社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2004－ＩＣＳ－135　(16)
2004／3／16

強化学習並列化による学習の高速化

森 紘一郎† 山名 早人‡

†早稲田大学大学院理工学研究科 〒169-8555 東京都新宿区大久保 3-4-1
‡早稲田大学理工学部 〒169-8555 東京都新宿区大久保 3-4-1
E-mail: †mori@yama.info.waseda.ac.jp, ‡yamana@waseda.jp

あらまし 強化学習は知識がない状態から試行錯誤によって学習を行う．そのため，学習が遅いという欠点があり，いかに学習を高速に行うかが大きな問題点となっている．このような問題に対して，従来，価値関数を分割して各プロセッサに割り当て，並列に更新する手法が提案されている．しかし，強化学習の性質上，分割された価値関数間で頻繁に経験を交換する必要があり，従来の研究ではプロセッサ間通信のオーバーヘッドが大きいことが問題であった．本論文では，共有する 1 つの価値関数を複数のエージェントが非同期並列的に更新するオーバーヘッドの少ない手法を提案する．共有メモリ型並列計算機 IBM pSeries 690 上で 127×127 の迷路問題を評価タスクにして実行したところ 24 プロセッサで 22.2 倍の速度向上を達成できた．

キーワード 強化学習，学習の高速化，並列計算

A Fast Learning Method for Reinforcement Learning on Shared Memory Multiprocessors

Kouichirou MORI† Hayato YAMANA‡

†Graduate School of Science and Engineering, Waseda University 3-4-1 Okubo, Shinjuku-ku, Tokyo, 169-8555 Japan
‡Faculty of Science and Engineering, Waseda University 3-4-1 Okubo, Shinjuku-ku, Tokyo, 169-8555 Japan
E-mail: †mori@yama.info.waseda.ac.jp, ‡yamana@waseda.jp

Abstract

In Reinforcement Learning, the agent learns by trial and error from a state without knowledge. Therefore, reinforcement learning has drawbacks that learning is slow. It is a serious problem how learns at high speed. In order to learn at high speed, some methods have been proposed. In the methods, the value function is divided. Then each divided value function is assigned to each processor, and updated in parallel. However, the method needs to exchange experiences frequently between the divided value function because of the character of reinforcement learning. It was a problem in the previous research that the overhead of communication between processors is large. In this paper, we propose the small overhead method that the parallel agents evaluate the shared value function asynchronously. As a result of the evaluation using shared memory type parallel machine, IBM pSeries 690, our method is approximately 22.2 times faster than sequential one in maze problem.

Keyword Reinforcement Learning, Speed-up Learning, Parallel Processing

1. はじめに

強化学習は知識がない状態から試行錯誤によって学習を行うため学習が遅いという欠点がある．強化学習を実問題に応用するには学習の高速化が必須であり盛んに研究が行われている．

強化学習の高速化に関する研究には以下の二つの立場がある．

(1) 学習の収束に要する経験数の減少

(2) 学習の収束に要する計算時間の短縮

(1) を目的とした研究には報酬を過去の行動に伝播する適正度の履歴[1]，一度得た経験を何度も更新に用いるプラニング[2]，教示の導入[3]，マルチエージェントによる経験の共有[4][5]などがある．一方，(2) を目的とした研究には，利得を近似して計算量を減らす Truncated Temporal Difference[6]，木構造を用いる TD(λ)の対数時間更新算法[7]，並列計算[8]などがある．

強化学習の高速化と言った場合，(1) を指向した研究が多い．これは，強化学習をロボットなどの実機に応用したとき，経験を得るのが難しく，少ない経験数で効率的に学習する方法が模索されてきたからだと考えられる．一方，シミュレーション上の学習では経験を容易に得ることができるため，(1) は問題にならず，(2) が問題となることが多い．

強化学習は，シミュレーションに比べ実機で用いられることが多いため，(1) に関する研究は重要である．しかし，実機に応用するにしても事前にシミュレーション上で学習を行うことが一般的である[9] ため，(2) に関する研究も重要であると考えられる．

本研究では，(2) の立場に立ち，並列計算機を用いて学習を高速化する手法を提案する．並列計算機を用いて学習を高速化する研究は，ニューラルネットワーク[10]や遺伝的アルゴリズム[11]で盛んに行われているが，強化学習では例が少ない．

Printista[8]は，分散メモリ型並列計算機を用いて，行動価値関数を各プロセッサに分割し，並列に更新する手法を提案している．しかし，強化学習アルゴリズムの性質上，分割されたプロセッサ間で頻繁な経験の交換が必要であり，プロセッサ間通信のオーバーヘッドが大きいことが問題となっている．本研究では，この問題を解決するため，共有メモリ上に存在する行動価値関数を非同期並列的に更新することにより，通信オーバーヘッドを減らす手法を提案する．

2 章では，強化学習の枠組みを説明し，どの部分を並列化の対象にするかを述べる．3 章では，強化学習並列化の従来手法について述べる．4 章では，提案手法について述べる．5 章では，本手法を迷路問題，車の山登り問題に適用し，計算時間が短縮されることを示す．6 章では，まとめを行う．

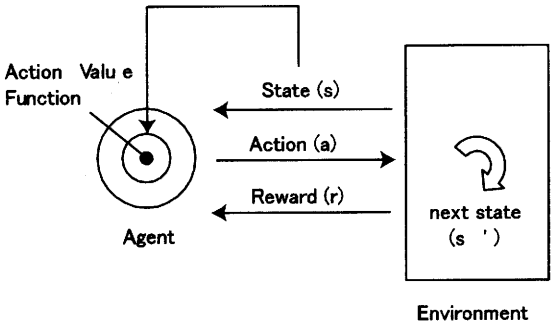


図 1: 強化学習の枠組み

2. 強化学習の概要

強化学習の枠組みを図 1 に示す．環境の状態を $s \in S$，エージェントの行動を $a \in A$ とする（S は状態の集合，A は行動の集合である）．状態 s で行動 a をとると環境の状態は s から $s' \in S$ に移り，このとき環境からエージェントに報酬 $r \in R$ が与えられる（R は実数の集合である）．エージェントは，ある状態でどの行動をとればより多くの報酬を得られるかを学習する．

強化学習アルゴリズムとして Q-Learning[12]がよく用いられる．Q-Learning では，状態 s，行動 a，次状態 s'，報酬 r の 4 つ組 (s,a,s',r)（経験と呼ぶ）を用い，式 (1) の行動価値関数 $Q(s,a)$ を更新することによって学習が行われる．

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')] \quad (1)$$

ここで，α は学習率[1]，γ は割引率[2]である．行動価値関数 $Q(s,a)$ は，状態 s において行動 a をとったときに得られる報酬の期待値を表している．エージェントはこの行動価値関数に基づき行動を選択する．つまり，行動価値関数の更新が進むにつれ，エージェントは適切な行動（期待獲得報酬を最大化する行動）を選択できるようになる．

エージェントの行動選択には ε-greedy という手法が使われる．この手法は確率 ε でランダムな行動を選択し，確率 $1-\varepsilon$ で行動価値関数に基づいて行動を選択する．ε は探査率と呼ばれる．エージェントがランダムに行動を選択する可能性を残すことによって局所解

[1] 学習率は，1 回の更新による行動価値関数の変化量を表すパラメータである．

[2] 割引率は，将来の報酬をどれだけ割り引くかを表すパラメータである．

に陥らないようにする効果がある．以上が強化学習の大まかな枠組みである．

強化学習が遅いのは，式(1)の更新を学習が収束するまでに大量に繰り返さなければならないからである．強化学習の並列化に関する従来の研究および本研究でも式(1)の更新部分を並列化の対象にしている．本研究では，並列計算機を用いて式(1)の計算を非同期並列的に行うことによって学習の高速化を達成する．

行動価値関数は主にテーブル，線形関数，ニューラルネットワークで表現される．本研究では，テーブルと線形関数を対象にし，どちらの場合にも本手法が有効であることを示す．

3. 強化学習並列化に関する従来の研究

従来の強化学習並列化手法として，Printista[8]，Kretchmar[5]，Antonova[13]など分散メモリ型並列計算機を対象とした手法が提案されている．

3.1 価値関数の分割更新

Printista[8]は，テーブル型の行動価値関数を分割して各プロセッサに割り当て，並列に更新することによって学習を高速化している（図 2）．この方法では，割り当てられた範囲以外の値が必要なときにプロセッサ間通信が生じる．そこで，得られた値をキャッシュすることによって通信頻度を抑える工夫をしている．分散メモリ型並列計算機 Power Mouse Parallel Machine 上で迷路問題に適用し，4 プロセッサで 2.32 倍の速度向上を達成している．しかし，8 プロセッサでは通信によるオーバーヘッドのため速度は低下している．

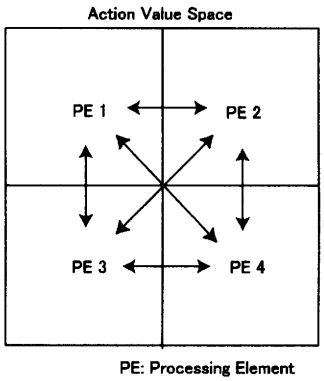


図 2: 価値関数の分割更新．価値関数を分割して各プロセッサに割り当て並列に更新する．

3.2 価値関数の結合

Kretchmar[5]は，複数のエージェントが独立した価値関数を並列に更新し，学習の途中で得られた価値関数を結合させることによって学習を高速化する方法を提案している（図 3）．Antonova[13]は，Kretchmar[5]を発展させ，価値関数の結合頻度を変えることによって学習速度がどのように変化するかを示している．Kretchmar[5]と Antonova[13]では，学習に要する経験数の減少を示しているだけで，並列計算機を用いて実行時間が短縮されることを示してはいない．しかし，分散メモリ型並列計算機を用いて，並列計算ができることを示唆している．

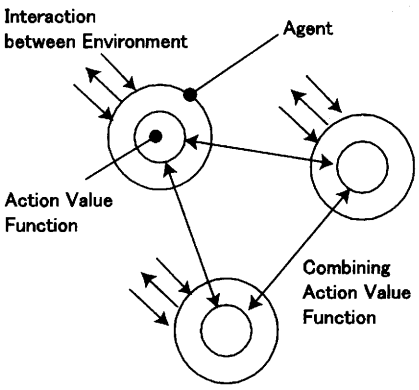


図 3: 価値関数の結合．複数のエージェントが独立した価値関数を並列に更新し，学習の途中で結合する．

3.3 従来手法の問題点

従来の手法における問題点は，

- 価値関数の分割を明示的に行う必要がある．
- プロセッサ間通信によるオーバーヘッドが大きい．
- 価値関数の結合コストが大きい．

が考えられる．本論文では，これらの問題点を解決する手法を提案する．

4. 提案手法

本論文では，共有する 1 つの価値関数を複数のエージェントが非同期並列的に更新する手法を提案する（図 4）．なお，プラットフォームとしては共有メモリ型並列計算機を想定する．

複数のエージェントを用いる点は，Kretchmar[5]と同じである．異なるのは，Kretchmar[5]では各エージェントが価値関数を独立に持っていたのに対し，本手法ではすべてのエージェントが価値関数を共有している点である．つまり，エージェントは異なるプロセッサ（Processing Element: PE）上で並列に動作するが，更新する価値関数は共有メモリ上にあり，すべてのエージェントがアクセスできる．また，同期処理による

遅延を避けるため，すべてのエージェントは共有された価値関数を非同期並列的に更新する．つまり，あるエージェントが価値関数を更新している間でも他のエージェントが価値関数にアクセスできるようにする．

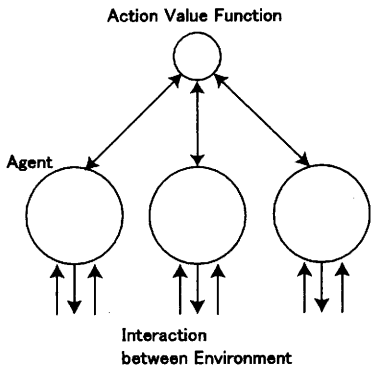


図 4: 提案手法．共有する 1 つの価値関数を複数のエージェントが非同期並列的に更新する．

5. 実験

本手法の有効性を確かめるため，迷路問題と車の山登り問題を評価タスクに選び，実験を行った．

5.1 迷路問題

迷路問題における環境は図 5(a)に示す迷路である．迷路の大きさは，63×63 と 127×127 の 2 種類を用意した[3]．状態はエージェントの位置であり，それぞれ 3969 (63×63)，16129 (127×127) の状態が存在する[4]．それぞれの状態でエージェントがとれる行動は，UP（上へ 1 マス進む），DOWN（下へ 1 マス進む），LEFT（左へ 1 マス進む），RIGHT（右へ 1 マス進む）の 4 種類である．エージェントは壁（図 5(a)中の黒いマス）には進むことができない．壁に進む行動をとったとき，状態は変化しない（その場にとどまる）ことにする．報酬はエージェントがゴール状態（図 5(a)中の右下端のマス）にいるとき 0，それ以外の状態では-1 とした．

行動価値関数にはテーブル形式を用いた．行動価値関数の更新は式(1)で行われる．強化学習のパラメータは，学習率 α を 0.1，探査率 ε を 0，割引率 γ を 0.9 とした．

5.2 車の山登り問題

車の山登り問題における環境は図 5(b)に示す凹型の山道である．エージェントは車であり，山道の右端最上部(図 5(b)中の Goal)にたどり着くのが目的である．状態は車の位置 x_t と速度 v_t である．車エージェントのとれる行動は，フルスロットル前進（右へ進む），フルスロットル後退（左へ進む），ゼロスロットル（惰性で進む）の 3 種類である．車エージェントの動きは，単純化した物理学モデルに従い，車の位置と速度は次式によって更新される．

$$x_{t+1} = bound[x_t + v_{t+1}] \quad (2)$$

$$v_{t+1} = bound[v_t + 0.001a_t - 0.0025\cos(3x_t)] \quad (3)$$

ここで，a_t は行動を表しており，フルスロットル前進のとき 1，フルスロットル後退のとき-1，ゼロスロットルのとき 0 とする．$bound$ は，位置，速度の範囲をそれぞれ $-1.2 \le x_{t+1} \le 0.5$（図 5(b)の左端の位置が -1.2，右端の位置が 0.5 である），$-0.07 \le v_{t+1} \le 0.07$ に制限する関数である．車の初期状態は $x_0 = -0.5$，$v_0 = 0$ とした．報酬は車がゴール状態（山道の右端最上部）にいるとき 0，それ以外の状態では-1 とした．

行動価値関数には線形関数形式を用いた．線形関数を用いた行動価値関数は式(4)のように表せる．

$$Q(s,a) = \vec{\theta}^T \vec{\phi}_{sa} \quad (4)$$

ここで，$\vec{\theta}$ は，パラメータベクトル，$\vec{\phi}_{sa}$ は，状態をタイルコーディング[14]と呼ばれる符号化アルゴリズムによって変換した特徴ベクトルである．テーブル形式が式(1)によって $Q(s,a)$ を直接更新していたのに対し，線形関数形式では式(6)によってパラメータベクトル $\vec{\theta}$ を更新する．

$$\delta = r + \gamma Q(s',a') - Q(s,a) \quad (5)$$

$$\vec{\theta} \leftarrow \vec{\theta} + \alpha\delta\vec{e} \quad (6)$$

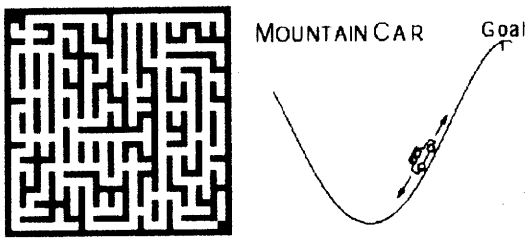
ここで，$\vec{e}$ は適正度を表している（1 章で述べた適正度の履歴という手法を合わせて用いている）．

強化学習のパラメータは，学習率 α を 0.1，探査率 ε を 0.1，割引率 γ を 1，履歴減衰パラメータ[5] λ を 0.9 とした．

[3] 棒倒し法と呼ばれる迷路生成アルゴリズムを用いた．

[4] エージェントは壁を通ることができないため探索すべき実際の状態数はより少ない．

[5] 適正度 $\vec{e}$ を 1 ステップでどれだけ減少させるかを表すパラメータである．



(a) 迷路問題　(b) 車の山登り問題[14]

図 5: 評価タスク

5.3 実験環境

プログラムの実装には，pthread を用いた．使用したマシンは共有メモリ型並列計算機 IBM pSeries 690（Power4 プロセッサ 24 台，メモリ 8GB，L2 キャッシュ 1.5MB，L3 キャッシュ 256MB，OS AIX Ver.5）である．スレッドのスケジューリングポリシーにはラウンドロビンを採用し，各スレッドの実行機会が均等になるようにした．

5.4 実験結果

5.4.1 学習曲線

図 6 は迷路問題（127×127）の学習曲線であり，並列に動作するエージェントの数と学習速度の関係を表している．横軸はエピソード数（迷路問題では，スタート地点からゴール地点へたどりつくまでが 1 エピソード），縦軸はステップ数（迷路問題では，エージェントの 1 回の行動が 1 ステップ）である．本手法は，全エージェント間で共有する行動価値関数を持っているため，どのエージェントの学習曲線をとってもほぼ一致する．そのため，複数のエージェントを用いた場合は，エージェント 1（はじめに起動したエージェント）の学習曲線を記録した．

図 6 から行動価値関数の更新に参加するエージェント数が増えるにつれ，学習の収束エピソードは速まることがわかる．エージェント数が 1 の場合，21200 エピソードで収束するのに対し，エージェント数が 32 の場合，500 エピソードで収束する．5.4.2 項で述べる学習の収束時間はプログラムの起動時から図 6 における収束点までの時間である．

5.4.2 迷路問題の学習時間

表 1 と表 2 は迷路問題でエージェント数を 1, 2, 4, 8, 16, 24, 32 としたときの収束までの時間を測定した結果である．プログラムの起動時から図 6 における収束点までの時間を測定した．計算機の負荷は一様でないため 10 回の試行でもっとも速かった結果を記録

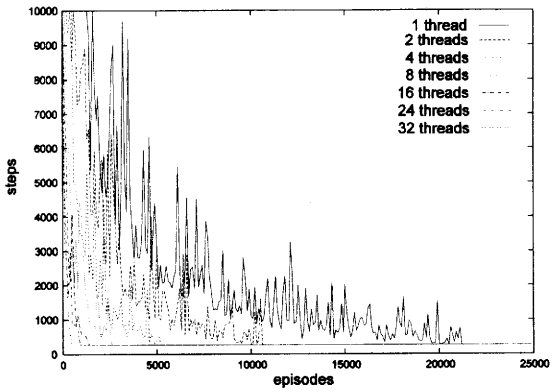


図 6：迷路問題（127×127）における学習曲線

した．

表 1 と表 2 から行動価値関数の更新に参加するエージェント数が増えるにつれ，学習の実行時間が短縮されることがわかる．エージェント数が 24 のときに比べ，32 のときの方が遅いのは，使用マシンのプロセッサが 24 台しかないためだと考えられる．また，規模の大きい迷路ほど速度向上比が高いのは，各エージェントが更新する範囲がばらつき，メモリアクセス競合が起きにくくなるためだと考えられる．

表 1：迷路問題（63×63）における学習時間

エージェント数	1	2	4	8	16	24	32
収束までの時間(s)	1.92	0.99	0.51	0.27	0.15	0.10	0.11
速度向上比	1	1.94	3.76	7.11	12.8	19.2	17.5

表 2：迷路問題（127×127）における学習時間

エージェント数	1	2	4	8	16	24	32
収束までの時間(s)	15.29	7.77	3.95	1.99	1.01	0.69	0.70
速度向上比	1	1.97	3.87	7.68	15.1	22.2	21.8

5.4.3 車の山登り問題の学習時間

表 3 は車の山登り問題でエージェント数を 1, 2, 4, 8, 16, 24, 32 としたときの収束までの時間を測定した結果である．

表 3 から行動価値関数の更新に参加するエージェント数が増えるにつれ，学習の実行時間が短縮されることがわかる．

車の山登り問題は迷路問題とは異なり，行動価値関

数が線形関数で表現されている．そのため，本手法が線形関数形式でも有効であることがわかる．

図 7 に各タスクにおけるエージェント数と学習時間の関係を示す．

表 3：車の山登り問題における学習時間

エージェント数	1	2	4	8	16	24	32
収束までの時間(s)	5.25	2.77	1.59	0.82	0.53	0.57	0.66
速度向上比	1	1.90	3.30	6.40	9.91	9.21	7.95

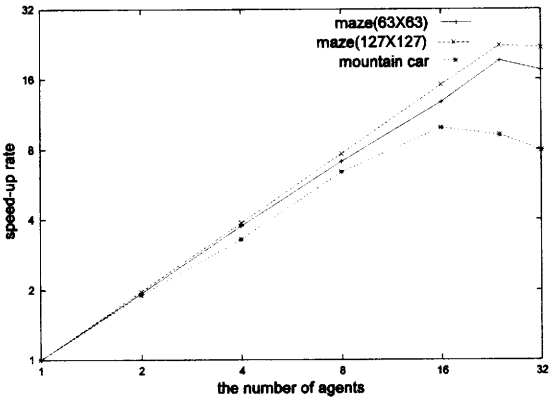


図 7: 各タスクにおけるエージェント数と速度向上比の関係

6. おわりに

本論文では，共有する行動価値関数を複数のエージェントが非同期並列的に更新し，学習を高速化する手法を提案した．迷路問題，車の山登り問題を用いて実験したところ，24 プロセッサで最大 22.2 倍の速度向上を達成できた．

本手法は，従来手法とは異なり，明示的に価値関数を分割する必要がないため実装が容易である．また，プロセッサ間通信や価値関数の結合が不要であるため従来手法よりオーバーヘッドが小さい．さらに，行動価値関数がテーブル，線形関数問わず，同じ手法が適用できる．ニューラルネットワークについては試していないが，山森[10]の重みパラメータ並列更新と同じ方法であるため高速化は可能であると考えられる．

問題点は，共有メモリ型並列計算機を用いているため，分散メモリ型並列計算機に比べて拡張性が低いこと，シミュレーション環境での学習のみにしか対応できず，実ロボットの学習では使えないことである．

文　献

[1] S. P. Singh and R. S. Sutton: “Reinforcement Learning with Replacing Eligibility Traces”, Machine Learning, Vol.22, pp.123-158, 1996.

[2] R. S. Sutton: “Dyna, an Integrated Architecture for Learning, Planning, and Reacting”, Working Notes of the AAAI Spring Symposium, pp.151-155, 1991.

[3] R. Maclin and J. W. Shavlik: “Creating Advice-Taking Reinforcement Learners”, Machine Learning, Vol.22, pp.251-281, 1996.

[4] M. Tan: “Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents”, Proc. of the 10th International Conf. on Machine Learning, pp.330-337, 1993.

[5] R. M. Kretchmar: “Parallel Reinforcement Learning”, The 6th World Conf. on Systemics, Cybernetics, and Informatics, 2002.

[6] P. Cichosz and J. J. Mulawka: “Fast and Efficient Reinforcement Learning with Truncated Temporal Differences”, Proc. of the 14th International Conf. on Machine Learning, pp.99-107, 1995.

[7] 片山晋，小林重信：“TD（λ）学習の対数時間更新算法”，人工知能学会誌，Vol.14, No.5, pp.879-890, 1999.

[8] A. M. Printista, M. L. Errecalde and C. I. Montoya: “A Parallel Implementation of Q-Learning Based on Communication with Cache”, Journal of Computer Science and Technology, Vol.6, 2002.

[9] 山口智浩，増渕元臣，田中康祐，谷内田正彦：“経験型強化学習における仮想個体から実ロボットへの学習行動の伝播”，人工知能学会誌，Vol.12, No.4, pp.570-581, 1997.

[10] 山森一人，堀口進：“並列計算機上での誤差逆伝播学習法の並列学習モデル”，信学論 (D-II), Vol.J81-D-II, No.2, pp.370-377, 1998.

[11] 北野宏明：“遺伝的アルゴリズム 4”，産業図書，2000.

[12] C. J. C. H. Watkins and P. Dayan: “Technical Note: Q-Learning”, Machine Learning, Vol.8, pp.55-68, 1992.

[13] D. Antonova: “Parallel Reinforcement Learning – Extending the Concept to Continuous Multi-State Tasks”, thesis, Denison University, 2003.

[14] R. S. Sutton and A. G. Barto: “Reinforcement Learning: An Introduction”, MIT Press, 1998.