情報処理学会論文誌 Vol.57 No.11 2328–2336 (Nov. 2016)

確率的勾配降下法を用いたLinUCTのスケーラビリティの改善

万代 悠作[1,a)] 金子 知適[2]

受付日 2016年2月19日, 採録日 2016年9月6日

概要：LinUCB は腕に割り当てられている特徴ベクトルがその腕の報酬の推定に有用である contextual multi-armed bandit problem のアルゴリズムとして考案された．LinUCB は現在モンテカルロ木探索の標準的手法とされている UCT に用いられている UCB1 と同じく regret の最小化を目的としている．LinUCB をモンテカルロ木探索へ応用した LinUCT と呼ばれるアルゴリズムの人工木における性能が調査され，一定の成果をあげている．LinUCB では特徴ベクトルから報酬への回帰を行う際に必要となるパラメータの更新に，特徴ベクトルの次元の二乗に比例する計算が必要となるが，効果的な結果を得るためには大きな次元を持つ特徴空間を必要とする場合が多い．本研究では，確率的勾配降下法を用いた LinUCB である fLinUCB-GD のモンテカルロ木探索への応用の有用性を検証する．評価実験として，既存の人工木と囲碁において UCT との性能比較を行った．最善手への収束の速さや勝率に統計的に有意な差は認められなかったが，実行速度の面では期待どおりの結果となった．

キーワード：モンテカルロ木探索，multi-armed bandit，contextual bandit，機械学習

Improving Scalability of LinUCT Using Stochastic Gradient Descent

Yusaku Mandai[1,a)] Tomoyuki Kaneko[2]

Received: February 19, 2016, Accepted: September 6, 2016

Abstract: In this paper, we tackle the problem of time complexity of LinUCT algorithm by using a fast stochastic gradient descent method. LinUCT is a variant of Monte Carlo tree search which employs LinUCB for its selection policy and LinUCB is an algorithm for contextual bandit problem where the expected reward of each arm depends on its feature vector. LinUCB computes ridge regression after each trial to improve its prediction formula. Therefore LinUCT has the same time complexity for each node selection. This time complexity becomes problematic when it uses high-dimensional feature vectors. To mitigate the time complexity problem, we propose a new variant of LinUCT, called fLinUCT-GD whose selection policy is fLinUCB-GD which enjoys the linear time complexity. fLinUCB-GD is an algorithm for contextual bandit problem as with LinUCB; however, it computes the prediction formula by means of a stochastic gradient method instead of ridge regression. We evaluated the effectiveness of fLinUCT-GD in artificial trees and the game of Go, particularly its running speed compared to the vanilla LinUCT and UCT algorithms. We could not find the significant differences between UCT and fLinUCT in terms of speed of convergence to optimal moves nor winning rates. Nevertheless, the running speed of fLinUCT-GD achieves to the same degree of UCT; therefore we could expect the promising potential of the algorithm.

Keywords: Monte Carlo tree search, multi-armed bandit, contextual bandit, machine learning

1. はじめに

近年コンピュータゲームプレイヤは目覚ましい進歩を遂げており，そのなかでもコンピュータ囲碁の棋力はモンテカルロ木探索という手法を用いることで飛躍的に向上し

[1] 東京大学大学院総合文化研究科
Graduate School of Arts and Sciences, The University of Tokyo, Meguro, Tokyo 153–8902, Japan
[2] 東京大学大学院情報学環
Interfaculty Initiative in Information Studies, The University of Tokyo, Bunkyo, Tokyo 113–8654, Japan
[a)] mandai@graco.c.u-tokyo.ac.jp

た [1]．現在は multi-armed bandit problem のアルゴリズムである UCB1 やその変種を用いる UCT がモンテカルロ木探索の主流である．UCT は最善手への収束性が証明されており，幅広いゲームで用いられている．

一方，近年のニューラルネットワークの再評価にともない，囲碁において盤面を入力，熟達者が指す確率を出力とするような高性能な評価関数の作成に成功したという報告も存在する．その他のゲームにおいても局面の特徴を用いる品質の高い評価関数の作成が行われており，特徴をうまく用いることによりコンピュータプレイヤの性能を向上させることができる．

また UCT が用いている UCB1 は多腕バンディット問題の最善アルゴリズムとして知られているが，多腕バンディット問題に付加的な情報としてそれぞれの腕に context と呼ばれる特徴ベクトルが備え付けられている，contextual bandit problem という問題が研究されており，特徴ベクトルをうまく利用することによって UCB1 以上の性能を発揮したという報告が存在する．なかでも LinUCB と呼ばれるアルゴリズムは多くの領域で利用されており，ゲーム木探索においても有用なアルゴリズムだと考えられる．

LinUCB とは Li らによって提案された, contextual multi-armed bandit problem において UCB1 以上の性能を示したアルゴリズムである [2]．Contextual multi-armed bandit problem とは各腕に特徴ベクトルが割り当てられており，その特徴ベクトルが報酬の期待値に影響する multi-armed bandit problem である．LinUCB はゲームにおいて不完全情報ゲームである麻雀において応用されている例 [3] があるほか，著者らは UCT の selection-policy において UCB1 の代わりに用いる LinUCT と呼ばれる手法を提案した [4]．

しかし現実的なゲームに LinUCT を適用する際には計算量が問題となる．ゲームにおける特徴ベクトルは大規模なもので数万から数百万の次元まである．しかし LinUCB の計算には特徴ベクトルの次元の二乗の計算量が必要となる．よって適切な特徴空間を設計しようとした際に特徴ベクトルの次元が増やせないといった問題が生じる．そこで本研究では確率的勾配降下法を用いた fLinUCB-GD [5] を用いる LinUCT に応用し，特徴ベクトルの次元が数千次元まで拡大しても現実的な時間で探索をすることができることを示す．

本研究では提案する fLinUCB-GD を用いた LinUCT の性能について，局面の特徴ベクトルを考慮に入れた人工木上で評価を行う．また実在のゲームとして，現在最も研究がさかんな領域の 1 つである囲碁においても調査を行う．囲碁ではゲーム木の探索手法としてモンテカルロ木探索が主流であり，特に UCT の有用性が十分に示されている [1], [6], [7], [8]．UCT との性能比較をすることにより，局面の特徴を用いたモンテカルロ木探索の有用性を検証する．

2. 関連研究

2.1 モンテカルロ木探索

モンテカルロ木探索は局面評価をモンテカルロシミュレーションで行う木探索手法である．ゲーム木探索においては，評価する局面からランダムで着手を行いゲーム終局まで進め，勝敗をもとに局面を評価する．具体的には以下の手順を繰り返す：

(1) 選択：根局面から葉局面まで選択していく．それぞれのノードにおいてはある選択基準（selection policy）によって次に進む局面を決定する．
(2) 展開：葉局面が展開する基準を満たしている場合，次の局面が探索木に加えられる．
(3) シミュレーション：葉局面から終端局面まで進行し，結果（典型的には勝敗）を観測する．局面進行を完全にランダムに行うのではなく，特定の確率分布に従うことでより性能が上がることが示されている [7], [9]．
(4) 伝播：シミュレーションの結果をたどってきた局面に反映する．

与えられた時間経過後に，開始局面の子局面のうち最も良いものを次の着手として採用する．最後の良さの基準としては訪れた回数が最大のものを選ぶという基準が典型的であり，本稿ではこの基準を採用する．

2.2 UCT

UCT（UCB applied to Trees）[10] はモンテカルロ木探索手法の 1 つであり，UCB（Upper Confidence Bound）[11] という基準をモンテカルロ木探索の selection policy としたアルゴリズムである．UCB のうち UCB1 という基準が広く用いられている．t 回目の試行における局面 a の UCB1 は以下の式で与えられる：

$$\mathrm{UCB1}(t,a) = \overline{X}_a + \sqrt{\frac{2\ln T(a)}{t}}.$$

ここで $\overline{X}_a$ は局面 a の勝率，$T(a)$ は局面 a を訪れた回数である．UCT は反復回数を増やすごとに，minimax 探索の結果に近づくという性質がある [10]．

2.3 LinUCB

LinUCB [2], [12] は contextual bandit problem のアルゴリズムである．Contextual bandit problem では，各腕に特徴ベクトルが与えられており，その特徴がその腕を選択した際の報酬の推定に有用である．文献 [2], [12] では腕の特徴ベクトルと腕の報酬の期待値が線形である．つまり時刻 t における腕 a の報酬の期待値は，腕 a の特徴ベクトル $\mathbf{x}_{t,a}$ を用いて以下の式，

$$\mathrm{E}[r_{t,a}|\mathbf{x}_{t,a}] = \mathbf{x}_{t,a}^{\top}\boldsymbol{\theta}_a^*$$

に従うという仮定をしている．$\boldsymbol{\theta}_a^*$ は未知のベクトルで，ア

Algorithm 1 LinUCB

Inputs: $\alpha \in \mathbb{R}_+$
for $t = 1, 2, 3, \ldots$ **do**
  for all $a \in \mathcal{A}_t$ **do** ▷ $\mathcal{A}_t$ is a set of available arms at t
    if a is new **then**
      $\mathbf{A}_a \leftarrow \mathbf{I}_{d\times d}$ ▷ d dimensional identity matrix
      $\mathbf{b}_a \leftarrow \mathbf{0}_{d\times 1}$ ▷ d dimensional zero vector
    $\hat{\boldsymbol{\theta}}_a \leftarrow \mathbf{A}_a^{-1}\mathbf{b}_a$
    $p_a \leftarrow \mathbf{x}_a^\top \hat{\boldsymbol{\theta}}_a + \alpha\sqrt{\mathbf{x}_a^\top \mathbf{A}_a^{-1}\mathbf{x}_a}$
  $a_t \leftarrow \arg\max_{a\in\mathcal{A}_t} p_a$ with ties broken arbitrarily
  Observe a real-valued payoff r_t
  $\mathbf{A}_{a_t} \leftarrow \mathbf{A}_{a_t} + \mathbf{x}_{a_t}\mathbf{x}_{a_t}^\top$
  $\mathbf{b}_{a_t} \leftarrow \mathbf{b}_{a_t} + r_t\mathbf{x}_{a_t}$

ルゴリズムは観察することができない．文献 [2], [12], [13] ではそれぞれ腕の特徴ベクトルは時間に依存するという仮定をおいている．

この仮定のもと，LinUCB は t 回目の試行において $\boldsymbol{\theta}_a^*$ を推定し，期待値の推定値の信頼区間の上端である以下の式，

$$\mathrm{LinUCB}(t,a) = \mathbf{x}_{t,a}^\top \hat{\boldsymbol{\theta}}_a + \alpha\sqrt{\mathbf{x}_{t,a}^\top \mathbf{A}_a^{-1}\mathbf{x}_{t,a}} \tag{1}$$

を評価値として計算し，評価値が最大の腕を次に引くというアルゴリズムである．ここで $\hat{\boldsymbol{\theta}}_a$ は腕 a の係数ベクトルの推定値，α は定数である．$\mathbf{A}_a$ は腕 a における推定の分散共分散行列を表しており，$\mathbf{A}_a = \sum_t \mathbf{x}_{t,a}\mathbf{x}_{t,a}^\top$ で定義される．つまり，腕 a を選択した際に観測した特徴ベクトル自身で直積をとったものの総和となる．LinUCB のアルゴリズムを Algorithm 1 に示す．

これまでの試行の結果から推定した $\hat{\boldsymbol{\theta}}$ を用いた腕の報酬の推定値と，その腕の実際の報酬の期待値との差は定数 α を用いて評価することができ，$\alpha = \sqrt{\ln(2/\delta)/2}$ としたとき確率 $1-\delta$ で

$$|\mathbf{x}_{t,a}^\top \hat{\boldsymbol{\theta}}_a - \mathbf{x}_{t,a}^\top \boldsymbol{\theta}_a^*| \le \alpha\sqrt{\mathbf{x}_{t,a}^\top \mathbf{A}_a \mathbf{x}_{t,a}} \tag{2}$$

の不等式を満たす [14]．

LinUCB はニュース記事の推薦において有効であると確認されており [2]，ゲームにおいても麻雀において LinUCB とモンテカルロ・シミュレーションを組み合わせた先行研究が存在する [3]．

2.4 fLinUCB-GD

前節では特徴ベクトルは時間によって変化し，それぞれの腕に対してそれぞれの $\boldsymbol{\theta}_a^*$ が存在すると仮定したが，以降の節では特徴ベクトルは時間によらず一定で，すべての腕の報酬が単一の $\boldsymbol{\theta}^*$ で計算されると仮定する．

LinUCB では重みベクトル $\hat{\boldsymbol{\theta}}$ の計算のために $\mathbf{A}$ の逆行列の計算が必要となる．d を特徴ベクトルの次元数とした場合，計算量は Sherman-Morrison の公式を用いて $O(d^2)$ [15] となり，次元数が大きい場合には多くの計算時間が必要となる．

他方で，LinUCB における ridge 回帰を確率的勾配降下法（stochastic gradient decent, SGD）で行うことで高速化を施した fLinUCB-GD という手法が提案されている [5]．fLinUCB-GD では $\hat{\boldsymbol{\theta}}$ を近似する際に fRLS-GD（fast Regularised online Least Squares - Gradient Descent）を用いる．fRLS-GD ではデータプールの中からランダムに 1 つデータを選び，そのデータを用いて重みベクトルを更新する，という手法を繰り返して回帰の近似計算を行う．

fLinUCB-GD では，毎時刻 t においてこれまでに観測した特徴ベクトル，報酬の訓練対の集合 $\{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(t-1)}, y^{(t-1)})\}$ の中からランダムに 1 つ選択し，以下の更新式によって現在の $\hat{\boldsymbol{\theta}}$ の予測を更新する：

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \gamma_t((y^{(i_t)} - \boldsymbol{\theta}_{t-1}^\top \mathbf{x}^{(i_t)})\mathbf{x}^{(i_t)} - \lambda_t\boldsymbol{\theta}_{t-1}). \tag{3}$$

ここで特徴ベクトル $\mathbf{x}^{(i_t)}$，報酬 $y^{(i_t)}$ はこれまでに観測した訓練対集合の中からランダムに選択した 1 つの訓練対を表している（$i_t \sim \mathcal{U}\{1,\ldots,t-1\}$，$\mathcal{U}\{a,\ldots,b\}$ は a から b までの要素をとりうる離散一様分布）．γ_t は学習率と呼ばれるパラメータで，時刻 t 時点でどの程度学習をすすめるかを示し，t が無限大のとき $\sum_t \gamma_t = \infty$，$\sum_t \gamma_t^2 < \infty$ となるような数列に設定する．λ_t は ridge 回帰における拘束条件の強さを表し，λ_t が大きいほど重みベクトルの要素が 0 に近づく．

式中 $(y^{(i_t)} - \boldsymbol{\theta}_{t-1}^\top \mathbf{x}^{(i_t)})\mathbf{x}^{(i_t)}$ が勾配ベクトルとなり，重みベクトルをその方向へ近づける．同時に正則化項である $-\lambda_t\boldsymbol{\theta}_{t-1}$ を加えることによって過学習を防いでいる．この式を反復することによって重みベクトル $\boldsymbol{\theta}$ がこれまでの学習データをより説明できるような値となる．

式 (1) の第 2 項にも逆行列の計算が必要となるが，$\mathbf{A}^{-1}\mathbf{x}_{t,a}$ を以下のような $\boldsymbol{\phi}$ で近似することで，逆行列の計算をせずに LinUCB 値の計算を行うことができる：

$$\boldsymbol{\phi}_{t,a} = \boldsymbol{\phi}_{t-1,a} + \gamma_t(\mathbf{x}_a/t - (\boldsymbol{\phi}_{t-1,a}^\top \mathbf{x}^{(i_t)})\mathbf{x}^{(i_t)}) \tag{4}$$

$$\sqrt{\mathbf{x}_{t,a}^\top \mathbf{A}^{-1}\mathbf{x}_{t,a}} \approx \frac{1}{t}\sqrt{\mathbf{x}_{t,a}^\top \boldsymbol{\phi}_{t,a}} \tag{5}$$

なお，文献 [2] では $\mathbf{A}$ を $\sum_i^t \mathbf{x}_i^\top \mathbf{x}_i$ と定義しているのに対し，文献 [15] では $\mathbf{A} = t^{-1}\sum_i^t \mathbf{x}_i^\top \mathbf{x}_i$ と定義している．そのため，文献 [5] と同じスケールとするために本稿では $\boldsymbol{\phi}_{t,a}$ に t^{-1} を乗じている．

式 (3) と式 (4) を用いてそれぞれの腕の LinUCB 値を近似し，最大の腕を選択して報酬を観測する．その際の特徴ベクトルと報酬の訓練対をデータプールへ格納し，次回以降の近似計算で用いる．これらの式の計算量は $O(d)$ であるため，逆行列の計算を必要とする通常の ridge 回帰よりも高速に行うことが可能である．

これまで説明した fLinUCB-GD のアルゴリズムを Algorithm 2 に示す．κ は定数である．

Algorithm 2 fLinUCB-GD

Initialization: Set $\hat{\boldsymbol{\theta}}_0$ arbitrarily, and set γ_k
for $t = 1, 2, \ldots$ **do**
  approximate $\hat{\boldsymbol{\theta}}_t$ using (3)
  for $a \in \mathcal{A}$ **do**
    Estimate confidence parameter $\boldsymbol{\phi}_{t,a}$ using (4)
    $p_a \leftarrow \mathbf{x}_a^\top \boldsymbol{\theta}_t + \frac{\kappa}{t}\sqrt{\mathbf{x}_a^\top \boldsymbol{\phi}_{t,a}}$
  $a_t \leftarrow \arg\max_{a \in \mathcal{A}} p_a$ with ties broken arbitrarily
  Observe a real-valued payoff r_t

2.5 LinUCT

モンテカルロ木探索の selection policy はこれまで様々なアルゴリズムが提案されており，アルゴリズムの違いにより性能は変化する．LinUCT は selection policy に LinUCB を採用したモンテカルロ木探索であり，これまでオセロや特徴ベクトル付きの P-game を模した人工木で性能が調査されている [4], [16].

LinUCB は特徴ベクトル付きの多腕バンディット問題である contextual bandit problem において UCB1 を超える性能を発揮したという報告 [2] があり，特徴ベクトルを利用することでモンテカルロ木探索における選択の性能を高めることができると期待できる．

完全情報ゲームにおける LinUCT では，1) 局面の特徴ベクトルとその局面の報酬は線形，2) すべての局面における報酬の期待値は 1 つの係数ベクトルから計算できる，という仮定をおいている．また完全情報ゲームでは局面の特徴ベクトルは時間に依存しないため，$\mathbf{x}_{t,a} = \mathbf{x}_a$ とする．

文献 [4] における LinUCT では 1 回のプレイアウトで行う係数ベクトルの回帰をプレイアウト中に通った局面の特徴すべてで行っている．このため，式 (1) における第 2 項が早い段階で小さい値となるので，以下の式に変更している：

$$\mathrm{LinUCB}'(a) = \mathbf{x}_a^\top \hat{\boldsymbol{\theta}} + \alpha\sqrt{\frac{T_{\mathrm{update}}}{T(a)}}\sqrt{\mathbf{x}_a^\top \mathbf{A}^{-1}\mathbf{x}_a}. \quad (6)$$

ここで T_{update} は $\hat{\boldsymbol{\theta}}$ を更新した回数であり，これまでのプレイアウトでたどった局面の総和に等しい．本稿でも上記仮定を採用し，式 (6) を用いる．

また，探索中に学習した重みベクトルを複数の局面で利用することにより，より正確な評価を行うことができると期待できるが，本稿では探索ごとに一から学習を行うことを仮定する．

3. fLinUCT-GD

文献 [4] では LinUCT の性能について，特徴ベクトルが葉局面の報酬の期待値となっているような人工木においては UCT を上回る性能を発揮したと報告している．そのため，実在のゲームにおいてもそのような特徴ベクトルをうまく設計できれば LinUCT が有効であるという期待がある．

Algorithm 3 fLinUCT-GD$_{\mathrm{PLAIN}}$

Initialization: Set $\hat{\boldsymbol{\theta}}_0$ arbitrarily
for $t = 1, 2, \ldots, T$ **do**
  approximate $\hat{\boldsymbol{\theta}}_t$ using (3)
  $i \leftarrow 0$
  $path[i] \leftarrow$ root node
  while $path[i]$ is not leaf nor terminal node **do**
    for a in all children of $path[i]$ **do**
      Estimate confidence parameter $\boldsymbol{\phi}_t$ using (4)
      $p_a \leftarrow \mathbf{x}_a^\top \hat{\boldsymbol{\theta}}_t + \frac{\kappa}{t}\sqrt{\frac{T_{\mathrm{update}}}{t}}\sqrt{\mathbf{x}_a^\top \boldsymbol{\phi}_t}$
    $path[i+1] \leftarrow \arg\max_a p_{t,a}$
    $i \leftarrow i + 1$
  if $path[i]$ should be expanded **then**
    Expand $path[i]$
  Simulate rest of the game from $path[i]$
   and observe the reward r_t
  Store the reward r_t and the feature vectors in $path$

しかし LinUCT では LinUCB で必要な $\hat{\boldsymbol{\theta}}$ の更新の際に $O(d^2)$ の計算量が必要となる．ゲームにおける特徴ベクトルの次元は大規模なもので数万 [17] から数百万次元 [18] 程度が主流であり，大きな特徴空間を利用することは前述のような特徴ベクトルを設計する際にも重要であると考えられる．

そこで LinUCT における $\hat{\boldsymbol{\theta}}$ の計算を fLinUCB-GD を用いて行う fLinUCT-GD について本稿では論じ，計算量を削減して数千次元程度の特徴ベクトルを現実的な時間で扱えるような手法について考察する．

文献 [4], [16] において著者らは LinUCB をそのまま適用した LinUCT$_{\mathrm{PLAIN}}$，LinUCB と UCB1 との組合せである LinUCT$_{\mathrm{RAVE}}$ などを提案しており，本稿では fLinUCB-GD を用いた LinUCT$_{\mathrm{PLAIN}}$，LinUCT$_{\mathrm{RAVE}}$ について評価する．

3.1 fLinUCT-GD$_{\mathrm{PLAIN}}$

fLinUCT-GD$_{\mathrm{PLAIN}}$ はモンテカルロ木探索の selection policy に fLinUCB-GD をそのまま適用したアルゴリズムである．fLinUCT-GD$_{\mathrm{PLAIN}}$ のアルゴリズムを Algorithm 3 に示す．

LinUCB を用いる LinUCT$_{\mathrm{PLAIN}}$ が全体の計算量が 1 プレイアウト中の平均訪問局面数を $\tilde{n}$ としたときに $O(T\tilde{n}d^2)$ であるのと比較して，fLinUCT-GD$_{\mathrm{PLAIN}}$ は $O(T\tilde{n}d)$ で行うことが可能である．

3.2 fLinUCT-GD$_{\mathrm{RAVE}}$

文献 [4] では LinUCT$_{\mathrm{PLAIN}}$ よりも優れた手法として，LinUCT$_{\mathrm{RAVE}}$ を提案している．P-game を拡張したそれぞれのノードが特徴ベクトルを持つ人工木において，LinUCT$_{\mathrm{PLAIN}}$ は正しく探索木中の最善手を見つけることができないということが報告されている．これは LinUCB だけでは探索木中の上部のノード選択において下部

のノードの情報を考慮することができないためであると考えられる．

$\mathrm{LinUCT_{RAVE}}$ は LinUCB と UCB1 を組み合わせた評価値をモンテカルロ木探索の selection policy にする [4]．$\mathrm{LinUCT_{RAVE}}$ では RAVE [6] で用いられる，AMAF ヒューリスティクスで計算した評価値と実際のシミュレーション結果で計算した評価値との和が評価値となることを参考に，LinUCB と UCB1 との和を selection policy の基準として用いる．具体的には，$\mathrm{LinUCT_{RAVE}}$ において，selection policy の基準値を以下の式で定める．

$$\begin{aligned}\mathrm{LinUCB_{RAVE}}(t,a) = {} & \beta(t,a)\,\mathrm{LinUCB}(t,a) \\ & + (1-\beta(t,a))\,\mathrm{UCB1}(t,a)\end{aligned}$$

ここで β は LinUCB で予測した勝率と UCB1 が表す実際のシミュレーションで観察した勝率のどちらを優先するかを決定するパラメータであり，

$$\beta(t,a) = \sqrt{\frac{k}{3T(a)+k}}$$

で計算する．k は定数である．

$\mathrm{LinUCT_{RAVE}}$ における LinUCB 値の計算を fLinUCB-GD で行うものが $\mathrm{fLinUCT\text{-}GD_{RAVE}}$ であり，局面評価以外のアルゴリズムは Algorithm 3 と同様である．

4. 実験環境

実験は**表 1** の環境で行った．線形代数計算は Boost.uBLAS [*1] を用いて実装を行った．

5. 人工木における性能

5.1 人工木の概要

本章では文献 [4] で提案されている，局面特徴を扱うようなアルゴリズムの性能を評価するための人工木について概説する．この人工木では，それぞれの内部ノードが d 次元の特徴ベクトルを保持しており，木のすべての葉局面についてその報酬の期待値はある 1 つの係数ベクトル $\boldsymbol{\theta}^*$ とその特徴ベクトルとの内積で計算され，その期待値に従う Bernoulli 試行によって終端局面の勝敗が決定する．特徴ベクトルは根の子局面に対してランダム二値ベクトルを割り当て，それ以外の局面は親の特徴ベクトルを引き継ぐが，ある一定の確率 p で特徴ベクトルの要素が反転するものとする．

この人工木は根の子局面に割り当てた情報が次第に薄れていくという incremental random tree（P-game）[10], [19] に似た性質を持つ．

また木内部の接点の特徴ベクトルはその PV の評価値と関連しておらず，木内部で学習した結果が必ずしも有用であるとは限らないため，LinUCT に特別有利な設定というわけではない．

またこの人工木においては，特徴ベクトルは局面に割り当てられ，着手に対応する差分ベクトルは葉局面の報酬に影響しないため，AMAF ヒューリスティクスを用いる RAVE は有効でないと考えられる．

以降木の分枝数と深さはあらかじめ定めた値で固定されているとする．

表 1 実験環境
Table 1 Hardware and software in experiments.

OS	Debian 3.16.0-4-amd64
CPU	Intel® Xeon™ E5645@2.40 GHz（5 章）
	Intel® Core™ i7-4790K@4.00 GHz（6 章）
メモリ	16 GB（6 章）
	48 GB（5 章）
コンパイラ	g++ 4.9.2
g++ オプション	`-O3 -DNDEBUG -march=native`
Boost	1.55

[*1] http://www.boost.org/doc/libs/1_55_0/libs/numeric/ublas/doc/index.htm

5.2 誤答率と実行速度

各手法の評価を誤答率と実行速度の 2 つの側面から行った．時刻 t において探索を打ち切ったとした場合に選ばれる着手について，それが最善手と異なることを誤答と定義する．時刻 t における誤答率はいくつかの実行結果の平均で算出する．各アルゴリズムの定数は後述の表 4 と同様であるが，モンテカルロ木探索における展開の閾値は 1 としている．

木は (分枝数, 深さ) が $(4,4)$，$(8,4)$，特徴ベクトルの次元が 8，32，128 であるような組合せの 6 種類の木を 100 本ずつ作成した．計測結果は 1 つの木に対して 1 回アルゴリズムを実行してすべての木について平均をとったものである．局面の特徴ベクトルの変異確率 p は 0.1 とした．**表 2**，**表 3** に $t = 10{,}000$ 時点での各アルゴリズムの誤答率を示す．**図 1**，**図 2** は 10,000 プレイアウトするのにかかった時間の平均である．

誤答率については，実験を行った回数が少ないことからどのアルゴリズムも有意に差があるとはいえないが，傾向として同じような性能を発揮することが分かり，また

表 2 各アルゴリズムの $t = 10{,}000$ 時点での誤答率（分枝数 4 深さ 4，() 内は 95%信頼区間の幅）
Table 2 Failure rates with 95% confidence intervals of each algorithm at $t = 10{,}000$ (branching factor 4, depth 4).

特徴ベクトルの次元	8	32	128
$\mathrm{LinUCT_{PLAIN}}$	24% (8.4%)	30% (9.0%)	16% (7.2%)
$\mathrm{LinUCT_{RAVE}}$	11% (6.1%)	25% (8.5%)	16% (7.2%)
$\mathrm{fLinUCT\text{-}GD_{PLAIN}}$	28% (8.8%)	25% (8.5%)	18% (7.5%)
$\mathrm{fLinUCT\text{-}GD_{RAVE}}$	13% (6.6%)	15% (7.0%)	12% (6.4%)
UCT	16% (7.1%)	12% (6.4%)	13% (6.5%)

表 3 各アルゴリズムの $t = 10{,}000$ 時点での誤答率（分枝数 8 深さ 4，() 内は 95%信頼区間の幅）

Table 3 Failure rates with 95% confidence intervals of each algorithm at $t = 10{,}000$ (branching factor 8, depth 4).

特徴ベクトルの次元	8	32	128
$\text{LinUCT}_{\text{PLAIN}}$	41% (9.6%)	34% (9.2%)	24% (8.4%)
$\text{LinUCT}_{\text{RAVE}}$	37% (9.3%)	34% (9.3%)	21% (8.0%)
$\text{fLinUCT-GD}_{\text{PLAIN}}$	38% (9.5%)	30% (9.0%)	24% (8.4%)
$\text{fLinUCT-GD}_{\text{RAVE}}$	24% (8.4%)	24% (8.3%)	22% (8.1%)
UCT	32% (9.1%)	29% (8.9%)	25% (8.5%)

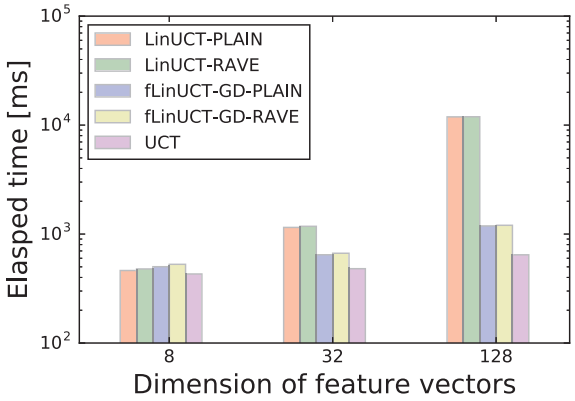

図 1 分枝数 4，深さ 4 の木における特徴ベクトルの次元と実行時間

Fig. 1 Dimension of feature vectors vs. elapsed time of each algorithm (branching factor 4, depth 4).

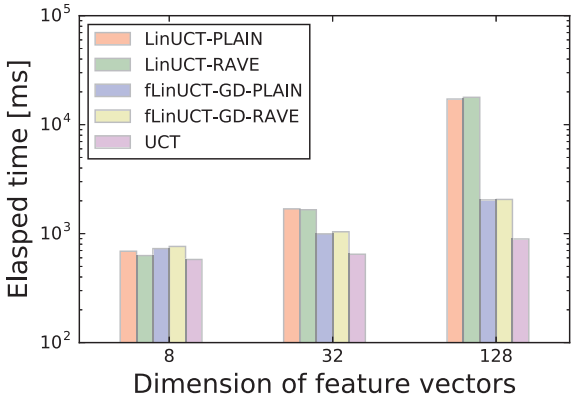

図 2 分枝数 8，深さ 4 の木における特徴ベクトルの次元と実行時間

Fig. 2 Dimension of feature vectors vs. elapsed time of each algorithm (branching factor 8, depth 4).

LinUCT，fLinUCT-GD ともに PLAIN よりも RAVE のほうが性能が良い傾向にある．また特徴ベクトルの次元で性能が異なるのは，次元によって加算される要素の数が異なり，評価値の分布が異なるからであると考えられる．実行時間を考えると，より少ない時間であるにもかかわらず fLinUCT-GD が LinUCT と同程度の結果を残すことが分かり，計算時間の面で有利であることがうかがえる．

6. 囲碁における性能

fLinUCT-GD が UCT と比較してどのような性能となるかを調査するため，対戦実験を行った．本章の実験はすべて 9 路盤で行った．

また本稿で用いる特徴の次元数は 2,000 次元程度としたため，もともとの LinUCB を用いる $\text{LinUCT}_{\text{PLAIN}}$，$\text{LinUCT}_{\text{RAVE}}$ では現実的な時間で探索が終了しない．よって本稿ではこれらは扱わない．

表 4 実験におけるパラメータ

Table 4 Configurations in experiments.

パラメータ名	値	関係するアルゴリズム
γ_t	$\frac{1}{100+t}$ [5]	fLinUCT-GD
λ_t	$\frac{1}{t^{1-\alpha}}$ [5]	fLinUCT-GD
α	0.6 [5]	fLinUCT-GD
κ	1.0	fLinUCT-GD
k	1,000	$\text{fLinUCT-GD}_{\text{RAVE}}$
展開の閾値	10	MCTS

6.1 実装

実装した UCT，$\text{fLinUCT-GD}_{\text{PLAIN}}$，$\text{fLinUCT-GD}_{\text{RAVE}}$ は，通常の囲碁プログラムでの評価に近づけるため progressive widening [20] と特徴を用いたプレイアウト中の行動選択 [7] を用いている．Progressive widening とプレイアウト中の行動選択確率 $\pi(s, a)$ で用いる特徴と重みはオープンソースの囲碁プログラム Libego [*2]が使用している 3×3 の石の配置パターンと，8 近傍点のアタリの情報（呼吸点の数が 1 または > 1）の特徴を使用した．プレイアウト中の行動選択確率は softmax 法で決定している．

各アルゴリズムのパラメータについては**表 4** のとおり設定した．

6.2 特徴

囲碁においては先行研究において様々な特徴が用いられてきた（文献 [21], [22] など）．しかしそれらのほとんどは局面の特徴ではなく着手の特徴であり，勝率の期待値が特徴ベクトルの線形結合で表されるという LinUCB の仮定にそわないと考えられる．そこで今回は既存の研究で示されているような局所的な特徴ではなく，盤面をそのまま表現するような特徴ベクトルの設計を行った．具体的には以下のとおりである．

- 8 近傍の石の組合せ（$81 \times 8 \times 3 = 1{,}944$ 次元）
- 定数（1 次元）

8 近傍の石の組合せとは，8 近傍の座標それぞれについて中心の石と同色/異色/空白もしくは盤外の 3 ビットずつの情報を持ち，9 路盤においては 8 近傍 $\times$ 81 点 $\times$ 3 種類の合計 1,944 次元の特徴を持つ特徴ベクトルである．例として，**図 3** における 8 近傍の石の組合せを表すベクトルは，左上の目から順に $(0,0,1|1,0,0|0,0,0|1,0,0|0,1,0|0,1,0|0,0,0|0,0,0)$

[*2] https://github.com/lukaszlew/libego (accessed 2016-02-19)

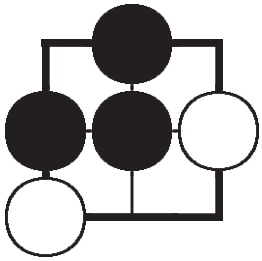

図 3 特徴ベクトルの例：中心の石と同色，異色，空白もしくは盤外の 3 つに各ビットが対応する．この例では $(0,0,1|1,0,0|0,0,0|1,0,0|0,1,0|0,1,0|0,0,0|0,0,0)$ が特徴ベクトルの部分ベクトルになる．9 路盤ではこれが $9 \times 9 = 81$ 個存在する

Fig. 3 Example of feature vector: every 3 bits correspond to same color as the center, different color, and empty/outside. The feature vector for this position is $(0,0,1|1,0,0|0,0,0|1,0,0|0,1,0|0,1,0|0,0,0|0,0,0)$.

表 5 同一プレイアウト回数下での UCT に対する fLinUCT-GD$_{\mathrm{PLAIN}}$ の対戦成績

Table 5 Win rates of fLinUCT-GD$_{\mathrm{PLAIN}}$ against UCT with equal number of playouts.

プレイアウト数	勝率	信頼区間	対戦回数
1,000	0.268	±0.045	100
2,000	0.258	±0.044	100
4,000	0.371	±0.049	100
8,000	0.433	±0.050	100

(|| の間が 1 つの目に対応し，左上の目は空白なので 3 つ目のビットが 1，真ん中上の目は同色なので 1 つ目のビットが 1，...）となる．囲碁の局面の特徴から勝率を予測する試みは深層畳み込みニューラルネットワークを用いた研究において最近大きな成功が報告された [17]．特徴の設計が適切であれば，囲碁においても局面特徴からの勝率予測も有効と考えられる．

6.3 UCT との比較

これまで述べた 2 つのアルゴリズムについての性能評価として，既存手法である UCT との対戦実験を行った．

はじめに，人工木での実験と同様にプレイアウト回数による性能の違いを計測するため，プレイアウト回数を同じに設定したアルゴリズムどうしの対戦を行い，勝率と信頼区間を計測した．次に高速化を施した fLinUCT-GD$_{\mathrm{PLAIN}}$，fLinUCT-GD$_{\mathrm{RAVE}}$ が UCT と比較してどの程度の速さで動作するかを確認した．

6.3.1 プレイアウト回数の比較

fLinUCT-GD$_{\mathrm{PLAIN}}$，fLinUCT-GD$_{\mathrm{RAVE}}$ と UCT とを同一プレイアウト回数の条件のもとで対戦させた結果を**表 5**，**表 6** に示す．

fLinUCT-GD$_{\mathrm{PLAIN}}$ は UCT と比較していずれのプレイアウト回数でも劣るという結果が得られた．しかしプレイアウト回数を増やすことによって勝率が向上するという結果が得られた．fLinUCT-GD$_{\mathrm{RAVE}}$ と UCT との対戦では，いずれのプレイアウト数でも統計的な有意差は見られなかったが，fLinUCT-GD$_{\mathrm{PLAIN}}$ と比較して性能が向上していることが確認できる．

表 6 同一プレイアウト回数下での UCT に対する fLinUCT-GD$_{\mathrm{RAVE}}$ の対戦成績

Table 6 Win rates of fLinUCT-GD$_{\mathrm{RAVE}}$ against UCT with equal number of playouts.

プレイアウト数	勝率	信頼区間	対戦回数
1,000	0.440	±0.068	100
2,000	0.474	±0.046	100
4,000	0.502	±0.049	100
8,000	0.521	±0.051	100

表 7 fLinUCT-GD$_{\mathrm{PLAIN}}$ と UCT との 1 試合あたりの実行時間（秒）の比較

Table 7 Average running time per match of fLinUCT-GD$_{\mathrm{PLAIN}}$ and UCT.

プレイアウト回数	1,000	2,000	4,000	8,000
UCT	14.3	28.7	58.6	116.6
fLinUCT-GD$_{\mathrm{PLAIN}}$	32.5	68.8	148.4	316.9
fLinUCT-GD$_{\mathrm{PLAIN}}$/ UCT	2.27	2.40	2.53	2.78

表 8 fLinUCT-GD$_{\mathrm{RAVE}}$ と UCT との 1 試合あたりの実行時間（秒）の比較

Table 8 Average running time per match of fLinUCT-GD$_{\mathrm{RAVE}}$ and UCT.

プレイアウト回数	1,000	2,000	4,000	8,000
UCT	13.6	28.4	59.4	129.6
fLinUCT-GD$_{\mathrm{RAVE}}$	23.1	46.2	97.1	204.3
fLinUCT-GD$_{\mathrm{RAVE}}$/ UCT	1.70	1.63	1.63	1.57

表 9 初期局面における 1 秒あたりの平均プレイアウト回数の比較（100 回試行の平均）

Table 9 Average number of playouts per seconds from initial position (over 100 trials).

アルゴリズム	プレイアウト回数/秒
UCT	2,246.19
fLinUCT-GD$_{\mathrm{PLAIN}}$	632.82
fLinUCT-GD$_{\mathrm{RAVE}}$	952.15

6.3.2 実行時間の比較

表 7，**表 8** にそれぞれのアルゴリズムの 1 試合あたりの実行時間の比較と，UCT を 1 としたときの fLinUCT-GD$_{\mathrm{PLAIN}}$，fLinUCT-GD$_{\mathrm{RAVE}}$ の実行時間を示す．

UCT と比較して，fLinUCB-GD を用いた fLinUCT-GD$_{\mathrm{PLAIN}}$，fLinUCT-GD$_{\mathrm{RAVE}}$ はおおよそ定数倍のオーダで計算を行うことが可能であることが分かる．fLinUCT-GD$_{\mathrm{PLAIN}}$ と fLinUCT-GD$_{\mathrm{RAVE}}$ を比較すると，fLinUCT-GD$_{\mathrm{PLAIN}}$ のほうが時間がかかっていることが分かる．**表 9** は開始局面から 1 秒間探索を実行した場合の平均プレイア

表 10 同一思考時間（5 秒）における UCT に対する勝率
Table 10 Winning ratio against UCT (5 seconds per turn).

アルゴリズム	勝率	信頼区間	対戦回数
fLinUCT-GD$_{\mathrm{PLAIN}}$	0.173	0.038	100
fLinUCT-GD$_{\mathrm{RAVE}}$	0.270	0.044	100

表 11 同一思考時間（5 秒）における UCT-RAVE に対する勝率
Table 11 Winning ratio against UCT-RAVE (5 seconds per turn).

アルゴリズム	勝率	信頼区間	対戦回数
fLinUCT-GD$_{\mathrm{PLAIN}}$	0.150	0.036	100
fLinUCT-GD$_{\mathrm{RAVE}}$	0.210	0.041	100

ウト回数を示している．この表からも fLinUCT-GD$_{\mathrm{PLAIN}}$ と比較して fLinUCT-GD$_{\mathrm{RAVE}}$ のほうがより多くのプレイアウトを実行することができていることが分かる．これはそれぞれのアルゴリズムの探索木の形状の傾向によってこのような差が生まれると思われるが，詳細な調査は今後の課題である．

6.4 同一思考時間における比較

最後に，同一の条件下での既存手法との比較を行った．fLinUCT-GD$_{\mathrm{PLAIN}}$，fLinUCT-GD$_{\mathrm{RAVE}}$ と UCT，UCT-RAVE とを思考時間を 5 秒に揃えて対戦させた．**表 10**，**表 11** に 2 つのアルゴリズムの結果を示す．

実行時間を同一にした場合は UCT，UCT-RAVE と比較して大きく劣ることが分かる．UCT との実行時間を考えると数倍のプレイアウトの差が存在し，その差を埋めるためより正確な評価が可能となる特徴の設計や，プレイアウト回数を増やせる次元数の少ない特徴の設計などを行う必要がある．

7. おわりに

本稿では確率的勾配降下法を用いた fLinUCB-GD を用いて，計算量を削減した LinUCT を実装し，ある程度大きな特徴空間であっても現実的な時間で探索が行えることを示した．計算時間は UCT の定数倍で行うことが可能であること，性能は ridge regression を行うもともとの LinUCT と遜色ないことを実験で示した．

囲碁においては既存手法である UCT，UCT-RAVE と比較して，本稿の範囲では fLinUCT-GD の優位性を示すことができなかった．また同一計算時間を用いた場合では大きく劣ることが確認できた．今後の課題として，特徴空間の設計などを見直してより良い性能を目指すことが第 1 にあげられる．今回用いた特徴は単純な石の配置だけで，勝敗に結び付いていなかった可能性が高い．

また今回は fLinUCT-GD$_{\mathrm{PLAIN}}$，fLinUCT-GD$_{\mathrm{RAVE}}$ の評価を行ったが，minimax 木の性質を考慮した他のアルゴリズム [4] の評価を囲碁で行い，実際のゲームでの有用性を調査する必要がある．

そのほかにもニューラルネットワークのような手法と組み合わせてより正確な評価を行う，などの方向性が考えられる．

謝辞 この研究の一部は JSPS 科研費 25330432 と 16H02927 の助成を受けています．

参考文献

[1] Gelly, S., Kocsis, L., Schoenauer, M., Sebag, M., Silver, D., Szepesvári, C. and Teytaud, O.: The grand challenge of computer Go: Monte Carlo tree search and extensions, *Comm. ACM*, Vol.55, No.3, pp.106–113 (2012).
[2] Li, L., Chu, W., Langford, J. and Schapire, R.E.: A contextual-bandit approach to personalized news article recommendation, *Proc. 19th International Conference on World Wide Web*, pp.661–670, ACM (2010).
[3] 中張遼太郎，水上直紀，浦　晃，三輪　誠，鶴岡慶雅，近山隆：LinUCB の 1 人麻雀への適用，ゲームプログラミングワークショップ 2013 論文集，pp.114–117 (2013).
[4] Mandai, Y. and Kaneko, T.: LinUCB Applied to Monte-Carlo Tree Search, *Advances in Computer Games – 14th International Conference*, *ACG 2015*, Leiden, The Netherlands, July 1–3, 2015, Revised Selected Papers, Plaat, A., van den Herik, H.J. and Kosters, W.A. (Eds.), Lecture Notes in Computer Science, Vol.9525, pp.41–52, Springer (online), DOI: 10.1007/978-3-319-27992-3_5 (2015).
[5] Korda, N., Prashanth, L.A. and Munos, R.: Fast Gradient Descent for Drifting Least Squares Regression, with Application to Bandits, *Proc. 29th AAAI Conference on Artificial Intelligence*, January 25–30, 2015, Austin, Texas, USA., Bonet, B. and Koenig, S. (Eds.), pp.2708–2714, AAAI Press (2015).
[6] Gelly, S. and Silver, D.: Monte-Carlo tree search and rapid action value estimation in computer Go, *Artificial Intelligence*, Vol.175, No.11, pp.1856–1875 (2011).
[7] Coulom, R.: Computing Elo ratings of move patterns in the game of Go, *ICGA Journal*, Vol.30, No.4, pp.198–208 (2007).
[8] Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S. and Colton, S.: A Survey of Monte Carlo Tree Search Methods, *IEEE Trans. Computational Intelligence and AI in Games*, Vol.4, No.1, pp.1–43 (online), DOI: 10.1109/TCIAIG.2012.2186810 (2012).
[9] Silver, D. and Tesauro, G.: Monte-Carlo simulation balancing, *Proc. 26th Annual ICML*, pp.945–952, ACM (online), DOI: http://doi.acm.org/10.1145/1553374.1553495 (2009).
[10] Kocsis, L. and Szepesvári, C.: Bandit based Monte-Carlo planning, *Machine Learning: ECML 2006*, pp.282–293, Springer (2006).
[11] 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).
[12] Chu, W., Li, L., Reyzin, L. and Schapire, R.E.: Contextual Bandits with Linear Payoff Functions, *Proc. 14th International Conference on Artificial Intelligence and Statistics*, *AISTATS 2011*, Gordon, G.J., Dunson, D.B. and Dudík, M. (Eds.), JMLR Proceedings, Vol.15, pp.208–214, JMLR.org (2011).
[13] Valko, M., Korda, N., Munos, R., Flaounas, I.N. and

Cristianini, N.: Finite-Time Analysis of Kernelised Contextual Bandits, *CoRR*, Vol.abs/1309.6869 (2013).
[14] Walsh, T.J., Szita, I., Diuk, C. and Littman, M.L.: Exploring Compact Reinforcement-learning Representations with Linear Regression, *Proc. 25th Conference on Uncertainty in Artificial Intelligence*, *UAI '09*, Arlington, Virginia, United States, pp.591–598, AUAI Press (2009).
[15] Korda, N., Prashanth, L.A. and Munos, R.: Online gradient descent for least squares regression: Non-asymptotic bounds and application to bandits, *CoRR*, Vol.abs/1307.3176 (2013).
[16] 万代悠作，金子知適：LinUCB のモンテカルロ木探索への応用，ゲームプログラミングワークショップ 2014 論文集，Vol.2014, pp.174–179 (2014).
[17] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T. and Hassabis, D.: Mastering the game of Go with deep neural networks and tree search, *Nature*, Vol.529, pp.484–503 (2016).
[18] Hoki, K. and Kaneko, T.: Large-scale optimization for evaluation functions with minimax search, *Journal of Artificial Intelligence Research*, pp.527–568 (2014).
[19] Smith, S.J. and Nau, D.S.: An analysis of forward pruning, *AAAI*, pp.1386–1391 (1994).
[20] Ikeda, K. and Viennot, S.: Efficiency of Static Knowledge Bias in Monte-Carlo Tree Search, *Computers and Games*, van den Herik, H.J., Iida, H. and Plaat, A. (Eds.), Lecture Notes in Computer Science, Vol.8427, pp.26–38, Springer International Publishing (2014).
[21] Huang, S.-C., Coulom, R. and Lin, S.-S.: Monte-Carlo Simulation Balancing in Practice, *Computers and Games*, van den Herik, H., Iida, H. and Plaat, A. (Eds.), Lecture Notes in Computer Science, Vol.6515, pp.81–92, Springer Berlin Heidelberg (2011).
[22] Graf, T. and Platzner, M.: Adaptive Playouts in Monte Carlo Tree Search with Policy Gradient Reinforcement Learning, *Advances in Computer Games* (2015).

万代 悠作 （正会員）

2014 年松江工業高等専門学校専攻科修了．2016 年東京大学大学院総合文化研究科広域科学専攻修了．修士（学術）．2016 年より東京大学大学院総合文化研究科博士課程在籍．

金子 知適 （正会員）

東京大学大学院総合文化研究科博士課程修了．博士（学術）．同大学院総合文化研究科助手，助教，准教授を経て，2015 年より情報学環准教授．