情報処理学会論文誌 Vol.53 No.2 557–565 (Feb. 2012)

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

高木 智彦[1,a)] 橋本 慎一朗[2] 八重樫 理人[1] 古川 善吾[1]

受付日 2011年6月14日, 採録日 2011年11月7日

概要：テスト戦術や投入可能なテスト労力などに基づいて最適化されたテストスイートを生成する新たな運用プロファイルベースドテスト法を提案する．本手法では，テスト労力に関する情報が付加された運用プロファイル（拡張運用プロファイル）を用いる．テスト労力をユーザの使用頻度に比例して配分する戦術と，逆に反比例して配分する戦術を選択することができる．遺伝的アルゴリズムによって，投入可能なテスト労力の上限を超えない範囲で，選択した戦術を徹底して反映させたテストスイートを生成する．本研究では，本手法を実装したツールを開発し，商用ソフトウェアに試験的に適用した．その結果，本手法によって運用プロファイルベースドテスト法におけるテストスイートの質を改善できることが分かった．本手法は，網羅性を指向する多くのテスト法とは異なるものであり，従来のテストを補完することが期待できる．また，近年の逼迫したテスト工程においても適用が可能である．

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，遺伝的アルゴリズム

The Optimized Test Suite Generation Method Based on Extended Operational Profiles

Tomohiko Takagi[1,a)] Shinichiro Hashimoto[2] Rihito Yaegashi[1]
Zengo Furukawa[1]

Received: June 14, 2011, Accepted: November 7, 2011

Abstract: This paper shows a new operational profile-based testing technique to generate optimized test suites based on a test strategy and available test labor. This technique employs an extended operational profile, which contains additional information about the test labor. It provides two kinds of test strategies; one is to distribute the available test labor to software under test (SUT) in proportion to the frequencies of use, and another is to distribute in inverse proportion to them. Test suites are generated by a genetic algorithm so as to exhaustively reflect the selected test strategy within the specified available test labor. In this study, we developed a tool in which this technique was implemented, and then applied it to commercial software on a trial basis. As a result, it was found that this technique improves the quality of test suites on operational profile-based testing. This technique can supplement conventional testing and can be introduced into recent tight test processes.

Keywords: software testing, operational profile, test case, genetic algorithm

1. はじめに

ソフトウェアテスト法は，ソフトウェアに要求される品質を実現するための重要な技術の1つである．ソフトウェア開発のテスト工程では，テスト法を駆使してフォールトを発見し解決する．これによって，出荷後にフォールトが

[1] 香川大学工学部
Faculty of Engineering, Kagawa University, Takamatsu, Kagawa 761–0396, Japan
[2] 香川大学大学院工学研究科
Graduate School of Engineering, Kagawa University, Takamatsu, Kagawa 761–0396, Japan
[a)] takagi@eng.kagawa-u.ac.jp

顕在化してユーザに不利益を与えるリスクを少なくすることができる．しかしながら，近年，ソフトウェアの大規模化，複雑化，短納期化などにより，出荷後に深刻なフォールトが顕在化する事例が後をたたない．従来のテスト工程では，主として，ソースコードや仕様書を網羅することを指向したテスト法が用いられているが，より高い品質を実現するためには，他の方向性でフォールトの発見を試みる新たなテスト法も必要である．

網羅性を指向するのとは異なる方向性のものとして，運用プロファイルベースドテスト（operational profile-based testing）がある [1], [2]．運用プロファイルは，テスト対象ソフトウェアの仕様を表す有限状態機械にユーザの利用確率をマッピングしたモデルである．運用プロファイルベースドテストでは，この運用プロファイルから確率的に生成した大量のテストケースを用いてテストを実施する．ここでいうテストケースとは，有限状態機械上の開始状態で始まり終了状態で終わる遷移列のことであり，たとえば，テスト対象ソフトウェアがインターネット通販システムだとすると，システムにログイン後，品物の選択や電子決済を行ってログオフするまでの入力や確認すべき項目などに関する情報の列が相当する．このようなテストケースの集合のことを本論文ではテストスイートと呼ぶ．運用プロファイルベースドテストによってソフトウェア信頼性 [3] に深刻な影響を与えるフォールトを発見できることが確認されており [4], [5]，従来のテストを補完する有効な手法の 1 つとして注目されている．「残存しているフォールトがたとえ 1 件だけであっても，それが使用頻度の高い機能に関係していれば高い頻度で顕在化する可能性があるのでリスクが高い」という考えに基づき，使用頻度に比例するようにテスト労力を配分するテスト戦術といえる．

ただし，運用プロファイルベースドテストには問題点が 2 つある．1 つは，近年のテスト工程では投入できるテスト労力が特に限られているため，大量のテストケースを使用することは現実的に困難であるという点である．もともと大数の法則に基づいたテスト法であるため，生成するテストケース数を少なくすればするほど，テストスイートがユーザの利用特性を反映しにくくなり，期待された効果が得にくくなる．たとえば，使用頻度の高い機能のためのテストケースが十分生成されず，高い頻度で顕在化するフォールトを見逃す可能性が高くなる．もう 1 つは，使用頻度に比例するのではなく，反比例するようにテスト労力を配分するテスト戦術をとるべき，との考えも成り立つ点である．つまり，使用頻度の低い機能は，開発者の考えが十分及んでいないことによるフォールトが残存する可能性が高いため，リスクが高いという点を考慮していない [6]．本論文では便宜上，これらの相反するテスト戦術についてそれぞれ以下のように称することとする．

戦術 a：残存しているフォールトがたとえ 1 件だけであっても，それが使用頻度の高い機能に関係していれば高い頻度で顕在化する可能性があるのでリスクが高い．したがって，使用頻度に比例するようにテスト労力を配分する．

戦術 b：使用頻度の低い機能は，開発者の考えが十分及んでいないことによるフォールトが残存する可能性が高いため，リスクが高い．したがって，使用頻度に反比例するようにテスト労力を配分する．

なお，戦術 a，b のどちらが正しいかについて，一般論として結論付けることはできない．各戦術の有効性は，テスト対象ソフトウェアにおけるフォールトの分布とユーザの利用方法の分布の間の関係によって決まる．したがって，テスト対象ソフトウェアの性質やプロジェクトの状況などに応じてテスト技術者が適切な戦術を判断する必要がある．

上記の問題点を解決できれば，運用プロファイルベースドテストの有効性を向上させ，ひいてはソフトウェアの高信頼化に寄与できると考えられる．そこで本研究では，限られたテスト労力の中で，戦術 a または b を徹底して反映させたテストスイートを生成するための，新たな運用プロファイルベースドテスト法を提案する．本手法の核心は以下の 2 点である．

- 戦術 a だけでなく戦術 b についても考慮する．テストスイートがどの程度戦術 a または b に基づいているかを評価するメトリクスとして，UDC（usage distribution coverage）[7] を導入する．
- テスト労力に関する情報を持つ，拡張された運用プロファイルを作成する．投入可能な労力をテスト技術者が指定すると，その労力の範囲内で戦術 a または b を最大限に具現化するように最適化されたテストスイートを生成する．

本研究では UDC を導入した遺伝的アルゴリズム [8] によって最適化を行う．本研究における最適化されたテストスイートとは，指定された労力の範囲内で UDC をできるだけ大きくしたテストスイートのことである[*1]．従来研究において，各戦術そのものの有効性についてはすでに議論されており [4], [5], [6]，またテストスイートがどれだけ各戦術を反映しているか（どれだけ達成しているか）を UDC によって評価できることも示されている [7]．したがって，本手法が従来の運用プロファイルベースドテスト法よりも優れていることを，UDC に基づいて明らかにすることが，本論文における最終的な目標である．本論文の構成は次のとおりである．まず 2 章で最適化されたテストスイートを生成する手法を示す．次に 3 章で，本手法を商用ソフトウェアに適用した結果について述べる．そして 4 章で関連研究について示し，最後に 5 章で本研究を総括する．

[*1] 一般的には，「発見できるフォールト数の期待値を最大化すること」をテストスイートの最適化と考える場合が多い．本研究の最適化はそれとは異なることに注意されたい．

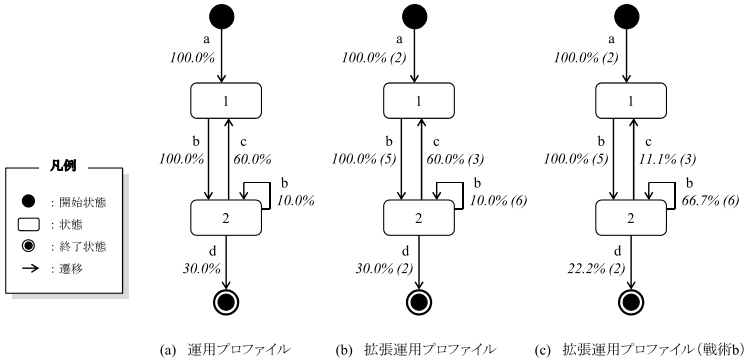

図 1 単純な拡張運用プロファイルの例
Fig. 1 Example of extended operational profiles.

2. 最適化されたテストスイートの生成手法

本章では，まず本手法の全体像を述べた後，テストスイートを最適化するための遺伝的表現および遺伝的操作の詳細について示す．

2.1 本手法の全体像

本手法は，主にシステムテストや受け入れテストにおいて，従来から行われているテスト（ソースコードや仕様書の網羅を指向したテストなど）を補完するためのものである．ソースコードや仕様書を網羅することには適さないので，本手法単体で十分なソフトウェアの品質を実現することは困難である．しかしながら，戦術 a，b に基づいて従来とは異なる側面からテストを行い，ソフトウェア信頼性に影響を与えうるフォールトを検出することが期待できる．フォールトが検出された場合はそれを除去することでソフトウェア信頼性を改善できるし，検出されなかった場合はソフトウェア信頼性について確信を深めることができる．本手法は，投入可能な労力に応じて最良の（すなわち，戦術 a または b を最大限に具現化するために UDC ができるだけ大きくなるような）テストスイートを生成するので，逼迫したテスト工程にも適用可能である．テストスイートを生成するモデルとして有限状態機械を用いるため，たとえば，他のシステムとの相互作用やユーザとの対話を行ったり，外部環境の変化に応じて振舞いを変えたりするような性質のソフトウェアをテストするのに適している．

本手法は以下の 5 つのステップから構成される．

ステップ 1. テスト技術者が運用プロファイルを作成する．運用プロファイルの構造，すなわち有限状態機械は，テスト対象ソフトウェアの仕様に基づいて作成する．有限状態機械の基本的な構成要素は，開始状態，状態，終了状態，遷移の 4 つである．遷移は，遷移元状態，イベント（テストデータの作成方法またはテストデータそのもの），遷移先状態，事前/事後条件（遷移の直前/直後においてテスト対象ソフトウェアが満たすべき条件）からなる．また，ユーザの利用特性を表す確率分布については，類似するソフトウェアの運用データや技術者の予測などに基づいて導出する．このステップは，従来の運用プロファイルベースドテストにおけるものと同じである．簡略表記された単純な運用プロファイルの例を**図 1** (a) に示す．状態には数字，イベントにはアルファベットの識別子を割り当て，イベントの識別子の直後に遷移確率を記述している．たとえば，状態 2 においてイベント b，c，d が生起する確率はそれぞれ 10%，60%，30%であることが示されている．

ステップ 2. 運用プロファイルのすべての遷移に対して，テスト実施に要する労力をテスト技術者が記述する．労力の尺度は，運用プロファイル全体で一貫してさえいれば何を使用してもよい．たとえば，テストデータの入力やテスト結果の判断などに必要な人時，あるいは，ある特定の遷移を基準にした相対的な労力の程度などが考えられる．何らかの理由で具体的な労力の値を決定できない場合は，すべての遷移を 1 としておく．以上で拡張運用プロファイルが完成する．図 1 (a) から作成された拡張運用プロファイルの例を図 1 (b) に示す．労力の値を遷移確率の直後の括弧内に記述している．

ステップ 3. テスト技術者が，テストスイートを最適化するための制約条件および各種パラメータを決定する．制約条件については，開発スケジュールに基づいて投入可能なテスト労力の上限を明らかにするとともに，戦術 a，b のどちらに基づいてテストを実施するべきかを状況に応じて判断する．さらに，テスト対象から除外するべき遷移や遷移列がある場合は非テスト対象遷移リストに登録し，テスト対象に含めるべき遷移や遷移列がある場合はテスト対象遷移リストに登録する．前者は，たとえば一部の機能の開発やデバッグがテスト工程に間に合わない事態において有用であり，後者は，回帰テストで特定の機能を再テストする必要が

ある場合に有用である．各種パラメータについては，2.3 節と 2.4 節で述べる．

ステップ 4. 遺伝的アルゴリズムを使用して拡張運用プロファイルから最適化されたテストスイートを生成する．テストスイートはテストケースの集合であり，テストケースは開始状態で始まり終了状態で終わる遷移列（イベントや事前/事後条件の列）である．

ステップ 5. テストスイートを用いてテスト対象ソフトウェアをテストする．

1 章で述べたとおり，本手法の目的は，限られた労力の範囲内で実行可能な，戦術 a または b に基づく最良のテストスイートを導出することである．この問題を解くために，本手法ではメタヒューリスティクスの 1 つである遺伝的アルゴリズムをステップ 4 で使用する．遺伝的アルゴリズムは，生物進化を模倣したシミュレーションによって最適化を行い，NP 困難あるいは定式化の困難な問題の最良解を求める手法である．本研究におけるテストスイート導出の問題を組合せ最適化問題としてとらえた場合，戦術や労力，非テスト対象遷移リスト，テスト対象遷移リストだけでなく，広義には遷移や遷移確率など，多くの複雑な制約条件が存在する．解の候補として検討するべき項目は，これらの制約条件によってすべての組合せの中の一部に限定されるため，遺伝的アルゴリズムで解くのに適している．なお，本問題は定式化によって解くことも可能であるが，計算時間の点で遺伝的アルゴリズムのほうが有効である．本手法ではどのような制約条件を与えるかによって様々なテストスイートを生成可能であり，テスト技術者は最終的にどのテストスイートを採用するかを判断する必要がある．それまで制約条件の変更をともないながらテストスイート生成が繰り返されることを考慮すると，計算時間への影響が少ない遺伝的アルゴリズムのほうが適している．また，厳しい制約条件が与えられた場合も考慮する必要がある．たとえば，逼迫したテスト工程では，非テスト対象遷移リストに多数の制約条件が設定されることが想定される．このように制約条件が厳しくても計算時間が短くすむという特長が遺伝的アルゴリズムにはあるので，やはり遺伝的アルゴリズムが適している．運用プロファイルベースドテストは非決定的に多様なテストスイートを導出できる点が特長であるので，同じく非決定的な手法である遺伝的アルゴリズムとの親和性は高い．

ステップ 4 は，さらに以下のステップに細分される．

ステップ 4.1. 染色体（すなわちテストスイート）の初期集団を生成する．染色体については 2.2 節で述べる．

ステップ 4.2. 交叉と突然変異により新たな染色体を生成する．交叉と突然変異については 2.3 節で述べる．

ステップ 4.3. 染色体の適合度を算出する．そして適合度に基づき次世代に残す染色体を選択する．選択および適合度については 2.4 節で述べる（労力や UDC の計算方法についても 2.4 節で述べる）．

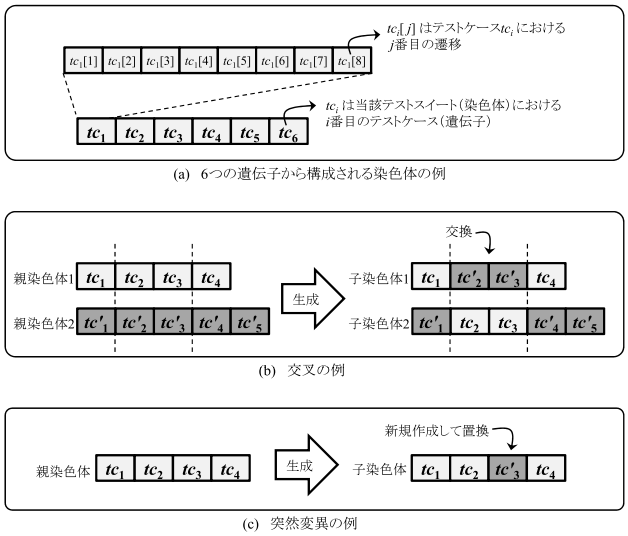

図 2 遺伝的表現と遺伝的操作の概要
Fig. 2 Overview of genetic representation and operations.

ステップ 4.4. 一定の世代交代回数に達していなければステップ 4.2 に戻る．

ステップ 4.5. 適合度の最も高い染色体を最良解（すなわち最適化されたテストスイート）として出力する．

以降では，染色体，交叉，突然変異，選択，適合度の詳細について述べる．

2.2 染色体

染色体は，解くべき問題に対する解の候補を，遺伝的アルゴリズムで扱える形式（遺伝的表現）で表現したものである．本手法では，**図 2** (a) に示すように，染色体はテストスイートに対応し，染色体を構成する遺伝子はテストケースに対応する．テストケースは，運用プロファイルにおける開始状態で始まり終了状態で終わる遷移列である．このように本手法の遺伝的表現は直接的であるため，特別なエンコーディングやデコーディングの処理は必要としない．

テストケースは運用プロファイルの確率分布に基づいてランダムに生成される．もしステップ 3 において戦術 a が選択されていれば，遷移先は遷移確率に比例して選択される．一方，戦術 b が選択されていれば，遷移先は遷移確率に反比例して選択される．これは，たとえば，図 1 (b) の確率分布を図 1 (c) の確率分布に変換したうえで，遷移先を遷移確率に比例して選択することによって実現できる．テストスイートは，そのようにして生成されたテストケースの集合である．ステップ 4.1 では，初期のテストスイートが，ステップ 3 で指定される労力の上限を超えない範囲で，できるだけ多くのテストケースを含むように生成される．その後，初期のテストスイートに対して交叉，突然変異などの遺伝的操作が加えられ，新たなテストスイートが生成される．

2.3 交叉と突然変異

交叉と突然変異は，既存の染色体（親染色体）から新たな染色体（子染色体）を生成する操作である．

交叉は，親染色体のペアの間で遺伝子を一部交換して子染色体を生成する操作である．本手法の交叉を図 2 (b) に示す．まず，現在のテストスイートの集合（染色体集団）の中から交叉確率 c（$0.0 \leq c \leq 1.0$）で親をランダムに選択してペアを作成する．次に，それぞれのペアについてランダムに決定した 2 カ所のカットポジションでテストケース（遺伝子）の交換を行う．その結果，1 つのペアから 2 つの新たなテストスイートが生成される．これは一般に二点交叉と呼ばれる手法である．

突然変異は，親染色体の一部の遺伝子を変化させて子染色体を生成する操作である．本手法の突然変異を図 2 (c) に示す．まず，テストケース（親染色体の遺伝子）を突然変異率 m（$0.0 \leq m \leq 1.0$）でランダムに選択する．そして，2.2 節で述べた方法によってテストケースを新たに生成し，選択したテストケースと置き換える．ただし例外的に，当該テストスイートの労力が上限を超えない場合は新たに生成したテストケースの追加のみ行い，また，当該テストスイートの労力がすでに上限を超えている場合は選択したテストケースの削除のみ行うものとする．1 つのテストスイートにおいて 1 つ以上のテストケースの変化が起こった結果，新たなテストスイートが 1 つ生成される．

2.4 選択と適合度

選択は，次世代を構成する染色体を適合度に基づいて選ぶ操作である．適合度は，与えられた制約条件にどの程度個々の染色体が適応しているかを表した値である．これが大きい染色体は良い解であるので，選ばれる可能性は高くなる．本手法では，エリート保存法とルーレット選択法を組み合わせた方法を用いる．すなわち，上位 e 個のテストスイートは必ず選択して次世代に残す．そして，残りのテストスイートの中から，適合度の大きさに比例する確率で $(s-e)$ 個を重複なく選択して次世代に残す．s は各世代を構成する親染色体の個数であり，集団サイズという．

本手法では，テストスイート ts の適合度を以下の評価関数 $eval(ts)$ によって導出する．

$$eval(ts) = dist(ts) - penalty(ts),$$

ここで，ts は n 個のテストケースの集合，すなわち，$ts = \{tc_1, tc_2, \cdots, tc_n\}$ であるとすると，$dist(ts)$ は以下の式によって定義される．

$$dist(ts) = \sum_{i=1}^{n} \prod_{j=1}^{\sharp tc_i} prob(tc_i[j]),$$

ここで $\sharp tc$ はテストケース tc の遷移数を意味する．また，$prob(tc[j])$ は，tc の先頭から j 番目の遷移確率（ステップ 3 において戦術 a を選択した場合）または遷移確率を反比例させた確率（ステップ 3 において戦術 b を選択した場合）を意味する．$dist(ts)$ は，UDC（usage distribution coverage）と呼ばれる特殊なテスト網羅基準 [7] であり，ユーザの利用特性を ts がどの程度網羅しているかを表す．たとえば，戦術 a を選択した場合において，遷移確率の低い遷移からなるテストスイート（すなわちユーザの利用特性を十分反映しないテストスイート）が生成されたとすると，その UDC は低いので次世代に残る可能性は低くなる．逆に遷移確率の高い遷移からなるテストスイートは，UDC が高いので次世代に残る可能性は高くなる．したがって，テストスイートの質を明らかにするためのメトリクスとして UDC を評価関数に導入することによって，戦術 a および b を徹底させることができる．

また，$penalty(ts)$ は ts に対するペナルティを意味する．ペナルティとは，テストスイートがステップ 3 で指定された労力の上限 l_{max} を超える場合[*2]や，非テスト対象遷移リストに登録された遷移や遷移列 $nobj$ をテストスイートが含む場合，テスト対象遷移リストに登録された遷移や遷移列 obj をテストスイートが含まない場合において，当該テストスイートの適合度から減じる値のことである．ペナルティ $penalty(ts)$ は以下の式によって定義される．

$$penalty(ts) = \begin{cases} p \times dist(ts), \\ \quad labor(ts) > l_{max} \vee ts \ni nobj \vee ts \not\ni obj, \\ 0, \qquad\qquad \text{otherwise} \end{cases}$$

ここで，p はペナルティの大きさを決定する値（$0.0 < p \leq 1.0$）を表している．$\ni$ は左辺のテストスイートが右辺の遷移などを含むことを意味する演算子であり，$\not\ni$ は左辺のテストスイートが右辺の遷移などを含まないことを意味する演算子である．ペナルティの適用条件を満たさない場合には，評価関数 $eval(ts)$ の値は $dist(ts)$ によって得られる値と等しくなる．また，労力 $labor(ts)$ は下式によって定義される．

$$labor(ts) = \sum_{i=1}^{n} \sum_{j=1}^{\sharp tc_i} lab(tc_i[j]),$$

ここで，$lab(tc[j])$ は tc の先頭から j 番目の遷移の労力を意味するものとする．

以上をまとめると，ステップ 3 で決定しなければならないものには，テスト戦術や労力の上限 l_{max} などのほかに，集団サイズ s，保存エリート数 e，交叉確率 c，突然変異率 m，ペナルティ p，世代交代回数 g がある．本手法によってより良い結果を得るためには，開発組織あるいはプロジェクトごとにこれらの適切な値を経験的に見つけていく必要がある．

[*2] 初期集団のテストスイートは l_{max} を超えないように生成される．しかし，交叉や突然変異によって新たに生成されるテストスイートは，l_{max} を超えることがある．

3. 適用例

本章では，本手法を商用ソフトウェアに試験的に適用した結果について述べる．

この商用ソフトウェアは，サーバ上に蓄積した種々の文書を高度に検索するシステム（オンライン文書検索システム）の一部で，規模は約 60 kLOC である．品質管理部門に所属するテスト技術者が，まず約 7 人日をかけて運用プロファイルを作成した．次に，約 0.5 人日をかけてテストの労力を検討し拡張運用プロファイルを完成させた．簡略表記したものを**図 3** に示す．戦術 b においては図 3 の確率分布は**図 4** の確率分布に変換される．テストの労力は，テストデータ入力に要する労力（**表 1**），システムの実行結果を待つのに要する労力（**表 2**），実行結果の正しさの判断に要する労力（**表 3**）の 3 つの要素の合計とした．

本研究では，本手法を実装したテストスイート生成ツールを開発した．このツールを用いてオンライン文書検索システムのテストスイートを戦術 a，b それぞれにつき 10 回生成した．パラメータの設定は，$l_{max} = 300$，$s = 10$，$e = 5$，$c = 0.2$，$m = 0.1$，$p = 0.5$，$g = 1{,}000$ とした．この結果を**図 5** と**図 6** に示す．図は，横軸を世代，縦軸を最良解の適合度とし，世代交代を重ねることによって最良解が成長していく様子を表している．10 回の生成は，それぞれ凡例に示すように線の種類によって区別している．

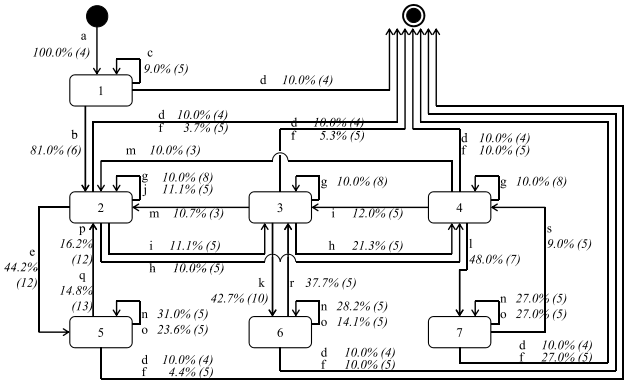

図 3 戦術 a に基づくオンライン文書検索システムの拡張運用プロファイル

Fig. 3 Strategy a-based extended operational profile of an online document search system.

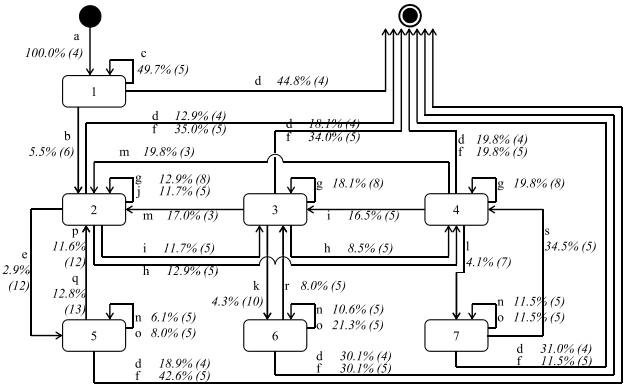

図 4 戦術 b に基づくオンライン文書検索システムの拡張運用プロファイル

Fig. 4 Strategy b-based extended operational profile of an online document search system.

表 1 テストデータ入力に要する労力

Table 1 Labor for inputting test data.

レベル	労力値	概要
高	3	入力フィールドが 2 個以上で入力する文字列が長い．
中	2	入力フィールドが 2 個以上で入力する文字列が短い．
低	1	入力フィールドが 1 個で入力する文字列が短い，またはボタン押下のみである．
なし	0	入力を必要としない．

表 2 システムの実行結果を待つのに要する労力

Table 2 Labor for waiting execution results of a system.

レベル	労力値	概要
高	4	テストデータ入力後実行結果を得るまでに 5〜20 秒程度を要する．
中	3	テストデータ入力後 3〜4 秒程度で実行結果を得ることができる．
低	2	テストデータ入力後 2 秒程度以下で実行結果を得ることができる．

表 3 実行結果の正しさの判断に要する労力

Table 3 Labor for judging execution results.

レベル	労力値	概要
高	6	確認すべき項目が 5 つ程度以上ある．
中	4	確認すべき項目が 3〜4 つ程度ある．
低	2	確認すべき項目が 1〜2 つ程度ある．
なし	0	確認すべき項目がない．

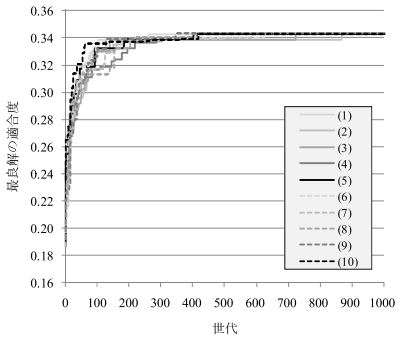

図 5 戦術 a の最良解の成長過程
Fig. 5 Process of growth of the best solution for strategy a.

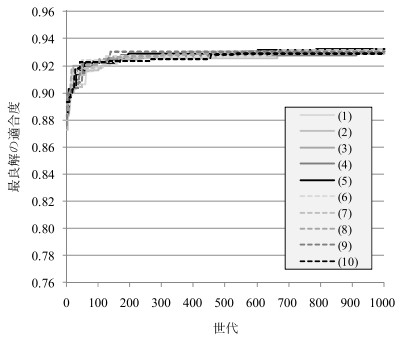

図 6 戦術 b の最良解の成長過程
Fig. 6 Process of growth of the best solution for strategy b.

最終的に得られた最良解の適合度の平均は，戦術 a では約 0.34，戦術 b では約 0.93 であった．これに対して，初期集団の適合度の平均は，戦術 a では約 0.09，戦術 b では約 0.83 であった．戦術 a では 0.25，戦術 b では 0.10 改善されたことになる．ここで重要なのは，どの世代においても最良解の適合度は UDC と等しくなるということである．これは，ペナルティの適用を受けない（すなわち適合度が UDC と等しくなる）ように初期集団を生成しており，さらに選択の手法としてエリート保存法を併用しているので，ペナルティの適用を受けない解の候補がどの世代においても必ず存在するからである．初期集団は，最適化を行わない従来の運用プロファイルベースドテスト法によって生成されたテストスイートと見なすことができるので，本手法は UDC を戦術 a では 0.25，戦術 b では 0.10 改善したということであり，従来手法よりも有効なテストスイートを生成できると結論できる．戦術 b の方が UDC の改善度が低いのは，図 3 と比較して図 4 の方が確率分布の偏りが大きく，高い UDC を持つテストケースの一部が初期集団においてすでに生成されていることが原因である．ゆえに，本手法は確率分布の偏りが少ない場合において特に大きな効果が期待できるといえる．また，戦術 b の方が UDC が高いのも同じ原因によるものであり，戦術 a よりも戦術 b が優れていることを意味しているわけではない．最適化されたテストスイートの内容についてテスト技術者を交えて確認したところ，戦術 a に基づくテストスイートは典型的な利用方法を集中的に実行するものであり，戦術 b に基づくテストスイートは想定外の利用方法を集中的に実行するものであった．通常実施しないようなテストケースが含まれており，より高いソフトウェア信頼性を実現するうえで有効であると考えられる．

なお，1 回のテストスイート生成に要した時間は 1 分未満であったため，本手法の適用に要した労力は，実質的に拡張運用プロファイル作成の 7.5 人日だけであった．今回は拡張運用プロファイルを新規に作成したが，これを開発資産として管理し，今後の類似するソフトウェアの開発時に再利用した場合，適用の労力を抑えると同時に確率分布やテスト労力の精度を向上させることができると考えられる．

4. 関連研究

本章では，本研究に関連のある従来研究について述べる．

運用プロファイルは，Musa [1] によって提案されて以来，研究や開発現場への応用が数多く行われ，テストの有効性を改善した事例が報告されている [4], [5]．しかしながらこれらの従来研究において，戦術 b は考慮されておらず，またテストスイートを最適化する試みも行われてこなかった．前者の原因は，運用プロファイルベースドテスト法がソフトウェア信頼性の効果的な改善を主眼として開発されたことと関係している．極論すれば，たとえフォールトが潜在するとしても，そのフォールトを含む機能が利用現場において実行される可能性がないのであれば，発見，除去する必要は必ずしもないという考えに基づいているからである．一方，後者の原因の 1 つは，テストスイートの質を運用プロファイルの観点から評価するメトリクスが存在しなかったことにあると考えられる．本研究では，UDC を導入することによって最適化を可能とした．UDC は，テストスイートやテストケースがユーザの利用特性をどれだけ網羅しているかを表すテスト網羅基準として提案された．文献 [7] では，ランダムテスト法に対する運用プロファイルベースドテスト法の優位性が UDC によって定量的に示されている．

近年では，遺伝的アルゴリズムを使用してテストスイートを生成する試みがさかんに行われている．たとえば，Andreou ら [9] は，ソースコードから生成したデータフローグラフに基づき，遺伝的アルゴリズムによって ALL-DU パス網羅基準を満たすテストスイートを生成するという手法を提案している．Mohapatra ら [10] は，モジュールの制御フローグラフを網羅するテストスイートを遺伝的アルゴリズムによって生成する手法を提案した．前者は，適合度の評価関数に網羅率を使用している点は本手法と同じであるが，カバーすることが困難なパスに対して加点する仕組みがあるのは特徴的である．また，後者は本手法と異なり，テストケースを染色体としている．サンプリング法を使用

することによって，網羅基準を満たし，かつテストケース数が少なくなるようなテストスイートを生成することができる．

テストスイートを生成するためのモデルとして本研究では有限状態機械を用いたが，類似するものとしては，たとえばUMLステートマシン図 [11] のような，コントロールフローだけでなくデータフローを含んだモデルを使用することが多い．このモデルは，有限状態機械と比較して高い表現力を持っている反面，各遷移に付随するアクションやガードの間にデータ依存性があるため，実行可能なパスを生成することが容易ではない [12]．この問題の解決策は2つ考えられる．1つは，パスの実行可能性に影響を与えるソフトウェアの内部変数や入力パラメータを状態変数に展開して有限状態機械を作成することである．有限状態機械の作成に手間がかかるという欠点はあるものの，通常のグラフ探索アルゴリズムによって実行可能なパスを生成できるようになる．もう1つの解決策は，モデルから実行可能なパスを探索的に生成することである．Kalajiら [13] は，すべての遷移を網羅し，かつ，遷移を発火させるための入力値を容易に発見できるような実行可能なパスを遺伝的アルゴリズムによって生成する手法を開発している．この手法では，遷移間のデータ依存関係を分析し，依存関係が深いものにペナルティを与える．ただし，Doungsa-ardら [14] は本質的に発火させることが困難な遷移の存在を指摘しており，探索的な手法における1つの課題となっている．本論文の手法では，先述のように有限状態機械を用いているためデータ依存性の問題は発生しないものの，表現力の高いモデルの導入に際しては考慮が必要となる可能性がある．

5. おわりに

本論文では，テスト労力についての情報を付加した拡張運用プロファイルに基づいて，テスト戦術や投入可能な労力をはじめとした各種制約条件を満たす，最適化されたテストスイートを生成する手法を提案した．ユーザの利用確率に比例して労力を配分する戦術aと，逆に反比例して労力を配分する戦術bがあり，テスト対象ソフトウェアの性格や状況などに応じて使用する戦術を選ぶことができる．最適化には遺伝的アルゴリズムを用いる．テストスイートを染色体，テストケースを遺伝子として，交叉，突然変異，選択などの遺伝的操作を行う．選択の際に用いる適合度の評価関数にUDCを導入している点が特徴的である．これによって最終的に，すべての制約条件を満たし，かつUDCができるだけ大きくなるようなテストスイートを得ることができる．テストスイートは，指定された労力の上限を超えないように生成されるので，逼迫したテスト工程においても本手法を適用することが可能である．本手法は，ソフトウェア信頼性を評価するために大量のテストケースを生成するという従来の運用プロファイルベースドテストとは異なっている．また，モデルの構成要素を網羅することを目的とした従来のモデルベースドテストとも異なっている．本手法を商用ソフトウェアに試験的に適用した結果，本手法によってテストスイートの質を改善できることが分かった．網羅性を指向した従来のテストを補完することが可能であり，ソフトウェアの高信頼化に寄与できると考えられる．

今後の研究では，拡張運用プロファイルをさらに拡張し，多様なテスト戦術を実現する方法について検討する．たとえば，ソフトウェアが各機能を果たせない場合の不利益の程度に関する情報を拡張運用プロファイルに付加すれば，致命的なフォールトを重点的に検出するためのテストスイートを生成できる可能性がある．また，実際の開発プロジェクトでの適用を通して，テスト戦術を効果的に実現するための各種パラメータ設定についても調査する予定である．

謝辞 本研究は平成22年度香川大学奨励研究経費および平成23年度日本学術振興会科研費若手研究（B）No.23700038によるものである．また，株式会社ジャストシステム西町和也，藤井禎，多田雅信，三橋尊志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] 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).
[2] 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).
[3] Rook, P.: *Software Reliability Handbook*, Elsevier Science (1990).
[4] Kelly, D.P. and Oshana, R.S.: Improving software quality using statistical testing techniques, *Information and Software Technology*, Vol.42, No.12, pp.801–807 (2000).
[5] 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).
[6] Beizer, B.: *Software Testing Techniques*, 2nd edition, Van Nostrand Reinhold (1990).
[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] Takagi, T., Hashimoto, S. and Furukawa, Z.: Operational Profile-based Test Suite Generation using a Genetic Algorithm, *Supplemental Proc. 20th International Symposium on Software Reliability Engineering*, Fast

Abstract, 2 pages, CD-ROM (2009).
[9] Andreou, A.S., Economides, K.A. and Sofokleous, A.A.: An automatic software test-data generation scheme based on data flow criteria and genetic algorithms, *Proc. 7th International Conference on Computer and Information Technology*, pp.867–872 (2007).
[10] Mohapatra, D., Bhuyan, P. and Mohapatra, D.P.: Automated Test Case Generation and Its Optimization for Path Testing Using Genetic Algorithm and Sampling, *Proc. WASE International Conference on Information Engineering*, pp.643–646 (2009).
[11] Object Management Group: *UNIFIED MODELING LANGUAGE*, Object Management Group (online), available from ⟨http://www.uml.org/⟩ (accessed 2011-08-22).
[12] Chanson, S.T. and Zhu, J.: A unified approach to protocol test sequence generation, *Proc. 12th Annual Joint Conference of the Computer and Communications Societies. Networking: Foundation for the Future, IEEE*, pp.106–114 (1993).
[13] 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).
[14] Doungsa-ard, 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).

高木 智彦 （正会員）

2002 年香川大学工学部卒業．2004 年同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．電子情報通信学会会員．

橋本 慎一朗 （学生会員）

2010 年香川大学工学部卒業．現在，同大学院工学研究科博士前期課程に在学．ソフトウェア工学，特にソフトウェアテスト法に興味を持つ．

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 年香川大学工学部信頼性情報システム工学科講師，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の教育，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正員．

古川 善吾 （正会員）

1975 年九州大学工学部卒業．1977 年同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．