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-026-08

モンテカルロ版 RGoal アルゴリズムの改良

Improvements to the Monte Carlo version of RGoal algorithm

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 a hierarchical reinforcement learning algorithm, RGoal, that allows recursive subroutine calls. In this paper, we improve the definition of the reference value for relative value in the Monte Carlo version of RGoal in order to stabilize learning when subroutines are shared between different tasks. The implemented algorithm was confirmed to work in several test tasks.

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、再帰的強化学習 RGoal を例外終了機構を持つように拡張したものを提案した [2]。本稿ではそこで述べたモンテカルロ版のアルゴリズムを改良し、異なるタスクでサブルーチンを再利用した時に生じる問題に対処する。

また、我々が以前に提案した行動計画の機構 [3] を新しい RGoal にあわせて再実装し、簡単な動作テストを行った。

2　モンテカルロ版 RGoal の改良

RGoal は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である。以前提案したアルゴリズム [2] の基本アイデアは、サブルーチンを正常終了した時点での状態の価値を基準値として、サブルーチン内の相対価値を学習するというものであった。基準値は呼び出し文脈ごとに異なるが、相対価値は文脈にあまり依存しないため、その学習結果を異なるタスクに再利用できることが期待できる。

しかし [2] で提案した２つのアルゴリズムのうちモンテカルロ版の方には、基準値と文脈が正しく対応付けられていないという問題があった。具体的には、個々の基準値がタスク全体のゴールにも依存するにも関わらず、その文脈を無視し、同じ値として学習してしまう。そのため異なるタスクで同じサブルーチンを共有する際、基準値の学習結果を再利用できない場合がある。そこで、基準値をタスクには依存せず、局所的な文脈のみに依存するような定義にしたい[1]。

そのために基準値と相対価値の定義を以下のように改める（図 1）[2]。

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

$$Base_\pi(s0, g0, a0) \equiv E_\pi[r + Q(s', g', a')] \qquad (1)$$

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、r はサブルーチンから return するステップの実行コスト、$Q(s', g', a')$ は return 直後の行動価値のサブルーチン $g0$ における相対価値である。（このとき $s' \in g$, $g' = g0$ である。）

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は、

$$\begin{aligned} Q_\pi(s, g, a) \quad &\equiv \\ E_\pi[T \quad &- \quad Base_\pi(s0, g0, a0) \\ &- \quad \text{<all } Base(caller) \text{ in the caller stack>}] \end{aligned} \qquad (2)$$

*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

[1] タスク全体のゴール集合を G とすると、基準値を区別する文脈を $(s0, g0, a0)$ ではなく $(s0, g0, a0, G)$ とするのが１つの解決策かもしれないが、それでは基準値の再利用ができないという問題がある。本稿では異なる解決策を取っている。

[2] 基準値の新しい定義は [2] の Sarsa 版のための定義と同一であるが、相対価値の定義は大きく異なる。

と定義される。ここで T はエピソード終了までの累積報酬であり、そこから呼び出しスタック内にあるすべての呼び出しに対応する基準値を引いたものが、呼び出し文脈にあまり依存しない相対価値となる。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 $Q_\pi(s,g,a)$ 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。

図 2 は提案アルゴリズムの動作確認のためのテスト用タスクの動作とその学習結果である。提案アルゴリズムと旧アルゴリズムの動作の違いを確認するためには、異なるタスクで１つのサブルーチンが共有されていて、そのサブルーチンがさらに別のサブルーチンを呼び出していることが必要である。

タスク１は状態 P1 から G1、タスク２は P2 から G2 への状態遷移を目指すタスクであり、その途中でどちらも状態 T1 をサブゴールとするサブルーチン {T1} を呼び出している。各ステップの状態遷移には -1 の報酬が与えられる。サブルーチン {U4} の基準値は、旧モンテカルロ版アルゴリズム [2] では、タスク１の文脈では -7, タスク２の文脈では -5 となってしまう。提案アルゴリズムでは基準値は常に -2 であり、設計目標どおり、タスクに依存しない基準値が学習できている。

3 行動計画を行う Pro5Lang プログラム

AGI エージェントが再利用性の高い知識を経験を通じて環境から学習できるようにすることは、我々の再帰的強化学習アルゴリズムの重要な設計目標の１つである。提案アルゴリズムがこの目標に必要な性質を持つことを、我々が以前に提案した行動計画のタスク [3] を用いて確認した。このタスクでは、エージェントがあらかじめ持っている地図に関する知識を用いて、スタートからゴールまで到達するために必要な中間目標を推論し、そのあとエージェントはその中間目標を経由して実際にゴールまで移動する。ここでもし学習アルゴリズムが我々が期待する性質を満たしているならば、最初に間違った地図の知識を持っていた場合でも、何度も経験を繰り返すことで、間違った知識の価値が下がり、正しい知識のみが利用されるようになるはずである。

行動計画には様々な戦略があり得るが、本稿では以下のようなシンプルな前向き連鎖による推論を用いる[3]。まず「状態 s のときサブゴール g' を設定すればいつかゴール g を達成できる」という命題を $F(s,g',g)$ と表記する。また、「状態 s から g を達成できる」という命題を $A(s,g)$ と表記し、エージェントは経験から得たこの形の「公理」をあらかじめ蓄えているものとする。エージェントは与えられたゴールを達成するために、以下の推論規則を繰り返し適用し、スタートからゴールまでの達成可能性を証明する。

$$A(g',g''),F(s,+,g') \vdash F(s,g',g'')$$

この推論規則は、「状態 g' から g'' を達成可能かつ s から g' を達成可能ならば s から g' を経由して g'' を達成可能である」ということを意味している。この推論によって、最初に実行すべきサブルーチン g' が得られる。エージェントは、サブルーチン g' を実行してサブゴール g' に到達したら、あらためて行動計画を立てて結果を実行するということを、ゴールに到達するまで繰り返す。

エージェントの脳内でこの行動計画戦略を実行するプログラムは、我々がプログラム合成の合成対象言語として設計を進めている Pro5Lang [4] を用いて実装されている[4]。行動計画を行う Pro5Lang プログラムの疑似コードを図 4 に示す。Pro5Lang プログラムは行動ルール $\mathrm{rule}(s,g,a)$ の集合として定義される。実行の１ステップごとに、エージェントの現在の状態 s とサブゴール g にマッチする s,g のパターンを持つ行動ルールが１つ非決定論的に選択され、その行動ルールが指定する行動 a が実行される。

図 5 は、テスト用環境の地図 (図 6) に関する知識を定義する行動ルール集合の疑似コードである。R1, R2, R3, R4 はそれぞれエージェントが部屋 1,2,3,4 にいる状態を表すとする。１行目の行動ルールは「R1 にいるとき R2 に移動するには Goto(R2) というプリミティブを実行する」という意味である。エージェントは R1 → R2, R2 → R4, R1 → R3, R3 → R4 のいずれかの移動しか行えず、R1 から R4 に行くためには、R1 → R2 と R2 → R4 、または R1 → R3 と R3 → R4 の移動を組み合わせる必要がある。６行目から９行目は行動計画に使われる知識であり、例えば６行目はエージェントが命題 $A(R1,R2)$ が成り立つことを知っていることを意味している。図 4 の５行目において call(A(R1,+)) というサブルーチンが呼び出しが実行されると、図 5 の６行目か８行目の行動ルールのいずれかが非決定論的に選択される。

このテスト用タスクのメインループは図 7 のプログラムで定義される。このプログラムは、与えられたゴール g に到達するまで、行動計画と行動を繰り返す。行動計画中、経路をスタート地点から前向きにたどっていき、それ以上進めなくなったときは図 5 の５行目のデフォルトの行動ルールが選択され Failed という状態

[3] 以前 [3] で提案したアルゴリズムは後ろ向き連鎖を用いていた。

[4] 本来はこの行動計画プログラム自体も、エージェントが経験から自律的に獲得することを想定している。

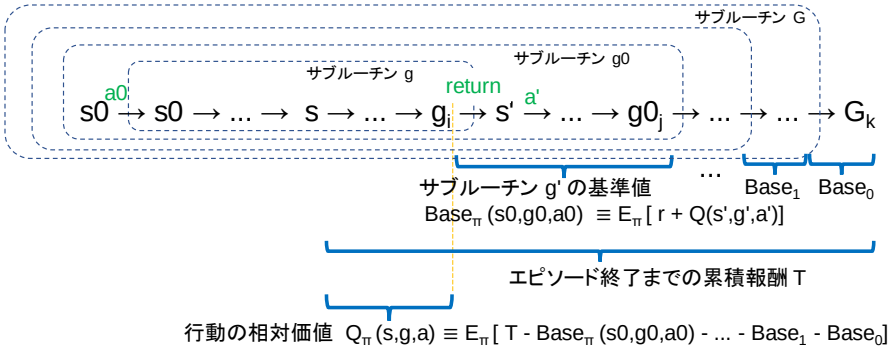

図 1: モンテカルロ法で学習する場合の行動価値の基準値と相対価値の定義

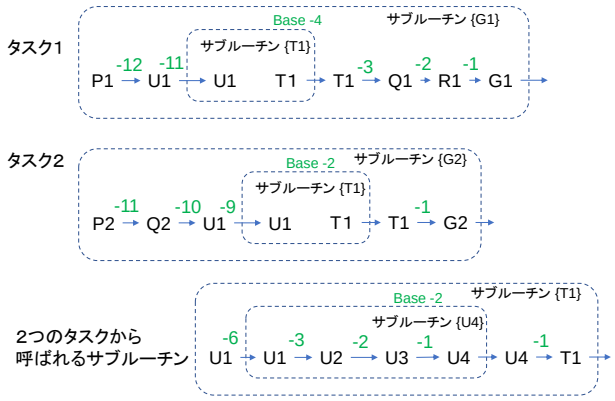

図 2: 改良アルゴリズムのテスト用タスクと学習結果

になる。これを受けて図 4 の 2 行目が実行され、行動計画プログラムの呼び出し側に制御が戻される。呼び出し側のプログラム（図 7）は必要に応じて繰り返し行動計画プログラムを呼び出す（2，3 行目）ことにより、いつかはゴールに到達可能であることが（もし到達可能であれば）証明され、推論された中間目標 a が 4 行目において実行される。それによりもしゴール g に到達すればタスクは終了、そうでなければ 2 行目に戻り、行動計画と行動をゴールに到達するまで繰り返す。

この行動計画プログラムは非決定論的に動作し、可能な中間目標の 1 つをランダムに選択する。この場合は選択される中間目標は R2 または R3 であり、R1 → R2 → R4 または R1 → R3 → R4 という経路でゴールに到着する。

ここで、エージェントがいる実際の環境において R1 → R2 という経路が通行不可であり、移動しようとするとスタート地点に戻されるとする[5]。この状況でエピソードを繰り返すと、図 5 における 1、6、7 行目の行動ルールの価値が下がり、R3 だけが中間目標に選択されるようになる。このように、モンテカルロ版 RGoal（もしくは適格度トレースを導入した RGoal）を用いることで、「間違った行動」だけでなく「間違った行動をのちに引き起こすことになる知識」の価値を下げ、使われなくすることができる。RGoal のこの特徴を用いることで、抽象度の高い宣言的知識や推論規則を、経験を通じて環境に接地させることが可能になると考えている。

なお、上記の設定おいて 6 行目の知識 $A(R1, R2)$ は確かに間違っているが、7 行目の知識 $A(R2, R4)$ は実際には間違っていないにもかかわらず価値が下がっている。これは、このテスト環境においては 2 つの知識が独立しておらず、必ず組にして使われるために生じる現象であると思われる。同じような現象が何らかのヒトの認知科学的実験により見出されるならば、ヒトの脳も RGoal と似たアルゴリズムを用いていることの証拠となるだろう。

4 関連研究

提案アルゴリズムの基準値の考え方は Recursive Q-learning (RQL) [5] に強く触発されている。ただし RQL は価値関数を厳密に学習することを重視しているが、我々の RGoal ではサブルーチンの抽象度を上げてパラメタ共有の機会を増やし、汎化性能を上げることを重視している。

改良版モンテカルロ法における価値関数の基準値と相対価値への分解は、MAXQ 価値関数分解 [6] とかなり似ている。MAXQ では行動価値関数は $Q(i, s, a) = V(a, s) + C(i, s, a)$ のように分解される。ここで i はタスクの番号（実質的に RGoal での g に相当）、$Q(i, s, a)$ は状態 s からサブタスク a を実行し親タスク i を終了するまでの間の報酬、$V(a, s)$ は s で開始した a を実行中の報酬、$C(i, s, a)$ は a 終了後 i 終了までの報酬である。$V(a, s)$ は親タスク i に依存しないため複数の親タスクから共有できる。$V(a, s)$ が RGoal における

[5]現在の実装では、エピソードがタイムアウトするまでそのまま同じ行動を繰り返す。

```
1: procedure EPISODE(S, G)
2:     # Init and choose the first action rule.
3:     ruleHistory, callerHistory, rewardHistory, callerStackHistory ← empty
4:     s ← S;  g ← G;  stack, callerStack ← empty
5:     stack.push(G) ;  caller ← StartRule ;  callerStack.push(ReturnRule)
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:        callerStackHistory.push(callerStack.copy())
26:        if a = RET then
27:            caller ← callerStack.pop()
28:        else if a is exit(m) then
29:            caller ← callerStack.pop()
30:        else if a is call(m) then
31:            callerStack.push(caller)
32:            caller ← rule
33:        #
34:        s ← s';  g ← g';  rule ← rule'
35:    Update(ruleHistory, callerHistory, rewardHistory, callerStackHistory)
36:
37: procedure UPDATE(ruleHistory, callerHistory, rewardHistory, callerStackHistory)
38:    rule ← ReturnRule
39:    T ← 0    # Total reward until the end of the episode.
40:    while ruleHistory is not empty do
41:        rule' ← rule
42:        rule ← ruleHistory.pop() ;  caller ← callerHistory.pop() ;  reward ← rewardHistory.pop() ;
   callerStack ← callerStackHistory.pop()
43:        T ← T + reward
44:        if rule = ReturnRule then
45:            Base(caller) ← Base(caller) + α(reward + Q(rule') − Base(caller))
46:        else
47:            Q(rule) ← Q(rule) + α(T − Base(caller) − <all Base(caller) in callerStack> − 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)=Base(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。

```
1: fp = w(a(F(s,+,g)));
2: rule(w(a(Failed)), fp, exit(a(Failed)));
// Start planning.
3: rule(w(), fp, call(a(A(s,+))));
4: rule(w(a(A(s,g’))), fp, set(a(F(s,g’,g’))));
// Loop.
5: rule(w(a(F(s,+,g’))), fp, call(a(A(g’,+))));
6: rule(w(a(A(g’,g”))), fp, recall(e(F(s,+,g’))));
7: rule(w(a(A(g’,g”)), e(F(s,+,g’))),
                        fp, set(a(F(s,g’,g”))));
```

図 4: 行動計画プログラムの疑似コード

```
// １つ隣の場所に移動するための行動ルール
1: rule(w(a(R1)), w(a(R2)), GoTo(R2));
2: rule(w(a(R2)), w(a(R4)), GoTo(R4));
3: rule(w(a(R1)), w(a(R3)), GoTo(R3));
4: rule(w(a(R3)), w(a(R4)), GoTo(R4));
// 行動計画に使われる知識
5: rule(w(), w(a(A(+,+))), exit(a(Failed)));
6: rule(w(), w(a(A(R1,+))), set(a(A(R1,R2))));
7: rule(w(), w(a(A(R2,+))), set(a(A(R2,R4))));
8: rule(w(), w(a(A(R1,+))), set(a(A(R1,R3))));
9: rule(w(), w(a(A(R3,+))), set(a(A(R3,R4))));
```

図 5: 地図に関する知識を表現する行動ルールの例

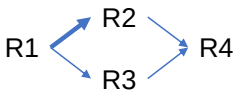

図 6: 行動計画のテストに用いる環境の地図

```
1: pg = w(a(g));
2: rule(w(), pg, RecogRoom()); // 現在位置を認識
3: rule(w(a(s)), pg, call(a(F(s,+,g))));
4: rule(w(a(F(s,a,g))), pg, call(a(a)));
```

図 7: 行動計画と行動を繰り返すプログラムの疑似コード

相対価値、$C(i,s,a)$ が RGoal における基準値とそれぞれ類似しているが、RGoal では基準値の設計に自由度があり、タスクの性質に特化させた様々な定義が許される。本稿での基準値の定義では、例外終了を基準値の学習に含めないことで、より汎化性能を高めることを狙っている。(ただしその効果については未検証である。)

MAXQ と RGoal は学習アルゴリズムも大きく異なっている。MAXQ は学習すべきパラメタ数は少ないが１ステップあたりの計算コストが階層の深さに対し指数関数的であるのに対し、RGoal はパラメタは少し多くなるが１ステップあたりの計算コストは少ないという特徴がある。

5　まとめと今後

本研究では、以前提案した再帰的強化学習アルゴリズム RGoal のモンテカルロ版の基準値の定義を改良し、簡単なテスト用タスクで動作を確認した。

なお、基準値と相対価値の定義および学習アルゴリズムにはまだ改良の余地があるだろう。また、現状はテストも完全ではなく、特殊なケースでは不具合が出る可能性がある。今回は実装の容易なモンテカルロ法を用いたが、Sarsa に適格度トレースを加えたアルゴリズムの方が、実用面でも脳のモデルとしても有望であろう。

再帰的強化学習は、ヒトの知能ような再帰的な構造を持った作業や言語処理を実現する上で、必要不可欠な要素技術であると考えている。多くの研究者による継続的な改良や実用化に向けた取り組みが始まることが期待される。

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] 一杉裕志, 中田秀基, 高橋直人, 竹内泉, 佐野崇. 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. 第 25 回 人工知能学会 汎用人工知能研究会 (SIG-AGI), 2023.

[3] 一杉裕志, 中田秀基, 高橋直人, 竹内泉, 佐野崇. 報酬最大化を目的とする行動計画・実行・対話・推論の統一的制御機構. 第 37 回 人工知能学会全国大会, 2023.

[4] 一杉裕志, 中田秀基, 高橋直人, 竹内泉, 佐野崇. 汎用人工知能のためのプログラム合成対象言語 Pro5Lang のエピソード記憶機構. 第 20 回 人工知能学会 汎用人工知能研究会 (SIG-AGI), 2022.

[5] 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.

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