情報処理学会論文誌 Vol.54 No.2 797–806 (Feb. 2013)

拡張有限状態機械を用いた運用プロファイルベースドテストのテストケース生成手法とツール構成

高木 智彦[1,a)] 八重樫 理人[1] 古川 善吾[1]

受付日 2012年5月14日, 採録日 2012年11月2日

概要：拡張有限状態機械に基づく運用プロファイルから，usage distribution coverage と N スイッチ網羅率が大きいテストケースを効果的に生成するためのテストケース生成手法とツール構成を提案する．拡張有限状態機械には遷移間にデータ依存性があるため，テストケースの実行可能性を考慮する必要がある．また，テスト工程に割り当てられる労力は有限なので，生成されるテストケースの実行に要する労力があらかじめ指定される上限を超えないようにする必要がある．本稿では，これらの問題にも対処した，遺伝的アルゴリズムによるテストケース生成手法を示す．さらに，本手法を実装したテストケース生成ツールと既存のモデリングツールを連携させたツール構成について検討する．商用ソフトウェアに適用した結果，拡張有限状態機械の使用によって複雑な振舞いを持つソフトウェアに対して効果的に適用できること，usage distribution coverage と N スイッチ網羅率を同時に改善できることが確認できた．

キーワード：ソフトウェアテスト，運用プロファイル，テストケース生成

Test Case Generation Method and Its Tool Configuration of Operational Profile-based Testing Using Extended Finite State Machines

Tomohiko Takagi[1,a)] Rihito Yaegashi[1] Zengo Furukawa[1]

Received: May 14, 2012, Accepted: November 2, 2012

Abstract: This paper shows a test case generation method and its tool configuration in which an operational profile that is based on an EFSM (extended finite state machine) automatically produces test cases to achieve high usage distribution coverage and high N-switch coverage effectively. The EFSM includes data dependencies among its transitions, which requires addressing the executability of test cases. Also, the effort devoted to executing given test cases should not exceed the finite efforts for a test process. In this paper, we propose a test case generation method using a genetic algorithm also to address these problems. Additionally, we consider a tool configuration that consists of an existing modeling tool and a test case generation tool that implements this method. The case study indicates that EFSMs make it possible to apply this method to software including complex behavior, and this method can improve both usage distribution coverage and N-switch coverage.

Keywords: software testing, operational profile, test case generation

1. はじめに

ソフトウェアテストはソフトウェアのフォールト（不具合）を出荷前に効果的に発見する技術であり，要求される品質を確保するための重要な役割をソフトウェア開発工程

[1] 香川大学工学部
Faculty of Engineering, Kagawa University, Takamatsu, Kagawa 761–0396, Japan
[a)] takagi@eng.kagawa-u.ac.jp

において担っている．近年ではテスト対象ソフトウェアの期待される振舞いを表すモデルに基づいてテストケースを設計する MBT（model-based testing）が注目されており，さかんに研究が行われている．たとえば，文献 [1], [2] は，UML（unified modeling language）のステートマシン図上の経路を網羅するテストケース設計手法を示している．このように，モデルの構成要素を網羅するテストケースを設計することによって満遍なくフォールトを洗い出すことを目的とした MBT が数多く提案されている．その一方で，ソフトウェア信頼性 [3] を評価したり，ソフトウェア信頼性に深刻な影響を与えるフォールトを重点的に発見したりすることの重要性も近年認識されつつあり，網羅性を指向するのではなくそのようなソフトウェア信頼性の観点でテストケースを設計する手法として OPBT（operational profile-based testing）[4], [5] が注目されている．

OPBT では，テスト対象ソフトウェアの振舞いを FSM（finite state machine）として定義し，これにユーザの利用特性を表す確率分布を付加したモデルを用いる．このモデルは運用プロファイル（operational profile）と呼ばれ，数学的には単純マルコフ連鎖と見なすことができる．OPBT におけるテストケースは，運用プロファイル上の開始状態で始まり終了状態で終わる遷移列であり，確率分布に基づいてランダムに生成される．我々は，OPBT の有効性を高めるために，GA（genetic algorithm）を用いてテストケースを最適化する手法を提案した [6]．この手法では，テストに投入可能な労力をあらかじめ指定しておくと，その範囲内で UDC（usage distribution coverage）[7] と呼ばれるメトリクスの値が大きいテストケースを選りすぐって生成するので，逼迫したテスト工程においても OPBT を効果的に導入することが可能である．OPBT は実際のソフトウェア開発に適用され成果をあげている [8], [9] ものの，以下に示す 2 つの課題が存在する．

- FSM は単純なモデルであるため，テスト対象ソフトウェアの複雑な振舞いを運用プロファイルによって表現することが困難な場合がある．
- 使用される可能性がほとんどない機能はテストする必要がないという考え方に基づいているが，使用頻度が低くてもテストすべき重要な機能は存在する．

従来研究において上記の課題に対する取り組みは行われていない．そこで本稿ではこれらの問題を解決するために，従来の我々の手法 [6] を発展させた OPBT のテストケース生成手法，およびその手法を実現するツール構成を提案する．本研究では，EFSM（extended finite state machine）に基づく運用プロファイルである拡張運用プロファイル（extended operational profile）を作成する．FSM よりも表現力が高い EFSM を導入することによって，1 つ目の課題を解決できる．そして UDC だけでなく，長さ $(N+1)$ の遷移列の網羅を目的とした N スイッチ網羅（$N \geq 0$）[10] も考慮したテストケースを拡張運用プロファイルから生成する．UDC によって使用頻度の高い機能を重点的にテストするだけでなく，N スイッチ網羅によって使用頻度の低い機能もテストするので，2 つ目の課題を解決できる．

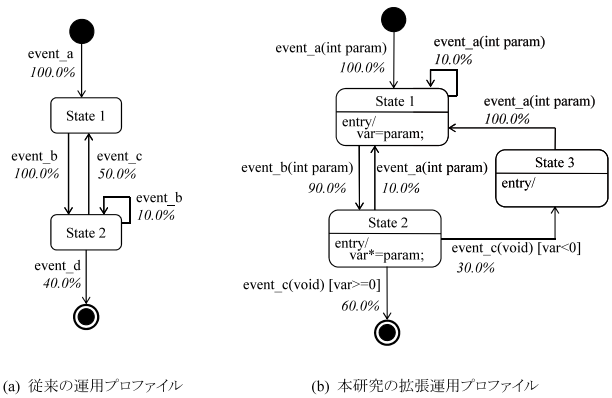
本稿の構成は以下のとおりである．まずは 2 章で本手法を構成する基本概念を述べる．3 章では拡張運用プロファイルからテストケースを生成するためのアルゴリズムを提案する．そして 4 章では本手法を実現するツール構成を提案したうえで，これを商用ソフトウェアに適用した事例に基づき有効性を議論する．5 章で考察を行い，6 章で関連研究についてまとめる．最後に 7 章で本研究を総括し，今後の課題を示す．

2. 基本概念

本章では，運用プロファイルやテストケース，メトリクスなど本手法を構成する基本概念について整理し，本研究が想定する OPBT の全体像を示す．

2.1 運用プロファイル

従来の運用プロファイルは，テスト対象ソフトウェアの振舞いを表す FSM に，ユーザの利用特性を表す確率分布を付加したモデルである．ノードはテスト対象ソフトウェアの状態，アークはイベント（ユーザの操作や外部システムからの入力）によるテスト対象ソフトウェアの状態の遷移を表す．FSM はテスト対象ソフトウェアの要求仕様に基づいて作成する．また，確率分布はテスト対象ソフトウェアの過去のバージョンや類似する他のソフトウェアの運用データ，技術者の予測に基づいて決定する．運用プロファイルの単純な例を**図 1** (a) に示す．たとえば，State 2 では event_b，event_c，event_d がそれぞれ 10%，50%，40%の確率で生起することを示している．従来の OPBT では，この確率に従ってランダムに遷移を選択していくことで，開始状態から終了状態に至る 1 本の遷移列を 1 つのテストケースとして生成する．テストケースの実行に要する労力を精密に考慮してテストケースを生成するために，各遷移



(a) 従来の運用プロファイル　(b) 本研究の拡張運用プロファイル

図 1 従来の運用プロファイルと拡張運用プロファイルの単純な例
Fig. 1 Example of conventional/extended operational profiles.

のテスト実行に必要な労力（たとえば，テストデータを適用する労力や，実行結果の正しさを確認する労力）を数値化して運用プロファイル中に記述することもできる [6]．この数値化された労力の記述を労力分布と呼ぶ．

FSM に基づく従来の運用プロファイルの問題は，テスト対象ソフトウェアの複雑な振舞いを記述することが困難な場合があるという点である．たとえば，テスト対象ソフトウェアの状態を特徴付ける変数が 2 つあり，それぞれ 10 通りの値をとりうるとすると，それだけで 100 個の状態を定義することになる．大規模で複雑な FSM の作成には相応の労力が必要であり，近年の逼迫したテスト工程においてそのような労力をかけることは現実的ではない．そこで本稿では EFSM に基づく運用プロファイルである拡張運用プロファイル[*1]を提案する．EFSM は，以下の 2 点で FSM よりも優れた表現力を備えている．

- イベントパラメータ（イベントの引数）を持ち，状態や遷移のアクションから参照することができる．イベントパラメータは，ユーザの操作や外部システムからの入力にともなうデータを表すもので，たとえば，インターネットショッピングシステムにおいてユーザが発注操作を行う際の商品 ID や発注数量などに相当する．
- テスト対象ソフトウェアの状態を特徴付ける変数を持ち，状態や遷移のアクションにおいてその変数の値を参照したり更新したりすることができる．さらに，その変数を用いて，遷移の発火条件であるガードを記述できる．アクションやガードはプログラミング言語あるいは何らかの形式言語で記述される．

EFSM の有用性は，UML ステートマシン図が開発現場で広く利用されていることからも明らかである．EFSM に確率分布や労力分布を付加したものが拡張運用プロファイルであるので，EFSM の表現力の高さはそのまま拡張運用プロファイルにも生かされる．ゆえに，拡張運用プロファイルによって従来では表現が困難であったテスト対象ソフトウェアの複雑な振舞いを表現し，その複雑な振舞いに関するテストケースを OPBT において効果的に生成できるようになる．拡張運用プロファイルの例を図 1 (b) に示す．この拡張運用プロファイルは変数 var を持ち，State 1 と State 2 のエントリーアクション（当該状態に遷移した直後に実行されるアクション）においては，当該状態への遷移のトリガとなったイベントのイベントパラメータを参照したうえで変数 var の更新を行う．たとえば，State 2 のエントリーアクションは event_b の int 型イベントパラメータ param を参照し，その値を変数 var の値に乗算して変数 var に格納する．イベントパラメータや変数などの概念が加わることによって表現力が強化される反面，拡張運用プロファイルには従来の運用プロファイルにはなかった実行可能性の問題が生じることになる．たとえば，図 1 (b) の State 2 においては，event_a と event_c がそれぞれ 10%と 90%の確率で生起し，event_c については，ガード var >= 0 を満たして終了状態に遷移する確率が 60%，ガード var < 0 を満たして State 3 に遷移する確率が 30%であることが示されている．仮に「開始状態 → event_a(−10) → State 1 → event_b(1) → State 2 → event_c() → 終了状態」という遷移列が与えられた場合，これの最後の遷移がガード var >= 0 を満たさないので実行不可能である．しかしながら，現在の状態とイベントの生起確率のみに基づいて遷移を選択する従来の OPBT のテストケース生成アルゴリズムでは，このような実行不可能な遷移列が生成される可能性がある．この問題を解決する新たなアルゴリズムを 3 章で提案する．

[*1] 従来の我々の手法 [6] では，労力分布を付加した運用プロファイルを拡張運用プロファイルと呼んでいる．本稿ではこれを EFSM の導入によってさらに拡張したものを拡張運用プロファイルと呼んでいる点に注意されたい．

2.2 テストケースとそのメトリクス

本研究におけるテストケースは，拡張運用プロファイル上の開始状態から始まり終了状態で終わる遷移列である．各遷移は，(a) 遷移元状態，(b) 遷移元状態における変数の値，(c) イベント，(d) イベントパラメータの値，(e) 遷移先状態，(f) 遷移先状態における変数の値から構成される．特に，(c)，(d) をテストデータ，(a)，(b)，(e)，(f) をテストオラクル（期待出力）と呼ぶ．(a)，(b) はテストデータ適用の事前条件，また (e)，(f) はテストデータ適用の事後条件と見なすことができる．遷移列は連続していなければならない．すなわち，(a)，(b) は直前の遷移における事後条件となるので，本手法では (a)，(b) もテストオラクルと見なす．

一般的に，テストケースは何らかのメトリクスに基づいて設計される．本研究をはじめとした FSM あるいは EFSM に基づくテスト技法におけるテストケースの設計とは，開始状態から終了状態に至るまでの間に実行する遷移を選択することを意味する．通常，単一のテストケースだけで十分なテストを行うことはできないので，メトリクスに基づいてテストケース集合（テストスイート）が構成される．また，メトリクスは，テストケースの設計や評価の基準となるものであり，従来の FSM に基づく網羅性を指向したテスト技法では，長さ $(N+1)$ の連続する遷移列をどれだけ網羅するかを表す N スイッチ網羅率が広く利用されている．たとえば，0 スイッチ網羅では，FSM のすべての遷移を網羅するテストケース集合が設計される．テストケース集合を ts とすると，ts の N スイッチ網羅率 $nsc(ts, N)$ は以下の式によって求められる．

$$nsc(ts, N) = \frac{|S(ts, N)|}{|S(N)|}$$

ここで，$S(N)$ は FSM における長さ $(N+1)$ の連続する

遷移列の集合，$S(ts, N)$ は ts によって実行される $S(N)$ の部分集合，$|S|$ は集合 S の要素数を表す．$S(N) = S(ts, N)$ のとき，「ts は N スイッチ網羅を満たす」という．

N の値を大きくするにつれて，網羅すべき遷移列の数，ひいては実行すべきテストケースの数は急激に増加する．加えて，EFSM に適用する場合では前節で述べたように実行不可能な遷移列が存在する場合があるので，すべての遷移列を網羅することは必ずしも要求されないが，できるだけ高い網羅率を達成することが望ましい．一方，1 章ですでに述べたとおり，従来の OPBT はソフトウェア信頼性に注目した手法である．本研究に導入する UDC は，テストケースあるいはテストケース集合がユーザの利用方法をどれだけ網羅するかを表す OPBT のメトリクスで，運用プロファイルの確率分布に基づいて計算される．ここで，n 個のテストケースから構成されるテストケース集合を $ts = \{tc_1, tc_2, \cdots, tc_n\}$，テストケース tc の長さ（すなわち遷移数）を $\sharp tc$，tc の先頭から j 番目の遷移確率を $p(tc[j])$ と表すとき，ts の UDC は以下の式によって求められる．

$$udc(ts) = \sum_{i=1}^{n} \prod_{j=1}^{\sharp tc_i} p(tc_i[j])$$

ただし，重複するテストケースを重複して加算しない．$udc(ts) = 1$ のとき，「ts は UDC を満たす」という．UDC は FSM に基づいて作成される運用プロファイルを前提としているが，EFSM に基づいて作成される本稿の拡張運用プロファイルにも適用可能である．

テストケースを構成する遷移の遷移確率が高いほど UDC は高くなる．UDC が高いテストケース集合ほど，ソフトウェア信頼性の評価やソフトウェア信頼性に深刻な影響を与えるフォールトの発見に効果的なので，OPBT ではできるだけ UDC の高いテストケース集合を生成する必要がある．テストケース集合を構成するテストケースの量が多いほど UDC は大きくなるものの，テスト工程に割り当てることのできる労力には限りがある．この問題を解決するために我々は，UDC ができるだけ大きく，かつ実行に要する労力があらかじめ指定される上限を超えないテストケース集合を生成する手法を提案し，その有効性を示した [6]．なお，テストケース集合 ts の実行に要する労力 $effort(ts)$ は，テストケース tc の先頭から j 番目の遷移の労力を $eff(tc[j])$ とすると以下の式によって定義される．

$$effort(ts) = \sum_{i=1}^{n} \sum_{j=1}^{\sharp tc_i} eff(tc_i[j])$$

運用プロファイルにおいて労力分布が定義されていない場合は，各遷移の労力値を一律で 1 と見なすので以下の式が成り立つ．

$$effort(ts) = \sum_{i=1}^{n} \sharp tc_i$$

従来の OPBT では，従来の FSM あるいは EFSM に基づくテスト技法において用いられる網羅性を指向したメトリクスは用いられていない．ゆえに，OPBT において生成されるテストケース集合では効果的に網羅率を高めることができず，しばしば OPBT の欠点として指摘されている．そこで本研究では UDC だけでなく N スイッチ網羅も導入し，あらかじめ指定される労力の上限を超えない範囲で，これらのメトリクスの値ができるだけ大きいテストケース集合を生成する．3 章で提案するテストケース生成アルゴリズムは，UDC と N スイッチ網羅の両方に基づくものである．なお，EFSM における UDC および N スイッチ網羅の評価に際しては，変数やイベントパラメータの値の違いによって状態や遷移を区別しないため，FSM における評価方法と同じとなる．

2.3 OPBT の概要

ソフトウェア信頼性に注目した手法である従来の OPBT は，主にシステムテストや受入れテストとして導入される．OPBT によってフォールトが発見されればそのフォールトを除去することで信頼性を効果的に改善できるし，フォールトが発見されなければテスト対象ソフトウェアの信頼性が出荷レベルに達したことの根拠の 1 つにすることができる．FSM に基づいた手法であるため，FSM を用いて表現することに適した性質を持つソフトウェア，たとえば，ユーザや外部システムからの入力に応じて状態や振舞いを変えるソフトウェアに対して特に効果的に適用できる．OPBT では運用プロファイルに基づいてテストケースを生成するため，ツールによる自動化が必須である．

本稿において提案する手法では，N スイッチ網羅によって EFSM の構成要素をできるだけ網羅することも目的とする．したがって，従来の OPBT では十分にテストを行うことが困難であった使用頻度の低い機能もテストできるようになる．また，EFSM の導入によってより複雑な振舞いを扱うことができるようになる．

本研究における OPBT の手順は以下のとおりである．

ステップ 1. テスト対象ソフトウェアの要求仕様に基づいて EFSM を作成する．

ステップ 2. EFSM 上の確率分布（および労力分布）を導出し，拡張運用プロファイルを完成させる．

ステップ 3. 開発プロジェクトの状況に応じて，生成するテストケース集合の性質や量に関するパラメータ値を決定する．

ステップ 4. 拡張運用プロファイルおよび上記パラメータに基づき，UDC および N スイッチ網羅率ができるだけ大きく，かつ実行可能なテストケース集合を生成する．

ステップ 5. 生成されたテストケース集合を実行する．

我々の従来の手法 [6] との手順上の違いは，(1) FSM で

はなくEFSMを作成すること，(2) N スイッチ網羅も考慮したテストケース集合を生成するためのパラメータ設定を行うことである．また，文献 [6] 以外の従来のOPBT（文献 [4], [5] など）との違いは，(3) GAを用いたテストケース生成アルゴリズムを使用すること，(4) 生成するテストケース集合の性質や量をコントロールするために各種パラメータ設定を行うことである．パラメータとしては，たとえば前節で述べた労力の上限がその1つとしてあげられるが，詳細は3章で述べる．

3. テストケース生成アルゴリズム

2.3節で示したステップ4では，拡張運用プロファイルからテストケースを生成するアルゴリズムが必要である．本章ではそのアルゴリズムを提案する．まずは，本手法のテストケースが満足しなければならない制約について以下に整理する．

(A) すべてのテストケースが実行可能であること．

(B) テストケース集合のUDCをできるだけ大きくすること．

(C) テストケース集合の N スイッチ網羅率をできるだけ大きくすること．

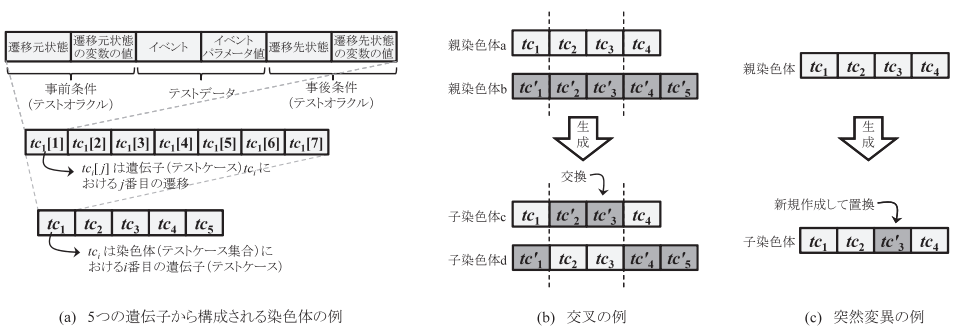
(D) テストケース集合の実行に要する労力が指定された値を超えないこと．

EFSMでは遷移間にデータ依存関係があるため，実行可能なテストケースを生成することが容易ではない．これは，ガードを満足するテストデータ集合を定式化によって求めることができない場合があることに起因している．この問題に対して，メタヒューリスティクスの一種であるGAを応用することで解決を図ろうとする研究が近年行われており，その有効性が示されている [1], [2]．加えて，OPBTでは多様なテストケース集合をランダムに繰り返し生成できることが重要である．様々な制約を考慮しつつ広大な空間をランダムに探索し，現実的な計算時間で最終的な解候補を導出できるGAはOPBTに適した技法の1つである [6]．そこで本研究では，上述の (A)〜(D) の制約を満足するテストケース集合を生成するためにGAを応用したアルゴリズムを提案する．このアルゴリズムでは，**図 2** (a) に示すように，染色体（1つの解候補をGAで扱える形式で表したもの）を1つのテストケース集合に，染色体を構成する各遺伝子を1つのテストケースに対応させる．そして，世代交代を繰り返し，自然淘汰を経て導出した最も優れた染色体を最終的な解候補として出力する．従来の我々の手法 [6] のアルゴリズムで (B)，(D) は考慮されているので，本研究では (A)，(C) にも対応するために遺伝子の作成方法と適合度関数を変更した．

本アルゴリズムは以下の手順から構成される．

ステップ 4.1. 初期集団（最初の世代を構成する染色体の集合）を生成する．この染色体は，(A)，(D) を満たすようにランダムに生成される．すなわち，拡張運用プロファイルの現在の状態において，状態のアクションの実行を終了した後，現在の状態を起点とする遷移に付随するガードをすべて評価することで次に発火可能な遷移を抽出する．そしてその中から遷移確率に比例する確率で1つの遷移をランダムに選択し実行する．イベントパラメータが存在する場合は，あらかじめ定義された値域に基づいてその値をランダムに生成する．また，遷移にアクションが付随する場合は遷移先の状態のアクションを実行する前に実行する．これらの操作を拡張運用プロファイルの開始状態から始め，終了状態に到達するまで行うことによって，1つの遺伝子を生成する．このように本アルゴリズムでは実際に拡張運用プロファイルを動作させることによって (A) を満たすことを保証する．さらに，遺伝子を (D) を満たさなくなる直前まで生成することで，1つの染色体を生成する．集団サイズ（1つの世代を構成する染色体の数）としてあらかじめ指定された s 個（$s > 1$）の染色体を生成すれば初期集団が完成する．

ステップ 4.2. 交叉と突然変異によって新たな染色体を生成し，現在の世代に追加する．本アルゴリズムにおける交叉と突然変異の概要を図2 (b)，(c) に示す．交叉とは，ランダムに選択した染色体（親染色体）のペアの間で遺伝子を交換することによって新たな染色体（子染色体）を生成する操作のことである．また，突然変異とは，ランダムに選択した染色体（親染色体）



(a) 5つの遺伝子から構成される染色体の例　(b) 交叉の例　(c) 突然変異の例

図 2 染色体，交叉，突然変異の概要

Fig. 2 Overview of chromosome, crossover, and mutation.

の一部の遺伝子を，ステップ 4.1 で示した方法で別途新たに作成した遺伝子と交換することによって新たな染色体（子染色体）を生成する操作のことである．交叉において親として選択される確率は交叉率，突然変異において交換するべき遺伝子として選択される確率は突然変異率といい，それぞれ c（$0.0 \leq c \leq 1.0$），m（$0.0 \leq m \leq 1.0$）としてあらかじめ指定される．

ステップ 4.3. 各染色体について適合度を求め，この適合度に基づいて次世代に残す染色体を選択する．適合度は各染色体がどれだけ優れているかを表す値であり，染色体 ts の適合度は以下の適合度関数 $ftn(ts)$ によって求められる．

$$ftn(ts) = f_1(ts) \cdot f_2(ts)$$

$$f_1(ts) = w_u \cdot udc(ts) + w_n \cdot \frac{1}{h+1} \sum_{i=0}^{h} nsc(ts, i)$$

$$f_2(ts) = 1 - pnl(ts)$$

ここで，$udc(ts)$ は ts の UDC を，$nsc(ts, i)$ は ts の N スイッチ網羅率（$N = i$）を意味する．h は，N スイッチ網羅率をどこまで加味するかを表す値で，あらかじめ指定される．たとえば $h = 5$ のときは 0 スイッチから 5 スイッチまでの網羅率を加味する．w_u と w_n は，それぞれ UDC と N スイッチ網羅にどの程度の重みを置くかというテスト戦術に基づいて決定される値で，$w_u + w_n = 1.0$，$0.0 \leq w_u \leq 1.0$，$0.0 \leq w_n \leq 1.0$ である．また，$pnl(ts)$ は，ts が (D) を満たさない場合に適合度から減じる値を求めるもので，これをペナルティと呼ぶ．(D) を満たさない場合とは，開発プロジェクトの状況に応じてあらかじめ指定される労力の上限 l を，ts の実行に要する労力 $effort(ts)$ が超える場合のことであり，$pnl(ts)$ は以下によって定義される．

$$pnl(ts) = \begin{cases} p, & effort(ts) > l \\ 0, & \text{otherwise} \end{cases}$$

p（$0.0 < p \leq 1.0$）はペナルティの大きさを決定する値であり，あらかじめ指定される．本アルゴリズムではエリート保存法とルーレット選択法を採用する．すなわち，適合度において上位 e 個の染色体を必ず選択し，残りの中から $(s - e)$ 個の染色体を適合度に比例する確率でランダムに選択する．

ステップ 4.4. 世代交代回数があらかじめ指定された回数 g に達するか，または最良の染色体が変化しない期間があらかじめ指定された世代交代回数 t に達する場合，その時点における最良の染色体を最終的な解候補として出力し終了する．そうでなければステップ 4.2 に戻る．

2.3 節のステップ 3 で決定するパラメータについて整理すると，集団サイズ s，交叉率 c，突然変異率 m，N スイッチの上限 h，UDC と N スイッチ網羅の重み w_u，w_n，ペナルティ p，労力の上限 l，保存エリート数 e，世代交代回数 g または t となる．これらに設定する値によって本アルゴリズムは様々なテストケース集合を生成することが可能であり，テスト技術者はテストケース生成を試行する中で開発プロジェクトの状況にふさわしいテストケース集合を最終的に採用することができる．

4. ツール構成とその適用例

2 章と 3 章で述べた手法を実現するツール構成について検討し，構築した．そして有効性を評価するために商用ソフトウェアを対象とした適用実験を行った．本章では，本研究が提案するツール構成と適用実験の結果について示す．

4.1 ツール構成

本研究では**図 3** に示すツール構成を提案する．概要は以下のとおりである．テスト技術者の手作業が必要なのは (1)，(2)，(5) であり，それ以外は自動的に行われる．

(1) テスト技術者が，テスト対象ソフトウェアの振舞いに関する要求仕様やユーザの利用特性，テストの労力に関する情報（図 3 (a)）を基に，MDD（model-driven development）をサポートしたモデリングツールを使用して拡張運用プロファイルを記述する．

(2) テスト技術者が，拡張運用プロファイルの実行可能 EFSM とデータファイルを生成するためにカスタマイズしたソースコード自動生成法（図 3 (b)）をモデリングツールに設定する．

(3) モデリングツールが，C++言語で記述された実行可能 EFSM を生成する（図 3 (c)）．実行可能 EFSM は EFSM の動作をシミュレーションするためのプログラムであり，テストデータを入力するとテストオラクルを出力する機能を持つ．

(4) モデリングツールが，拡張運用プロファイルのデータファイル（図 3 (d)）を生成する．このデータファイルは，3 章のアルゴリズムを実装したテストケース生成

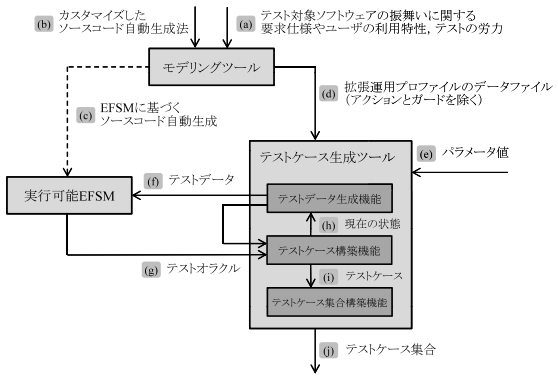


図 3 本手法のツール構成
Fig. 3 Tool configuration in this method.

ツールに入力するためのもので，アクションとガードを除く拡張運用プロファイルに関する基本的な情報が含まれる．

(5) テスト技術者が，生成するテストケース集合の性質や量を指定するためのパラメータ値（図 3 (e)）を決定する．実行可能 EFSM，データファイル，パラメータ値の準備が完了すると，テストケース生成ツールはテストケースの自動生成を開始できる状態になる．

(6) テストケースの自動生成が開始されると，テストケース生成ツールは，拡張運用プロファイルに基づいて生成したテストデータ（図 3 (f)）を実行可能 EFSM に入力する．

(7) 実行可能 EFSM は，入力されたテストデータを用いてアクションの実行やガードの評価，状態の遷移を実行し，その結果をテストオラクル（図 3 (g)）としてテストケース生成ツールに入力する．

(8) テストケース生成ツールは，実行可能 EFSM からのテストオラクルに基づいて現在の状態（図 3 (h)）を更新し，テストデータおよびテストオラクルからテストケースを構築する．

(9) テストケース生成ツールは，構築されたテストケース（図 3 (i)）を用いてテストケース集合を構成する．

(10) テストケース生成ツールは，最終的な解候補であるテストケース集合（図 3 (j)）をファイルに出力する．

このツール構成の特徴は，近年普及しつつある MDD をサポートしたモデリングツールを使用する点である．MDD はモデルに基づいてソフトウェア開発を行う方法論の総称であり，これをサポートしたモデリングツールは，UML をはじめとした図式表記法による要求仕様のモデル記述機能や，モデルからのソースコード自動生成機能，表記法や自動生成法のカスタマイズ機能などを備えている．モデリングツールを使用するこのツール構成には，テスト技術者にとって 2 つのメリットがある．1 つは，テストケース生成ツールの開発や利用に要する手数が少なくてすむという点である．本手法に基づくテストケース生成ツールの開発にあたり，拡張運用プロファイルの記述機能や EFSM の実行環境を組み込む必要がなく，3 章のアルゴリズムを実装するだけでよい．既存のモデリングツールの機能をできるだけ利用する方が開発コストが少ないし，使い慣れている分利便性も高い．もう 1 つは，アクションを作り込んだりソースコード自動生成法を拡張したりすることによって，テストオラクルを詳細化できるという点である．モデリングツールのソースコード自動生成機能は，単に EFSM を実行するだけのプログラムを生成するというよりは，ソフトウェアそのもの，あるいはそのプロトタイプを生成することを本来の目的としているので，実際のテスト対象ソフトウェアにより近い複雑な動作ロジックを実行可能 EFSM に組み込むことも可能である．このツール構成は，モデルやプロトタイプを用いてテスト対象ソフトウェアの期待される振舞いをシミュレーションすることによってテストオラクルを生成する手法であるシミュレーションオラクル [11] を効果的に実現する方式といえる．

4.2 適用実験

本手法の有効性を確認するために 4.1 節で述べたツール構成を用いて適用実験を行った．本適用実験におけるテスト対象ソフトウェアは，ある商用のテスト支援ツール中の機能の一部であり，そのツール全体の規模はおよそ 1,000 KLOC，テスト対象とする機能の規模はおよそ 45 KLOC である．ソフトウェア開発会社でのテスト工程において 3 件のフォールトが発見されたことが分かっている．本適用実験の目的は，以下の 4 つの観点で評価を行うことである．

(i) 拡張運用プロファイルを作成できるか．

(ii) UDC と N スイッチ網羅率が改善されるか．

(iii) フォールトの発見可能性があるか．

(iv) 本手法の適用に要する労力は現実的か．

本適用実験の方法は次のとおりである．まず，ソフトウェア開発会社の技術者がテスト対象ソフトウェアの拡張運用プロファイルを記述するとともに，モデリングツールのソースコード自動生成法のカスタマイズを行った．これは 4.1 節の (1)，(2) に対応する．次に，この拡張運用プロファイルから実行可能 EFSM とデータファイルを生成した．これは 4.1 節の (3)，(4) に対応する．そしてテストケース生成ツールに対して $s = 5$，$c = 0.2$，$m = 0.1$，$h = 4$，$w_u = 0.5$，$w_n = 0.5$，$p = 0.5$，$e = 2$，$g = 1{,}000$，l に 1 人日程度を想定した値を設定した．これは 4.1 節の (5) に対応する．そのうえで，テストケース生成ツールによってテストケース集合を生成した．これは 4.1 節の (6)〜(10) に対応する．なお，テストケース生成は 3 章で示したとおりランダムな方法で行われるため，10 回試行した結果をもって評価することとした．最後に，10 回の試行によって得られた 10 個のテストケース集合のそれぞれについて，3 件のフォールトを発見できる可能性があるか否かを確認した．ここでは，あらかじめ分かっているフォールトの顕在化条件を EFSM 上で満たした場合に「テスト対象ソフトウェア上で当該フォールトを発見できる可能性がある」と判定する方法で行った．

本適用実験の結果は以下のとおりである．まず (i) については，問題なく拡張運用プロファイルを作成することができた．このテスト対象ソフトウェアの振舞いを FSM によって表す場合ではその状態数が数百に及ぶと考えられる．そのような大規模な FSM および運用プロファイルをテスト工程の限られた時間で正確に作成することは困難である．これに対して本適用実験で作成した EFSM の規模は状態数 10，遷移数 19 であった．ゆえに，FSM ではなく

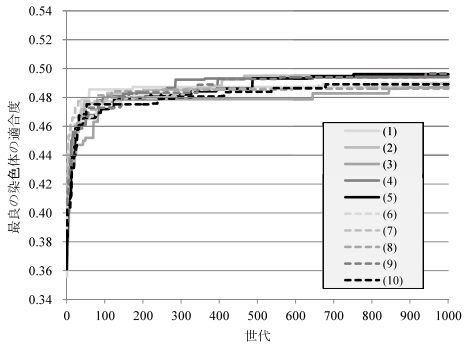


図 4 各試行における適合度の成長
Fig. 4 Growth of fitness in each experiment.

表 1 各メトリクスの改善の程度
Table 1 Improvement of each metric.

メトリクス	初期集団の平均	最終的な解候補	改善の幅
UDC	0.301	0.565	0.264
0 スイッチ	0.855	1.000	0.145
1 スイッチ	0.440	0.636	0.196
2 スイッチ	0.207	0.318	0.111
3 スイッチ	0.073	0.101	0.028
4 スイッチ	0.024	0.029	0.005

EFSM を用いることで運用プロファイルを作成することができたといえる．次に (ii) について，テストケース集合生成の 10 回の各試行の過程において適合度が成長する様子を**図 4** の (1)〜(10) として示す．いずれも似た曲線を描いており，ばらつきも少なく安定した結果が得られているといえる．さらに，初期集団（第 0 世代）および最終的な解候補のテストケース集合における各メトリクスの平均値を**表 1** に示す．ここで初期集団を構成するテストケース集合は，UDC や N スイッチ網羅率の改善を行わない従来の方法（文献 [4], [5] など）で生成したものと見なすことができる．すべてのメトリクスにおいて改善がみられていることから，本手法によって UDC と N スイッチ網羅率を改善することができたといえる．l による制限があるにもかかわらず，UDC と N スイッチ網羅率を同時に改善できる点は特筆すべきである．なお，各テストケース集合のテストケース数の平均は 13 であった．次に (iii) に関しては，各テストケース集合の内容を確認した結果，10 個すべてのテストケース集合について先述の 3 件のフォールトを発見できる可能性があることが分かった．本手法は従来のテストで発見されるフォールトを発見可能であり，従来のテストの一部を体系化するものといえる．なお，3 件のフォールトは初期集団によっても発見できる可能性があることが分かった．フォールトやソフトウェアに応じた発見可能性について今後も調査，検討する必要がある．(iv) については合計 13 人日を要した．その内訳は，拡張運用プロファイルの記述に 3 人日，ソースコード自動生成法のカスタマイズに 10 人日であった．後者が全体の労力の大部分を占めるが，これは次回以降の適用時には不要となる．したがって，本手法の適用に要する労力は現実的であるといえる．

5. 考察

適用実験では評価観点 (i)〜(iv) について期待した結果を得ることができたので，本手法の有効性を確認することができたといえる．ただし，(iii) については，フォールトが発見できる可能性を示すことはできたが，必ず発見できると断定するには至らなかった．これは，フォールトの顕在化条件の一部が EFSM では記述できない部分によっていたためである．たとえば，今回のテスト対象ソフトウェアは複雑な構造のファイルを入力として受け付けるが，このファイルを生成するための形式的定義を EFSM に組み込むことができなかったことが原因である．現実の複雑なソフトウェアのフォールトをより効果的に発見するには EFSM だけでは必ずしも十分ではないため，MBT における共通の課題として今後取り組む必要がある．

他のソフトウェアをテスト対象としたときに，今回の適用実験と同様の効果が得られるか否かは，次に述べる 2 つのポイントに依存すると考えられる．1 つ目のポイントは，拡張運用プロファイルを作成するための技術者の技量である．本手法をはじめとした MBT はテストケースを自動生成できることが最大のメリットであるが，正しいモデルを限られた時間で作成できる人材がいることが前提である．一般的にソフトウェアのモデルを適切に作成できるようになるには一定の経験が必要である．2 つ目のポイントは，テスト工程にどの程度余裕があるかである．労力の上限 l に大きい値を指定することができる余裕のあるテスト工程では，多くのテストケースからなるテストケース集合を生成することができる．テストケース数が多いと UDC や N スイッチ網羅率は最初から大きい値となるため，本手法による改善の幅は小さくなる．逆に逼迫したテスト工程では改善の幅は大きくなるので，本手法は逼迫したテスト工程において特に効果が期待できると考えられる．

3 章で示したテストケース生成アルゴリズムでは，遺伝子を生成する段階で実際に拡張運用プロファイルを動作させ，最終的なテストケース集合の実行可能性を保証している．これは我々の従来手法のアルゴリズム [6] との大きな違いの 1 つであり，従来手法と比較してテストケース集合生成に時間を要する原因となっている．今回の適用実験においてテストケース集合生成の 1 回の試行に要した時間はおよそ 15 分であり，実用上問題にはならない．しかしながら生成時間は短い方が望ましいので，最終的なテストケース集合のみに対して実行可能性の確認を行い，実行不可能なテストケースを後で除外することで効率化を図るという方法も考えられる．しかしながら，この方法では効果的に UDC や N スイッチ網羅率を改善できない．たとえば，あ

るテストケース t が実行不可能であるとして後から除外されると，t によって実行されるはずであった使用方法や遷移列が実行されなくなる．GA によって最適化されているので，他のテストケースは UDC や N スイッチ網羅の観点で t と重複しないように生成されている．したがって，他のテストケースによる補完は期待できないし，t と同等かつ実行可能な別のテストケースが生成可能であるとは限らない．拡張運用プロファイルに含まれるアクションが複雑になればなるほど，実行不可能なテストケースが生成されやすくなるので，先述の方法は困難となる．

6. 関連研究

本章では関連研究について述べる．

OPBT はソフトウェア信頼性に着目した MBT として Musa [4] によって提案され，MTTF（mean time to failure）や Kullback 判別式 [12], [13] がメトリクスとして用いられてきた．MTTF を用いる場合では，運用プロファイルに基づくテストケースを目標の MTTF に達するまで生成し実行し続ける．また，Kullback 判別式を用いる場合では，運用プロファイルとテストモデル（テストケースの実行結果に基づいて作成した特殊な運用プロファイル）の間の差異が一定以下になるまでテストケースを生成し実行し続ける．このように MTTF や Kullback 判別式は大量のテストケースの使用が前提となっており，本研究が想定している，テスト工程に割り当てることが可能な労力が限定される状況においては適していない．これに対して，本研究に導入した UDC は大量のテストケースの使用を前提としていない．MTTF や Kullback 判別式との最大の違いは，同じテストケースを重複して実行しても評価しない点である．

OPBT は研究や開発の現場に適用され成果をあげた事例が報告されている．たとえば，Popovic ら [14] は通信プロトコルのテストに OPBT を導入した．運用プロファイルにおいてイベントを形式的に定義しておき，最終的に JUnit によるテストドライバを生成する仕組みを構築している．テストドライバの生成が可能なのは，通信プロトコルのテストに必要なルーチンやインタフェースがあらかじめ分かっているからである．適用対象を限定すれば，我々の研究においても自動化の範囲を拡大できると考えられる．また，Chruscielski ら [8] は航空機関連システムに対して，Hartmann ら [9] は医療関連機器に対して OPBT を適用した．いずれも，ソフトウェア信頼性だけでなく設計の改善にも有効であったと報告している．運用プロファイルの作成には手数を要するものの，その作成作業自体が設計のレビューとして機能すると考えられる．OPBT のツールとしては MaTeLo [15] や JUMBLE [16] が知られており，運用プロファイルの定義やテストケース生成，実行のための機能を提供している．本章でここまでに紹介した従来の OPBT の研究は，1 章で述べた課題，すなわち，テスト対象ソフトウェアの複雑な振舞いを扱うことや使用頻度の低い機能をテストすることが困難であるという課題を十分考慮しておらず，またメタヒューリスティクスによるテストケースの改善も行っていない．

OPBT 以外の MBT においては，メタヒューリスティクスを応用した手法が提案されている．Kalaji ら [2] は，テスト対象ソフトウェアの振舞いを表す EFSM から，実行可能かつ 0 スイッチ網羅を満たすテストケース集合を GA によって生成する手法を提案した．遷移間のデータ依存関係が少ないテストケースを選りすぐって生成するため，イベントパラメータの値を容易に発見することができる．ただし，特に複雑なアクションを持つ EFSM では，発火させることが本質的に困難な遷移が存在する場合がある [1]．我々の手法も含め，より大規模で複雑なソフトウェアに効果的に適用できるようにするためには，発火が困難な遷移に対して有効なアルゴリズムを検討する必要があると考えられる．

7. おわりに

本稿では，UDC と N スイッチ網羅率が大きいテストケース集合を拡張運用プロファイルから効果的に生成する手法を提案した．EFSM に基づいて作成される拡張運用プロファイルは従来の運用プロファイルよりも表現力が高いため OPBT の適用範囲を広げることができるが，遷移間のデータ依存性に起因する実行可能性の問題に対処する必要がある．さらに，現実のテスト工程を考慮すると，テストケース集合は限られた労力の範囲で実施できるものでなければならない．これらを解決するために本稿では GA を応用したテストケース生成アルゴリズムを示した．また，本手法を実現するために，当該アルゴリズムを実装したテストケース生成ツールと既存のモデリングツールを連携させたツール構成を提案し構築した．商用ソフトウェアに適用した結果，EFSM の使用によって複雑な振舞いを持つソフトウェアに対して OPBT が効果的に適用できること，UDC と N スイッチ網羅率を同時に改善可能であることなどが確認できた．したがって，本手法は OPBT の有効性をより高めるものであると結論できる．

テストの現場では文書化されていない有用なテスト戦術が数多く存在するが，我々は本手法を拡張することによってそれらの一部を体系化できると考えている．今後の研究では本手法を拡張し，様々なソフトウェアに適用することによってその有効性を調査する予定である．

謝辞 本研究は JSPS 科研費 23700038 の助成を受けた．また，ガイオ・テクノロジー株式会社市川忠彦，大西建児，大城戸薫の諸氏には本研究に関して有益なご意見をいただいた．ここに深く感謝の意を表する．

参考文献

[1] Doungsaard, C., Dahal, K., Hossain, A. and Suwannasart, T.: Test Data Generation from UML State Machine Diagrams using GAs, *Proc. International Conference on Software Engineering Advances*, p.47 (2007).

[2] Kalaji, A., Hierons, R.M. and Swift, S.: Generating Feasible Transition Paths for Testing from an Extended Finite State Machine (EFSM), *Proc. International Conference on Software Testing Verification and Validation*, pp.230–239 (2009).

[3] Rook, P.: *Software Reliability Handbook*, Elsevier Science (1990).

[4] Musa, J.D.: The Operational Profile, *Reliability and Maintenance of Complex Systems*, NATO ASI Series F: Computer and Systems Sciences, Vol.154, pp.333–344 (1996).

[5] Walton, G.H., Poore, J.H. and Trammell, C.J.: Statistical Testing of Software Based on a Usage Model, *Software Practice and Experience*, Vol.25, No.1, pp.97–108 (1995).

[6] 高木智彦，橋本慎一朗，八重樫理人，古川善吾：拡張運用プロファイルに基づく最適化されたテストスイートの生成手法，情報処理学会論文誌，Vol.53, No.2, pp.557–565 (2012).

[7] Takagi, T., Nishimachi, K., Muragishi, M., Mitsuhashi, T. and Furukawa, Z.: Usage Distribution Coverage: What Percentage of Expected Use Has Been Executed in Software Testing?, *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, Studies in Computational Intelligence, Vol.209, pp.57–67, Springer (2009).

[8] Chruscielski, K. and Tian, J.: An Operational Profile for the Cartridge Support Software, *Proc. 8th International Symposium on Software Reliability Engineering*, pp.203–212 (1997).

[9] Hartmann, H., Bokkerink, J. and Ronteltap, V.: How to reduce your test process with 30% – The application of Operational Profiles at Philips Medical Systems, *Supplementary Proc. 17th International Symposium on Software Reliability Engineering*, CD-ROM (2006).

[10] Chow, T.S.: Testing Software Design Modeled by Finite-State Machines, *IEEE Trans. Softw. Eng.*, Vol.SE-4, No.3, pp.178–187 (1978).

[11] Binder, R.V.: *Testing Object-Oriented Systems: Models, Patterns, and Tools*, Addison-Wesley (1999).

[12] Whittaker, J.A. and Thomason, M.G.: A Markov Chain Model for Statistical Software Testing, *IEEE Trans. Softw. Eng.*, Vol.20, No.10, pp.812–824 (1994).

[13] Sayre, K. and Poore, J.H.: Stopping criteria for statistical testing, *Information and Software Technology*, Vol.42, No.12, pp.851–857 (2000).

[14] Popovic, M., Basicevic, I., Velikic, I. and Tatic, J.: A Model-Based Statistical Usage Testing of Communication Protocols, *Proc. 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems*, pp.377–386 (2006).

[15] Guen, H.L. and Thelin, T.: Practical Experiences with Statistical Usage Testing, *Proc. 11th Annual International Workshop on Software Technology and Engineering Practice*, pp.87–93 (2003).

[16] Prowell, S.J.: Using Markov Chain Usage Models to Test Complex Systems, *Proc. 38th Hawaii International Conference on System Sciences*, USA, p.318c (2005).

高木 智彦 （正会員）

2002 年香川大学工学部卒業．2004 年同大学大学院工学研究科修士課程修了．2007 年同大学院博士後期課程修了．2008 年香川大学工学部助教，現在に至る．博士（工学）．ソフトウェア工学，特にソフトウェアテスト法の研究に従事．電子情報通信学会，ソフトウェア科学会各会員．

八重樫 理人 （正会員）

2005 年芝浦工業大学大学院博士（後期）課程修了．同年豊田工業大学総合情報センターポストドクトラル研究員．2006 年芝浦工業大学 JAD プログラム講師．2009 年香川大学総合情報センター助教，2010 年香川大学工学部講師，現在に至る．博士（工学）．ソフトウェア開発を支援するツール，グループでの活動を支援するツールおよび教育支援システム等の研究に従事．電子情報通信学会，教育工学会，教育システム情報学会各会員．

古川 善吾 （正会員）

1975 年九州大学工学部卒業．1977 年同大学大学院工学研究科修士課程修了．同年日立製作所システム開発研究所勤務．1986 年九州大学工学部情報工学科助手，1990 年九州大学工学部講師，1992 年九州大学情報処理教育センター助教授，1998 年香川大学工学部教授，現在に至る．博士（工学）．ソフトウェア工学，特にソフトウェアテスト法，分散システム/インターネットの運用管理等の研究に従事．電子情報通信学会，ソフトウェア科学会，ACM，IEEE-CS，ISOC 各会員．