人工知能学会第二種研究会資料
汎用人工知能研究会
SIG-AGI-025-08

再帰的な階層型強化学習 RGoal への サブルーチン例外終了機能の導入

Introducing Exceptional Termination of Subroutines to Recursive Hierarchical Reinforcement Learning RGoal

一杉裕志[1*]　中田秀基[1]　高橋直人[1]　竹内泉[1]　佐野崇[2]
Yuuji Ichisugi[1]　Hidemoto Nakada[1]　Naoto Takahashi[1]　Izumi Takeuti[1]　Takashi Sano[2]

[1] 産業技術総合研究所 人工知能研究センター
[1] National Institute of Advanced Industrial Science and Technology (AIST), AIRC
[2] 東洋大学 情報連携学部情報連携学科
[2] Faculty of Information Networking for Innovation And Design, Toyo University

Abstract: We previously proposed RGoal, a hierarchical reinforcement learning algorithm with recursive subroutine calls, but it had a strong limitation of only one subroutine exit. In this paper, we extend RGoal to allow subroutines to have multiple exits and also introduce an exception termination feature. Two versions of the algorithm, Monte Carlo and Sarsa, were implemented and tested on multiple test tasks. We also discussed problems with the current Sarsa version and proposed improvements.

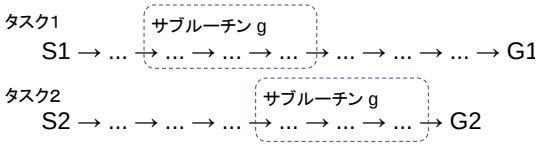

図 1: サブルーチンは様々な異なる文脈で呼び出されるが、サブルーチン内部の最適方策が文脈に依存せずほぼ同じであれば、学習結果を再利用して学習効率を上げられるはずである。

1　はじめに

我々は再帰的強化学習・生成モデル・プログラム合成の３つを中核技術とした脳型ＡＧＩアーキテクチャの早期実現を目指している [1]。

その一環として我々は以前、再帰的なサブルーチン呼び出しが可能な階層型強化学習アルゴリズム RGoal [2] （本稿では以降、旧 RGoal と呼ぶ）を提案した。

強化学習にサブルーチンを導入する動機の１つは、サブタスク共有による学習の加速である [3]。例えば、枝を切るときに使った「はしごをかける」という動作（サブタスク）を学習した結果得られたサブルーチンは、高いところのものを取る時や、電球を取り替える時などに再利用可能であろう（図 1)。汎用人工知能は未経験のタスクも少しの練習で解けるようになることが望まれるため、サブタスク共有による学習の加速は必要不可欠であろう。

旧 RGoal ではサブルーチンの出口が１つの状態のみという強い制限があった。しかし実用的なサブルーチンは、以下のように複数の出口の状態を持つことが普通である。

1. 推論や観測行動を行うサブルーチン。例えば「戸棚におかしがあるかどうか確認する」というサブルーチンの出口における状態は、「あることを確認した」「ないことを確認した」の２つである。
2. 目的の達成の仕方が複数あるサブルーチン。例えば「はしごを置く」というサブルーチンの終了状態は、しっかり置けている状態や、不安定な状態など、様々である。
3. 注目するサブゴール以外への副作用があるサブルーチン。例えば「おかしを食べる」という動作の結果の、まわりのちらかり具合など。

これらはサブルーチンが**正常終了**した結果の状態が複数ある例だが、より一般には**例外終了**も想定しておく必要がある。本稿では例外終了とは、サブゴールの達成が難しいと判断されたときにやむを得ず行うサブルーチンの終了のことを言う。例えば、「おかしを探したが

*連絡先：産業技術総合研究所
茨城県つくば市梅園１－１－１ 中央事業所
E-mail: y-ichisugi@aist.go.jp

見つからない」、「おかしを食べようとしたが袋が開けられない」、「親におかしの場所を教えてもらおうとしたが親も知らない」といった場合である。

本稿では、サブルーチンが複数の出口を持てる上、例外終了の機能も持つ場合の学習アルゴリズムについて述べる。モンテカルロ法と Sarsa の２つのバージョンのアルゴリズムを実装し、複数のテスト用タスクを用いて動作確認した。

提案アルゴリズムの重要な設計目標の１つは、正常終了を目指すべきか、別のサブルーチンを呼び出すべきか、あるいはただちに例外終了すべきかを、報酬最大化という原理のもとで、エージェントが合理的に意思決定できるようにすることである。この目標は、通常の行動、サブルーチン呼び出し、例外終了の行動価値をすべて同一の価値基準に基づいて表現することで達成される。

本稿は以下のような構成になっている。2 節で関連研究について述べ、3 節で旧 RGoal の問題点と解決方法、4 節で提案アルゴリズムについて説明し、5 節でテスト用タスクの説明、6 節で議論を行い、最後に 7 節でまとめを述べる。

2　関連研究

階層型強化学習は、強化学習の学習の加速（サンプル効率の向上）などを目的として研究されている [4][5]。

MAXQ [3] は２層以上の構造を持てる階層型強化学習アーキテクチャである。価値関数分解によるサブタスク共有と、時間抽象、状態抽象により、学習の加速を図っている。ただし階層構造は固定で、あらかじめ設計者が決める必要がある。他のほとんどの階層型強化学習アーキテクチャは２層構造である。我々が提案した RGoal は階層構造があらかじめ決められておらず、深さの制限なしにサブルーチンが他のサブルーチンを再帰的に呼び出せる点に特徴がある。

RGoal と同様に再帰的な呼び出しを可能にするアルゴリズムとして Recursive Q-learning (RQL) [6] が提案されている。このアルゴリズムは、再帰的な構造を持ったタスクを厳密に定式化する Recursive MDP (RMDP) に基づいている。RMDP の可能な応用としては、プログラム合成や確率的文脈自由文法の処理などが挙げられている。RQL ではサブルーチンが複数の出口を持つことを最初から想定しており、例外終了を特別扱いする必要はない。RQL では最も価値の低い出口の価値を基準値とした相対価値を学習することで厳密さを損なわずにパラメタ共有による学習速度の改善を図っている。本稿の提案アルゴリズムはこの研究に強く触発されているが、価値の厳密な学習は追求せず、代わりにより多くのパラメタ共有を行っている。

例外終了に似た概念としては、サブルーチンの早期終了がある。HDG [7] では、ゴールの中間目標となるランドマークを目指している途中であっても、よりよいランドマークに中間目標を切り替える行動をとり得る。一般に早期終了を許す階層型強化学習アーキテクチャは、そうでないものより行動の自由度が高いため、より優れた行動系列が得られる。しかし、この行動は、サブゴール到達が困難な時に選択される例外終了とは性質が異なるものである。

提案アルゴリズムは、プロダクションシステムの構造を持った認知アーキテクチャの実装に用いることを計画している [1]。似た構造の認知アーキテクチャとしては ACT-R [8], Soar [9] が有名である。認知アーキテクチャ Soar のルール選択に強化学習を用いる方法が提案されている [10] が、階層構造は持っていない。

ChatGPT [11] は大規模言語モデルを Reinforcement Learning from Human Feedback (RLHF) という手法でファインチューニングすることで、適切に質問応答できるようにしたシステムである。強化学習がヒトに近い知能の一側面を実現できることを示した画期的な例であると言える。しかし大規模言語モデル GPT-4 の初期バージョンに対して、複雑な行動計画、計算、推論の能力は十分でないという指摘もある [12]。再帰的なサブルーチンを実行する機構は、これらのタスクに対し有効となる可能性があるだろう。

3　旧 RGoal の問題点と解決方法

3.1　RGoal の概要

RGoal は深さの制限なくサブルーチン呼び出しができる階層型強化学習アルゴリズムである。旧 RGoal では、任意の状態からサブゴール g に向かう方策を「サブルーチン g 」と呼ぶ。エージェントは普通の行動の他に、サブルーチン呼び出し call(g) という行動を常に選択可能である。エージェントは内部状態としてサブゴール g とサブゴールのスタックを持つ。エージェントがサブゴール g に到達したら、サブゴールの値はスタックから取り出された値に再設定される。

RGoal では、エージェントがサブゴールに到達したときに設計者が恣意的に設計した報酬を与えるようなことはしない。かわりに、すべての行動に対し、小さな負の報酬（コスト）が与えられるような状況を想定する。この想定の下では、いわゆる「スパースな報酬の問題」は存在しない。サブゴール到達時の報酬がなくても、エージェントはできるだけ小さいコストでサブゴールに到達しようとする。この小さな負の報酬は、生物や機械においては、単位時間あたりのエネルギー消費や、他の有用な行動（エサの捕食や安全の確保など）

に費やせるはずの時間を失うことによる機会損失として解釈でき、定量的な設計が原理的には可能である。

RGoal は異なる新しいタスクが次々に与えられるようなマルチタスクの状況（すなわち生物や汎用人工知能がおかれる状況）において、少ない経験で近似解を得ることと、それにより高いオンライン性能を得ることを目的としている。多くの強化学習研究は膨大な時間をかけた網羅的な探索によって少しでもよい解を得ることを目的としているが、本研究ではそれは目的ではない。

RGoal の設計の重要な動機の１つは、Prolog 言語のような再帰的な推論の強化学習による実現である [13]。Prolog 言語は後ろ向き連鎖によって、ゴールとなる命題を証明するために、必要なサブゴールを再帰的に証明していく。RGoal のサブゴールも、似た目的で使うことを想定している。

我々は RGoal をヒトの脳の前頭前野の計算論的モデルの候補と考えており、神経科学的妥当性の高さも設計の重要な目標である。

旧 RGoal ではサブゴール g は単一の状態であったが、本稿では他の多くの階層型強化学習アーキテクチャ [4][5] と同様に、g を状態の集合とする。したがってサブルーチン g は「**任意の状態からサブゴール集合 g のいずれかの要素に向かう方策**」として再定義される。なお RGoal ではサブゴール集合がサブルーチンの識別子そのものである。理論上は、サブゴール集合の数だけサブルーチンが存在する。

本稿で導入するもう１つの機能拡張が例外終了である。エージェントは exit(s') という行動をいつでも取ることができる。この行動をとると、現在のサブルーチンはただちに実行を終了し、サブゴールはスタックから取り出した値に設定され、状態は s' に強制的に遷移する[1]。

3.2 旧 RGoal の問題点

旧 RGoal では、行動価値関数（ゴールに到達するまでの累積報酬期待値）をサブゴール到達の前と後に分割し、サブゴール到達前の報酬の総和 $Q(s,g,a)$ だけを学習し、異なるタスク間で共有することで学習の加速を実現した。しかし g を状態の集合に拡張した場合、

1. 出口が複数あるときはサブゴール到達後の累積報酬期待値の計算が簡単でない。
2. 例外終了への対処方法が自明でない。

という問題がある。この問題を解決するには、行動価値関数 $Q(s,g,a)$ の定義を見直す必要がある。

[1]本稿ではアルゴリズムのテストを容易にするために、exit の引数 s' に任意の値を指定できることとした。実際に応用する際は、環境の状態 s は変えずに exit する仕様の方が妥当と思われる。

3.3 解決方法

図 1 に示したように、サブルーチンは様々な異なる文脈で呼び出されるが、サブルーチン内では文脈にあまり依存しない何らかの**基準値**に対する**相対価値**を学習することで、学習結果を再利用できるようになる。

よい基準値の定義には、学習が容易なこと、収束が速いこと、環境が変化しても基準が比較的安定していること、などの性質が求められる。本稿では１つの案として、正常終了時の状態の平均価値を基準値として用いることを提案する。サブルーチンは少数のサブゴールのいずれかに到達して正常終了するので、正常終了時の状態の価値が安定した基準値としてふさわしいと考えられるからである。

具体的な実現方針としては、行動価値関数 $Q(s,g,a)$ を学習すると同時に、サブルーチン内の価値の基準値を呼び出しの文脈ごとに学習する。基準値は $Base(s0,g0,a0)$ で表現される。これは、状態 $s0$, サブゴール $g0$ のときに、サブルーチン g を呼び出す行動 $a0 = \mathrm{call}(g)$ を選択したという文脈における g 内の基準値を表す。

この方針に基づいて設計した行動価値関数の定義と学習アルゴリズムを、次の 4 節で説明する。

4 提案アルゴリズム

4.1 モンテカルロ法と Sarsa

実装にはモンテカルロ法と Sarsa を用いた。この２つについて以下に簡単に説明する。（詳細は [14] 参照。）

モンテカルロ法はエピソードが終了した時に、得られた実行サンプル（各ステップごとの状態、行動、報酬の履歴）を用いて、エピソード中で使用されたすべての行動価値を一度に更新するアルゴリズムである。１つの実装方法としては、下記の学習則で行動価値関数 $Q(s_t,a_t)$ を学習する。

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha(G_t - Q(s_t,a_t)) \qquad (1)$$

ただし G_t は、この実行サンプルにおいて状態 s_t で行動 a_t をとった直後からエピソード終了までに得られた報酬の総和で、$G_t = r_t + r_{t+1} + \cdots$ である。（本稿では報酬割引は行わない。すなわち割引率 $\gamma = 1$ である。）

Sarsa は状態 s で行動 a をとって報酬 r が得られた時、次のステップの状態と行動を s', a' とすると

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + Q(s',a') - Q(s,a)) \qquad (2)$$

という学習則で行動価値関数 $Q(s,a)$ を学習するアルゴリズムである。

Sarsa を一般化したものとして、適格度トレースを用いた Sarsa(λ) がある。使用された直後の状態行動対

には高い適格度を持たせる一方、すべての適格度を時間とともに減衰係数λで減衰させる。行動の1ステップごとに、すべての状態行動対を、それぞれの適格度に応じた強さで学習する。普通の Sarsa は Sarsa(0)（λ=0 で適格度が瞬時に減衰する）であり、モンテカルロ法は Sarsa(1)（λ=1 で適格度が全く減衰しない）であると考えることができる。

一般に性能を最大にするλの値はタスクごとに異なる。Sarsa(λ) は神経回路による実現が比較的素直に行えるため、生物の脳内でも使われている可能性があるだろう。本稿では手始めとして、実装が容易なモンテカルロ法と普通の Sarsa のみを実装した。

4.2 モンテカルロ法による学習

状態 $s0$、サブゴール $g0$ において行動 $a0$ =call(g) を実行したという文脈[2]におけるサブルーチン g 内の相対価値の基準値を

$$Base_\pi(s0, g0, a0) \equiv E_\pi[G_{Base}] \tag{3}$$

と定義する。ただし、E_π は方策 π のもとで得られた実行サンプルのもとでの平均、G_{Base} は方策 π の実行サンプルにおいて、文脈 $(s0, g0, a0)$ で呼び出されたサブルーチン g が正常終了してからエピソードが終了するまでの累積報酬である（図 2）。例外終了した場合はこの基準値に寄与しない。（学習に使う実行サンプルにおいて少なくとも一度は正常終了するものと仮定する。一度も正常終了が起きない場合は基準値が定義できない。）

サブルーチン g 内の状態 s において行動 a を選択する相対価値を表現する行動価値関数の値は、

$$Q_\pi(s, g, a) \equiv E_\pi[G - Base_\pi(s0, g0, a0)] \tag{4}$$

と定義される。ここで G はエピソード終了までの累積報酬である。例外終了の場合 $(a = \text{exit}(s'))$ であっても同じ定義式が用いられる（図 2）。サブルーチンは方策 π のもとでの様々な文脈で呼び出されるが、行動価値関数はあらゆる呼び出しにおける相対価値の平均である。

以上の定義に近づくように基準値と行動価値関数を学習する疑似コードを図 3 に示す。

なお $Base_\pi(s0, g0, a0)$ と $Q_\pi(s, g, a)$ は方策 π だけでなく、学習に用いるタスクの確率分布 $P(S, G)$（S はエピソードの初期状態、G はゴール集合）にも依存する。訓練時 $P^{Training}(S, G)$ のもとでの学習結果がテスト時 $P^{Test}(S, G)$ にもある程度汎化すること、もしくは2つの分布が同一であることが、この提案アルゴリズムを利用する前提である。

[2]モンテカルロ法においては文脈情報に g0 は必須ではないが、本稿では後述の Sarsa と形をそろえるために入れている。

4.3 Sarsa による学習

本節では Sarsa による学習アルゴリズムについて述べるが、現状版には5節で述べるように若干の問題がある。この問題の解決の方針については 6.1 節で述べる。

Sarsa は実行中にオンラインで学習するアルゴリズムであるため、モンテカルロ法のようにエピソード終了までの累積報酬 G が利用できない。そのため基準値は「正常終了時の状態の呼び出し元における相対価値の平均」と定義する。形式的には、状態 $s0$、サブゴール $g0$ において行動 $a0 = \text{call}(g)$ を実行したという文脈におけるサブルーチン g 内の基準値は

$$Base_\pi(s0, g0, a0) \equiv E_\pi[r + Q_\pi(s', g', a')] \tag{5}$$

と定義される（図 4）。ただし r, s', g', a' は、方策 π における実行サンプルにおいて、文脈 $(s0, g0, a0)$ で呼び出されたサブルーチン g が正常終了した時の報酬と直後の状態、サブゴール、行動である。（このとき $g' = g0$ である。）また、$Q_\pi(s', g', a')$ はサブルーチン g' における状態 s' での行動 a' の相対価値である。行動価値関数 $Q_\pi(s, g, a)$ は形式的には以下に定義される。

サブルーチン g の内部から再帰的にサブルーチン g' 呼び出す時（$a = \text{call}(g')$）の行動価値は、呼び出しの前後で行動価値関数の基準値が変わることに配慮して、

$$\begin{aligned} &Q_\pi(s, g, \text{call}(g')) \\ &\equiv E_\pi[r + Q_\pi(s', g', a') + Base_\pi(s, g, a)] \end{aligned} \tag{6}$$

と定義される（図 5）。

RGoal では $s \in g$ となったときにサブルーチンは正常終了する。このときの行動の相対価値は基準値そのものであり常に0である。

$$Q_\pi(s, g, RET) \equiv 0 \quad (s \in g) \tag{7}$$

文脈 $(s0, g0, a0)$ で呼び出されたサブルーチン g が例外終了するときの行動価値（$a = \text{exit}(s')$）は、

$$\begin{aligned} &Q_\pi(s, g, \text{exit}(s')) \\ &\equiv E_\pi[r + Q_\pi(s', g', a') - Base_\pi(s0, g0, a0)] \end{aligned} \tag{8}$$

と定義される（図 6）。

それ以外の場合は基準値の切り替えがないことから、

$$Q_\pi(s, g, a) \equiv E_\pi[r + Q_\pi(s', g', a')] \tag{9}$$

と定義される。

以上の定義に近づくように基準値と行動価値関数を学習する疑似コードを図 7 に示す。

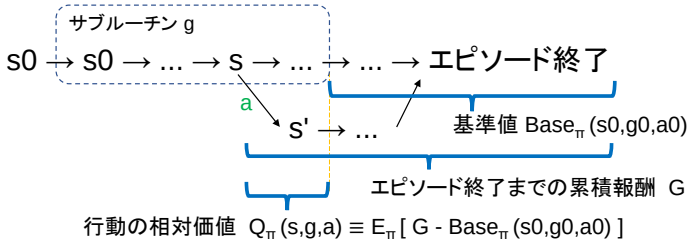

図 2: モンテカルロ法の基準値と、例外終了時の行動価値の定義

5 テスト用タスク

提案アルゴリズムの動作を確認するため、１０個程度の小さなテスト用のタスクを作成して、学習結果が予想どおりになることを確かめた。テスト用タスクには、サブルーチンが自分自身を再帰呼び出しするものやプログラムが exit したあと即座に再 exit するものが含まれる。

ここではテスト用タスクの１つを示す。また現状の Sarsa 版のアルゴリズムの問題点について説明する。

図 8 は、サブルーチンの正常終了を目指すか、別のサブルーチンを呼び出すか、例外終了するかを合理的に選択することを確認するテスト用タスクである。３つの経路を選択するところでのみエージェントの意思決定が必要で、他の状態遷移は１本道でゴールに向かうものとする。太い矢印の行動コスト（負の値の報酬）は可変、他は固定で、デフォルトではすべての行動コストは -1 とする。エージェントは学習の結果、最小コストの経路を高い確率で選択することが期待される。デフォルトでは３つの経路のコストは同じである。

３つの経路をほぼ等確率で選択する方策のもとでの行動価値を、図 9 はモンテカルロ法、図 10 は Sarsa で学習した結果である。学習率 $\alpha = 0.001$ 、逆温度 $\beta = 0.001$ で１０万エピソードを学習した結果の各行動の相対価値の小数点第二位を四捨五入したものを矢印の上に図示した。

このタスクでは、図 9 のモンテカルロ法では正しい相対価値が学習されている。３つの経路選択の相対価値はどれも -3.5 となっている。（エピソード終了までのコストはそれに基準値 -3.5 を足した値 -7 である。）一方、図 10 の Sarsa では３つの選択肢の相対価値はそれぞれ -3, -4, -3.5 となっている。Sarsa では大域的な文脈を見ないでサブルーチンを正常終了するまでの局所的なコストを学習するため、もっとも低いコストでサブルーチン {S1, S2} を抜ける経路の価値が最も高くなっている。

次に逆温度 $\beta = 1$ として、価値を学習しつつその結果を用いて行動選択させたときにどうなるかを確認した。モンテカルロ法では、３つの経路のうち１つのコストを下げたら、その経路が他より高い確率で選択されることが確認された。一例として P から R2 への状態遷移コストを -1 から -0.5 に変更した場合のモンテカルロ法での学習結果を図 11 に示す。３つの選択肢の行動価値はそれぞれ -3.6, -3.1, -3.6 と学習され、ゴールまでのコストが行動価値に正しく反映されている。

一方、おなじ条件での Sarsa の学習結果を図 12 に示す。この場合は３つの選択肢の価値は -3, -3.5, -3.4 となり、最も低コストな経路でない、１つめの経路の価値が最も高くなっている。この問題に対する改良案は 6.1 節で述べる。

6 議論

6.1 Sarsa による学習アルゴリズムの改良案

現状の Sarsa 版でサブゴール到達後の報酬が反映されない問題に対処するためには、正常終了する行動の相対価値を一律に 0 とするのではなく、正常終了直後の行動の価値 $Q(s', g', a')$ を反映させるようにすればよいだろう。そのような改良を現在計画している。

本稿では説明していないが、テスト用タスクで確認されている別の問題もある。環境の状態を抽象化して表現する場合や POMDP の状況では、普通の Sarsa では、状態の価値がうしろに正確に伝搬しないことがあるという問題である。このこともまた、長期のコストを無視し短期的な利益を優先した行動を引き起こす。この問題は適格度トレースを導入することで改善するはずである。

このような改良は実用上必要であると考える。短期的にはサブルーチンを速く終了できても、のちのち不都合を引き起こすような「間違った推論」や「間違った行動」の価値は下がることが望ましい。例えば、算数のテストで計算をさぼって答えを全部 0 と書くような行動は、テストは楽に終えられるがあとで困ることになる。

なお、筆者らは [13] で推論規則の価値を強化学習で学習する機構を提案したが、そこで用いた学習アルゴリズムは旧 RGoal に基づくもので不完全であった。そこで提案アルゴリズムおよびその改良版を用いて、よ

り複雑なタスクを用いることで「間違った推論」や「間違った行動」の価値が適切に下がることを今後示していく予定である。

6.2 パラメタ共有がうまくいかないサブルーチンの扱い

本稿ではサブルーチン内部の最適方策が文脈に依存せずほぼ同じであることを仮定したが、任意に決めたサブゴール集合に対してそのような性質が成り立つとは考えにくい。将来の汎用人工知能においては、有用なサブゴール集合（サブルーチン）が対話や模倣によって得られることが多いと想定しているが、そうでない場合は、有用なサブゴール集合とそうでないサブゴール集合を自らの経験のみで選別する機構が別途必要になるだろう。

サブルーチン内部の最適方策が文脈に強く依存する場合は、文脈ごとに細かくサブルーチンを分けて個別に方策を学習すれば学習で得られる解は優れたものになるが、学習すべきパラメタは増えて学習速度は遅くなる。つまり、この問題は、パラメタが多すぎると訓練データへのフィッティングはよくなるが汎化性能が落ちるという、機械学習でよく見られる現象の一例であると解釈できる。したがってこの問題に対処するには、交差確認法のようなことをオンラインで行いモデル選択するなどの方法が考えらえれる。

6.3 相対価値の安定性

テスト用タスクで学習中の行動価値関数の値の変化を可視化すると、Base と Q の学習速度の不一致により、相対価値が一度大きく下がった後ゆっくり上がるなど、不安定な動きが観察されることがあった。この現象はオンライン性能に悪い影響を及ぼす可能性がある。この問題は、 Base と Q のテーブルの各要素ごとに個別に学習率を持たせ、それぞれ独立にスケジューリングすることで緩和するかもしれない。

6.4 Fail の価値

我々は以前、 RGoal にスタックを破棄する Fail という行動を追加し、その価値の学習則を提案した [13] が、本稿が仮定するようなサブルーチンが異なる文脈で呼び出され得る場合においては、価値が意味のある値にならないことが懸念される。スタックを一度に破棄する Fail 命令は廃止し、サブルーチンを１つずつ順に exit することで代替することを検討中である。

6.5 生物学的妥当性

提案アルゴリズムは理論的妥当性とシンプルさを優先しており、生物学的妥当性の検討は現時点では十分ではない。

第一に脳内に深さ制限のないスタックがあるとは考えにくい。脳はスタックの機能をエミュレーションするなんらか機構を持っているのだろうと推測している。

基準値と相対価値の学習が神経回路で可能なのかも疑問がある。しかし、大脳皮質は確率的生成モデルを学習する器官であるという前提に立てば、行動価値関数を適切なバイアスのもとで生成モデルとして学習すれば、基準値と相対価値は独立な因子としてそれぞれ自然に学習されるかもしれないと考えている。

7 まとめ

本研究では、まず再帰的な階層型強化学習 RGoal を複数の出口を持てるように拡張した。また、サブゴールに到達せずにサブルーチンを抜けるための例外終了のための行動 exit(s') を導入した。そして、モンテカルロ法と Sarsa に基づく２つのアルゴリズムを実装し、複数のテスト用タスクを用いて、学習結果が想定される値になっていることを確認した。また、現状の Sarsa アルゴリズムの問題点と改良案について考察した。

今後はより複雑なデモタスクを用いて提案アルゴリズムの有用性を示していく予定である。

謝辞

本研究は JSPS 科研費 JP22K12188 の助成を受けたものです。

参考文献

[1] 一杉裕志. 報酬最大化原理にもとづく脳型ＡＧＩアーキテクチャの構想. 第 18 回 人工知能学会 汎用人工知能研究会 (SIG-AGI), 2021.

[2] Yuuji Ichisugi, Naoto Takahashi, Hidemoto Nakada, and Takashi Sano. Hierarchical reinforcement learning with unlimited recursive subroutine calls. In *Artificial Neural Networks and Machine Learning – ICANN 2019: Deep Learning*, pp. 103–114, Cham, 2019.

[3] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*, Vol. 13, pp. 227–303, 2000.

[4] Shubham Pateria, Budhitama Subagdja, Ah-hwee Tan, and Chai Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Comput. Surv.*, Vol. 54, No. 5, jun 2021.

[5] Matthias Hutsebaut-Buysse, Kevin Mets, and Steven Latré. Hierarchical reinforcement learning: A survey and open research challenges. *Machine Learning and Knowledge Extraction*, Vol. 4, No. 1, pp. 172–221, 2022.

[6] Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. Recursive reinforcement learning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[7] Leslie Pack Kaelbling. Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the tenth international conference on machine learning*, Vol. 951, pp. 167–173, 1993.

[8] ACT-R 7 Tutorial Units. `http://act-r.psy.cmu.edu/software/`, 2020.

[9] Soar Tutorial 9.6.0. `https://soar.eecs.umich.edu/articles/downloads/soar-suite/228-soar-tutorial-9-6-0`, 2017.

[10] Shelley Nason and John E. Laird. Soar-RL: integrating reinforcement learning with soar. *Cognitive Systems Research*, Vol. 6, No. 1, pp. 51–59, 2005.

[11] Introducing ChatGPT. `https://openai.com/blog/chatgpt/`, 2022.

[12] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with GPT-4. *arXiv preprint arXiv:2303.12712*, 2023.

[13] 一杉裕志, 中田秀基, 高橋直人, 佐野崇. 推論規則の価値を階層型強化学習 RGoal を用いて学習する手法の提案. 第 14 回 人工知能学会 汎用人工知能研究会 (SIG-AGI), 2020.

[14] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

```
1: procedure EPISODE(S, G)
2:     # Init and choose the first action rule.
3:     ruleHistory, callerHistory, rewardHistory ← empty
4:     s ← S;  g ← G;  stack, callerStack ← empty
5:     stack.push(G) ;  caller ← StartRule ;  callerStack.push(dummy)
6:     Choose rule from s, g using policy derived from Q
7:     while stack is not empty do
8:         # Take action.
9:         a ← getAction(rule)
10:        if a = RET then
11:            s' ← s;  g' ← stack.pop();  reward ← R^Return
12:        else if a is exit(m) then
13:            s' ← m;  g' ← stack.pop();  reward ← R^Exit
14:        else if a is call(m) then
15:            stack.push(g);  s' ← s;  g' ← m;  reward ← R^Call
16:        else
17:            Take action a, observe reward and s';  g' ← g
18:        # Choose action rule.
19:        if s' ∈ g' then
20:            rule' ← ReturnRule
21:        else
22:            Choose rule' from s', g' using policy derived from Q
23:        # Remember history for Update.
24:        ruleHistory.push(rule) ;  rewardHistory.push(reward) ;  callerHistory.push(caller)
25:        if a = RET then
26:            caller ← callerStack.pop()
27:        else if a is exit(m) then
28:            caller ← callerStack.pop()
29:        else if a is call(m) then
30:            callerStack.push(caller)
31:            caller ← rule
32:        #
33:        s ← s';  g ← g';  rule ← rule'
34:    Update(ruleHistory, callerHistory, rewardHistory)
35:
36: procedure UPDATE(ruleHistory, callerHistory, rewardHistory)
37:    G ← 0    # Total reward until the end of the episode.
38:    while ruleHistory is not empty do
39:        rule ← ruleHistory.pop() ;  caller ← callerHistory.pop() ;  G ← G + rewardHistory.pop()
40:        if rule = ReturnRule then
41:            Base(caller) ← Base(caller) + α(G − Base(caller))
42:        else
43:            Q(rule) ← Q(rule) + α(G − Base(caller) − Q(rule))
```

図 3: モンテカルロ法による学習の疑似コード。ここで rule とは (s, g, a) の組を表すデータ構造である。ReturnRule は定数で getAction(ReturnRule)=RET, Q(ReturnRule)=0 とする。StartRule はエピソードを開始するダミーの行動ルールを表す定数。$R^{Call}, R^{Return}, R^{Exit}$ はそれぞれサブルーチン呼び出し、 RET、 exit の報酬（実行コスト）を表す定数。

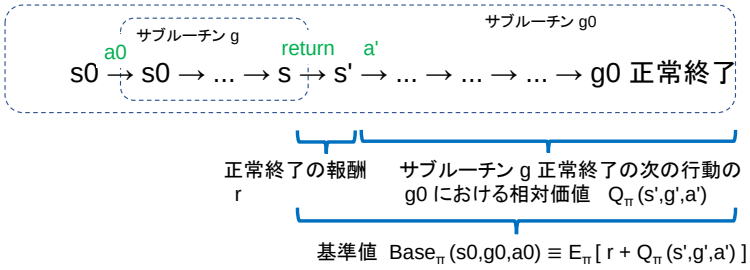

図 4: Sarsa で学習する時の基準値の定義

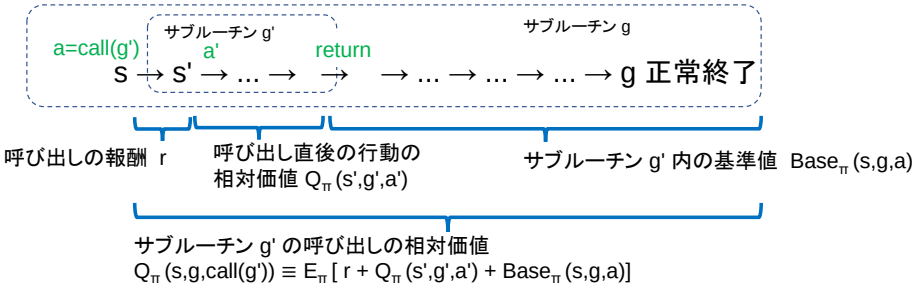

図 5: Sarsa で学習する時のサブルーチン呼び出しの行動価値の定義

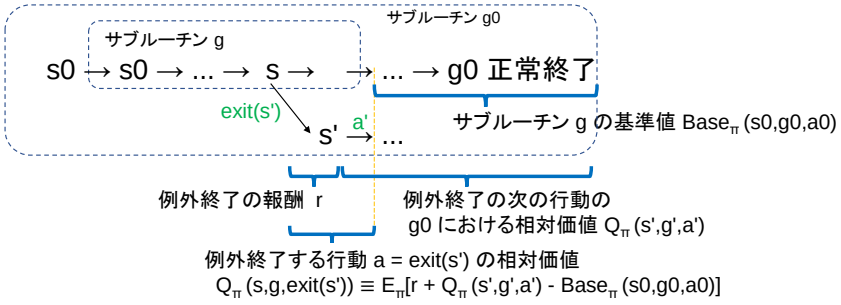

図 6: Sarsa で学習する時の例外終了の行動価値の定義

```
1: procedure EPISODE(S, G)
2:     # Init and choose the first action rule.
3:     s ← S;  g ← G;  stack, callerStack ← empty
4:     stack.push(G) ;  caller ← StartRule ;  callerStack.push(dummy)
5:     Choose rule from s, g using policy derived from Q
6:     while stack is not empty do
7:         # Take action.
8:         a ← getAction(rule)
9:         if a = RET then
10:            s' ← s;  g' ← stack.pop();  reward ← R^Return
11:        else if a is exit(m) then
12:            s' ← m;  g' ← stack.pop();  reward ← R^Exit
13:        else if a is call(m) then
14:            stack.push(g);  s' ← s;  g' ← m;  reward ← R^Call
15:        else
16:            Take action a, observe reward and s';  g' ← g
17:        # Choose action rule.
18:        if s' ∈ g' then
19:            rule' ← ReturnRule
20:        else
21:            Choose rule' from s', g' using policy derived from Q
22:        # Update.
23:        if a = RET then
24:            Base(caller) ← Base(caller) + α (reward + Q(rule') − Base(caller))
25:            caller ← callerStack.pop()
26:            # Do not update Q(rule) because Q(ReturnRule) is always 0 .
27:        else if a is exit(m) then
28:            Q(rule) ← Q(rule) + α (reward + Q(rule') − Base(caller) − Q(rule))
29:            caller ← callerStack.pop()
30:        else if a is call(m) then
31:            callerStack.push(caller)
32:            caller ← rule
33:            Q(rule) ← Q(rule) + α (reward + Q(rule') + Base(rule) − Q(rule))
34:        else
35:            Q(rule) ← Q(rule) + α (reward + Q(rule') − Q(rule))
36:        #
37:        s ← s';  g ← g';  rule ← rule'
```

図 7: Sarsa による学習の疑似コード

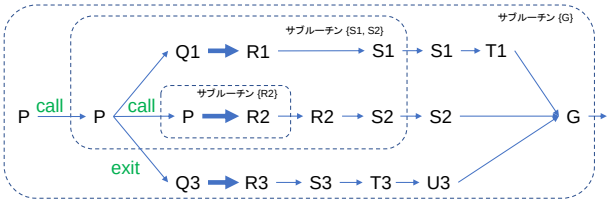

図 8: テスト用タスクの 1 つ

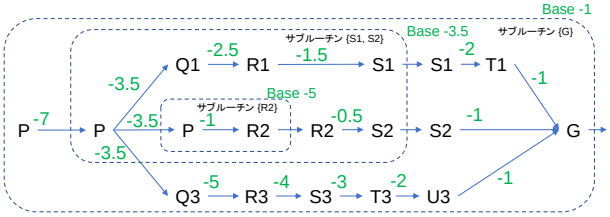

図 9: 3 つの経路を等確率で選択する方策における行動価値をモンテカルロ法で学習した結果

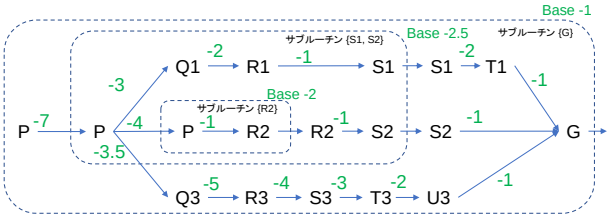

図 10: 3 つの経路を等確率で選択する方策における行動価値を Sarsa で学習した結果

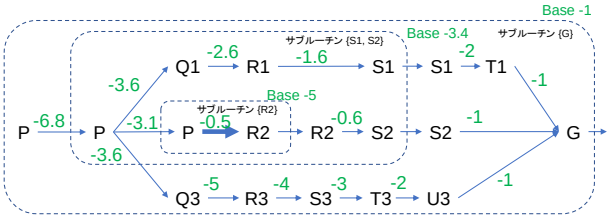

図 11: 2 つめの経路のコストが小さい場合の行動価値をモンテカルロ法で学習した結果

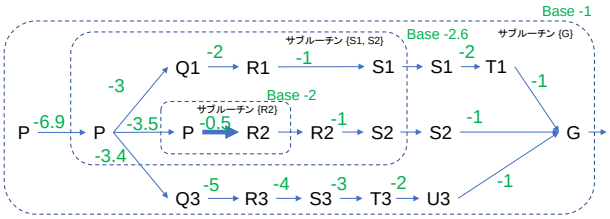

図 12: 2 つめの経路のコストが小さい場合の行動価値を Sarsa で学習した結果