行い設定をした.

4. 評価

4.1 実験概要

提案手法で提示した 3 種類の LinUCB を 1 人麻雀に適用し, 性能を評価した. また, 比較対象として UCB1, 事前学習を利用する手法を 1 人麻雀に適用した. それぞれに対して 100 局のテストデータと 10,000 局のテストデータを与え, あがることができた局数を計測した. テストデータとは, 全ての牌をランダムに並べたものを 100 セット用意したデータである. このデータから 13 牌を初期牌として与え, その後牌を引いて切る動作を 27 回行い, その中であがれたかどうかを確認した. 100 局のテストデータで人間のプレイヤとの比較を行い, 10,000 局のテストデータで各手法の性能の評価を行った.

プレイアウトを行う手法についてはプレイアウトの回数は 1000 回とした. プレイアウト時には次に引く牌は分からないため, まだ出現していない牌の中から無作為に選択する. しかし, 捨てる牌については, シャンテン数を小さくする牌があればその中から, なければシャンテン数を変えない牌から無作為に選択するようにした.

学習を行う手法については事前に 1,000,000 局の牌譜を学習させた. 学習に用いた牌譜は, インターネット上の麻雀サイトである天鳳 [7] の上位のプレイヤのみが対局したものを用いた. ここでの上位のプレイヤは麻雀サイト天鳳 [7] において鳳凰卓でプレイすることができるプレイヤのこと

を指す. 鳳凰卓でプレイできるプレイヤは, 全プレイヤの中でも 0.5%程度である. また, 元々の牌譜は 4 人麻雀のものであったため学習に適したものを抽出している. 4 人麻雀は自分以外のプレイヤの動向, 点数などを考慮して最短であがりを目指さなかったり, あがりを放棄したりすることがある. しかし 1 人麻雀では自分以外のプレイヤの要素はなく, 最短でのあがりを目指すため, 4 人麻雀の牌譜をそのまま学習に用いるのは不適切である. 麻雀ではあがりまであと 1 牌という状態でリーチと宣言することができる. 本実験では, 4 人の中で一番早くリーチを宣言したプレイヤは最短であがりを目指したプレイヤであるとして, このようなプレイヤの牌譜のみを抽出し, 1,000,000 局分用意した. 学習は牌譜と同じ牌を捨てるかに応じて報酬を与えることで行った.

4.2 結果

実験結果を表 1, 表 2 に示す. 表 1 より今回適用したいずれの手法も人間の平均プレイヤにも及んでいないという結果となった. 表 2 を見ると UCB1 と近い結果が期待された独立 LinUCB は UCB1 よりも悪い結果となっている. これは UCB1 と LinUCB の評価値計算において信頼度の計算式が異なることが原因であると考えられる. 式 (1) より, UCB1 では信頼度の計算に対数を用いている. 一方で LinUCB では式 (2) より信頼度は線形で変化する. このため LinUCB は UCB に比べて早く信頼度が小さくなる. 信頼度が早く小さくなるので, UCB1 と比べて有望そうな手を早く絞ってしまうことになり, 結果に差が生じたと考えられる. 表 2 では事前学習のみの結果が UCB1 にやや劣る結果となった. しかし, 探索を行わず事前学習のみを用いる手法で探索を行う UCB1 と近い性能が出たということから, 不完全情報ゲームに対して事前学習がある程度有効であるということが分かった. また, Plain LinUCB, 事前学習+LinUCB の結果は UCB1 に劣っている. 原因として特徴量の設計が不適切であったことが考えられる.

表 1 100 局でのあがった局数

手法	あがった局数 (%)
上級者	51
平均プレイヤ	34
UCB1	24
事前学習のみ	25
独立 LinUCB	22
Plain LinUCB	0
事前学習+LinUCB	9

表 2 10,000 局でのあがった局数

手法	あがった局数 (%)
UCB1	31.2
事前学習のみ	27.7
独立 LinUCB	25.5
Plain LinUCB	0.01
事前学習+LinUCB	0.20

5. おわりに

本稿では, 牌譜の局面からの教師あり学習や異なる探索の結果の共有ができる LinUCB を 1 人麻雀に適用することを提案した. LinUCB と, 比較手法として UCB1 と事前学習を利用する手法を 1 人麻雀に適用し, あがった局数を比較することで評価を行った. その結果いずれの手法も人間のプレイヤには及ばず, 3 つの手法では UCB1 が最も良いあがり率を出す結果となった. しかし, UCB1 と事前学習のみを用いる手法の性能が近かったことから, 事前学習が不完全情報ゲームに対してある程度有効であるということが分かった. また, Plain LinUCB, 事前学習+LinUCB の結果から特徴量の設計が不適切であったことが考えられる.

今後は特徴量の見直して実験を行い, LinUCB の性能評価を再度行いたい. 本実験では, LinUCB と事前学習を利用する手法を分けて評価を行ったが, LinUCB で成果が出るようになれば, これらを組み合わせることで更に性能が向上することが期待されるのでこれについても評価を行いたい. また, UCB をモンテカルロ木探索に適用した UCT [8] が成果をあげていることから LinUCB をモンテカルロ木探索へ適用することも検討している.

参考文献

[1] 北川竜平, 三輪誠, 近山隆：麻雀の牌譜からの打ち手評価関数の学習, *Proceedings of 12th Game Programming Workshop*, pp. 76–83 (2007).

[2] 三木理斗, 三輪誠, 近山隆：木カーネルを用いた SVM による麻雀打ち手の順位学習, *Proceedings of 13th Game Programming Workshop*, pp. 60–66 (2008).

[3] 根本佳典, 古宮嘉那子, 小谷善行：CRF を用いた麻雀の不完全情報推定, ゲームプログラミングワークショップ 2012 論文集, Vol. 2012, No. 6, pp. 155–158 (2012).

[4] Auer, P., Cesa-Bianchi, N. and Fischer, P.: Finite-time analysis of the multiarmed bandit problem, *Machine learning*, Vol. 47, No. 2-3, pp. 235–256 (2002).

[5] Li, L., Chu, W., Langford, J. and Schapire, R. E.: A contextual-bandit approach to personalized news article recommendation, *Proceedings of the 19th international conference on World wide web*, ACM, pp. 661–670 (2010).

[6] Crammer, K. and Gentile, C.: Multiclass classification with bandit feedback using adaptive regularization, *Machine Learning*, Vol. 90, No. 3, pp. 347–383 (2013).

[7] 天鳳：. http://tenhou.net/.

[8] Kocsis, L. and Szepesvári, C.: Bandit based monte-carlo planning, *Machine Learning: ECML 2006*, Springer, pp. 282–293 (2006).