情報処理学会論文誌 Vol.60 No.10 1617–1630 (Oct. 2019)

自己適応システムのための環境モデル実行時更新手法

田邉 萌香[1,a)] 鄭 顕志[1,b)] 本位田 真一[1,c)]

受付日 2019年1月25日, 採録日 2019年7月3日

概要：ソフトウェアシステムは実行時に環境変化が生じても要求を充足し続ける自己適応性を備えることが望ましい．近年では，これまで開発時にのみ用いられてきた要求，環境，振舞い仕様モデルをシステム実行中にも活用し，モデル検査やコントローラ生成手法と組み合わせることで要求充足を保証した自己適応を実現する実行時モデル（Models@run.time）技術が開発されている．しかしながら，これらの実行時モデル技術によってもたらされる保証は，実行環境に生じた変化が正確に環境モデルに反映されていることを前提としている．環境モデルが実行環境を正確にモデル化していない場合，モデル上での検証や意思決定の結果は何の保証も持たない．本論文では，環境変化を正確，迅速，かつ少ない計算オーバヘッドでモデルに反映する環境モデル実行時更新技術を提案する．具体的には，Labeled Transition System（LTS）形式の環境モデルを対象とし，システムから得られた実行トレースをもとに環境モデルを実行時に更新する．環境モデル更新に必要となる実行中の計算オーバヘッドを軽減するために，確率的勾配降下法を応用した差分学習手法を提案する．自動倉庫管理システムを例題としたケーススタディを通して，従来のバッチ処理形式の環境モデル更新手法と比較して，提案手法は同程度の正確度，より早い収束時間，より少ない計算オーバヘッドで環境モデルを実行時更新可能であることを確認した．

キーワード：自己適応システム，環境モデル，Labeled Transition System，実行時モデル

Updating Environment Model at Runtime for Self-adaptive System

Moeka Tanabe[1,a)] Kenji Tei[1,b)] Shinichi Honiden[1,c)]

Received: January 25, 2019, Accepted: July 3, 2019

Abstract: Software systems should be able to adapt in response to changes in the environment in order to keep satisfying its requirements. Models@run.time approach, which utilizes models such as requirements, environment, and specification models, at runtime for verification and/or decision making, is one of the promising approaches to provide a formal guarantee of self-adaptation. However, the guarantee depends on the correctness of the environment model. If the environment model becomes inconsistent with the environment due to changes, the system works without any formal guarantee. In this paper, we aim to enable runtime update of the environment model. We propose a difference learning technique to reflect changes in the environment to environment model accurately, in a fast settling time, and with low computational overhead. We evaluate our techniques through case studies based on automated warehouse scenario. The results show that our runtime update achieves close accuracy to, faster settling time than, and lower computational overhead than the existing batch-based update technique.

Keywords: self-adaptive systems, environment model, labeled transition system, Models@run.time

1. はじめに

サイバーフィジカルシステム，Internet-of-Things，ロボットシステム等，ソフトウェアシステムは実世界とより密に連動することが求められている．一般にソフトウェアシステムの開発においては，開発対象となるソフトウェア

[1] 早稲田大学
Waseda University, Shinjuku, Tokyo 169–8555, Japan
[a)] marron.620@ruri.waseda.jp
[b)] ktei@aoni.waseda.jp
[c)] honiden@nii.ac.jp

がインタフェースを通じて相互作用する外部要素（以降，実行環境と呼ぶ）をモデル化し，実行環境に関する仮定に基づいて，開発するソフトウェアの妥当性を検証する．たとえば，倉庫でロボットが荷物の運搬を行うような自動倉庫管理システムにおいて，運搬ロボットを制御するソフトウェアシステムを考えた場合，センサ等のインタフェースを通じて把握することができるエネルギー消費量等のロボットの状態，またロボットの周辺の物理環境が実行環境となる．また，クラウドシステムでは，監視エージェント等を通じて把握することができるリクエスト数，サーバリソース等が実行環境となる．このようなソフトウェアシステムの実行環境はシステム実行時に変化し，かつその変化を開発時に完全に予測することは困難である [12]．一方，ソフトウェアシステムは変化する環境下で要求を充足し続けることが求められる．

そのような背景の中，ソフトウェア工学分野においても，環境の変化に対して，要求を充足し続けるよう振舞いを実行時に変更する，自己適応システム [3], [19] に関する研究が進められている．特に，従来上流工程でのみ扱われていた要求，環境，仕様モデルを実行時にも活用する，いわゆる実行時モデル（Models@run.time）に関する研究が進められている．文献 [11], [13], [14], [17] らの研究では，実行中に起きた環境変化を環境モデルに反映し，開発時に用意された振舞い仕様モデルと環境モデルに対して実行時にモデル検査を行うことで，現環境下で要求を充足することが保証される振舞い仕様を選び出す．これらの手法では候補となる振舞い仕様モデルは開発者によって準備されなければならない．より柔軟かつ保証を備えた自己適応を実現するために，文献 [5], [8] では，環境モデルと要求モデルから，要求を満たすことが保証された振舞いモデルを自動生成するコントローラ生成（controller synthesis）技術を活用した自己適応手法を提案している．文献 [5], [8] では，Labeled Transition System（LTS）形式で記述された環境モデルと，Linear Temporal Logic（LTL）で記述された要求モデルを入力とし，LTS 形式の振舞いモデルを自動生成する．出力された振舞いモデルは要求モデルとして記述された安全性や活性といった性質を満たすことが保証される．

しかしながら，これらの既存の実行時モデル手法を用いた自己適応は，環境モデルの正しさを前提としている．実行環境と一致しない不正確な環境モデルを基に振舞いを決定した場合，システムの要求充足は保証されない．実行環境は不確実性を持つため，開発時にすべての起こりうる変化を反映した環境モデルを構築することは本質的に困難であり，開発時に置かれる仮定のもと環境モデルは構築される．その際に，環境の変化により，開発時に構築された環境モデルが持つ仮定から逸脱してしまうリスクを軽減するため，あらかじめ弱い仮定を持つ，様々な理想的でない事象が起こることを想定した環境モデルを構築することがある．弱い仮定のもと構築されたモデルでは，高度な要求は保証できないが，環境が変化した際にも弱い仮定のもと動作を続けることが可能である．一方，強い仮定のもと，理想的な事象のみが起こることを想定して構築された環境モデルでは，高度な要求が保証可能となるが，仮定からの逸脱が頻繁に発生し，対応できなくなってしまう [12]．したがって，実行環境との整合性を維持するため，実行時に環境モデルを更新する必要がある．

文献 [4], [21] 等の既存研究では環境モデルの構築手法を提案している．開発時にシステムをテスト実行し，システムの状態遷移を実行トレースとして記録し，そこで得られた多量の実行トレースをバッチ処理で一度に解析して，状態遷移モデルである環境モデルを構築している．これらの手法は非実行時に環境モデルを構築することを想定している．多量の実行トレースをバッチ処理で一度に分析することから，モデル構築に時間を要する．自動倉庫管理システムの場合，ロボットが動作を止めることなく実行を続けるためには，実行時に起こる変化を 1 秒以内程度で環境モデルに反映することが求められるが，これらの手法では環境モデルの構築に 10 秒から 20 秒程度を要する．実行時に起こる変化を環境モデルに反映するには，実行時に迅速に更新する必要があるため，既存手法は実行時に用いるのには適していない．

そこで本論文では，実行時に生じた環境変化を，正確，迅速，かつ少ない計算オーバヘッドで環境モデルに反映する環境モデルの実行時更新手法を提案する．本手法では文献 [5], [8] で提案されているコントローラ自動生成を用いた自己適応技術の構成要素であり，環境とシステムの相互作用を表す LTS 形式の環境モデルを扱う．本手法では，文献 [21] のようなバッチ処理によるモデル構築を避け，実行時に新たに得られた実行トレースのみから環境モデルを差分更新することで，一度の更新に要する時間を削減し，実行時の効率的なモデル更新を可能にする．そのために，確率的勾配降下法を応用した，環境モデルの差分更新手法を提案する．評価では，自動倉庫管理システムの事例をもとに，既存の更新手法と本論文で提案する差分更新手法について，推定された環境モデルの正確度，変化が環境モデルに反映されるまでの時間，モデル更新の実行時計算オーバヘッドの観点から比較を行う．

論文の構成は以下のとおりである．2 章では自己適応システム，環境モデル構築に関する従来手法と，その課題について説明する．3 章では背景技術となる Labeled Transition System と勾配降下法について説明する．4 章では本論文で扱う例題について説明する．5 章では実行時に適用可能な環境モデルの差分更新手法を提案する．6 章では 5 章で提案した手法と従来手法の比較，評価を行う．7 章で考察を行い，8 章で結論を述べる．

2. 関連研究

2.1 自己適応システム

システムの実行環境に変化が生じた場合，開発時の仕様では与えられた要求が充足されなくなる可能性がある．その際に，新しい環境のもとで要求を充足する新しい仕様を決定，切り替える必要があり，それをシステム自身で行うのが自己適応システムである [3], [19].

Braberman ら [2] の研究では，高い保守性を持つ自己適応システムのアーキテクチャ（MORPH アーキテクチャ）を提案している．MORPH アーキテクチャは，**図 1** のように，要求分析を行う Goal Management，振舞い仕様を決定する Strategy Management，振舞い仕様を実行する Strategy Enactment という 3 つの層と Knowledge Repository により構成されており，適応ロジックとアプリケーションロジックを分離して自己適応システムを構成することで，高い保守性を得ている．Knowledge Repository では，環境の変化を検知するため，実行時に得られるシステムの情報を監視し，監視結果に基づいて system state，system goals，environment assumptions の実行時更新を行う．3 つの層は Knowledge Repository が持つ情報の正しさを前提として実行されるため，環境に変化が生じた場合は Knowledge Repository が持つ情報を正確に更新することが求められる．またこのアーキテクチャでは，自己適応システムの仕様の生成にコントローラ生成技術 [6] を用いることを想定している．コントローラ生成技術は，LTS 形式で記述された環境モデルと LTL 形式で記述された要求モデルから，要求を充足することが保証された振舞いモデルであるコントローラを自動生成する技術である．本研究では特に，MORPH アーキテクチャの Knowledge Repository が持つ情報の一部である environment assumptions であり，コントローラ生成技術の入力となる環境モデルの実行時更新に焦点を当てている．

D'Ippolito ら [5], [7], [8] の研究では，コントローラ生成技術を活用した自己適応手法を提案している．これらの研究では，異なる仮定を持つ複数の環境をあらかじめ想定している．段階的な仮定を持つ複数の環境モデル，その環境モデル下で充足可能な要求，コントローラの組を用意し，動作の観測結果に応じて実行時にコントローラを切り替えることで，環境の変化に対処している．強い仮定を持つ環境モデルの組から弱い仮定を持つ環境モデルの組へ切り替える際の要求緩和も実現している．しかしながら，あらかじめ想定した環境と変化後の環境が一致するとは限らず，その差によっては過剰な要求緩和が行われてしまう場合も存在する．したがって，どの程度の段階に分けて環境モデルを用意するかが課題となる．本研究では，あらかじめ段階的な仮定を持つ環境モデルを想定するのではなく，動作の観測結果から，実行時に実行環境を正確に表現する環境モデルを学習することを目的とする．

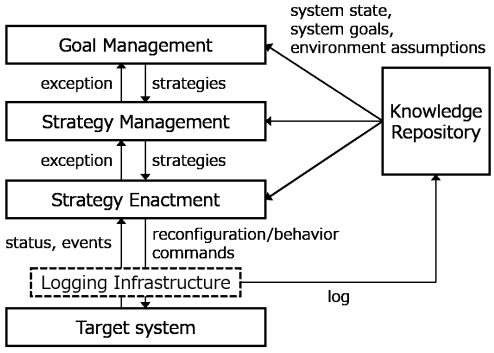


図 1 MORPH アーキテクチャ [2]
Fig. 1 MORPH architecture [2].

2.2 環境モデル構築

Fahland ら [10] や Ding ら [4] の研究では，ペトリネットを基にした環境モデル更新を扱っている．ペトリネットとシステムの実行ログを用意し，ペトリネット上で実行ログが再現できなくなった場合に，それが再現可能かつできる限り直前のモデルに近いようなペトリネットをプロセスマイニングにより更新する手法が提案されている．マイニング技術を用いた更新手法は，Yuan ら [23] の研究においても提案されている．また Sykes ら [21] は LTS ベースの環境モデル構築手法を提案している．Sykes らが提案する NoMPRoL アプローチでは，LTS ベースの環境モデルを論理プログラムモデルに変換し，論理プログラムが持つ規則の尤もらしさを実行トレースから学習する．学習は，テスト実行により得られた実行トレースをもとに，勾配降下法を用いて行っている．これらの研究では環境モデル構築は開発時に行うことを想定しているが，本手法ではモデルの更新を実行時に行う．

Filieri ら [11] は環境を離散時間マルコフ連鎖で，Ghezzi ら [13] はマルコフ決定過程で，Iftikhar ら [14] は時間オートマトンでモデル化している．これらの手法では，各遷移に付随するパラメータ（実行時間等）を観測結果をもとに実行時に更新し，確率モデル検査やパラメトリックモデル検査を実行時に行い，実行時間等の定量要求について検証を行っている．本手法では，遷移に付随するパラメータではなく，実行トレースから最も確からしい遷移を推定し，環境モデルに反映する．

強化学習を用いた環境モデル構築に関する研究も複数存在する．強化学習では，エージェントは環境に関する知識を探索・活用する．Menashe ら [16] の研究では，Factored MDP として表されるモデルの学習を強化学習により行っている．Sharifloo ら [20] の研究では，フューチャーモデルを強化学習によって学習している．強化学習を用いた場合，システムが知識を探索する試行の際に，システムが持つ機能的要求が充足されなくなる可能性がある．したがっ

て，要求充足の保証を扱う本手法では強化学習は行わない．

2.3 従来手法の課題

2.2 節で述べた Sykes ら [21] の NoMPRoL アプローチを用いることで，環境とシステムの相互作用を表す LTS ベースの環境モデルの更新が可能である．NoMPRoL では，LTS を論理プログラムモデルに変換し，システムのテスト実行により得られた実行トレースをもとに，勾配降下法を用いてモデル構築を行う．この NoMPRoL で用いられている勾配降下法を実行時のモデル更新に拡張することを考える．

実行時に起こる変化を環境モデルに反映するには，実行時にモデルを更新する必要がある．しかしながら，勾配降下法を用いた従来手法では，与えられた実行トレース中のすべてのデータを用いて学習を行うため，実行トレース長が大きくなるほど学習の正確度は高まるが，計算回数が増加する．また過去のデータの影響を大きく受けることから，環境が変化した際にその変化を認識するまでの実行動作数は実行トレース長が大きくなるほど増加する．したがって，正確，迅速かつ少ない計算オーバヘッドでの環境モデルの更新は困難である．

3. 背景技術

3.1 Labelled Transition System

Labelled Transition System（LTS）は，離散システムの振舞いを表現するために用いられるモデルであり，$E = (S, A, \Delta, s_0)$ と定義される．S は状態の集合，A は動作の集合，$\Delta \subseteq (S \times A \times S)$ は状態の遷移関係であり，ある遷移 $\delta = (s, a, s')$ における動作 a を δ が持つラベルと呼ぶ．また $s_0 \in S$ は E の初期状態である．

本研究で扱う環境モデルは，システムとその外部環境との相互作用を LTS によって離散的にモデル化したものである．遷移に付随する動作 $a \in A$ は，制御可能または観測可能な動作を表している．システムは，制御可能な動作を実行することで，環境側に何かしらの影響を与え，その影響を観測可能な動作として受け取る．環境モデルは，これらの 2 種類の動作が交互に行われるような状態遷移を持ち，それによりシステムとその外部環境との相互作用を表している．

3.2 勾配降下法

勾配降下法（GD：Gradient Descent）[18] は，パラメータ $\mathbf{p}$ を引数とする目的関数 $L(\mathbf{p})$ の値を最小化するための手法である．目的関数の勾配をもとにパラメータの調整をすることで，目的関数が凸であれば最小値を求めることができる．目的関数が凸でない場合は局所解が得られる可能性がある．

勾配降下法の目的関数は，学習のために与えられた N 個のデータのうち i 番目のデータによって得られる値を $l_i(\mathbf{p})$ とすると，次の式 (1) のように表すことができる．

$$L(\mathbf{p}) = \sum_{i=1}^{N} l_i(\mathbf{p}) \tag{1}$$

パラメータの更新は，次の式 (2) を用いて目的関数が収束するまで行われる．

$$\mathbf{p} = \mathbf{p} - \eta \nabla L(\mathbf{p}) \tag{2}$$

∇ は勾配，η は学習率である．学習率を変化させることにより，解を得るまでの計算回数も変化する．

4. 例題

4.1 自動倉庫管理システム

本研究では，自動倉庫管理システムを例題として扱う．自動倉庫管理システムは，**図 2** に示すような 6 つのエリアで構成されている倉庫内をロボットが移動し，荷物の出荷準備を行うようなシステムを想定している．エリア w1 は荷物の出荷準備を行うエリア，エリア e1 は荷物が保管されているエリアである．ロボットは（1）`move.{e, w}`：{ 東方向，西方向 } に向かって移動する，（2）`pickup`：荷物を持ち上げる動作をする，（3）`putdown`：荷物を下ろす動作をする，という 3 種類の動作を実行することができる．そしてロボットの動作の結果，システムは（1）`arrive.{e1, m1, w1, e2, m2, w2}`：ロボットがエリア {e1，m1，w1，e2，m2，w2} へ到達した，（2）`pickupsuccess`：ロボットが `pickup` に成功した，（3）`putsuccess`：ロボットが `putdown` に成功した，という 3 種類の動作を観測することができる．またこのシステムは，実行中に充足しなければならない機能的な要求を複数持っている．たとえば，「ロボットはエリア e1 からエリア w1 へ荷物を運ばなければならない」，「ロボットは荷物を持っている間に `pickup` を行ってはならない」，「ロボットは荷物を持っている間はエリア w1 に向かって移動しなければならない」等である．

この例題における環境モデルの一部を**図 3** に示す．たとえば，図 3 の環境モデルにおいて，初期状態（状態 0）からシステムが実行を開始し，ロボットがエリア w1 にいる場合（すなわち，`arrive.w1` を受け取った状態 1 において），システムは `move.e` または `putdown` という制御可能な動作が実行可能である．ここで `move.e` を実行して状態

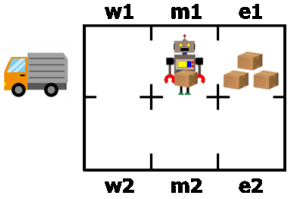


図 2 自動倉庫管理システム
Fig. 2 Automated warehouse management system.

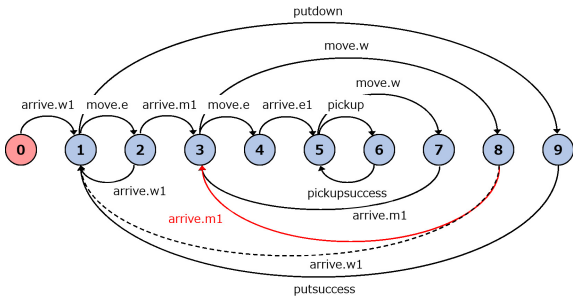


図 3 自動倉庫管理システムの環境モデル
Fig. 3 Environment model of automated warehouse management system.

2 に遷移した場合，エリア m1 に到達する（`arrive.m1` を受け取って状態 3 に遷移する），またはエリア w1 に到達する（`arrive.w1` を受け取って状態 1 に遷移する）ことが想定されている．

4.2 環境の変化

自動倉庫管理システムの例題において，「エリア w1 とエリア m1 の間が荷物で塞がれて通れなくなる」，という物理環境の変化が生じたとする．このとき，図 3 において，赤い矢印によって表される遷移が，環境の変化によって新たに起こるようになり，実行トレースに記録されるようになった遷移を表している．また点線の矢印によって表される遷移が，環境の変化によって起こらなくなり，実行トレースに記録されなくなった遷移を表している．この変化をシステムが認識することができなかった場合，エリア m1 で荷物を持っているロボットは，与えられた振舞い仕様に基づいて `move.w` という動作を繰り返す．そして，与えられた要求は充足不可能となってしまう．

環境が変化した場合でも，変化を環境モデルに反映することができれば，変化後の環境下で要求充足が可能なシステムの振舞い仕様を再生成することができる．そこで本論文では，要求充足が保証された自己適応システムの実現のため，実行時に生じた変化を正確，迅速，かつ少ない計算オーバヘッドで環境モデルに反映させることを目的とする．

5. 環境モデルの実行時差分更新

本手法では，実行時に得られる実行トレースから効率良く環境モデルを更新するため，環境モデルを差分更新する手法を提案する．確率的勾配降下法 [1] を応用し，実行時のモデル更新を現実的な時間内で可能にする．

5.1 手法の全体像

環境モデルの差分学習手法の概要を図 4 に示す．学習の入力は，(1) 規則群 R，(2) アクションセット，(3) 閾値 ζ，の 3 つである．本手法では，LTS が持つ遷移群 Δ の尤もらしさを学習するため，開発時に LTS を確率的なパラメータを持つ R に変換し，R が持つ確率的なパラメータを学習により更新する．アクションセットは実行トレースの一部であり，ある制御可能な動作を実行した際の事前条件と観測結果を抜き出したものである．R が制御可能な動作を実行した際の事前条件と観測結果からなるため，そのもっともらしさの学習のため，アクションセットという形で実行トレースから情報を抽出する．学習は新たなアクションセットが得られるたびに行う．更新されたパラメータは次の学習と LTS を構築する遷移の決定に用いる．LTS は確率付きモデルではないため，ζ をもとにもっともらしい遷移を決定し，それらを用いて LTS を更新する．

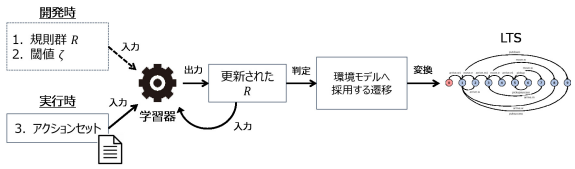
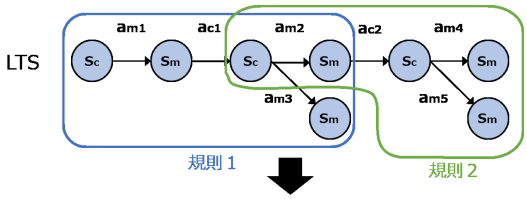


図 4 差分更新手法の全体像
Fig. 4 Overview of differential updating.



規則 1 ⟨a_{m1}, a_{c1}, {⟨a_{m2}, 0.5⟩, ⟨a_{m3} 0.5⟩}⟩
規則 2 ⟨a_{m2}, a_{c2}, {⟨a_{m4}, 0.5⟩, ⟨a_{m5}, 0.5⟩}⟩

図 5 LTS から規則への変換
Fig. 5 From LTS to rules.

5.2 差分学習の入出力

R は規則の集合であり，i 番目の規則を r_i とすると，r_i は次のように表される．

$$r_i = \langle pre_i, act_i, Post_i \rangle \tag{3}$$

制御可能な動作，観測可能な動作の集合をそれぞれ A_c, $A_m (A = A_c \cup A_m)$ とすると，$act_i \in A_c$，$pre_i \in A_m$ である．$Post_i$ は $post$ の集合であり，j 番目の $post$ を $post_j$ とすると，$post_j = \langle a_{mj}, \theta_j \rangle$ と表される．θ_j は a_{mj} の観測確率であり，$a_{mj} \in A_m$，$0 \leq \theta_j \leq 1$ である．pre_i，act_i，$Post_i$ はそれぞれ「事前条件 (*pre-condition*)」，「制御動作 (*control-action*)」，「事後条件群 (*post-conditions*)」と呼ぶこととする．本研究では，$a_m \in A_m$ と $a_c \in A_c$ を交互に受け取る環境モデルを対象としており，pre_i は act_i を受け取る直前に受け取る観測可能な動作，a_{mj} は act_i を受け取った直後に受け取る可能性のある観測可能な動作である．規則は事前条件と制御動作の組ごとに生成され，同じ事前条件と制御動作の組を持つ規則が複数存在することはない．例として，図 5 の LTS では，$a_{m1}, a_{m2}, a_{m3}, a_{m4}, a_{m5} \in A_m$，

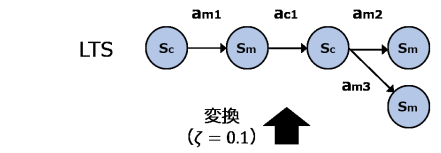


図 6 規則から LTS への変換
Fig. 6 From a rule to LTS.

$a_{c1}, a_{c2} \in A_c$ であり，$\langle a_{m1}, a_{c1}\rangle$，$\langle a_{m2}, a_{c2}\rangle$ という 2 つの事前条件と制御動作の組が存在する．$\langle a_{m1}, a_{c1}\rangle$ からは，a_{c1} の後に受け取る可能性のある a_{m2}，a_{m3} を事後条件とした規則 $1\langle a_{m1}, a_{c1}, \{\langle a_{m2}, 0.5\rangle, \langle a_{m3}, 0.5\rangle\}\rangle$ が生成され，$\langle a_{m2}, a_{c2}\rangle$ からは，a_{c2} の後に受け取る可能性のある a_{m4}，a_{m5} を事後条件とした規則 $2\langle a_{m2}, a_{c2}, \{\langle a_{m4}, 0.5\rangle, \langle a_{m5}, 0.5\rangle\}\rangle$ が生成される．事後条件に付与される観測確率の初期値には任意の値を設定することができる．

システムは，実行した制御可能な動作と受け取った観測可能な動作を，実行トレースとして記録する．実行時に入力するアクションセットは，実行トレースから制御可能な動作とその前後の観測可能な動作を抽出したものであり，$actionset = \langle pre, act, post\rangle$ と表される．このとき，$pre, post \in A_m$，$act \in A_c$ である．得られたアクションセットから θ の値を学習し，学習された値をもとにもっともらしい遷移を決定する．もっともらしい遷移の決定には ζ を用いる．

ζ は，環境モデルへ採用する事後条件を決定する際に用いる推定観測確率の閾値であり，開発時に任意に決定，入力する．ζ の値は，大きく設定することでより観測確率の高い動作によって環境モデルが構築され，小さく設定することで観測確率の低い動作も考慮した環境モデルが構築される．したがって，ζ の値により，環境モデルが持つ仮定の強弱を調整することができる．

学習の出力は，事後条件の観測確率が更新された R である．$r(r \in R)$ が持つ事後条件の数が N 個のとき，推定観測確率を θ_i とすると，式 (4) が成り立つ．

$$\sum_{i=1}^{N} \theta_i = 1 \tag{4}$$

更新された観測確率は，次の学習と環境モデルの構築に用いられる．環境モデルの構築では，更新された各事後条件の推定観測確率と ζ の値を比較し，ζ の値を上回る推定観測確率を持つ事後条件を環境モデルへ採用する．ζ の値を下回り不採用となった事後条件は，環境モデルからは削除されるが，R から削除されることはない（R 内の確率は更新される）．したがって，観測結果により再度環境モデルへ採用される場合がある．例として，**図 6** の規則が得られた場合，a_{m2}，a_{m3} は ζ の値を上回るため，環境モデルへ採用される．一方，a_{m4} は ζ の値を下回るため不採用となり，a_{m2}，a_{m3} のみを含む図 6 のような環境モデルが構築される．

Algorithm 1 差分学習のアルゴリズム

Input: R, ζ, $\langle pre_o, act_o, post_o\rangle$(Obtained data)
Output: updated R

```
1: for all r ∈ R do
2:   if r.pre == pre_o and r.act == act_o then
3:     for all post ∈ r.Post do
4:       θ_post ← θ_post − η ∂MSE/∂θ_post
5:     end for
6:     for all post ∈ r.Post do
7:       θ_post ← θ_post / sum(r.Post)
8:       if θ_post ≤ ζ then
9:         post.rule ← false
10:      else
11:        post.rule ← true
12:      end if
13:    end for
14:  end if
15: end for
16: return R
```

5.3 実行時差分更新

差分学習のアルゴリズムを Algorithm 1 に示す．$\langle pre_o, act_o, post_o\rangle$ は学習時点で新規に得られたアクションセットである．新しいアクションセットが得られるたびに，差分学習を行う．まず，得られたアクションセットと同様の事前条件，制御動作を持つ規則 r を抽出する（2 列目）．抽出された r について，r が持つ事後条件群が $Post$ のとき，$Post$ に含まれる各事後条件 $post \in Post$ の観測確率 θ_{post} を，確率的勾配降下法と同様の計算により推定する（4 列目）．計算結果と入力した ζ をもとに，高い推定観測確率を持つ $post$ は環境モデルへ採用される（6 から 13 列目）．$sum(r.Post)$ は $\theta_{post \in Post}$ の合計値である．$post.rule$ が $true$ であれば $post$ は採用され，$post.rule$ が $false$ であれば $post$ は不採用となる．

従来手法では，学習時点から過去のある一定期間に得られたアクションセットをもとに，勾配降下法を用いて計算を行う．パラメータ更新は，誤差関数が収束するまで繰り返される．これに対して本手法では，新しいアクションセットを得る度に，確率的勾配降下法を用いて計算を行い，パラメータ更新は 1 度だけ行う．確率的勾配降下法は，勾配法の 1 種であり，1 つのデータを読み込んだ際にそのデータのみを使って勾配を計算し，パラメータを更新する手法である．通常の確率的勾配降下法では，与えられた全データから，

(1) ランダムに 1 つのデータを選択
(2) 選択したデータを用いてそれまでに計算した勾配を更新

を繰り返す．本手法では，(1) のランダムな選択の代わりに，時系列上の最新データを利用する．すなわち，

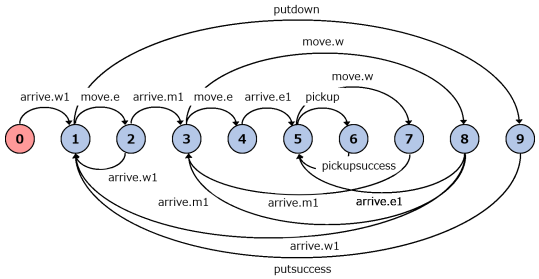


図 7 更新前の環境モデル
Fig. 7 Environment model before updating.

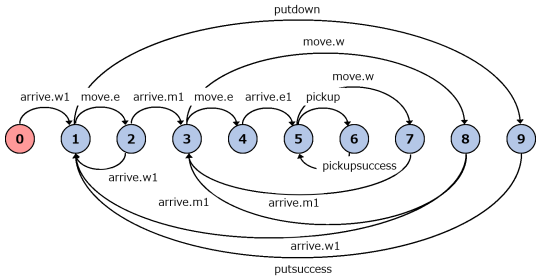


図 8 更新後の環境モデル
Fig. 8 Environment model after updating.

(1) 実行時に得られた最新データを取得
(2) 得られたデータを用いて過去に計算した勾配を更新
を実行時にデータを得るたびに実行し，モデルを更新する．

誤差関数は式 (5) のように定義する．誤差関数を最小化するようなパラメータ $\mathbf{p}$ を求め，事後条件の観測確率を推定する．

$$MSE(\mathbf{p}) = (1 - P(post_o|Post_{\langle pre_o, act_o\rangle}))^2 \tag{5}$$

上式において，$Post_{\langle pre_o, act_o\rangle}$ を C と置くと，C は pre_o と act_o を含む r が持つ事後条件群である．$\mathbf{p}$ は $\theta_{post\in C}$ のベクトルである．$P(post_o|C)$ の計算には式 (6) を，また $\mathbf{p}$ の更新には式 (7) を用いる．

$$P(post_o|C) = \frac{\sum_{\{post\in C, post\models post_o\}} \theta_{post}}{\sum_{\{post\in C\}} \theta_{post}} \tag{6}$$

$$\mathbf{p}_{t+1} = \mathbf{p}_t - \eta\nabla MSE(\mathbf{p}_t) \tag{7}$$

上式によって推定された $\mathbf{p}$ の値から，事後条件の推定観測確率を求め，あらかじめ与えている ζ と比較し，環境モデルへ採用するか否かを決定する．最終的には，ζ の値を上回る推定観測確率を持つ事後条件，事前条件，制御動作の集合を LTS 形式の環境モデルに変換する．

ここで，自動倉庫管理システムの例題を用いて，具体的な計算例を示す．更新前の環境モデルを**図 7** に示す．システムの実行中に，次のアクションセットが得られたとする．

$$\langle arrive.m1, move.w, arrive.m1\rangle$$

この場合，学習のために抽出される規則は次のような規則である．

$$\langle arrive.m1, move.w, \{arrive.m1, arrive.e1, arrive.w1\}\rangle$$

この規則は，得られたアクションセットと同じ事前条件と制御動作，また 3 つの事後条件 $post1$, $post2$, $post3$ を持つ．各事後条件は，それぞれパラメータ θ_{post1}, θ_{post2}, θ_{post3} を持つ．これらのパラメータは，式 (7) をもとに更新される．更新されたパラメータの値が $\theta_{post1} = 1.4$，$\theta_{post2} = 0.2$，$\theta_{post3} = 0.4$ であった場合，各事後条件の推定観測確率は次のようにして求められる．

$$post1 : \frac{\theta_{post1}}{\theta_{post1} + \theta_{post2} + \theta_{post3}} = 0.7$$

$$post2 : \frac{\theta_{post2}}{\theta_{post1} + \theta_{post2} + \theta_{post3}} = 0.1$$

$$post3 : \frac{\theta_{post3}}{\theta_{post1} + \theta_{post2} + \theta_{post3}} = 0.2$$

ここで得られた値は事前に入力された ζ と比較される．ζ が 0.15 の場合，$\theta_{post1} = 0.7$ と $\theta_{post3} = 0.2$ は ζ よりも大きいため，$post1$ と $post3$ は環境モデルへ採用される．一方，$\theta_{post2} = 0.1$ は ζ よりも小さいため，$post2$ は不採用となる．したがって，環境モデルの構築に用いられる規則は次のようになる．

$$\langle arrive.m1, move.w, \{arrive.m1, arrive.w1\}\rangle$$

この規則がその時点での環境モデルが持つ規則と異なる場合は，得られた規則をもとに環境モデルを更新する．更新後の環境モデルを**図 8** に示す．図 7 の環境モデルと比較すると，図 7 の状態 8 から 5 に出ていた遷移（arrive.e1）が図 8 では削除されていることが確認できる．

6. 評価

得られた環境モデルの正確度，その収束時間，モデル更新に要する計算時間について，勾配降下法を用いた従来手法 [21] との比較により評価する．以下の 3 つを研究課題とする．

研究課題 1 どの程度の正確度でモデル更新ができるのか．
研究課題 2 正確度の収束時間に違いはあるか．
研究課題 3 一度のモデル更新に要する計算時間はどの程度削減できるのか．

上記 3 つの研究課題について，2 つの異なる規模の自動倉庫管理システムを例題としたケーススタディを行う．各例題の設定を**表 1** に示す．

6.1 評価指標

研究課題 1 では，学習結果の正確度と，得られた環境モデルの正確度を評価する．学習結果の正確度は誤差の大きさをもとに評価する．誤差は，「実行トレース生成時に設定した事後条件の真の観測確率と，従来手法・提案

表 1 各例題の設定
Table 1 Settings.

設定	小規模	大規模
エリア数	3	153
規則数	10	1,171
事後条件数	26	5,259

表 2 予測結果と真の結果の関係
Table 2 Relationship between prediction result and true result.

		真の結果	
		正	負
予測結果	正	TP	FP
	負	FN	TN

手法を用いた計算によって得られた推定観測確率の値の差」とする．ある事後条件 i に関する学習結果の正確度を $Accuracy_{learning\_i}$，真の観測確率を p_{true_i}，推定観測確率を p_i とすると，学習結果の正確度は次の式で表される．

$$Accuracy_{learning\_i} = 1 - |p_{true\_i} - p_i|$$

環境モデルの正確度は，予測結果と真の結果を**表 2** のように分類し，正や負と予測したデータのうち実際にそうであるものの割合とし，次の式で表される．ここで，正は環境モデルへ採用，負は環境モデルへ不採用，を表す．

$$Accuracy_{model} = \frac{TP + TN}{TP + FP + TN + FN}$$

研究課題 2 については，環境に変化が生じてから環境モデルの正確度が収束するまでに実行する動作数を評価する．研究課題 3 については，1 度のモデル更新において実行トレースの読み込みから最後のパラメータ更新までに要した時間を計算時間とし，評価する．

6.2 評価設定

従来手法の入力とする実行トレースは，計算時点までの一定のタイムウィンドウ内で得られたデータとする．タイムウィンドウ長は小規模な例題では計算時点から過去 3,001 動作分，大規模な例題では 300,001 動作分とする．自動倉庫管理システムの例題において，実行トレース長を変化させて従来手法の正確度に関する予備実験を行い，これらの実行トレース長を用いることで学習結果が収束するという結果が得られたためである．また，従来手法・提案手法ともに，パラメータ $\mathbf{p}$ の初期値は 0.5，環境モデルへ採用・不採用とする基準となる閾値 ζ は 0.1 とする．

実験を行うにあたり，大小 2 つの例題において次の 2 つの環境を用意した．

- 環境 1：観測可能な動作が決定的である環境
- 環境 2：観測可能な動作が非決定的である環境

環境 1 から環境 2 への変化，環境 2 から環境 1 への変化を

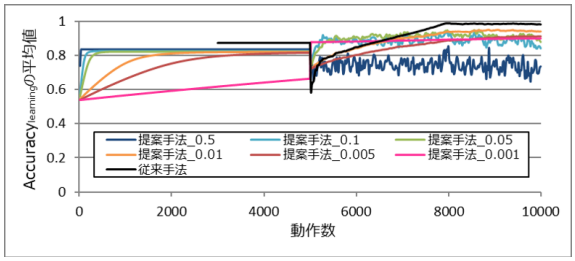


図 9 学習率を変化させた場合の学習結果の正確度（小規模な例題，環境 1→環境 2）
Fig. 9 Accuracy of learning results (small case, env.1 → env.2).

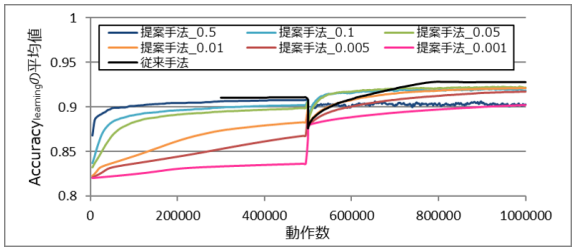


図 10 学習率を変化させた場合の学習結果の正確度（大規模な例題，環境 1→環境 2）
Fig. 10 Accuracy of learning results (large case, env.1 → env.2).

含む実行トレースを用意し，それぞれについて実験を行った．小規模な例題では行った動作数が 5,001 となった時点で，大規模な例題では行った動作数が 500,001 となった時点で環境を変化させた．

6.3 パラメータの更新手法

勾配法におけるパラメータの更新手法は複数存在する．今回の実験では，固定の学習率 $\eta = 0.5$，0.1，0.05，0.01，0.005，0.001 に加え，既存のパラメータ更新手法を用いた実験を行った．適用した既存のパラメータ更新手法は，Adam [15]，AdaDelta [24]，RMSProp [22]，AdaGrad [9] の 4 つである．これらの手法は，学習率を計算時に調整することで，誤差関数の素早い収束と振動の抑制を目指す手法である．AdaGrad では，各パラメータはそれぞれ異なる学習率を持ち，その学習率は計算をするたびに更新される．ここで，急速な学習率の低下を防ぐために AdaGrad を改良したものが，RMSProp，AdaDelta，Adam である．

6.4 研究課題 1：学習結果と環境モデルの正確度

本節では，各手法を用いて得られた学習結果と環境モデルの正確度について，比較評価する．

まず，学習結果の正確度について比較を行う．環境 1 から環境 2 へ変化した場合の小規模な例題の結果を**図 9** に，大規模な例題の結果を**図 10**，**図 11** に示す．また環境 2 から環境 1 へ変化した場合の小規模な例題の結果を**図 12** に，大規模な例題の結果を**図 13**，**図 14** に示す．縦軸は全

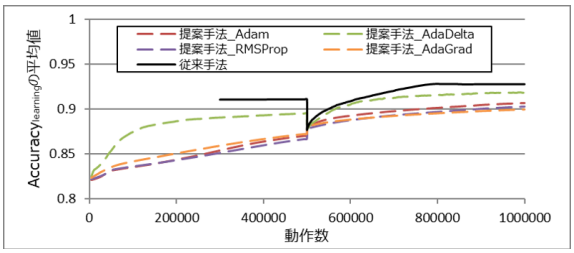


図 11 既存の学習率更新手法を用いた場合の学習結果の正確度（大規模な例題，環境 1→環境 2）

Fig. 11 Accuracy of learning results with existing method (large case, env.1 → env.2).

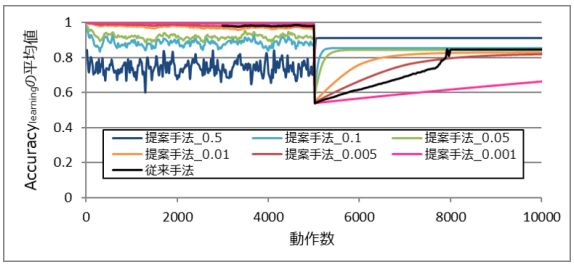


図 12 学習率を変化させた場合の学習結果の正確度（小規模な例題，環境 2→環境 1）

Fig. 12 Accuracy of learning results (small case, env.2 → env.1).

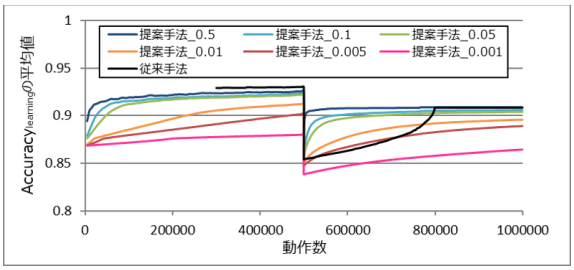


図 13 学習率を変化させた場合の学習結果の正確度（大規模な例題，環境 2→環境 1）

Fig. 13 Accuracy of learning results (large case, env.2 → env.1).

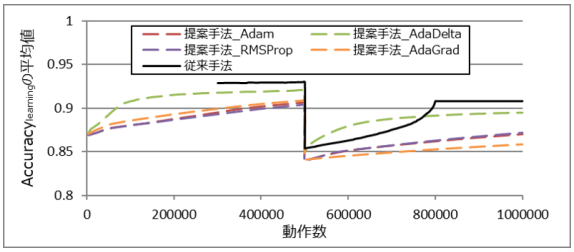


図 14 既存の学習率更新手法を用いた場合の学習結果の正確度（大規模な例題，環境 2→環境 1）

Fig. 14 Accuracy of learning results with existing method (large case, env.2 → env.1).

事後条件における正確度の平均値であり，横軸は実行動作数である．従来手法については，学習率を変化させた場合でも正確度にほとんど差は生じず，グラフの形に変化がなかったため，学習率 0.5 の場合の結果のみ載せている．小規模な例題における既存のパラメータ更新手法を用いた提案手法のグラフは省略する．

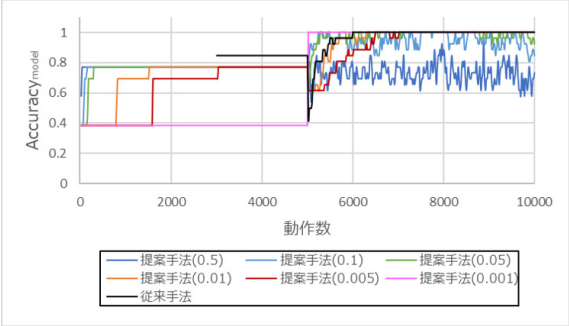


図 15 学習率を変化させた場合の環境モデルの正確度（小規模な例題，環境 1→環境 2）

Fig. 15 Model accuracy (small case, env.1 → env.2).

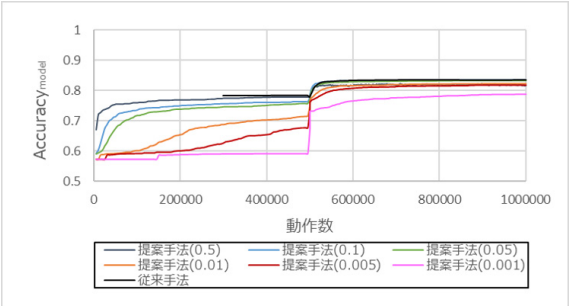


図 16 学習率を変化させた場合の環境モデルの正確度（大規模な例題，環境 1→環境 2）

Fig. 16 Model accuracy (large case, env.1 → env.2).

変化前後において，図 9 の結果を除き，環境 1 における正確度は従来手法と提案手法で同程度となっている．従来手法は十分なタイムウィンドウが設定されており，変化が生じない限りは高い正確度で推定している．提案手法では，学習率によって収束時の正確度が異なり，高い学習率を設定するほど，収束時の正確度が高くなっている．

また，環境 2 における正確度は従来手法が優れている．提案手法では，学習結果が収束したのち振動している．従来手法と適切に学習率が設定されている提案手法（学習率 0.1）を比較すると，収束後の正確度の差は 0.05 以内程度である．また既存の 4 つのパラメータ更新手法を用いた提案手法では，従来手法や固定の適した学習率を用いた提案手法よりも正確度が低くなっている．

次に，環境モデルの正確度を比較する．環境 1 から環境 2 へ変化した場合の小規模な例題の結果を**図 15** に，大規模な例題の結果を**図 16** に示す．また環境 2 から環境 1 へ変化した場合の小規模な例題の結果を**図 17** に，大規模な例題の結果を**図 18** に示す．縦軸は正確度，横軸は実行動作数である．従来手法については，学習率を変化させた場合でも正確度にほとんど差は生じず，グラフの形に変化がなかったため，学習率 0.5 の場合の結果のみ載せている．

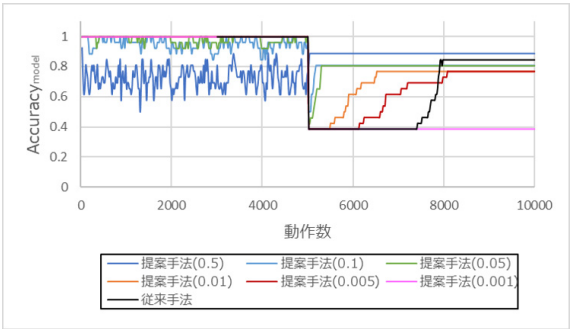


図 17 学習率を変化させた場合の環境モデルの正確度（小規模な例題，環境 2→環境 1）

Fig. 17 Model accuracy (small case, env.2 → env.1).

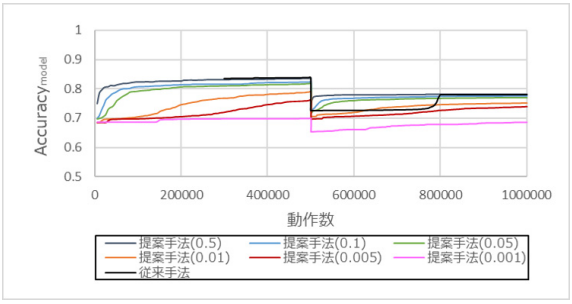


図 18 学習率を変化させた場合の環境モデルの正確度（大規模な例題，環境 2→環境 1）

Fig. 18 Model accuracy (large case, env.2 → env.1).

既存の 4 つのパラメータ更新手法を用いた提案手法のグラフは省略する．

小規模な例題の結果において，変化前後の環境 1 における正確度は従来手法と同程度，環境 2 における正確度は従来手法が優れており，学習結果の正確度とほとんど同様の結果となっている．一方，大規模な例題の結果において，変化前後の環境 1 における正確度は従来手法と同程度，環境 2 における正確度も従来手法と同程度となっている．既存の 4 つのパラメータ更新手法を用いた提案手法の結果は，従来手法や固定の適した学習率を用いた提案手法よりも正確度が低かった．

6.5 研究課題 2：学習結果の収束性

本節では，環境が変化した際の正確度の収束の早さについて，比較評価する．

従来手法では，収束するまでに小規模な場合で約 3,000 動作数分，大規模な場合で約 300,000 動作数分の時間がかかっている．一方で，提案手法では適切な学習率が設定されている場合，迅速に収束する．小規模環境下では，適した学習率を用いた場合に数十～数百動作数分で収束しており，大規模環境下では，学習率 0.1，0.05 の場合，数万動作数分で収束している．既存のパラメータ更新手法を用いた提案手法では，図 14 の結果において AdaDelta では従来手法よりも迅速に収束しているが，その他の結果では従

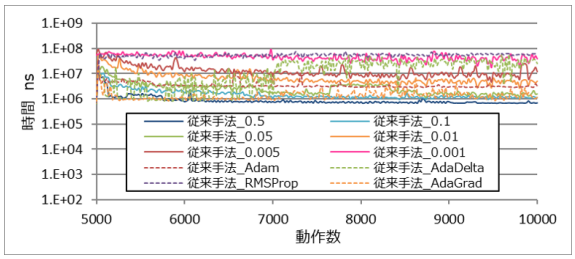


図 19 従来手法の計算時間（小規模な例題）

Fig. 19 Computation time (small case, existing method).

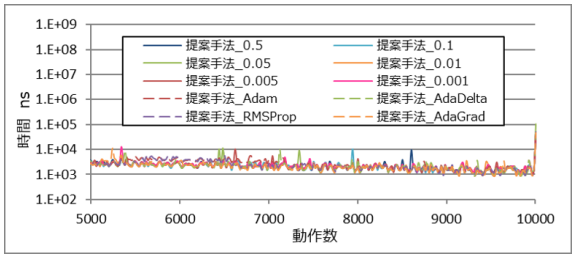


図 20 提案手法の計算時間（小規模な例題）

Fig. 20 Computation time (small case, proposed method).

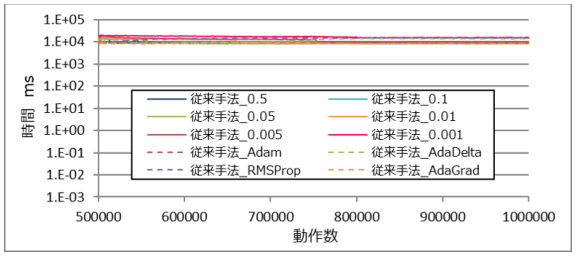


図 21 従来手法の計算時間（大規模な例題）

Fig. 21 Computation time (large case, existing method).

来手法よりも収束までの動作数が多くなっている．

また提案手法では，学習率が極端に高い，もしくは低い場合は，性能が悪化する．学習率が極端に小さい 0.001 の場合，変化をモデルに反映するまでに多大な時間を要し，収束時間が従来手法より遅くなっている．一方で，学習率が極端に大きい 0.5 の場合，環境に変化が生じていなくともモデルが安定せず，収束しない，もしくは収束しても正確度が低くなっている．今回実験した範囲内では，学習率を 0.1 から 0.05 の範囲にすることで，従来手法よりも迅速に収束し，変化直後においてより高い正確度の環境モデルを得ることができる．

6.6 研究課題 3：計算時間

本節では，1 度のモデル更新に要する計算時間について評価する．

環境 1 から環境 2 へ変化した場合の計算時間について，小規模な例題に関する結果を**図 19**，**図 20** に，大規模な例題に関する結果を**図 21**，**図 22** に示す．縦軸は計算時間，横軸は行った動作数である．

図 19，図 20 を比較すると，小規模な例題では，従来手

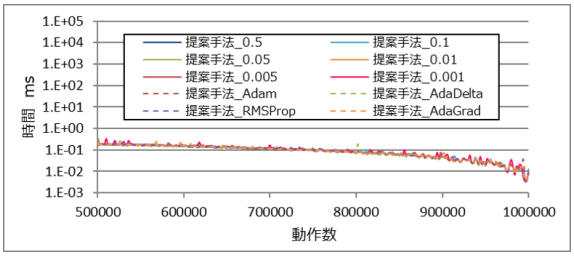


図 22 提案手法の計算時間（大規模な例題）
Fig. 22 Computation time (large case, proposed method).

法の計算時間は提案手法の 1,000～10,000 倍となっている．また図 21，図 22 を比較すると，大規模な例題では，従来手法の計算時間は提案手法の 100,000 倍となっている．規則数が約 117 倍となることで，計算時間は従来手法では 1,000～10,000 倍，提案手法では 10 倍となっていることが分かる．また図 19 から図 22 の結果において，計算時間の平均増加量は，従来手法では約 12,000 ミリ秒，提案手法では約 0.12 ミリ秒となっており，従来手法では計算時間が大幅に増加していることが分かる．

6.7 評価結果のまとめ

3 つの研究課題について実験を行い，それぞれ次のような結果が得られた．

研究課題 1 従来手法と提案手法によって得られる環境モデルの正確度は，決定的な環境下においては同程度，非決定的な環境下においては従来手法が優れている．

研究課題 2 適した学習率を用いた場合，従来手法よりも提案手法の方が正確度の収束時間が早い．

研究課題 3 提案手法を用いることで，モデル更新に要する計算時間は大幅に削減可能である．

自己適応システムは変化に対して迅速に意思決定することが求められる．その点において，提案手法は変化に対してより迅速に正確度の高い環境モデルを得ることができることが確認できた．

提案手法の性能は学習率により異なり，今回の実験の例題においては 0.1 の学習率が適していることが分かった．変化する環境において正確かつ迅速な環境モデル更新を実現するにあたり，適した学習率を設定する必要がある．適した学習率はドメイン依存であることが予想され，最適なものを事前に特定することは困難である．実行時情報を用いた学習率の更新は将来研究とする．

7. 考察

7.1 研究課題 1 の考察

図 9，図 15 の変化前の決定的な環境 1 において，従来手法の正確度が優れているが，この差は従来手法によって得られた十分に学習されていない結果が偶発的に正解に近くなったために得られたものである．小規模な例題の結果では，1 度しか観測されていないある 1 つの規則の学習結果により，図 9，図 15 に示されるような差が生じている．1 度しか観測されていない規則が存在する場合，観測された事後条件の推定観測確率は，従来手法では 1.0，提案手法では初期値に近い値となり，学習結果に大きく差が生じてしまう．1 度しか観測されていない規則を除いて正しい事後条件数を比較すると，その差はなくなることから，すべての規則が十分に観測された場合は正確度の差は微小となり，どちらの手法を用いても同程度の正確度での学習が可能であることが推測できる．

提案手法では，学習率が大きくなるほど新しく得られた観測結果を学習結果に大きく反映するようになる．そのため，学習率を大きくした場合に，環境 1 のような決定的な環境下では良い学習結果が得られるが，環境 2 のような非決定的である環境下では，学習結果が不安定となり正確度が落ちてしまう場合がある．一方，学習率が小さい場合，新しく得られた観測結果は学習結果に小さく反映される．そのため，安定した学習が可能となり，図 9 から図 18 に示される結果が得られたと考えることができる．

図 9 から図 18 の結果より，非決定的な環境 2 において，提案手法の学習結果が振動している．学習結果の振動により，環境モデルの更新が頻発してしまう場合がある．環境モデルの更新によるコントローラの再生成には時間を要するため，環境モデルの更新が頻発した場合はシステムの実行に影響を与えてしまう．学習結果は閾値により変化するため，閾値を変更することで学習結果の振動を回避することが可能であるが，そのような閾値を開発時に決定することは難しい．非決定的な環境下で提案手法を用いて安定した学習を実現するためには，閾値の実行時の調整が必要となることが推測できる．

環境モデルの正確度について，図 16，図 18 より，大規模な例題では環境 2 においても従来手法と同程度であり，今回の実験設定では，学習結果の誤差が環境モデルの構築に与える影響は小さかった．しかしながら，例題や閾値により，学習結果の誤差が環境モデルの構築に与える影響が大きくなる場合もあると考えられる．

7.2 研究課題 2 の考察

従来手法では，勾配の計算時に過去の環境における観測結果の影響を大きく受けてしまう．それに対し，提案手法では計算時点で得られた観測結果のみを考慮していることから，過去の環境における観測結果の影響を受けにくい．また，提案手法では学習率が大きいほど新しく得られた観測結果が学習結果に大きく反映されやすい．これらのことから，大きな学習率を用いた場合の提案手法の学習結果において良い結果が得られたと考えられる．

図 11，図 14 より，既存の 4 つのパラメータ更新手法を用いた提案手法において，環境の変化後の収束性が悪く

なっているのは，既存のパラメータ更新手法が変化のない環境下で用いることを想定した手法であるためと考えることができる．変化する環境下において有用な学習率の実行時更新手法を提案し，本手法に適用することで，より収束性の良い学習が実現可能となることが推測できる．

今回の実験では，環境を 1 度だけ変化させて実験を行っているが，実際には環境の変化はたびたび発生するものである．その際，従来手法では学習結果の収束に時間がかかるために，学習結果の収束前に新たな環境の変化が発生する場合が多く存在することが考えられる．そのような場合，変化前の環境における観測結果の影響により，実際の環境を正確に表現するモデルの学習が困難になってしまうことが予想される．たびたび変化する環境下で実環境を正確に表現するモデルを実行時に構築するためには，学習結果が素早く収束することが重要であると考えられる．

7.3 研究課題 3 の考察

一度のパラメータ更新に要する計算量は，従来手法では $O(n)$，提案手法では $O(1)$ である．従来手法では規則数の増加によりパラメータの更新回数も増加するため，システムの規模が大きくなるにつれて計算時間が大幅に増加し，提案手法では従来手法に比べて計算時間の増加量は少なくなることが予想される．

大規模な例題では，従来手法を用いた場合の計算時間が 10〜20 秒程度となっている．新しいアクションセットが得られるたびにモデル更新を行うことを考えると，1 度のモデル更新に 10 秒以上要するのは現実的ではない．自動倉庫管理システムにおいても，計算時間の増加はロボットの作業効率の低下につながってしまう．一方，提案手法では一度のモデル更新は 1 ミリ秒以下で行うことが可能である．これはロボットの作業効率にほとんど影響を与えることなく，無視することができる値である．

7.4 本手法の限界

本手法は，変化する環境下での正確かつ迅速な学習を実現するにあたり，適した学習率を用いた場合に有用である．しかしながら，適した学習率はシステムの規模や仕様によって異なることが予想され，最適な値を事前に特定することは困難である．したがって，実行時により適した学習率を特定することが求められる．実験で用いた既存のパラメータ更新手法は学習結果に応じて学習率を変化させる手法であるが，これらの手法は変化する環境下で用いるには収束性の点であまり適していなかった．変化する環境下で有用なパラメータ更新手法とあわせて本手法を用いることができれば，適した学習率を事前に特定することなく，正確かつ収束性の良い学習が実現可能であると考えられる．

環境モデルの構成要素である規則は，あらかじめ用意しており，規則中に存在しないアクションセットを観測した場合は学習されない．したがって，観測しうるアクションセットはシステムの開発時に洗い出し，規則として用意する必要がある．またアクションセットは，制御可能な動作，観測可能な動作を交互に実行・受け取ることを想定し，制御可能な動作とその前後の観測可能な動作の組としている．しかしながら，制御可能な動作を複数回実行するようなシステム，ある 1 つの制御可能な動作に対して複数の観測可能な動作を受け取るようなシステムも存在すると考えられる．

7.5 本手法の妥当性への脅威

実験結果は，調整するパラメータの初期値や，従来手法において用いる実行トレース長，閾値によって変化する可能性がある．既存のパラメータ更新手法を用いた場合についても，設定値を変更することで結果が変わる可能性がある．

8. まとめ

本論文では，環境変化を正確，迅速，かつ少ない計算オーバヘッドで環境モデルに反映することを目的とし，実行時に得られるデータから効率良く環境モデルを実行時更新するための差分学習手法を提案した．確率的勾配降下法を応用して差分学習をすることで，一度のモデル更新に要するデータ量と時間を削減し，実用的な実行時間でのモデル更新を可能とした．評価では，環境モデルの正確度，収束時間，計算時間について，従来手法との比較を行い，(1) 従来手法と提案手法の正確度は決定的な環境下においては同程度，非決定的な環境下においては従来手法が優れていること，(2) 提案手法では適した学習率を用いることで迅速に高い正確度の環境モデルを構築可能であること，(3) 提案手法を用いることで 1 度のモデル更新に要する計算時間は大幅に削減可能であること，を確認した．変化する環境下で環境モデルを実行時に構築するためには，素早く正確にモデル更新することが重要であり，本手法は有用であるといえる．

本手法では，環境モデルの構成要素である規則はあらかじめ用意する必要がある．想定外のアクションを観測した場合の学習手法は将来研究とする．また，実験結果より，本手法では適切に学習率を設定する必要があることが示された．本手法では学習率は固定としたが，将来研究では，変化する環境下で有用な実行時の学習率調整手法を本手法に応用し，適切な学習率に自動で調整するように拡張する予定である．また，本論文で提案した LTS ベースの環境モデル更新手法を，同じく LTS ベースの環境モデルを扱うコントローラ生成手法に基づいて意思決定する自己適応システムに組み込み，環境モデル更新の正確度，収束時間，計算時間がシステムの品質維持にどのような影響を与えるか，評価を行う予定である．

謝辞 本研究の一部はJSPS科研費18H03225，17H00732，および独立行政法人情報通信研究機構（NICT）の委託研究「欧州との連携によるハイパーコネクテッド社会のためのセキュリティ技術の研究開発」の成果です．

参考文献

[1] Bottou, L.: Online learning and stochastic approximations, *On-line Learning in Neural Networks*, Vol.17, No.9, p.142 (1998).

[2] Braberman, V., D'Ippolito, N., Kramer, J., Sykes, D. and Uchitel, S.: Morph: A reference architecture for configuration and behaviour self-adaptation, *Proc. 1st International Workshop on Control Theory for Software Engineering*, pp.9–16, ACM (2015).

[3] De Lemos, R., Giese, H., Müller, H.A., Shaw, M., Andersson, J., Litoiu, M., Schmerl, B., Tamura, G., Villegas, N.M., Vogel, T., et al.: Software engineering for self-adaptive systems: A second research roadmap, *Software Engineering for Self-Adaptive Systems II*, pp.1–32, Springer (2013).

[4] Ding, Z., Zhou, Y. and Zhou, M.: Modeling self-adaptive software systems with learning Petri nets, *IEEE Trans. Systems, Man, and Cybernetics: Systems*, Vol.46, No.4, pp.483–498 (2016).

[5] D'Ippolito, N., Braberman, V., Kramer, J., Magee, J., Sykes, D. and Uchitel, S.: Hope for the best, prepare for the worst: Multi-tier control for adaptive systems, *Proc. 36th International Conference on Software Engineering*, pp.688–699, ACM (2014).

[6] D'ippolito, N., Braberman, V., Piterman, N. and Uchitel, S.: Synthesizing nonanomalous event-based controllers for liveness goals, *ACM Trans. Software Engineering and Methodology* (*TOSEM*), Vol.22, No.1, p.9 (2013).

[7] D'Ippolito, N., Braberman, V., Sykes, D. and Uchitel, S.: Robust degradation and enhancement of robot mission behaviour in unpredictable environments, *Proc. 1st International Workshop on Control Theory for Software Engineering*, pp.26–33, ACM (2015).

[8] D'Ippolito, N.R., Braberman, V., Piterman, N. and Uchitel, S.: Synthesis of live behaviour models, *Proc. 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp.77–86, ACM (2010).

[9] Duchi, J., Hazan, E. and Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization, *Journal of Machine Learning Research*, Vol.12, No.Jul, pp.2121–2159 (2011).

[10] Fahland, D. and van der Aalst, W.M.: Repairing process models to reflect reality, *International Conference on Business Process Management*, pp.229–245, Springer (2012).

[11] Filieri, A., Ghezzi, C. and Tamburrelli, G.: Run-time efficient probabilistic model checking, *Proc. 33rd International Conference on Software Engineering*, pp.341–350, ACM (2011).

[12] Garlan, D.: Software engineering in an uncertain world, *Proc. FSE/SDP Workshop on Future of Software Engineering Research*, pp.125–128, ACM (2010).

[13] Ghezzi, C., Pinto, L.S., Spoletini, P. and Tamburrelli, G.: Managing non-functional uncertainty via model-driven adaptivity, *2013 35th International Conference on Software Engineering* (*ICSE*), pp.33–42, IEEE (2013).

[14] Iftikhar, M.U. and Weyns, D.: Activforms: Active formal models for self-adaptation, *Proc. 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pp.125–134, ACM (2014).

[15] Kingma, D.P. and Ba, J.: Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).

[16] Menashe, J. and Stone, P.: Monte carlo hierarchical model learning, *Proc. 2015 International Conference on Autonomous Agents and Multiagent Systems*, International Foundation for Autonomous Agents and Multiagent Systems, pp.771–779 (2015).

[17] Moreno, G.A., Cámara, J., Garlan, D. and Schmerl, B.: Proactive self-adaptation under uncertainty: A probabilistic model checking approach, *Proc. 2015 10th Joint Meeting on Foundations of Software Engineering*, pp.1–12, ACM (2015).

[18] Polak, E.: *Optimization: Algorithms and consistent approximations*, Vol.124, Springer Science & Business Media (2012).

[19] Salehie, M. and Tahvildari, L.: Self-adaptive software: Landscape and research challenges, *ACM Trans. Autonomous and Adaptive Systems* (*TAAS*), Vol.4, No.2, p.14 (2009).

[20] Sharifloo, A.M., Metzger, A., Quinton, C., Baresi, L. and Pohl, K.: Learning and evolution in dynamic software product lines, *2016 IEEE/ACM 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems* (*SEAMS*), pp.158–164, IEEE (2016).

[21] Sykes, D., Corapi, D., Magee, J., Kramer, J., Russo, A. and Inoue, K.: Learning revised models for planning in adaptive systems, *2013 35th International Conference on Software Engineering* (*ICSE*), pp.63–71, IEEE (2013).

[22] Tieleman, T. and Hinton, G.: Lecture 6.5-RMSProp, COURSERA: Neural networks for machine learning, University of Toronto, Technical Report (2012).

[23] Yuan, E., Esfahani, N. and Malek, S.: Automated mining of software component interactions for self-adaptation, *Proc. 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pp.27–36, ACM (2014).

[24] Zeiler, M.D.: ADADELTA: An adaptive learning rate method, arXiv preprint arXiv:1212.5701 (2012).

田邉 萌香

2017年早稲田大学大学院基幹理工学研究科情報理工・情報通信専攻修士課程修了．株式会社NTTデータ・アイを経て2018年より早稲田大学大学院基幹理工学研究科情報理工・情報通信専攻博士課程．現在に至る．

鄭 顕志 （正会員）

2008年早稲田大学大学院理工学研究科情報・ネットワーク専攻博士課程修了．2006年同大学理工学部助手．2007年同大学基幹理工学部助手．2008年同大学メディアネットワークセンター助教．2010年国立情報学研究所アーキテクチャ科学研究系助教．2015年同准教授．2018年より早稲田大学理工学術院総合研究所研究院准教授/主任研究員．現在に至る．博士（工学）（早稲田大学）．自己適応ソフトウェア，ソフトウェアアーキテクチャ，モデル駆動工学の研究に従事．

本位田 真一 （正会員）

1978年早稲田大学大学院理工学研究科修士課程修了．（株）東芝を経て2000年国立情報学研究所教授，2012年同研究所副所長を兼務．2001年東京大学大学院情報理工学系研究科教授を兼任，2018年より早稲田大学理工学術院教授，現在に至る．現在，英国UCL客員教授ならびに国立情報学研究所GRACEセンター長を兼任．2005年度パリ第6大学招聘教授．2015年度リヨン第1大学招聘教授．日本学術会議連携会員．本会フェロー．