囲碁における LinUCT の性能評価

万代悠作[1,a)]　金子知適[1]

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期待値に影響する contextual multi-armed bandit problem のアルゴリズムとして考案された. LinUCB は現在多くのモンテカルロ木探索の標準的手法である UCT にて用いられている UCB1 と本質的に同じ計算をしており, モンテカルロ木探索において有用であると考えられる. LinUCB を UCB1 の代わりに用いたモンテカルロ木探索である LinUCT はこれまでオセロや人工木において性能が調査され, 一定の成果を上げている. 本研究では現在ゲーム情報学において主要な調査領域である囲碁において LinUCT の性能を調査し, 囲碁における有用性を検証する. また既存の LinUCB アルゴリズムの計算量は特徴ベクトルの次元の二乗に比例し, 特徴ベクトルの次元数が増大するにつれて現実的な時間で探索を終えることができない. 他方で LinUCB を確率的勾配降下法を用いて近似する fLinUCB-GD という手法が提案されている. 本研究では fLinUCB-GD を用いて計算量の削減を行い, 2000 次元程度の特徴空間で現実的な探索時間をもつ新たな LinUCT を実装した. 対戦実験においては UCT と LinUCT との有意な性能差を認めることはできなかった.

Performances of LinUCT on Game of Go

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

Abstract: LinUCB is a variant of UCB specialized for contextual multi-armed bandit problems, where the expected reward of each arm depends on its feature vector. The LinUCB behaves similarly to that of UCB1, which is employed in the standard MCTS algorithm called UCT. Therefore, LinUCB should be effective in game tree search and it has been applied to Othello and an artificial tree model similar to P-game. In this study, we investigated performances of a family of LinUCT algorithms on the game of Go, which is one of the most challenging domains in game informatics. In addition, to achieve better performance in practice, the LinUCB needs to handle high dimensional feature vectors. However, the computational complexity of the vanilla LinUCB algorithm is proportional to the square of the dimension of its feature space. Therefore, it is intractable to search with LinUCT when the dimension of its feature vector becomes greater. On the other hand, a new LinUCB approximation algorithm called fLinUCB-GD was proposed. The algorithm employs stochastic gradient descent method to compute the regression in LinUCB. We also studied the effectiveness of fLinUCB-GD for LinUCT algorithm and implemented a new LinUCT algorithm which employs fLinUCB-GD. Experimental result showed that it is feasible to search game trees where their nodes are associated with a feature vector whose dimension is approximately 2,000. Finally we compared the LinUCT algorithms with the UCT in the game of Go, however, we could not find significant differences between their performances.

1. はじめに

近年コンピュータゲームプレイヤー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.

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-

[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
[a)] mandai@graco.c.u-tokyo.ac.jp

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, 著者らによって UCT の selection-policy において UCB1 の代わりに用いる LinUCT と呼ばれる手法が提案された [4], [5].

本研究では LinUCT の性能について, 現在最も研究が盛んな領域のひとつである囲碁において調査を行う. 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.

また LinUCB に必要な逆行列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. これは特徴空間の次元を増やそうとしたときに大きな問題となる. そこで本研究では LinUCB の計算に確率的勾配降下法を用いた fLinUCB-GD [9] を用いる LinUCT を実装し, 特徴ベクトルの次元が数千次元の場合でも現実的な時間で探索をすることができることを示す.

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局面を進行していき, ゲーム終局の勝敗を観測し, その観測結果を元に局面を評価する. 具体的には,

(1) 選択: 開始局面から葉局面まで選択していく. それぞれの内部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], [10].
(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.

という手順を繰り返し, 開始局面の子局面のうち最も良いものを次の選択として採用する. 最後の選択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仮定する.

2.2 UCT

UCT (UCB applied to Trees) [11] はモンテカルロ木探索の一つであり, UCB (Upper Confidence Bound) [12] 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:

$$\mathrm{UCB1}(t,a) = \overline{X}_a + \sqrt{\frac{2\ln T(a)}{t}}. \tag{1}$$

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 [11].

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
  $\mathbf{A}_{a_t} \leftarrow \mathbf{A}_{a_t} + \mathbf{x}_{a_t}\mathbf{x}_{a_t}^\top$
  $\mathbf{b}_{a_t} \leftarrow \mathbf{b}_{a_t} + r_t\mathbf{x}_{a_t}$

2.3 LinUCB

LinUCB は各腕と腕を引くユーザに特徴ベクトルが与えられており, その特徴が行動の期待値に影響する contextual multi-armed bandit problem のアルゴリズムとして提案された [2], [13].

LinUCB では各腕の報酬の期待値が

$$\mathrm{E}[r_{t,a}|\mathbf{x}_{t,a}] = \mathbf{x}_{t,a}^\top \boldsymbol{\theta}_a^* \tag{2}$$

に従うという仮定をしている. ここで $r_{t,a}$ と $\mathbf{x}_{t,a}$ は t 回目の試行における腕 a の報酬と特徴ベクトル, $\boldsymbol{\theta}_a^*$ は係数ベクトルで, アルゴリズムは観察することができない. 文献 [2] では各試行ごとに異なるユーザが腕を選択するという状況をモデル化したため, 腕の特徴ベクトルは t にも依存する. この仮定のもと, 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,

$$\mathrm{LinUCB}(t,a) = \mathbf{x}_{t,a}^\top \hat{\boldsymbol{\theta}}_a + \alpha\sqrt{\mathbf{x}_{t,a}^\top \mathbf{A}_a^{-1}\mathbf{x}_{t,a}} \tag{3}$$

を評価値として計算し, 評価値が最高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 の係数ベクトルの予測値, α は定数, $\mathbf{A}_a = \sum_{i=1}^{t} \mathbf{x}_{i,a}\mathbf{x}_{i,a}^\top + \mathbf{I}$ である.

2.3.1 fLinUCB-GD

これ以降の章では簡便のため, すべての腕の報酬の期待値が同じ $\boldsymbol{\theta}^*$ で計算でき, 腕の集合は時間によって変化しないと仮定する.

LinUCB では値の計算のために $\mathbf{A}$ の逆行列の計算が必要となるが, その計算量は逐次計算の場合 Strassen のアルゴリズムを用いたとき $O(d^{2.807})$ であり, Sherman-Morrison

Algorithm 2 fLinUCB-GD

Initialization: Set $\hat{\boldsymbol{\theta}}_0$, γ_k
for $t = 1, 2, ...$ **do**
  approximate $\hat{\boldsymbol{\theta}}_t$ using $\boldsymbol{\theta}_t$ (4)
  for $a \in \mathcal{A}$ **do**
    Estimate confidence parameter $\boldsymbol{\phi}_{t,a}$ using (5)
    $p_a \leftarrow \boldsymbol{\theta}_t^\top \mathbf{x}_a + \kappa\sqrt{\mathbf{x}_a^\top \boldsymbol{\phi}_{t,a}}$
  $a_t \leftarrow \arg\max_{a \in \mathcal{A}} p_a$ with ties broken arbitrarily
  Observe a real-valued payoff r_t

の公式を用いたとき $O(d^2)$ となる．ここで d は行列の次元である．他方で, ridge 回帰を確率的勾配降下法 (SGD; stochastic gradient decent) で行う fRLS-GD (fast Regularised online Least Squares - Gradient Descent) で $\hat{\boldsymbol{\theta}}$ を近似する手法があり, 以下の更新式

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \gamma_t((y^{(i_t)} - \boldsymbol{\theta}_t^\top \mathbf{x}^{(i_t)})\mathbf{x}^{(i_t)} - \lambda_t \boldsymbol{\theta}_{t-1}) \quad (4)$$

を用いることで計算量 $O(d)$ で行うことができる．ここで γ_t はステップサイズであり, $t \to \infty \Rightarrow \sum_t \gamma_t = \infty, \sum_t \gamma_t^2 < \infty$ を満たす．λ_t は複雑度パラメータである．$(\mathbf{x}^{(t)}, y^{(t)})$ は t 回目の試行において選択した腕の特徴ベクトルと報酬を表しており, $i_t \sim \mathcal{U}(1, .., t-1)$ である．

この fRLS-GD を用いて LinUCB 値を計算する fLinUCB-GD という手法が提案されている [9]．LinUCB を計算するためには, $\boldsymbol{\theta}_t$ を用いるとともに以下の更新式

$$\boldsymbol{\phi}_{t,a} = \boldsymbol{\phi}_{t-1,a} + \gamma_t((\mathbf{x}_a/t - (\phi_{t-1,a}^\top \mathbf{x}^{(i_t)})\mathbf{x}^{(i_t)}) \quad (5)$$

用いて $\boldsymbol{\phi}_{t,a}$ を計算したのち, 以下の式

$$\mathrm{LinUCB}(t,a) = \mathbf{x}_{t,a}^\top \boldsymbol{\theta}_t + \kappa\sqrt{\mathbf{x}_{t,a}^\top \boldsymbol{\phi}_{t,a}} \quad (6)$$

で LinUCB を近似できる．κ は定数である．

fLinUCB-GD のアルゴリズムを 2 に示す．注釈として文献 [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 [9] では $\mathbf{A} = t^{-1}\sum_i^t \mathbf{x}_i^\top \mathbf{x}_i$ と定義している．本稿では [2] と値を近づけるため $\boldsymbol{\phi}_{t,a}$ に t^{-1} を乗じている．

3. 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 [5]．本稿では逆行列を直接計算する LinUCB ではなく, SGD を用いて回帰を行う fLinUCB-GD を中心に考える．また [2] ではそれぞれの腕が独立な係数ベクトルによって勝率の期待値が定まると仮定していたが, 本稿ではすべての腕 (すなわち合法手) の勝率の期待値が一つの係数ベクトルに従うと仮定する．LinUCB のみでモンテカルロ木探索の selection policy を構成するモンテカルロ木探索を本稿では $\mathrm{LinUCT_{PLAIN}}$ と呼ぶ．

$\mathrm{LinUCT_{PLAIN}}$ のアルゴリズムを Algorithm 3 に示す．

Algorithm 3 LinUCT (fLinUCB-GD)

for $t = 0, 1, ..., T$ **do**
  approximate $\hat{\boldsymbol{\theta}}_t$ using $\boldsymbol{\theta}_t$ (4)
  $i \leftarrow 0$
  $path[i] \leftarrow$ 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 (5)
      $p_a \leftarrow \boldsymbol{\theta}_t^\top \mathbf{x}_a + \kappa\sqrt{\mathbf{x}_a^\top \boldsymbol{\phi}_{t,a}}$
    $path[i+1] \leftarrow \arg\max_a p_{t,a}$
    $i \leftarrow i + 1$
  if $path[i]$ should be expanded **then**
    Expand $path[i]$
  Simulate the rest of game from $path[i]$ and observe the reward r_t

3.1 $\mathrm{LinUCT_{RAVE}}$

[5] では P-game を拡張したそれぞれのノードが特徴ベクトルをもつ人工木において, $\mathrm{LinUCT_{PLAIN}}$ 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 minimax 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, 性能が落ちると思われる．

そこで 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 $\mathrm{LinUCT_{RAVE}}$ という手法が提案されている [4], [5]．$\mathrm{LinUCT_{RAVE}}$ では RAVE で用いられる, AMAF ヒューリスティック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．

4. 実験

4.1 実験環境

実験は表 1 の環境で行った．線形代数計算の実装には

Boost.uBLAS [*1] を用いた.

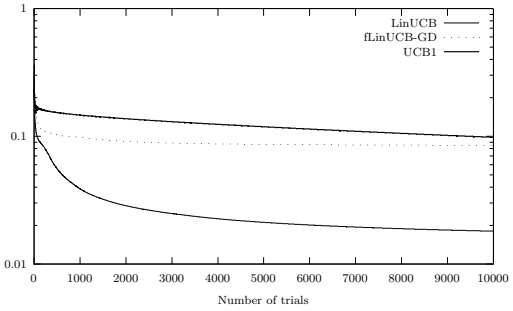
OS	Debian 3.16.0-4-amd64
CPU	Intel® Core™ i7-4790K CPU @ 4.00GHz
メモリ	16GB
コンパイラ	g++ 4.9.2
コンパイルオプション	`-O3 -DNDEBUG -DBOOST_UBLAS_NDEBUG`
Boost	1.55

表 1: 実験環境

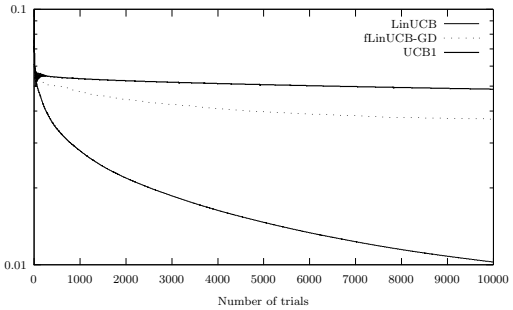
4.2 予備実験: fLinUCB-GD の性能

[9] では [2] における, 腕の集合が時間によって変化し, 特定の腕の間のみで係数ベクトルを共有するモデルについて実験が行われた. この章では腕の集合は時間に依存せず, すべての腕の間で係数ベクトルを共有するモデルについて追実験を行った結果を記す. 図 1 は LinUCB, fLinUCB-GD, UCB1 それぞれの average regret の時間経過のグラフである. Average regret とは cumulative regret の増加率で, $R/n = \mu^* - n^{-1}\sum_{t=1}^{n}\mu_{i_t}$ で定義する. 実験結果は 100 試行の平均をプロットしたもので, それぞれの試行において, 25 の腕を作成しそれぞれにランダムに特徴ベクトルを割り当て, 係数ベクトルをランダムに作成する. 腕の報酬の期待値は特徴ベクトルと係数ベクトルの内積で与える. 特徴ベクトルの次元は $10, 100, 1000$ である. パラメータは後述の表 2 と同様である.

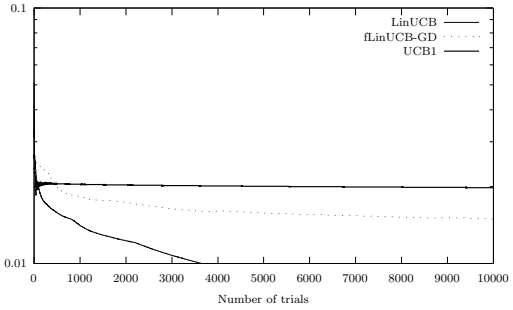
どのアルゴリズムも減少傾向が見られ, 最善の腕に収束すると思われる. 中でも LinUCB は最も早く減少し, 最良の性能を発揮している. fLinUCB-GD は LinUCB ほどではないが, UCB1 よりもよい性能であり, 一定の効果があるとみられる.



(a) 次元 10



(b) 次元 100



(c) 次元 1000

図 1: Contextual bandit problem におけるそれぞれのアルゴリズムの average regret

4.3 囲碁における実験

LinUCT 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. 本稿の実験はすべて 9 路盤で行った.

4.3.1 実装

実験のため UCT, $\text{LinUCT}_{\text{PLAIN}}$, $\text{LinUCT}_{\text{RAVE}}$ を実装した. 探索の効率化のため progressive widening [14] 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ソフトウェアである 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以下のとおり設定した.

パラメータ名	値	アルゴリズム
γ_t	$\frac{1}{100+t}$ [9]	fLinUCB-GD
λ_t	$\frac{1}{t^{1-\alpha}}$, $\alpha=0.6$ [9]	fLinUCB-GD
κ	1.0	$\text{LinUCT}_{\text{PLAIN}}$, $\text{LinUCT}_{\text{RAVE}}$
k	1000	$\text{LinUCT}_{\text{RAVE}}$
展開のしきい値	10	MCTS

表 2: 実験におけるパラメータ

4.3.2 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 ([15], [16] など). しかしそれらのほとんどは局面の特徴ではなく手の特徴であり, LinUCB の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仮定に沿わない

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

[*2] https://github.com/lukaszlew/libego

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 近傍の石の組み合わせ ($81 \times 8 \times 3 = 1944$ 次元)
- 定数 (1 次元)

8 近傍の石の組み合わせとは，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もち，9 路盤においては 8 近傍 $\times$ 81 点 $\times$ 3 種類の合計 1944 次元の特徴をもつ特徴ベクトルである．例として，図 2 における 8 近傍の石の組み合わせを表すベクトルは，$(0,0,1|1,0,0|0,0,0|1,0,0|0,1,0|0,1,0|0,0,0|0,0,0)$ となる．

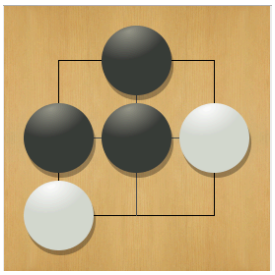


図 2: 特徴ベクトルの例: 中心の石と同色，異色，空白もしくは盤外の 3 ビットの情報を持つ．この例では $(0,0,1|1,0,0|0,0,0|1,0,0|0,1,0|0,1,0|0,0,0|0,0,0)$ となる．9 路盤ではこれが $9 \times 9 = 81$ 個存在する

4.3.3 $\mathrm{LinUCT_{PLAIN}}$: プレイアウト回数による性能

LinUCB をモンテカルロ木探索にそのまま適用した $\mathrm{LinUCT_{PLAIN}}$ は人工木の実験において正しく探索できていないということが示されている．その性質が囲碁においても成り立つかどうかを実験した．実験はプレイアウト数が 2000 の $\mathrm{LinUCT_{PLAIN}}$ と，4000, 8000 の $\mathrm{LinUCT_{PLAIN}}$ をそれぞれ対戦させた．結果を 表 3 に示す．

プレイアウト数	勝率 (%)	対戦回数
4000	39.1	100
8000	17.4	100

表 3: $\mathrm{LinUCT_{PLAIN}}(n = 2000)$ との対戦成績

対戦結果から，$\mathrm{LinUCT_{PLAIN}}$ ではプレイアウト数を増やすとともに性能が落ちるということが観察できる．先行研究 [5] で示されている通り，$\mathrm{LinUCT_{PLAIN}}$ はプレイアウト回数が多くなるに連れて探索木の上部の特徴ベクトルの差異に引きずられてしまい，最善手を選ぶことができなくなってしまうという原因が考えられる．

4.3.4 $\mathrm{LinUCT_{RAVE}}$: UCT との比較

LinUCB をそのまま適用した $\mathrm{LinUCT_{PLAIN}}$ はゲーム木探索において良い性能を発揮できないということがこれまでの結果からわかる．そこで LinUCB と UCB1 を組み合わせた $\mathrm{LinUCT_{RAVE}}$ が UCT と比較したときどのような性能になるかを対戦実験において観察した．対戦実験の結果を表 4, 図 3 に示す．

プレイアウト数	勝率 (%)	対戦回数
1000	0.43 ($\pm$0.10)	100
2000	0.53 ($\pm$0.10)	100
4000	0.45 ($\pm$0.068)	217
8000	0.49 ($\pm$0.10)	100

表 4: $\mathrm{LinUCT_{PLAIN}}(n = 2000)$ との対戦成績

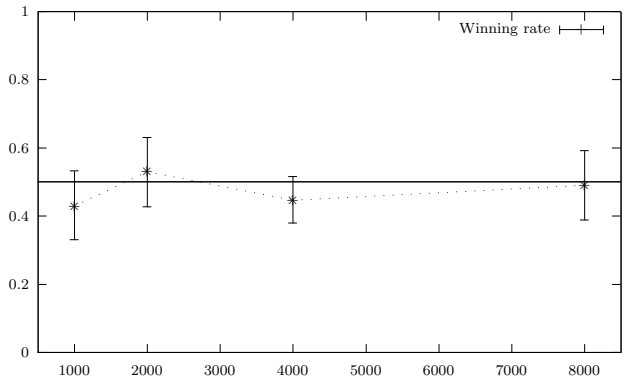


図 3: UCT に対する $\mathrm{LinUCT_{RAVE}}$ の勝率と 95 % 信頼区間

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．

4.3.5 実行速度

最後に，本稿で取り上げたアルゴリズムの実行時間についてそれぞれ比較した．図 4 に比較結果を示す．表ではプレイアウト回数が 1000 の UCT を 1 としたときの実行速度を示している．

UCT と比較して，fLinUCB-GD を用いた LinUCT は定数倍のオーダーで計算を行うことが可能である．また興味深い結果として，LinUCB のみを用いる $\mathrm{LinUCT_{PLAIN}}$ と UCB1 と組み合わせた $\mathrm{LinUCT_{RAVE}}$ とを比較すると後者の方が計算時間が少ない．$\mathrm{LinUCT_{PLAIN}}$ は探索が分散されず，特定のノードにプレイアウトが集中してしまい，結果的に探索木が深くなりより多くの内積計算をする傾向があるためであると考えられる．

5. おわりに

本稿では LinUCB をモンテカルロ木探索へと応用した LinUCT の囲碁における性能について議論した．また

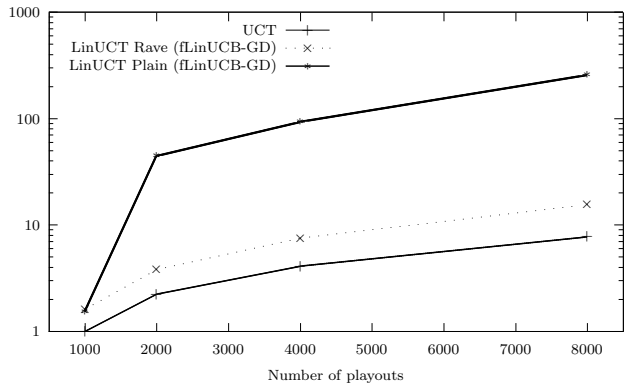


図 4: それぞれのアルゴリズムの実行時間の比較: UCT (プレイアウト数 1000) を 1 としたときの速度比較

LinUCB を高速化する方法である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.

既存手法である UCT と比較して, 本稿の範囲では LinUCT の優位性を示すことができなかった. 今後は特徴空間の設計やアルゴリズムなどを見直してより良い性能を目指す.

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, *Commun. 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, *Proceedings of the 19th international conference on World wide web*, ACM, pp. 661–670 (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] 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（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/170000087238/⟩ (2014).

[5] Mandai, Y. and Kaneko, T.: LinUCB Applied to Monte Carlo Tree Search, *Advances in Computer Games* (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, *Computational Intelligence and AI in Games, IEEE Transactions on*, Vol. 4, No. 1, pp. 1–43 (online), DOI: 10.1109/TCIAIG.2012.2186810 (2012).

[9] Korda, N., A., P. L. and Munos, R.: On-line gradient descent for least squares regression: Non-asymptotic bounds and application to bandits, *CoRR*, Vol. abs/1307.3176 (online), available from ⟨http://arxiv.org/abs/1307.3176⟩ (2013).

[10] Silver, D. and Tesauro, G.: Monte-Carlo simulation balancing, *Proceedings of the 26th Annual ICML*, ACM, pp. 945–952 (online), DOI: http://doi.acm.org/10.1145/1553374.1553495 (2009).

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

[12] 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).

[13] Chu, W., Li, L., Reyzin, L. and Schapire, R. E.: Contextual Bandits with Linear Payoff Functions, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011* (Gordon, G. J., Dunson, D. B. and Dudík, M., eds.), JMLR Proceedings, Vol. 15, JMLR.org, pp. 208–214 (2011).

[14] 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, Springer International Publishing, pp. 26–38 (online), available from ⟨http://dx.doi.org/10.1007/978-3-319-09165-5_3⟩ (2014).

[15] 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, Springer Berlin Heidelberg, pp. 81–92 (online), available from ⟨http://dx.doi.org/10.1007/978-3-642-17928-0_8⟩ (2011).

[16] Graf, T. and Platzner, M.: Adaptive Playouts in Monte Carlo Tree Search with Policy Gradient Reinforcement Learning, *Advances in Computer Games* (2015).