UCT でオンライン知識とオフライン知識を組み合わせる

Sylvain Gelly SYLVAIN.GELLY@LRI.FR
Univ. Paris Sud, LRI, CNRS, INRIA, France

David Silver SILVER@CS.UALVERTA.CA
University of Alberta, Edmonton, Alberta

概　要

UCTアルゴリズムはサンプル・ベースの探索を使って価値関数をオンラインで学習する. *TD*(λ) アルゴリズムは方策に沿った分布の価値関数をオフラインで学習できる. 我々はUCTアルゴリズムでオフラインとオンラインの価値関数を組み合わせる3つのアプローチを考察する. 1, オフラインの価値関数をモンテ・カルロ・シミュレーションのデフォルト方策として使う. 2, UCTの価値関数に行動価値の迅速なオンライン推定を組み合わせる. 3, オフラインの価値関数をUCT探索木での事前知識として使う. これらのアルゴリズムをGNU Go 3.7.10との9路碁の対局で評価する. 最初のアルゴリズムはランダム・シミュレーション方策とUCTの組み合わせより良いが, 驚くべきことに, より弱い手作りのシミュレーション方策とUCTの組み合わせより悪い. 2番目のアルゴリズムは全体的にUCTより良い結果を与える. 3番目のアルゴリズムは手作りの事前知識とUCTの組合わせより優れている. これらのアルゴリズムを世界で最も強い9路碁プログラムである*MoGo*に組み込む. 各手法は*MoGo*の強さを大きく改善する.[*]

1. はじめに

価値ベースの強化学習アルゴリズムは多くの注目すべき成果を挙げた. 例えば *TD*(λ) アルゴリズムの色々な変種は, チェス (Baxter 他, 1998), チェッカー (Schaeffer 他, 2001) そしてオセロ (Buro, 1999) で, 学習によりマスター・レベルに到達した. どれも価値関数は2値特徴の線型和で近似されている. 重みは各特徴の期待値への相対的な寄与を見つけるように調整され, 方策は新しい価値関数について改善される. このアプローチで, エージェントは方策に沿った状態の分布を跨いで適合する知識を学習する.

UCT はサンプル・ベースの探索アルゴリズム (Kocsis と Szepesvari, 2006) で, より挑戦的なゲームである囲碁に於いて注目すべき成功を収めた (Gelly 他, 2006). UCT アルゴリズムは各時刻毎, モンテ・カルロ・シミュレーションで各状態の価値関数を推定しながら探索木を作る. 各シミュレートされたエピソードの終了後, 木の中の値はオンラインで更新され, シミュレーション方策が新しい値について改善される. この値は現在の状態に対して高度に特殊化された局所的な知識を表現している.

本論文で我々は両アプローチを組合わせる方法を探る. 我々はオフラインの強化学習アルゴリズムによって蓄えられた一般的な知識と, サンプル・ベースの探索によってオンラインで得た知識とを組み合わせる新しいアルゴリズムを 2 つ導入する. 我々はまた, サンプル・ベースの探索によってオンラインで得た知識の 2 つの源: 一方は偏りがなくもう一方は学習は速いが偏りがある, を組み合わせる 3 つ目のアルゴリズムを導入する.

我々のアルゴリズムのテストに 9 路碁を使う (図 1). UCT アルゴリズム・ベースのプログラム *MoGo* は, KGS コンピュータ碁トーナメントの 9 路, 13 路, そして 19 路で優勝し, Computer Go Online Server で最も高いレートを持つプログラムである (Gelly 他, 2006; Wang と Gelly, 2007). 2 値特徴の線型和をベースに *TD*(0) アルゴリズムを使ったプログラム *RLGO* は, 大した事前の分野知識なしで 9 路碁用の最強既知価値関数を学習した (Silver 他, 2007). 我々はここで *MoGo* のオンライン知識と *RLGO* のオフライン知識とを組み合わせて, 総合的により良い性能を得られるかどうかを追求する.

[*]Appearing in *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, OR, 2007.

2. 価値ベースの強化学習

価値ベースの強化学習手法は，方策を計算する中間ステップとして価値関数 (*value function*) を使う．エピソード・タスクの場合，収益R_tはそのエピソードが時刻tから時刻Tで停止する迄の間に累積された全報酬である．後に続く全行動を方策πを使って選ぶ場合，行動価値関数$Q^\pi(s, a)$ は状態$s \in S$ で行動$a \in \mathcal{A}$ を選んだ後で期待される収益である．

$$\begin{aligned} R_t &= \sum_{k=t+1}^{T} r_k \\ Q^\pi(s, a) &= E_\pi\left[R_t \mid s_t = s,\ a_t = a\right] \end{aligned}$$

価値関数は，例えば $TD(\lambda)$ アルゴリズムの様な種々の異なるアルゴリズムを使って推定できる (Sutton, 1988)．そして方策は新しい価値関数に関して，例えば収穫 (価値が最大の行動を選ぶ) と探検 (ランダムな行動を選ぶ) をバランスさせるために ε-greedy 方策を使って，**改善** (*improve*) することができる．**方策反復** (*policy iteration*) として知られる方策の評価と改良を繰り返す過程は，価値ベースの強化学習アルゴリズムの基礎を成す (Sutton & Barto, 1998).

もし環境モデルが利用できるか学習可能なら，**サンプル・ベースの探索** (*sample-based search*) に，価値ベースの強化学習アルゴリズムを使うことができる．実際の経験から学ぶよりむしろ，シミュレートされたエピソードをモデルからサンプルすることができる．そして価値関数はシミュレートされた経験を使って更新される．Dyna アーキテクチャはサンプル・ベースの探索の一例を与えている (Sutton, 1990).

3. UCT

UCT アルゴリズム (Kocsis と Szepesvari, 2006) は，もっぱら開始状態とそれ以後の状態の木に焦点を当てた価値ベースの強化学習アルゴリズムである．

行動価値関数$Q_{UCT}(s, a)$ は，全 (状態，行動) 対の部分集合を含んだ部分的な表表現$\mathcal{T} \subseteq S \times \mathcal{A}$ によって近似される．これは開始状態を根とする，訪れた状態の探索木とみなすことができる．木の中の各状態と行動用の個別の価値はモンテ・カルロ・シミュレーションによって推定される．

UCT が使う方策は，多碗バンディット・アルゴリズム UCB (Auer 他, 2002) をベースに，探検と収穫とがバランスする様に設計されている．

もし現在の状態 s から選べる全行動が木の中に表現されていたら，$\forall a \in \mathcal{A}(s)$, $(s, a) \in \mathcal{T}$, UCT は行動の価値の信頼上限 (upper confidence bound) を最大化する行動を選ぶ．

$$\begin{aligned} Q^{\oplus}_{UCT}(s, a) &= Q_{UCT}(s, a) + c\sqrt{\frac{\log n(s)}{n(s, a)}} \\ \pi_{UCT}(s) &= \operatorname*{argmax}_a Q^{\oplus}_{UCT}(s, a) \end{aligned}$$

ここで$n(s,\ a)$ は状態sから行動aが選ばれた回数，$n(s)$ はある状態を訪れた回数の合計，$n(s) = \Sigma_a n(s,\ a)$ である．

もし現在の状態sから選べるある行動が木の中に表現されていなければ，$\exists a \in \mathcal{A}(s)$, $(s, a) \notin \mathcal{T}$, 一様ランダム方策$\pi_{random}$を使って木にない全ての行動 $\tilde{\mathcal{A}}(s) = \{ a \mid (s, a) \notin \mathcal{T} \}$ の中から一つ選ぶ．

あるエピソード$s_1, a_1, s_2, a_2, \ldots, s_T$, が終わった時，探索木中の各状態行動対 $(s_t,\ a_t) \in \mathcal{T}$ がこのエピソードの収益を使って更新される．

$$n(s_t, a_t) \leftarrow n(s_t, a_t) + 1 \tag{1}$$

$$\begin{aligned} Q_{UCT}(s_t, a_t) \leftarrow\ & Q_{UCT}(s_t, a_t) \\ &+ \frac{1}{n(s_t, a_t)}\left[R_t - Q_{UCT}(s_t, a_t)\right] \end{aligned} \tag{2}$$

このエピソードからの新しい状態と行動，$(s_t,\ a_t) \notin \mathcal{T}$, は，初期値$Q(s_t,\ a_t) = R_t$, $n(s_t,\ a_t) = 1$ として木に付加される．場合によっては初めて訪れた $(s_t,\ a_t) \notin \mathcal{T}$ な状態と行動だけを加える方がメモリ消費が少ない (Coulom, 2006; Gelly他, 2006)．この手続きはエピソードをn回経験すると節点数nの探索木を作り上げる．

UCT 方策にはステージが 2 つあると考えることができる．各エピソードの始めの方では，探索木に含まれた知識にしたがって行動を選ぶ．しかし一旦探索木の範囲を離れると知識はなくランダムに振舞う．そこで木の中の各状態はその価値をモンテ・カルロ・シミュレーションで推定する．より多くの情報が木を遡って伝播するにつれて方策は進歩し，モンテ・カルロ推定はより精密な収益に基づく．

モデルが利用できる場合，UCT をサンプル・ベースの探索アルゴリズムとして使うことができる．エピソードは実際の状態 $\hat{s}$ で始まるモデルからサンプルされる．シミュレートされた経験を使って，実際のタイム・ステップ毎に新しい表現 $\mathcal{T}(\hat{s})$ が作成される．典型的には各ステップ毎に数千のエピソードをシミュレートできるので，価値関数は現在の状態 $\hat{s}$ の詳細な探索木を含む．

2 人ゲームの場合，相手をエージェント自身の方策でモデル化できるので，エピソードは自己対戦でシミュレートできる．UCT はエージェントの価値の信頼上限の最大化と相手の価値の信頼下限の最小化に使われる．非定常性に関する一定の仮定の下で UCT はミニマックス値に収束する (Kocsis と Szepesvari, 2006)．しかしアルファ・ベータ探索の様な他のミニマックス探索アルゴリズムと異なり，UCT は状態を評価したり手を並び替えるために事前の分野知識を必要としない．さらに UCT の探索木は一様ではなく，最も有望な方向をえこひいきする．これらの性質から，UCT は大きな状態空間と分岐係数を持ち，強い評価関数が知られてない囲碁の様なゲームに理想的である．

4．線形価値関数近似

UCT は表形式の手法で，状態を跨ぐ一般化は行わない．他の価値ベースの強化学習手法は，複雑な領域中の状態を抽象化する豊富な関数近似手法を提供している (Schraudolph 他, 1994; Enzenberger, 2003; Sutton, 1996)．ここでは最低限の事前分野知識しか必要とせず (Silver 他, 2007)，また他の多くの分野で成功が証明されている (Baxter 他, 1998; Schaeffer 他, 2001; Buro, 1999)，簡単な関数近似アプローチを考察する．

エージェントがそのゲームに勝てば $r = 1$，負ければ $r = 0$ という単純な報酬関数を推定しよう．価値関数は 2 値特徴 ϕ の重み θ による線型和で近似される．

$$Q_{RLGO}(s, a) = \sigma(\phi(s, a)^T \theta)$$

ここで S 字状圧縮関数 σ は価値関数をゲームに勝つ確率に写像する．各タイム・ステップの後，重みは *TD*(0) アルゴリズム (Sutton, 1988) を使って更新される．価値関数が確率なので，現在の価値と後続の価値の相互エントロピーが最小になるように損失関数を修正する．

$$\delta = r_{t+1} + Q_{RLGO}(s_{t+1}, a_{t+1}) - Q_{RLGO}(s_t, a_t)$$

$$\Delta\theta = \frac{\alpha}{|\phi(s_t, a_t)|}\delta\phi(s_t, a_t)$$

ここで δ は TD 誤差，α は刻み幅である．

囲碁に於いて**形** (*shape*) の概念は戦略的に重要である．この理由から，我々は石の局所的なパターンを識別する 2 値特徴 $\phi(s, a)$ を使う (Silver 他, 2007)．各**局所形状特徴** (*local shape feature*) は，盤上のある特定の矩形内の指定された石と空の交点の構成にマッチする (図 1)．局所形状特徴は盤上全ての位置の 1×1 から 3×3 までの全構成が創られている．2 種類の重みが使われている: 最初のセットでは重みは回転あるいは鏡像対称な全特徴で共有されている．2 つ目のセットでは同様に，重みは同じパターンから変換された局所形状特徴間で共有されている．

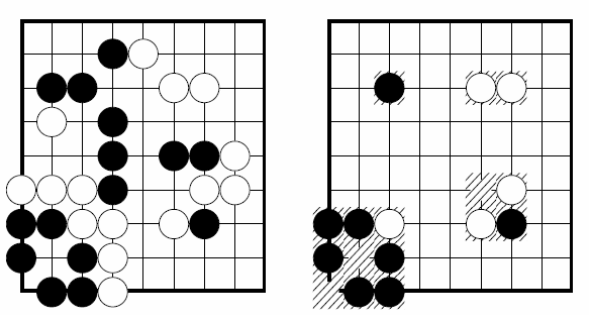

図 1. (左) 9 路碁の局面の例．黒と白は交互に石を置く．石は決して動くことはできないが，完全に囲まれたら取り上げられる．地を多く囲んだ対局者がゲームに勝つ．(右) 形は囲碁の戦略に重要な部分である．この図は (左上から時計回りに) 局面の例にある 1×1，2×1，2×2，3×3 の局所的な形の特徴を表している．

トレーニングでは同じエージェントの 2 つの版同士がどちらも ε-greedy 方策を使って対局する．各ゲームは空の盤面から終局まで打たれ，方策に沿った状態の分布に対する損失が最小化される．こうして近似価値関数は，自己対局中に出会う局面の全分布を跨いで，各局所形状特徴の勝利への相対的な貢献を学習する．

5．UCT とデフォルト方策

UCT アルゴリズムはその探索木の範囲を超えれば何の知識も持ってない．もし木の中にない状態に遭遇したらそれはランダムに振舞う．Gelly ら (Gelly 他, 2006) は，UCT が一度探索木を離れた後のエピソードを完了するのに使う**デフォルト方策** (*default policy*) を UCT に組み合わせた．我々はデフォルト方策 π に対し，元の UCT アルゴリズムを $UCT(\pi_{random})$ と，この拡張されたアルゴリズムを $UCT(\pi)$ と記す．

世界で最も強い 9 路のコンピュータ碁プログラム *MoGo* は UCT に手作りのデフォルト方策 π_{MoGo} を組み合わせている (Gelly 他, 2006)．しかし多くの分野で，良いデフォルト方策を作るのは困難である．エキスパートの知識が利用できる時でさえ，それを解釈してコードにするのは難しいだろう．さらに，エピソードを数多くシミュレートして大きな探索木を作ることができる様に，デフォルト方策は高速でなければならない．最小限の分野知

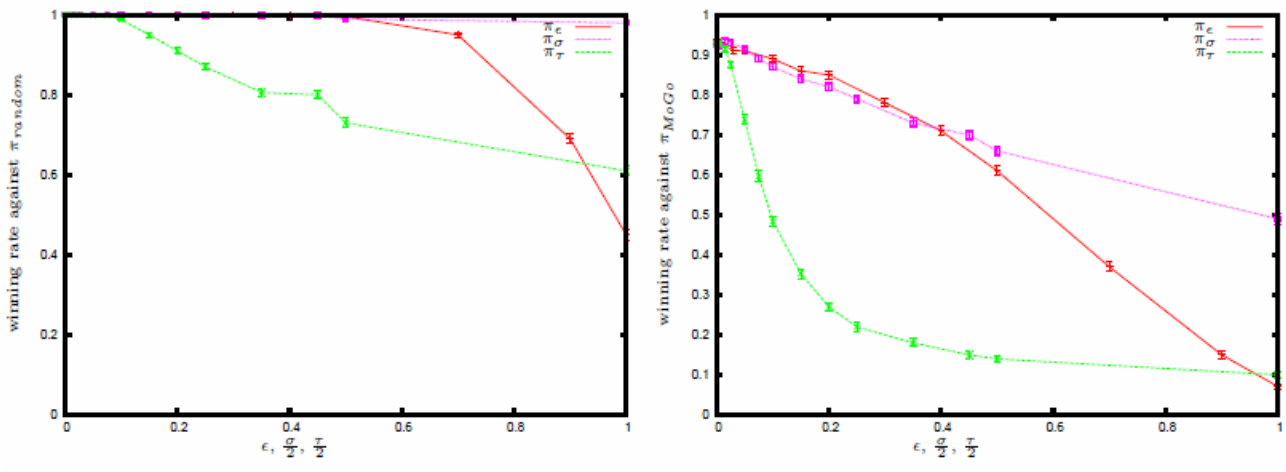

図 2. デフォルト・ポリシーの各クラスの，ランダム・ポリシーπ_{random} (左) と手作りポリシーπ_{MoGo} (右) に対する相対的な強さ．横軸は各ポリシーのランダム性．

識で高性能のデフォルト方策を学習する手法が好ましいだろう．

2 値特徴の線型和はそういう手法の一つを与える．2 値特徴は評価するにも速くかつインクリメンタルに更新できる．このアプローチで学習される表現は多くの分野で良い性能を示すことが知られている (Baxter 他, 1998; Schaeffer 他, 2001)．特徴を指定するために最小限の分野知識が必要である．

9 路碁分野用の価値関数Q_{RLGO}を，2 値特徴の線型和を使って学習する (4 節参照)．それはオフラインの自己対局で学習される．この価値関数を使ってUCTのデフォルト方策を生成する．モンテ・カルロ・シミュレーションは確率的なデフォルト方策と組み合わせると最も良く働くので，我々は価値関数Q_{RLGO}から確率的な方策を生成する 3 つの異なるアプローチを考察する．

初めに ε-greedy 方策を考察する．

$$\pi_\varepsilon(s, a) = \begin{cases} 1 - \varepsilon + \dfrac{\varepsilon}{|A(s)|} & \text{if } a = \operatorname{argmax}_{a'} Q_{RLGO}(s, a') \\ \dfrac{\varepsilon}{|A(s)|} & \text{otherwise} \end{cases}$$

次にガウス雑音$\eta(s, a) \sim N(0, \sigma^2)$ で崩した，雑音の多い価値関数上のgreedy方策を考察する．

$$\pi_\sigma(s, a) = \begin{cases} 1 & \text{if } a = \operatorname{argmax}_{a'} Q_{RLGO}(s, a') + \eta(s, a') \\ 0 & \text{otherwise} \end{cases}$$

3 番目，温度パラメタ τ のソフトマックス分布を使って手を選ぶ．

$$\pi_\tau(s, a) = \frac{e^{Q_{RLGO}(s,a)/\tau}}{\sum_{a'} e^{Q_{RLGO}(s,a')/\tau}}$$

我々はデフォルト方策π_{random}, π_{MoGo}, π_ε, π_σ, そしてπ_τの各クラスの性能を比較した．方策の各組当たり 6,000 試合のラウンド・ロビン・トーナメントで測定した，各デフォルト方策の (囲碁対局者としての) 相対的な強さの評価結果を図 2 に示す．少ししか，あるいは全くランダム化しない場合，Q_{RLGO}ベースの方策群がランダム方策π_{random} とMoGoの手作り方策π_{MoGo}の両方を 90% を超えるマージンで上回っている．ランダム化のレベルが上がるにつれて，これらの方策はランダム方策π_{random} へと退化する．

次に我々は，デフォルト方策 π の各々のモンテ・カルロ・シミュレーションに於ける精度を，手でラベルを付与した 200 局面のテスト・スイートを使って比較した．各テスト局面から，方策 π の各々を使って自己対戦 1,000 局を行った．我々は収益の平均値 (即ちモンテ・カルロ推定値) と手で付与したラベルの値 (図 3 参照) の差の MSE (訳注: mean square error; 平均自乗誤差) を測定した．一般に $UCT(\pi)$ に良い方策はモンテ・カルロ・シミュレーションで正確な評価ができなければならない．我々の *MoGo* での経験からは，MSE と対局時の強さは密接な関係がある様に思われる．

MSEは，より強くかつ適切にランダム化されたデフォルト方策が使われた時，一様ランダムなシミュレーションから進歩する．もしデフォルト方策が決定的過ぎると，モンテ・カルロ・シミュレーションは何らかの寄与を提供することに失敗しπの性能は劇的に低下する．もしデ

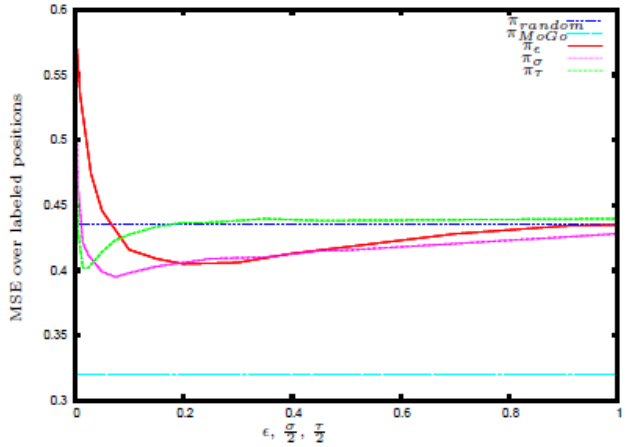

図 3. 手でラベルを付けた 200 局面のテスト・スイートをモンテ・カルロ・シミュレーションで評価した時の各ポリシーπ の MSE. 横軸はポリシーのランダム性.

フォルト方策がランダム過ぎるとランダム方策π_{random} と同等になる.

直感的に, モンテ・カルロ・シミュレーション中, より強くかつ適切にランダム化された方策はより弱い方策を凌駕すると期待するかも知れない. しかし, これらデフォルト方策の対局時の強さは大きく改善されるにも関わらず, π_e, π_o, そしてπ_τの精度は決して手作り方策π_{MoGo}の精度に迫ることはない. Q_{RLGO}ベースのデフォルト方策が実際に強いことを我々のテスト・スイートの局面で検証するために, 我々はこの各局面から交互に開始するラウンド・ロビン・トーナメントを再び行い, デフォルト方策の相対的な強さは同じ位のままであることを確認した. 我々はまた, 完全なUCTアルゴリズムの性能を, 最善のQ_{RLGO}ベースのデフォルト方策と, パラメタを最小化したMSEを使って比較した (表 1). この実験はMSEの結果が実際の対局に適合することを示した.

客観的により強いデフォルト方策が UCT でより良い性能を導かないのは意外である. さらに, この結論にはモンテ・カルロ・シミュレーションしか必要ないので, 他のサンプル・ベースの探索アルゴリズムも含まれる. シミュレーション方策の性質はその客観的な性能と**同等かそれ以上に**重要と思われる. 各方策は固有の偏りを持ち, モンテ・カルロ・シミュレーションに於いてエピソードのある特定の分布を導く. もしその分布が客観的にありそうもない方に歪んでいれば探索アルゴリズムの予測精度は損なわれるかも知れない.

6. 迅速行動価値推定

UCTアルゴリズムは, 価値を比較する根拠を得る前に, ある状態$s \in \mathcal{T}$ から選べる全行動をサンプルしなければならない. さらに, 分散の小さな推定値を生成するために状態sの各行動は複数回サンプルされなければならない. 行動の空間が広い時はこれが学習速度の低下の原因になり得る. この問題を解決するために新しいアルゴリズムUCT_{RAVE} を導入する. これは状態sの行動aの迅速行動価値推定(*rapid action value estimate*)を形成し, このオンライン知識をUCTに組み込む.

アルゴリズム	勝率 (対 GnuGo)
$UCT(\pi_{random})$	8.88 ± 0.42 %
$UCT(\pi_o)$	9.38 ± 1.9 %
$UCT(\pi_{MoGo})$	48.62 ± 1.1 %

表 1. 異なるデフォルト方策を使ったUCTアルゴリズムの GnuGo 3.7.10 (level 0) に対する勝率. 一手当たり 5,000 シミュレーション. ± の後の数は完全な対局数千局の標準誤差に相当. π_o は$o = 0.15$ を使った.

通常, モンテ・カルロ法は直ちに a が選ばれた全エピソードの収益を平均して a の価値を推定する. 代わりに我々は, a が後続の**任意の**時点で選ばれた全エピソードの収益を平均する. このアイデアはコンピュータ囲碁分野では *all-moves-as-first* ヒューリスティックとして知られている (Brügmann, 1993) が, 同じアイデアは行動列を転置可能な任意の分野に適用できる.

$Q_{RAVE}(s, a)$ を状態sの行動aの迅速価値推定としよう. 各エピソード$s_1, a_1, s_2, a_2, \ldots, s_T$, の後, 各状態 $s_{t_1} \in \mathcal{T}$ で $a_{t_2} \in \mathcal{A}(s_{t_1}), t_1 \le t_2$, かつ$\forall t < t_2, a_t \neq a_{t_2}$ を満たす後続の行動 a_{t_2} の価値が更新される.

$$\begin{aligned} m(s_{t_1}, a_{t_2}) &\leftarrow m(s_{t_1}, a_{t_2}) + 1 \\ Q_{RAVE}(s_{t_1}, a_{t_2}) &\leftarrow Q_{RAVE}(s_{t_1}, a_{t_2}) \\ &+ 1/m(s_{t_1}, a_{t_2}) [R_{t_1} - Q_{RAVE}(s_{t_1}, a_{t_2})] \end{aligned}$$

ここで $m(s, a)$ は行動 a が状態 s 以後の任意の時点で選ばれた回数.

迅速行動価値推定は, 各行動の価値を小さな分散で素早く学習することができる. しかしこれは, 行動の価値は通常それが選ばれる厳密な状態に依存するので, 何らかの偏りを導入するかも知れない. それゆえ, できれば初期は迅速推定を, 限界では元の UCT 推定を使いたい. これを達成すべく我々は, 徐々に減少する重み β による, この 2 つの推定の線型和を使う.

$$Q^{\oplus}_{RAVE}(s, a) = Q_{RAVE}(s, a) + c\sqrt{\frac{\log m(s)}{m(s, a)}}$$

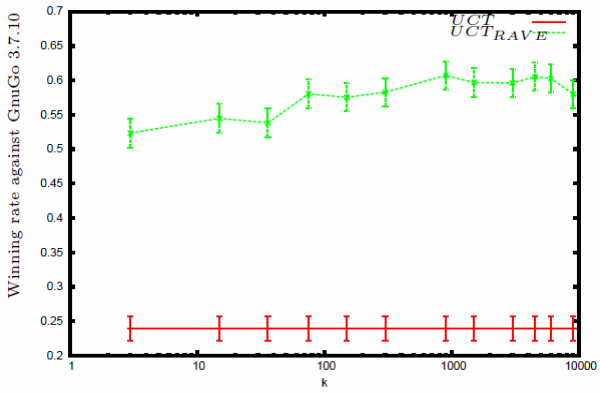

図 4. 異なる等価パラメタkの設定に対する$UCT_{RAVE}(\pi_{MoGo})$のGnuGo 3.7.10 (level 10) に対する勝率．一手当たり 3,000 シミュレーション．縦棒は標準誤差．各点は 2,300 対局の平均．

$$\beta(s,a) = \sqrt{\frac{k}{3n(s)+k}}$$

$$Q^{\oplus}_{UR}(s,a) = \beta(s,a)\, Q^{\oplus}_{RAVE}(s,a) + (1-\beta(s,a))\, Q^{\oplus}_{UCT}(s,a)$$

$$\pi_{UR}(s) = \operatorname*{argmax}_a Q^{\oplus}_{UR}(s,a)$$

ここで$m(s) = \sum_a m(s,a)$．等価パラメタ (*equivalence parameter*) kは両推定の重みが等しくなるエピソードの経験回数を制御する．

デフォルト方策 π_{MoGo} を使い，異なる等価パラメタkの設定で新しいアルゴリズム$UCT_{RAVE}(\pi_{MoGo})$ を調べた．各設定毎にGNU Go 3.7.10 (level 10) と2,300回対局させた．結果と，一手当たり 3,000 シミュレーションの$UCT(\pi_{MoGo})$ との比較を図 4 に示す．UCT_{RAVE}を使った勝率は50%から60%の間で変化し，比べた迅速推定なしは 24%である．最高性能は等価パラメタが 1,000 あるいはそれ以上で達成される．これは，迅速行動価値推定がおよそUCTシミュレーション数千エピソードに匹敵する価値があることを示している．

7. UCT と事前知識

UCT アルゴリズムは各状態の価値をモンテ・カルロ・シミュレーションで推定する．しかし多くの場合我々は，ある状態のらしい価値に関する事前知識を持っている．我々はオフライン知識を利用する簡単な手法を導入する．これは UCT の漸近的な価値推定を偏らせずに学習レートを増加する．

既存の価値関数$Q_{prior}(s,a)$ を組み込める様にUCTを修正する．ある新しい状態と行動 (s, a) がUCTの表現$\mathcal{T}$に付け加えられた時，その価値を事前知識にしたがって初期化する．

$$n(s,a) \leftarrow n_{prior}(s,a)$$
$$Q_{UCT}(s,a) \leftarrow Q_{prior}(s,a)$$

数n_{prior}は事前価値関数の等価な経験 (*equivalent experience*) を推定する．これはUCTが同じ位の精度の推定を達成するのに必要であろうエピソード数を示す．初期化後，価値関数は通常のUCTの更新 (等式 1, 2 参照) を使って更新される 我々はデフォルト方策πを使ったこの新しいUCTアルゴリズムを$UCT(\pi, Q_{prior})$ と記す．

UCT_{RAVE}アルゴリズムに対しても迅速推定を事前知識にしたがって初期化するという同様の修正を行う．

$$m(s,a) \leftarrow m_{prior}(s,a)$$
$$Q_{RAVE}(s,a) \leftarrow Q_{prior}(s,a)$$

9 路碁の事前知識を生成するいくつかの手法を比較した．最初は互角 (*even-game*) ヒューリスティック，$Q_{even}(s,a) = 0.5$, を使う．これは方策に沿って遭遇する局面はほとんど全部，ほぼ対等らしかろうということを示している．次にお祖父ちゃん (*grandfather*) ヒューリスティック，$Q_{grand}(s_t, a) = Q_{UCT}(S_{t-2}, a)$, を使う．これは通常，対局者$P$の手番の価値は一つ前の$P$の手番の状態の価値と同じ位だろうということを示している．3 番目に手作りのヒューリスティック$Q_{MoGo}(s,a)$ を使う．このヒューリスティックは，greedyな行動選択が，知られている中で最善のデフォルト方策$\pi_{MoGo}(s,a)$ を産むだろうということを意図されている．最後に，$TD(\lambda)$ でオフライン学習された 2 値特徴の線型和，$Q_{RLGO}(s,a)$ を使う (4 節参照)．

M_{eq}の色々な定数値で，各事前知識源に等価経験$m_{prior}(s,a) = M_{eq}$ を割り当てる．$UCT_{RAVE}(\pi_{MoGo}, Q_{prior})$ とGnuGo 3.7.10 (level 10) を，手番を対局毎に替えながら2,300回対局させた．UCTアルゴリズムは，一手当たりの時間を固定するのではなく，各手毎に3,000エピソードの経験をサンプルした (図 5 参照)．事実上どのアルゴリズムの実行時間も同等である (表 4)．

オフラインで学習された価値関数Q_{RLGO}は，他の全ヒューリスティックを上回り，UCT_{RAVE}アルゴリズムの勝率を 60% から 69% に上げている．最高性能は等価経験$M_{eq} = 50$ を使った時で，これはQ_{RLGO}にUCT_{RAVE}のシミュレーション 50 エピソードと同程度の価値があることを示している．等価経験を事前価値推定の分散に

アルゴリズム	対 GnuGo の勝率
$UCT(\pi_{random})$	1.84 ± 0.22 %
$UCT(\pi_{MoGo})$	23.88 ± 0.85 %
$UCT_{RAVE}(\pi_{MoGo})$	60.47 ± 0.79 %
$UCT_{RAVE}(\pi_{MoGo}, Q_{RLGO})$	69 ± 0.91 %

表 2. 異なる UCT アルゴリズムの GnuGo 3.7.10 (level 10) に対する勝率. 一手当たり 3,000 シミュレーション. ± の後の数字は数千局の標準誤差に相当.

応じて変えることで, これらの結果をさらに改良できそうに思われる.

8. 結論

UCT アルゴリズムを, オンライン学習とオフライン学習を組み合わせる 3 つの異なるテクニックで大きく改良することができる. 1, UCT 探索木を超えたエピソードを完結するのにデフォルト方策を使うことができる. 2, 初期学習を加速するのに迅速行動価値推定を使うことができる. 3, UCT 木の中で価値関数の初期化に事前知識を使うことができる.

この 3 つのアイデアをコンピュータ囲碁プログラム *MoGo* に適用し, その性能をUCTアルゴリズム・ベースではない 9 路碁で最強のプログラムの一つ, GnuGo 3.7.10 (level 10) との対局で測定した. 新しいテクニックの各々は勝率を, 基本的な UCT アルゴリズムのたった2% から3つの技法全てを使った69% まで, 以前のアルゴリズムから大きく向上させた. 表 2 にこれらの改良結果をまとめた. 条件は, 各アルゴリズムのパラメタ設定は最善, 一手当たり 3,000 シミュレーション. 表 4 は各アルゴリズムの実行速度.

これらの結果は一手当たりたった 3,000 シミュレーション実行をベースにしている. シミュレーション回数が増えた時*MoGo*の全体的な性能は対応して向上した. 例えば, 協同アルゴリズム$UCT_{RAVE}(\pi_{MoGo}, Q_{MoGo})$ を使った場合, 一手当たりのシミュレーション回数を増やすと勝率は 92% まで向上する. *MoGo*のこの版は Computer Go ServerでEloレーティング 2,320 を達成した. これはUCTベースではないプログラムをおよそ 500 ポイント上回り, UCTベースのプログラムより 200 ポイント以上高い[1] (表 3 参照).

この版の*MoGo*が要求する分野知識はデフォルト方策π_{MoGo} と事前価値関数Q_{MoGo}に含まれている. 我々は, $TD(\lambda)$ を使った価値関数のオフライン学習により,

[1] Eloレーティング・システムは, 対局時の強さの統計的な計量で, 200 ポイントの差は上手の勝率の期待値およそ 75% を示す.

シミュレーション回数	対 GnuGo 勝率	CGOS のレート
3,000	69 %	1,960
10,000	82 %	2,110
70,000	92 %	2,320*

表 3. 一手当たりのシミュレーション回数を増やした時の $UCT_{RAVE}(\pi_{MoGo}, Q_{MoGo})$ のGnuGo 3.7.10 (level 10) に対する勝率. アスタリスクが付いたCGOSで使われた版の一手当たりのシミュレーション回数は, 利用可能な時間に応じて序盤の 300,000 から 20,000 迄変化する.

アルゴリズム	速度
$UCT(\pi_{random})$	6,000 g/s
$UCT(\pi_{MoGo})$	4,300 g/s
$UCT(\pi_{\varepsilon}), UCT(\pi_{o}), UCT(\pi_{\tau})$	150 g/s
$UCT_{RAVE}(\pi_{MoGo})$	4,300 g/s
$UCT_{RAVE}(\pi_{MoGo}, Q_{MoGo})$	4,200 g/s
$UCT_{RAVE}(\pi_{MoGo}, Q_{RLGO})$	3,600 g/s

表 4. P4 3.4GHz上での各アルゴリズムの対局開始時の毎秒シミュレーション回数. $UCT(\pi_{random})$ は高速だが一手当たりの時間を同じだけ与えても非常に弱い. $UCT_{RAVE}(\pi_{MoGo}, Q_{RLGO})$ 以外のアルゴリズムの実行速度は同等である.

この分野知識への要求を如何に完全に取り除くことができるかを証明した. 学習された価値関数は, 木の内部で事前知識として使われた時, ヒューリスティック価値関数より優れている. 驚くべきことに, デフォルト方策として使われた場合, その客観的優位性にも関わらず学習された価値関数の性能はヒューリスティック価値関数に劣る. モンテ・カルロ・シミュレーションに於いて, なぜ特定の方策が他のより良い性能を示すかを理解することは, 将来の進歩の鍵になるかも知れない.

我々は 9 路碁の分野に集中してきた. 13 路や 19 路の碁の様なより大きな分野ではこの 3 つのアルゴリズムの重要性が増しそうである. 分岐係数が大きい場合, 正確なシミュレーション, 初期学習の加速, そして事前知識の組込みがますます重要になる. これらのアイデアが組み込まれた時, UCT アルゴリズムは他の多くの挑戦的な分野での成功を証明するだろう.

参考文献

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multi-armed bandit problem. *Machine Learning, 47*, 235–256.

Baxter, J., Tridgell, A., & Weaver, L. (1998). Experiments in parameter learning using temporal differences. *International Computer Chess Association Journal, 21*, 84–99.

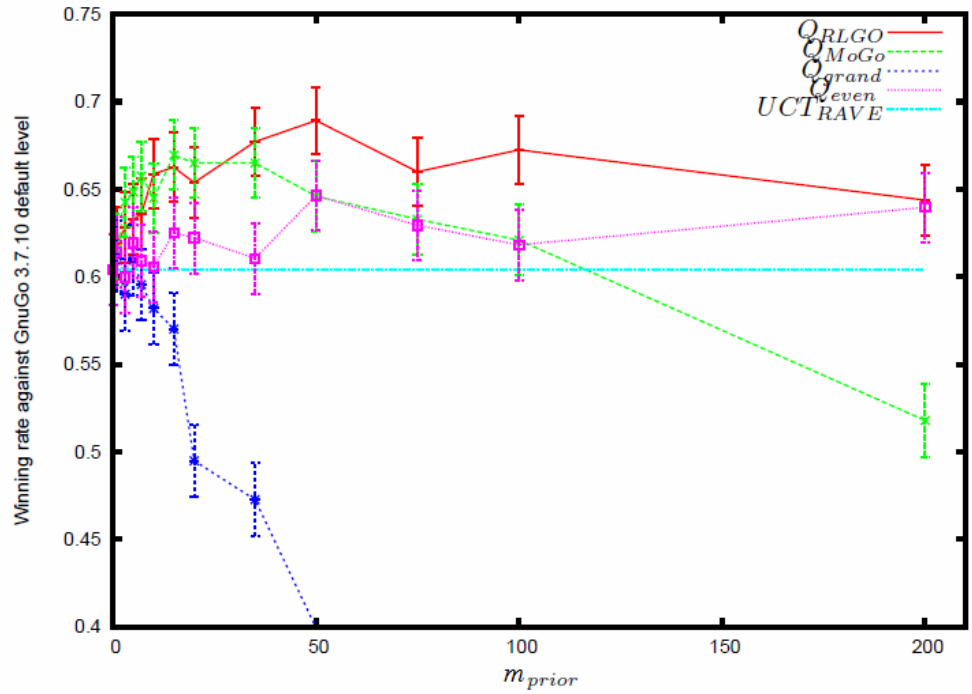

図 5. 初期化に異なる事前知識を使った時の$UCT_{RAVE}(\pi_{MoGo})$ の対GnuGo 3.7.10 (level 10) の勝率. 一手当たり 3,000 シミュレーション. 棒は標準誤差. 各点は 2,300 対局の平均.

Brügmann, B. (1993). Monte-Carlo Go. *http://www.cgl.ucsf.edu/go/Programs/Gobble html.*

Buro, M. (1999). From simple features to sophisticated evaluation functions. *1st International Conference on Computers and Games* (pp. 126–145).

Coulom, R. (2006). Efficient selectivity and backup operators in Monte-Carlo tree search. *5th International Conference on Computer and Games, 2006-05-29*. Turin, Italy.

Enzenberger, M. (2003). Evaluation in Go by a neural network using soft segmentation. *10th Advances in Computer Games Conference* (pp. 97–108).

Gelly, S., Wang, Y., Munos, R., & Teytaud, O. (2006). *Modification of UCT with patterns in Monte-Carlo Go* (Technical Report 6062). INRIA.

Kocsis, L., & Szepesvari, C. (2006). Bandit based Monte-Carlo planning. *15th European Conference on Machine Learning* (pp. 282–293).

Schaeffer, J., Hlynka, M., & Jussila, V. (2001). Temporal difference learning applied to a high-performance game-playing program. *17th International Joint Conference on Artificial Intelligence* (pp. 529–534).

Schraudolph, N., Dayan, P., & Sejnowski, T. (1994). Temporal difference learning of position evaluation in the game of Go. *Advances in Neural Information Processing Systems 6* (pp. 817–824). San Francisco: Morgan Kaufmann.

Silver, D., Sutton, R., & Müller, M. (2007). Reinforcement learning of local shape in the game of Go. *20th International Joint Conference on Artificial Intelligence* (pp. 1053–1058).

Sutton, R. (1988). Learning to predict by the method of temporal differences. *Machine Learning*, *3*, 9–44.

Sutton, R. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *7th International Conference on Machine Learning* (pp. 216–224).

Sutton, R. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems 8* (pp. 1038–1044).

Sutton, R., & Barto, A. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Wang, Y., & Gelly, S. (2007). Modifications of UCT and sequence-like simulations for Monte-Carlo Go. *IEEE Symposium on Computational Intelligence and Games*, Honolulu, Hawaii (pp. 175–182).

訳者より: 本文書は,

S. Gelly and D. Silver: Combining online and offline knowledge in UCT, *Zoubin Ghahramani (Ed.) Proceedings of the 24th International Conference on Machine Learning*, pp. 273–280, Omni Press, 2007.

http://www.machinelearning.org/proceedings/icml2007/papers/387.pdf

を原著者の許諾を得て日本語に翻訳したものです．本翻訳ならびにその公開は専ら日本のコンピュータ囲碁界への貢献を目的としたもので，原著者らの権利を侵害する意図は全くありません．論文等での引用 (citation) に際しては必ず原論文を先に記して下さい．本翻訳の公開を快諾してくれた原著者に厚く感謝します．翻訳には細心の注意を払ったつもりですが，もし誤訳等を見つけた時は mailto:gg@nue.ci.i.u-tokyo.ac.jp まで連絡して頂ければ幸いです．なお，原論文の公開後に原著者が報告した誤りは修正してあります．

下のリンクは原著者らがICML07の発表で使用したスライドです．参考にどうぞ．

http://www.cs.ualberta.ca/~silver/research/presentations/files/sylvain-silver.pdf

Translator's note: This document is a Japanese translation of the paper,

S. Gelly and D. Silver: Combining online and offline knowledge in UCT, *Zoubin Ghahramani* (*Ed.*) *Proceedings of the 24th International Conference on Machine Learning*, pp. 273–280, Omni Press, 2007.

http://www.machinelearning.org/proceedings/icml2007/papers/387.pdf

under the permission by the original authors.

The object of this translation and publication is entirely to contribute to Japanese computer Go community and there is no intention of violating original authors' rights. Upon citing, cite the original paper first. The translator wish to express my gratitude to original authors for their permission. Though the translator paid close attention if the reader find any error, please mailto:gg@nue.ci.i.u-tokyo.ac.jp. The errors reported by the authors after the original paper was published are corrected.

Following is a link to the slides used at ICML07 by original author(s), which may help understand this paper.

http://www.cs.ualberta.ca/~silver/research/presentations/files/sylvain-silver.pdf