修 士 論 文

題　目　　組み合わせテストにおける
　　不具合誘発パラメータ組の効率的特定技法

主任指導教員　　水野　修　教授

指導教員　　辻野　嘉宏　教授

京都工芸繊維大学大学院 工芸科学研究科

情報工学専攻

学生番号　　16622031

氏　　名　　西浦　生成

平成30年2月9日提出

様式 2 号

学位論文の要旨（和文）

平成 30 年 2 月 9 日

京都工芸繊維大学大学院
工芸科学研究科長　殿

工芸科学研究科　　情報工学専攻
平成 28 年入学
学生番号　　16622031
氏　　名　　西浦　生成　㊞

（主任指導教員　　水野　修　㊞　）

本学学位規則第 4 条に基づき、下記のとおり学位論文内容の要旨を提出いたします。

1.　論文題目

組み合わせテストにおける不具合誘発パラメータ組の効率的特定技法

2.　論文内容の要旨（400 字程度）

ソフトウェアテストにおいて、組み合わせテストの結果から不具合を誘発しているパラメータの組を特定すること（FIL）は、開発者が不具合の原因箇所を特定し修正するうえで有益な情報をもたらす。従来の FIL 手法では、得られたテスト結果を分析し、必要に応じてテストの追加実行を行いながら不具合誘発の可能性のあるパラメータ組全てを網羅的に検証していくことで不具合誘発パラメータ組の確実な特定を行う方式をとっているが、そういった手法では組み合わせの大きさに応じて組み合わせ爆発が発生し膨大な処理時間が必要となるため、大規模な組み合わせテストに適用するのは現実的ではない。本論文では従来手法に代わり、全ての可能性を考えることなく不具合を誘発している可能性が最も高いパラメータ組を直接導出することで組み合わせ爆発を回避しつつ効果的に FIL を行う方法を提案し、評価を行った。提案手法は SSCC 法と名付けられた、再帰処理をベースとするシンプルなアルゴリズムである。本論文ではまず SSCC 法のアイディアと基本的な実装を述べ、次にその動作が失敗する場合の原因を明らかにし、改良を施すことで完成に至るまでを記述した。実証実験には実在するソフトウェアモデルを使用し、従来手法との比較を行った。実証実験の結果として、提案手法の持つ効果の高さを明らかにした。また大規模なテストであるほど提案手法が有用となることを示し、研究成果の意義を示した。

Efficient identification technique of fault induced parameter set in combinatorial testing

2018 16622031 *NISHIURA Kinari*

Abstract

In software testing, detecting a combination of parameters that induces a defect from the result of a combinatorial testing (Faulty Interaction Localization, FIL) brings useful information to debugging. In the conventional FIL method, by analyzing the obtained test results and comprehensively verifying all parameter sets that may be induced as faults while performing additional tests as necessary, it is possible to comprehend the combination of fault inducing parameters. However, in such a method, combinatorial explosion occurs according to the size of the combination, which requires enormous processing time. So it is not practical to apply to large scale combination tests. In this paper, I propose new FIL method named SSCC method and evaluate it. SSCC method directly detects the set of combination of parameter with the highest possibility of inducing fault in without considering all the possibilities. Therefore SSCC method can avoid combinatorial explosion and do FIL effectively. In this paper, I describe the basic idea and implementation of SSCC method, then clarify the cause of the failure of the operation and improve it. We used a real software model for the demonstration experiment and compared SSCC method with the conventional method. As a result of demonstration experiment, I clarified the high effectiveness of the proposed method. In addition, the significance of the research results was established by showing that the larger the test the more useful the proposed method is.

目　次

1. 緒言 **1**

2. 準備 **3**

2.1 用語の定義 3

2.2 組み合わせテスト 3

2.3 Faulty Interaction Localization (FIL)・関連研究 4

2.4 前提 7

3. 提案手法 **8**

3.1 基礎アイディア 8

3.2 初期実装 10

3.2.1 SSCC 法のアルゴリズム説明 11

3.2.2 SSCC 法を用いた FIL の動作汎例 12

3.3 初期実装における失敗要因とその対応 13

3.3.1 失敗要因 1 14

3.3.2 失敗要因 2 15

3.3.3 失敗要因 3 16

3.3.4 失敗要因 4 17

3.3.5 失敗要因 5 17

3.3.6 複数の解の取り扱い 18

3.4 改良型 20

3.4.1 改良型 SSCC 法のアルゴリズム 20

3.5 制約への対応 22

3.5.1 入力とする制約仕様 22

3.5.2 スーパーパラメータの導入 22

3.5.3 制約を用いた Search 結果の補正 23

3.5.4 制約を用いた Select 結果の補正 23

4. SSCC 法の適用例 **25**

4.1 適用対象 25

4.2 適用動作例 25

5. 実証実験 **31**

5.1 研究設問 31

5.2 実験準備 31

5.2.1 実験対象 31

5.2.2 実験内容 32

5.2.3 比較対象 35

5.3 実験結果 36

6. 考察 **43**

6.1 研究設問への回答 43

6.2 妥当性の検証 50

6.2.1 内的妥当性 50

6.2.2 外的妥当性 50

6.3 今後の課題 51

7. 結言 **53**

謝辞 **53**

1. 緒言

ソフトウェアを開発する際——特に責任ある立場からソフトウェアを社会にリリースする際に——組み合わせテストの重要性は論を俟たない。一度世に出したソフトウェアに大きな不具合が見つかることは顧客の機会損失や信頼低下を招き、重大な損害に繋がることも珍しくない。リリース後の不具合の発覚を防ぐためには、リリース前に万全なテストを実施することが重要である。複数のパラメータの組み合わせによって誘発される不具合を網羅的に検出する組み合わせテストは、開発者にとって思いもよらない不具合を発見するのに大きな効力を持つ。ただし、不具合を発見しただけで満足してはならず、開発者は次にその不具合を修正する必要がある。「失敗したテストがある」という情報は、単に「ソフトウェアにバグがある」という情報しかもたらさない。ではそのバグはどこにあり、何を直せばいいのか？　例えば『実行環境が XX かつ入力が YY の場合に必ず失敗する』というように、組み合わせテストの結果から不具合を誘発しているパラメータ値の組み合わせを特定することができれば、その情報は必ずバグ原因箇所特定の大きな一助となるだろう。

組み合わせテストの結果から不具合を誘発するパラメータ値の組み合わせを特定する技法として Faulty Interaction Localization(FIL) がある。FIL を解くための手法がいくつか提案されているが、既存の手法では組み合わせ数を十分に減らすことができない問題や、組み合わせ数が爆発する問題が発生する場合がある。

筆者はその解決策として、ロジスティック回帰分析が不具合誘発パラメータ組の特定に有効であることを示し、それを利用した不具合誘発パラメータ組のサブセット予測による軽量な FIL 手法を提案した。しかしこの手法は特定精度と時間削減性の両方で不完全であることがその後の研究で明らかになったため、別方向からのアプローチによるさらなる手法開発を実施してきた。

本論文ではその結論として、新たな事後分析的 FIL 手法を提案する。この提案手法はシンプルな構造でありながら、組み合わせ爆発を回避し、追加で行うテストの実行も原則必要とせず、現実的な処理時間での効率的な不具合誘発パラメータ組の特定を実現するものである。提案手法は、4 つの実行容易なコマンドによって構成された再帰処理をベースとする極めてシンプルなアルゴリズムで与えられ、これらのコマンドの頭文字をとって SSCC 法と名付けられた。SSCC 法は、不具合を誘発し

ている可能性のある全てのパラメータの組み合わせを網羅的に考慮することなく、最も不具合誘発した可能性の高いパラメータの組み合わせを直接特定することで、組み合わせ爆発を起こさずにFILを行うことを目的としている。本論文ではまず、SSCC法を基本的なアイディアのみに基づく必要最小限のアルゴリズムとして構築し、初期実装を行った。その後、いくつかの要因により初期実装では期待した動作が行われないことを確認し、それらの失敗要因に対して補完的に改良を行ったものを改良型として提案する。またそれをパラメータ間に制約のあるシステムモデルに対応させる方法についても述べる。

また提案手法の有効性を評価するための実証実験も行った。実験には実際に存在する5種類のシステムモデルを対象に設計された合計16個の組み合わせテストを対象とし、ランダムに生成された不具合誘発パラメータ組を組み合わせテストの結果から特定させることで、SSCC法の特定精度と時間効率性を従来手法と比較した。評価の結果、提案手法の持つ効果の高さを明らかにした。また大規模なテストであるほど提案手法が有用となることを示し、研究成果の意義を示した。

以降の論文の構成は次の通りである。2章では、本研究の対象である組み合わせテストとFILについて説明する。3章では、提案する不具合誘発パラメータ組の効率的特定手法について説明し、4章では提案手法の具体的な適用例を説明する。5章で提案手法の評価実験とその結果について説明し、6章では実験結果に対する考察を述べ、7章で結言とする。

2. 準備

2.1 用語の定義

本論文での説明や議論に用いる基本的な用語を定義しておく。

パラメータは、ここでは複数の選択肢を持つようなシステムの入力を指す。表 2.1 に示すシステムは X,Y,Z の 3 つのパラメータを持つ。

パラメータ値 (Parameter-value, Pv) は、パラメータが取る値を指す。表 2.1 に示すシステムの持つパラメータ X は 1 または 2 という 2 種類のパラメータ値を取り得る。またパラメータ X が 1 という値を取っている状態のことを $p_X.1$ と表す場合がある。パラメータの定義からパラメータ値は必ず 2 種類以上の値を取る。

パラメータ組は、複数のパラメータ値が組み合わさった状態を指す。表 2.1 に示すシステムのパラメータがそれぞれ X = 1, Y = 2, Z = 3 であるとき、大きさが 2 のパラメータ組として (X,Y)=(1,2), (Y, Z)=(2,3), (Z,X)=(3,1) の 3 つが存在すると表現する場合がある。またこれらのパラメータ組を $\{p_X.1, p_Y.2\}$ のように表す場合がある。パラメータ組を構成するパラメータ値の数をそのパラメータ組の**因子数**と呼称する。またパラメータ組を構成するパラメータ値をそのパラメータ組の**構成要素**と呼ぶ場合がある。また因子数が 1 の場合も便宜的にパラメータ組と呼称する。

不具合誘発パラメータ組は、そのパラメータ組が存在するときに限り不具合が発生するようなパラメータ組を指す。$\{p_X.1, p_Y.2\}$ が不具合誘発パラメータ組であると言った場合、パラメータ X が 1 かつパラメータ Y が 2 である場合にシステムが正常でなくなるといった状態を表している。

テストケースは、テストを目的としたシステムの各パラメータへの値の割り当てを指す。テストケースを X = 1, Y = 2, Z = 3 としたとき、このテストケースはパラメータ組 $\{p_X.1, p_Y.2\}$ を含むといったように表す場合がある。

2.2 組み合わせテスト

組み合わせテスト [?, ?] は，パラメータ組によって引き起こされる不具合の検出を目的としたブラックボックステスト手法である。一般的には単体テストや結合テストを終えたシステムに対して、「これで完成としてもよいか」を確認するために実

施される。

組み合わせテストではパラメータ値の組み合わせパターンを網羅的にテストするため、パラメータ値を変えてテストケースを複数回実行する必要があり、このテストケースの集合を**テストスイート**と呼ぶ。全てのパラメータ値の組み合わせパターンを網羅するテストスイートは組み合わせテストを行う上で最も理想的だが、そのテストのサイズはシステムサイズに対して指数的に増加するため、現実には難しい場合が多い。そのためある組み合わせ数 t を設定し、t 個のパラメータ間でとる値のパターン全てを網羅するようにテストケース数を最適化されたテストスイートが実用的とされており、このテストスイートは t-way テストと呼ばれる。また t-way テストが網羅している最大の組み合わせ因子数 t を組み合わせテストにおける網羅因子数と呼称する。

例として表 2.1 のパラメータ仕様が与えられたときの 2-way テストを生成する。このパラメータ仕様を持つシステムは、表 2.2 のような SUT（System Under Test）モデルで表記される。ここでモデルサイズの $2^2 3^1$ という表記は 2 種類の値を取るパラメータが 2 つと 3 種類のパラメータ値を取るパラメータが 1 つあることを表している。まず全網羅のテストスイートなら、3 種類の値を取るパラメータが 1 つ、2 種類の値を取るパラメータが 2 つなので、全てのパターンを網羅するのに必要なテストケース数は $3 \times 2 \times 2 = 12$ となる。対して 2-way テストならば表 2.3 のテストスイートになり、テストケース数は 6 となる。考えられる全ての 2 つのパラメータ値の組み合わせ (X,Y), (Y,Z), (Z,X) を確認すると、確かに全てのパターンを網羅している。

組み合わせテスト生成ツールには、Microsoft 社の Czerwonka らが開発した PICT、産業技術総合技術研究所の Choi らが開発した pricot、Bryce らが開発した DDA などがある。

2.3 Faulty Interaction Localization (FIL)・関連研究

Faulty Interaction Localization (FIL) は不具合誘発パラメータ組特定とも呼ばれ、不具合原因の特定を目的として、組み合わせテストの結果からテストの失敗を誘発する原因となったパラメータ組を特定する技法である。例として表 2.3 の組み合わせテストを全て実行させたところ、最下部の $tc6$ だけが失敗したという結果が得ら

表 2.1　例題システムのパラメータ仕様

パラメータ	パラメータ値
X	1, 2
Y	1, 2
Z	1, 2, 3

表 2.2　例題システムの SUT モデル

システム名	パラメータ数	モデルサイズ	制約サイズ
例題システム	3	$2^2 3^1$	-

表 2.3　例題システムの 2-way テスト

テストケース	X	Y	Z
$tc1$	1	1	1
$tc2$	2	2	1
$tc3$	1	2	2
$tc4$	2	1	2
$tc5$	1	2	3
$tc6$	2	1	3

れた。このときテストの失敗を誘発したパラメータ組は $tc6$ 中に存在する $X = 2$、$Y = 1$、$Z = 3$、$(X, Y) = (2, 1)$、$(Y, Z) = (1, 3)$、$(Z, X) = (3, 2)$、$(X, Y, Z) = (2, 1, 3)$ という 7 つのパラメータ組が候補として考えられるが、$X = 2$ は $tc2$ と $tc4$ に、$Y = 1$ は $tc1$ と $tc4$ に、$Z = 3$ は $tc5$ に、また $(X, Y) = (2, 1)$ は $tc4$ にそれぞれ含まれておりそれらのテストは成功しているため、この 4 つの候補がテストの失敗を誘発している可能性は排除される。残り 3 つの候補のうちどのパラメータ組が実際にテストの失敗を引き起こしているのかは現時点で判断できないが、例えば $(X, Y, Z) = (1, 1, 3)$ と $(X, Y, Z) = (2, 2, 3)$ という 2 つのテストケースを新しく作って実行させ、$(X, Y, Z) = (1, 1, 3)$ だけが失敗したなら、$(Y, Z) = (1, 3)$ のパラメータ組がテストの失敗を誘発していることがただちに判明する。

ある程度大きな規模の組み合わせテストになるとこのように目視で FIL を行うことは困難なので、計算機を用いて FIL を行う様々な手法が考えられている。

既存の FIL 手法は事前設計的な方法と事後分析的な方法に大別できる。事前設計的な方法とは、組み合わせテストの設計時にそのテスト結果によって不具合誘発パラメータ組を一意に特定可能であるようなテストを設計してしまう方法である。この方法は FIL のプロセスを簡潔にできるが、テストケースの必要数が増えるうえ、特定の因子数の不具合誘発パラメータ組の存在を特定の個数だけ仮定して設計する必要があり、仮定と異なった場合には特定できないので手法としての汎用性に欠ける。また事後分析的な方法とは、組み合わせテストの実行結果を分析してどの組み合わせが不具合を誘発しているのかを特定する方法である。これには様々な種類の手法があるが、例えば Zhang らは Delta Debugging というテクニックを用いて、ある 1 つの失敗したテストケースのパラメータの値を部分的に変化させたテストケースを新たに逐次実行することで、そのテストケースを失敗に至らしめたパラメータ組を特定する手法を挙げている。また Ghandhari らや Zheng らは、テストスイート全体のテスト結果から不具合を誘発した可能性のあるパラメータ組を全て求め、その中から実際に不具合を誘発しているパラメータ組を追加テストによって検証していく虱潰し式の手法を提案している。これらの手法は確実に不具合誘発パラメータ組を特定可能である一方で、FIL のためにテストそのものを多数回に渡って実行する必要があり、テストあたりの時間的または金銭的コストの大きいソフトウェアのテストには不向きである。また、テスト中に存在する全てのパラメータ組から不具合

誘発の可能性があるパラメータ組と不具合誘発の可能性が皆無であるパラメータ組を区別しておく必要があるが、対象システムの持つパラメータ数や組み合わせテストの網羅因子数がわずかに大きくなるだけで組み合わせの総数は爆発的に増加するため、大規模な組み合わせテストではFILに必要な処理時間が容易に膨大なものとなってしまう。そのため入力規模の大きなソフトウェアや網羅因子数の高い組み合わせテストの場合、FILの完了までに数十分から数時間といった待機時間を要する場合もあり、実際の開発現場に導入することは現実的ではない。(注 1)

2.4 前提

新手法の提案に当たり、前提条件を以下のように設定しておく。これらは従来手法においても同様に前提とされているものである。

- 与えられたSUTモデル、パラメータ制約、組み合わせテストの設計、テストの結果は正確であるとする。
- テストの結果は常に同じであるとする。ランダムに発生する不具合や考慮外の要素を要因とした不具合は発生せず存在しないものとして扱う。
- FILの入力として使用する組み合わせテストのテスト結果は、全て成功あるいは全て失敗ではないものとする。(注 2)

(注 1):デバッガツールの豊富な現在ならば、こうしたFILによる長時間の自動特定を待つまでもなく、開発者は追跡デバッグを行うことで数分から数十分のうちに不具合箇所を発見してしまうだろう。ただしその作業には大きなストレスが伴う。
(注 2):組み合わせテストが全て成功したならば不具合を誘発しているパラメータ組は存在せず、組み合わせテストが全て失敗したならば原因特定は不可能である。どちらの場合もFILをわざわざ行う必要はない。

3. 提案手法

3.1 基礎アイディア

まず初めに、提案手法の目的は組み合わせ爆発を回避しつつ FIL を行うことであることを繰り返しておく。しかし、従来手法が組み合わせ爆発を起こすのは、決してそれが冗長な方法であるからではなく、全ての組み合わせを網羅的に確かめることで FIL 結果の確実性を担保しているからである。対して提案手法で行うのは、不具合を誘発した原因として最も怪しい、最も妥当と思われるパラメータ組合わせの直接的な特定であり、それが実際に不具合を引き起こしたかどうかまでは関与しない。本提案手法では、組み合わせ爆発を回避するためにこうした厳密さを捨てているに過ぎないことを承知頂きたい。

提案手法は、以下に述べる 3 つの仮定を利用する。

仮定 1： 不具合誘発パラメータ組は、その全ての構成要素がテストケース中に含まれている場合にテストが失敗する。

仮定 1 は紛れもない事実である。不具合を誘発するパラメータ組が全てテストケース内に含まれている場合にのみ不具合が誘発されテストは失敗するが、逆に不具合誘発パラメータ組を構成するパラメータ値のうち 1 つでも欠けていれば不具合は誘発されない。

仮定 2： 不具合誘発パラメータ組が 1 つしか存在しなければ、失敗した全てのテストケースに共通するパラメータ値を調べることで、不具合誘発パラメータ組を特定できることが期待できる。

仮定 2 の期待は直観的に理解できる。ただ一種類の不具合誘発パラメータ組のみが存在し、不具合を誘発しているとすると、テストが失敗したテストケースには必ずその不具合誘発パラメータ組が含まれているはずである。であれば、失敗した全てのテストケースには共通するパラメータ値が存在し、それらを統合させたものが不具合誘発パラメータ組であることを期待してもよいだろう。また逆に、複数の不具合誘発パラメータ組が存在するならば、失敗した全てのテストケースに共通する不具合誘発パラメータ組は存在しないため、失敗テストケースに共通要素は現れな

いはずである。

> **仮定 3：** あるパラメータの失敗率を「そのパラメータ値が含まれているテストケースのうち失敗したテストケースの割合」と定義する。失敗率の高いパラメータ値ほど、不具合誘発パラメータ組の構成要素である可能性が高いと期待できる。

仮定 3 の期待は簡単に言えば、失敗したテストケースに多く含まれているパラメータ値は失敗に関与した可能性が高いということである。これは次のように説明できる。

いま、A, B, C というパラメータ値で構成される不具合誘発パラメータ組がある。これをテストする 1 つのテストケースがあり、そのテストケースのパラメータ値はランダムに決まるが、A が含まれることだけはすでに分かっている。このとき、このテストケースに B と C が含まれることが決まればそのテストは失敗する。つまり、この A を含むテストケースが失敗する確率は、B と C が共にこのテストケースに含まれる確率に等しい。また、A, B, C に無関係なパラメータ値 D を考え、同様にテストケースにすでに D だけが含まれており、残りのパラメータ値がランダムに決定されるとする。このとき、このテストケースが失敗するためには、A, B, C を全て含む必要がある。つまり、D を含むテストケースが失敗する確率は、A と B と C が全て含まれる確率に等しい。全てのパラメータが 2 種類以上の値を選択できるという前提に立てば、あるテストケースが A,B,C のパラメータ 3 つを全て含む確率は、B,C のパラメータ 2 つを全て含む確率よりも小さい。このため、A を含むテストケースが失敗する確率は、D を含むテストケースが失敗する確率より小さいことになる。これは、不具合誘発パラメータ組の構成要素いずれかを含むテストケースは、構成要素でないパラメータ値いずれかを含むテストケースよりも失敗率が高いことを表す。翻って、失敗率が高いパラメータ値ほど、不具合誘発パラメータ組の構成要素である可能性が高いと期待できる。

以上の三つの仮定を使って、以下のような方法で全ての不具合誘発パラメータ組を特定できることが期待できる。

不具合誘発パラメータ組が一つだけ存在する場合、**仮定 2** から、失敗したテストケースに共通するパラメータ値を調べることでそれを特定できる。

不具合誘発パラメータ組が複数存在する場合、**仮定 2** から、失敗したテストケースに共通するパラメータ値を調べてもそれらを特定することはできない。この場合は次に、全てのパラメータ値のうち最も失敗率の高いパラメータ値 Pv_{max} を求める。**仮定 3** から、Pv_{max} は不具合誘発パラメータ組いずれかの構成要素であることが期待できる。元のテスト結果集合から Pv_{max} を含んでいるテストケースを除外すると、このとき**仮定 1** から、テストケースの除外後に残ったテストの失敗は Pv_{max} を構成要素に含む不具合誘発パラメータ組によるものではないことが確定する。仮に不具合誘発パラメータ組が合計で 2 つ存在していた場合、残りのテスト失敗はただ 1 つの不具合誘発パラメータ組が誘発しているので、**仮定 2** から、失敗テストの共通パラメータ値を調べることでその不具合誘発パラメータ組を特定することができる。そうして特定した不具合誘発パラメータ組を含むテストケースを元のテストケース集合から除外することで、もう片方の不具合誘発パラメータ組も特定できる。不具合誘発パラメータ組の個数が 2 個ではなく n 個の場合でも、$(n-1)$ 回の「失敗率が最大であることによる構成要素の特定」および「構成要素を含むテストケースの除外」を行うことでただ 1 つの不具合誘発パラメータ組のみがテストを失敗させている状態にすることができ、そこから順に n 個の不具合パラメータ組を特定することができる。

3.2 初期実装

前述した基礎アイディアを利用し、不具合誘発パラメータ組の特定を効率的に行う新手法を考案した。新手法は 4 つの実行容易な操作概念 Search（探索）、Select（選択）、Cut（切除）、Call（呼出）によって構成されることから、頭文字をとって SSCC 法と命名した (注 3)。SSCC 法では自己の呼び出しを備えたサブルーチンである *SSCC* 関数を使用し、再帰的な処理を行うことによって全てのテスト失敗を最小の個数で

(注 3)：SSCC は Smart Search of Contributive Combination の頭字語でもありダブルミーニングである。

説明するようなパラメータ組を直接特定する。以下にSSCC法のアルゴリズムの説明と動作例を示す。

3.2.1 SSCC法のアルゴリズム説明

- 組み合わせテストを行ったテストスイートと各テストケースのテスト結果を対応付けたテスト結果集合を入力とする。
- 関数 $SSCC$ は渡されたテスト結果集合を T_0 とし、処理を開始する。
- T_0 のうち失敗したテストケースの個数が0または1であれば、特定不可能として処理を終了する。(注4)
- [**Search**$_1$] (注5) まず、T_0 における全ての失敗したテストケースに共通するパラメータ値を調べる。存在すればそれらを構成要素とするパラメータの組を不具合誘発パラメータ組として返却し、存在しなければ処理を続行する。
- [**Select**] 各パラメータ値の失敗率を算出し、最も失敗率の高いパラメータ値を Pv_{max} とする。ただし、あるパラメータ値の失敗率は

$$\text{失敗率} = \frac{\text{そのパラメータ値を含み失敗したテストケースの数}}{\text{そのパラメータ値を含むテストケースの数}} \tag{3.1}$$

とする。
- [**Cut**$_1$] T_0 から Pv_{max} を含むテストケースを除外した新たなテスト結果集合 T_1 を作成する。
- [**Call**] 関数 $SSCC$ に T_1 を渡し、既に特定された不具合誘発パラメータ組のセットである返り値 R を受け取る。
- [**Cut**$_2$] T_0 から R のうちいずれかでも含むテストケースを除外した新たなテスト結果集合 T_2 を作成する。
- T_2 のうち失敗したテストケースの個数が0または1であれば、特定不可能として処理を終了する。
- [**Search**$_2$] T_2 にはただ1つの不具合誘発パラメータ組が存在していると考えら

(注4):失敗したテストケースがただ1つのとき、そのテストケースに含まれるパラメータ値は全て「失敗したテストケース値に共通するパラメータ値」となるため、原因であるパラメータの組み合わせを特定できない。
(注5):**Search**, **Cut** は同じ $SSCC$ 関数内で2度使用するため、位置により区別する場合には **Search**$_1$ 等とし、特に区別しない場合には **Search** と表記する。

れるので、T_2 における全ての失敗したテストケースに共通するパラメータ値を調べ、それらを構成要素とするパラメータ組を既に受け取った返り値 R に加えて返却する。

3.2.2 SSCC 法を用いた FIL の動作汎例

いま、入力として複数の失敗を含むテスト結果集合がある。これらのテスト失敗の原因である不具合を誘発しているパラメータ組として、

$$F_1 = \{F_1a, F_1b, F_1c\}, \quad F_2 = \{F_2a, F_2b, F_2c\}, \quad F_3 = \{F_3a, F_3b, F_3c\} \tag{3.2}$$

という因子数が 3 のパラメータ組が 3 組存在しているが、FIL 実行者には未知である。

このテスト結果集合を関数 $SSCC$ に渡し、関数 $SSCC$ はこれを T_0 として受け取る。今後省略するが、常にテスト結果集合の中に失敗したテストケースは 2 件以上あったとする。まず T_0 の失敗テストケースに共通するパラメータ値を調べるが、3 種類の原因があるため共通のパラメータ値は見つからない。次に最も高い失敗率を持つパラメータ値を調べたところ F_1a だった。よって T_0 から F_1a を含むテストケースを除いたテスト結果集合 T_1 を作成し、T_1 を引数として関数 $SSCC$ を呼び出した。

最初に呼び出された関数 $SSCC$ 内の処理を一層目とし、一層目で呼び出した $SSCC$ 関数内の処理を二層目と呼称する（同様に n 層で呼び出した $SSCC$ 関数内の処理を $n+1$ 層目と呼称する）。二層目の入力である T_0' は一層目の T_1 なので、F_1a を含むテストケースは存在せず、従って不具合誘発パラメータ組 F_1 によって引き起こされたテストの失敗は T_0' には存在しない。しかしまだ 2 種類の原因が存在しているため、失敗したテストケースに共通するパラメータ値は見つからない。次に T_0' における最も高い失敗率を持つパラメータ値を調べたところ F_2a だった。よって T_0' から F_2a を含むテストケースを除いたテスト結果集合 T_1' を作成し、T_1' を引数として関数 $SSCC$ を呼び出した。

三層目の入力である T_0'' は二層目の T_1' なので、F_1a を含むテストケースが一層目で、F_2a を含むテストケースが二層目でそれぞれ除外されており、従って不具合誘発パラメータ組 F_1 と F_2 によって引き起こされたテストの失敗は T_0'' には存在しない。よって T_0'' における全てのテストの失敗は 1 種類の不具合誘発パラメータ組 F_3

によって引き起こされているので、全ての失敗に共通するパラメータ値を調べることで F_3a, F_3b, F_3c の 3 つのパラメータ値が見つかる。これを構成要素とするパラメータ組を二層目に返却する。

二層目では返り値 R としてパラメータ組 $\{F_3a, F_3b, F_3c\}$ を受け取り、T_0' から受け取ったパラメータ組を含むテストケースを除いたテスト結果集合 T_2' を作成する。T_2' は一層目で F_1a を含むテストケースが、また三層目からの返り値によって F_3 を含むテストケースが除外されているため、現在 T_2' におけるテストの失敗は全て F_2 によって引き起こされている。このため全ての失敗に共通するパラメータ値を調べると F_2a, F_2b, F_2c が見つかる。これらを構成要素とするパラメータ組と三層目からの返り値 R を合わせ、$\{F_2a, F_2b, F_2c\}$ と $\{F_3a, F_3b, F_3c\}$ の 2 つのパラメータ組を一層目に返却する。

一層目で受け取った返り値 R に含まれるパラメータ組、すなわち $\{F_2a, F_2b, F_2c\}$ と $\{F_3a, F_3b, F_3c\}$ のいずれかを含むテストケースを T_0 から除いたテスト結果集合 T_2 を作成する。ここで不具合誘発パラメータ組 F_2 と F_3 によるテストの失敗は既に除かれているので、T_2 におけるテストの失敗は全て F_1 によって引き起こされたものであり、失敗したテストケースに共通するパラメータ値を調べると F_1 の構成要素である F_1a, F_1b, F_1c が見つかる。これらを構成要素とするパラメータ組と二層目からの返り値を加え、$\{F_1a, F_1b, F_1c\}$, $\{F_2a, F_2b, F_2c\}$, $\{F_3a, F_3b, F_3c\}$ という 3 つのパラメータ組を返却する。

この結果として、初めに $SSCC$ 関数に入力したテスト結果集合に対し、そこに含まれる全ての不具合誘発パラメータ組が出力として得られることになる。

3.3 初期実装における失敗要因とその対応

3.2 節に示した初期実装で動作確認を行ったところ、入力とする組み合わせテスト結果によって、期待通りの動作と出力が得られる場合もある一方で期待した動作が行われずに終了する場合もあることがわかった。調査の結果、そうした失敗の原因を 5 つの要因に分類できることが判明した。以下に各要因と、それに対処するためアルゴリズムに対して行った対応をそれぞれ示す。

3.3.1　失敗要因 1

失敗要因 1

失敗率により不具合誘発パラメータ組の構成要素を推測する **Select** において、どの不具合誘発パラメータ組の構成要素でもないパラメータ値の失敗率が最大になってしまうことによるもの。

失敗要因 1 への対応

この失敗の発生は、3.1 節における **仮定 3** の期待が外れた場合を示している。不具合誘発パラメータ組の構成要素であるパラメータ値の失敗率が高くなる仕組みは 3.1 節で述べたが、ここで述べた失敗率が意味するのはあくまで「失敗するであろう確率」であり、言わば失敗率の理想値である。比べて、提案手法における **Select** で扱う失敗率は「実際に失敗した割合」であり、言わば失敗率の実測値である。あるパラメータ値の失敗率を求める際に、そのパラメータ値を含むテストケースの全てのパターンが用意されていれば、そこから算出される「実際に失敗した」失敗率は理想値と一致するだろう。しかし実際に用いられるテストスイートは t-way テスト等、テストケース数が圧縮削減されたものであるため、「あるパラメータ値を含むテストケースの全てのパターン」は得られない場合が多い。そういったギャップから、「現実に得られた」失敗率が理想値よりも偶然高く、あるいは低くなってしまい、その結果として不具合誘発パラメータ組の構成要素でないパラメータ値の失敗率が構成要素のものよりも高くなる事態が起こりうる。失敗要因 1 による特定失敗はその結果である。

構成要素ではないにも関わらず失敗率が最も高くなったパラメータ値は、そのパラメータのとる値の選択肢の多いものに多数見られた。これは値の選択肢の多いパラメータでは、値の選択肢が少ないパラメータに比べ、1 つのパラメータ値当たりの出現するテストケース数が少ないことが原因であると考えられる。例えば選択肢が 2 通りのパラメータ $P_a : \{1,\ 2\}$ の取るパラメータ値 $P_a.1$ は全テストケースのおよそ半分に含まれていると考えられるが、選択肢が 10 通りのパラメータ $P_b : \{1,\ 2, ...,\ 10\}$ の取るパラメータ値 $P_b.1$ は全テストケースのおよそ 10 分の 1 にしか含まれておらず、そのためそれぞれの失敗率を算出する際の母数には約 5 倍の開きがある。すな

わち、選択肢の多いパラメータのとるパラメータ値の失敗率は選択肢の少ないものと比べて理想値とのギャップが大きくなる可能性が高く、信頼度が低いと言える。

この問題を解決するため、信頼度を基に失敗率を補正する**失敗率スコア**を導入する。あるパラメータ値の失敗値スコアは、そのパラメータ値が属するパラメータのとる選択肢の数を N_p とすると、

$$\text{失敗率スコア} = \text{失敗率} \times \frac{N_p + 1}{N_p} \tag{3.3}$$

で表される。失敗率に掛けられる倍率は、N_p が小さいほど高く（最大で 1.5 倍 (注 6)）、また N_p が大きいほど 1 倍に近づく。**Select** で失敗率の代わりに失敗率スコアを使用することによって、信頼度の低いパラメータ値の順位を抑制し、また構成要素の中でもより信頼度の高いパラメータ値を優先的に選択することができる。

3.3.2 失敗要因 2

失敗要因 2

失敗したテストケースに共通するパラメータ値を調べる **Search** において、ただ一つ存在する不具合誘発パラメータ組の構成要素ではないにも関わらず、失敗した全てのテストケースに偶然含まれているパラメータ値が存在することで誤った不具合誘発パラメータ組を導いてしまうことによるもの。

失敗要因 2 への対応

この失敗は、3.1 節における**仮定 2** の期待が外れた場合を示している。失敗要因 2 の状況において我々は偶然共通しているパラメータ値と不具合誘発パラメータ組の構成要素を区別することができない。またこれらのパラメータ値を組み合わせたパラメータ組は確かに不具合を誘発するが、それはその組み合わせの部分集合に実際の不具合誘発パラメータ組が存在しているからに他ならない。正しい不具合誘発パラメータを特定するためには、そういった最小の大きさで不具合を誘発させている **Search** 結果の部分集合を特定する必要がある。

失敗要因 2 への対応として、まず **Search** で見つかった全ての失敗テストケースに共通するパラメータ値を軽率にパラメータ組とすることを止める。代わりに、次のような条件を満たすパラメータ組を特定する操作を行う。

(注 6)：2.1 節で定義したようにパラメータは必ず 2 つ以上の選択肢を持つため

いま、**Search** の対象となっているテスト結果集合を T_{now}、入力となった本来のテスト結果集合を T_{org} とする。目的とするパラメータ組は、**Search** によって検出されたパラメータ値を構成要素とし、

1. T_{now} においてそのパラメータ組を含むテストケースは全て失敗している。
2. T_{org} においてそのパラメータ組は失敗したテストケースにのみ含まれている。

という 2 つの条件を同時に満たすパラメータ組のうち、最も少ない因子数のパラメータ組である。パラメータ値の集合からこうしたパラメータ組を見つける処理を**最小組抽出**と呼称する。また最小組抽出によって条件を満たすパラメータ組を特定できた場合、最小組抽出を満たしたと表現する。最小組抽出によって条件を満たす組を発見できればそれを返却する。条件を満たす組が複数存在する場合はそれらのセットを返却する。**Search** によって失敗テストケースに共通するパラメータ値が検出されても、最小組抽出が満たされなければ返却を行わず処理を続行する。

3.3.3 失敗要因 3

失敗要因 3

n 個の不具合誘発パラメータ組を特定するにあたり、$n-1$ 個の構成要素であるパラメータ値を含むテストケースを切除していく過程で、n 層目、あるいはそれ以前の層におけるテストの失敗数が偶然に 0 あるいは 1 になってしまい、特定不可能として終了したことによるもの。

失敗要因 3 への対応

失敗要因 3 の状況に加え、初期実装の段階では仕様上、入力となるテスト結果集合のうち失敗したテストケースが 1 件しかない場合はただちに特定失敗として処理を終了していたが、最小組抽出を導入したことで失敗したテストケースが 1 件のみの場合でも特定処理を行えるようになる。

一方で、失敗したテストケースが処理中で 0 件となった場合の処理は場合分けを考える必要がある。$\mathbf{Cut}_2$ によって T_2 の失敗テストケース数が 0 になった場合、下層で特定した不具合誘発パラメータ組によって現時点での全ての失敗が説明されている状態であり、この層でさらに特定すべき不具合誘発パラメータは存在しないと考

えられるため、下層からの返却 R をそのまま上層に返却すればよい。$\mathbf{Cut}_1$ によって T_1 の失敗テストケース数が 0 になった場合だが、この場合については少し保留しておく。

3.3.4 失敗要因 4

失敗要因 4

複数ある不具合誘発パラメータ組の全てが共通して持つパラメータ値 Pv_{common} が存在する場合、$\mathbf{Search}_1$ で全ての失敗したテストケースに共通するパラメータ値として Pv_{common} が該当し出力してしまうことによるもの。

失敗要因 4 への対応

最小組抽出を導入したことで、複数の不具合誘発パラメータ組に共通するパラメータ値が存在していてもそれが返されることはなくなる。しかしそうすると次の問題が発生する。**Select** でその共通パラメータ値が選択されると、その共通パラメータ値を含むテストケースを切除することで全てのテスト失敗が取り除かれることになるのである。そうなると特定が不可能であるため、対策を講じる必要がある。

n 層目の $\mathbf{Search}_1$ によって失敗した全てのテストケースに共通するパラメータ値が検出されたにも関わらず、最小組抽出を満たすパラメータ組が存在しなかった場合、そのパラメータ値は n 層目の T_0 におけるテスト失敗を引き起こしている不具合誘発パラメータ組すべてに共通するパラメータ値であると見なすことができる。従ってそのパラメータ値を**重複要素**として登録し、続けて行う **Select** 処理では重複要素であるパラメータ値を存在しないものとして扱う。

重複要素が **Select** で選ばれなくすることで、あるパラメータ値を含むテストケースを削除することで失敗したテストケースの件数が 0 になるといった事態は発生しえなくなる。そのため 3.3.3 節で保留した、$\mathbf{Cut}_1$ によって T_1 の失敗テストケース数が 0 になった場合の対応は考えなくてよい。

3.3.5 失敗要因 5

失敗要因 5

不具合誘発パラメータ組の一部の組が共通して持つパラメータ値がある場合——例えば三つの不具合誘発パラメータ組である F_1, F_2, F_3 のうち F_1 と F_2 が共通のパラメータ値 Pv_{common} を持っている場合、**Select** で Pv_{common} が選ばれると、切除後のテスト結果集合には F_3 によるテストの失敗しか存在しなくなる。この場合、F_3 の特定とその切除を経て $\mathbf{Search}_2$ が行われるが、三つある原因のうちまだ二つが残っており、また共通パラメータ値 Pv_{common} が存在するために、$\mathbf{Search}_2$ の出力結果が F_3 と Pv_{common} になってしまうことによるもの。

失敗要因 5 への対応

こちらは言わば失敗要因 4 の状況が $\mathbf{Search}_2$ で起こった場合である。**Select** で選ばれるパラメータ値を含むテストケースの切除が複数の不具合誘発パラメータ組による失敗を切除してしまうと、$\mathbf{Search}_2$ を行う段階で複数の不具合誘発パラメータ組がまだ残っているため特定ができない。$\mathbf{Search}_2$ の結果によって複数の不具合誘発パラメータ組に共通するパラメータ値 Pv_{common} が検出されるが、Pv_{common} は不具合誘発パラメータ組ではないため最小組抽出を満たさない。初期実装では $\mathbf{Search}_2$ が SSCC 関数の最後の処理なので、$\mathbf{Search}_2$ で不具合誘発パラメータ組を特定できなければ続けて行うべき処理がなく特定失敗となる。

この問題への対応として、もし $\mathbf{Search}_2$ で全ての失敗テストケースに共通するが最小組抽出を満たさないパラメータ値 Pv_{common} が存在した場合、その $\mathbf{Search}_2$ の処理を $\mathbf{Search}_1$ だとして扱い、続けて再度 **Select**、$\mathbf{Cut}_1$、**Call**、$\mathbf{Cut}_2$、$\mathbf{Search}_2$ を行う。その際、失敗要因 4 への対応と同じように **Select** から重複要素 Pv_{common} を除外する。

3.3.6 複数の解の取り扱い

諸々の失敗要因への対応として最小組抽出を導入したが、最小組抽出を満たすパラメータ組が複数存在することもある。その場合には FIL 結果として複数の解が出力されることになる。このとき、本当にそれら全ての解が成り立つ場合もあるが、誤った解が含まれている場合もある。誤った解が現れるのは、その最小組抽出が行われた層より前の層でテストケースが切除されていたために、必要な情報が失われていたことが原因である。そうした誤った解を除外するため、解の出力時にそのパ

ラメータ組が全てのテストケース失敗を説明できているか検査する。この検査を**正解検査**と呼称する。

以下に正解検査の具体例を述べる。いま、

$$F_1 = \{F_1a, F_1b, F_1c\}, \quad F_2 = \{F_2a, F_2b, F_2c\}, \quad F_3 = \{F_3a, F_3b, F_3c\} \tag{3.4}$$

という 3 組の不具合誘発パラメータ組によってテストの失敗が引き起こされたテスト結果集合があり、ここから一層目と二層目で F_1a と F_2a を含むテストケースを切除することで三層目の T_0'' へと至った。T_0'' に対して失敗したテストケースに共通するパラメータ値を調べることで、F_3 の構成要素である F_3a, F_3b, F_3c に加え、偶然共通していた Pv_i の 4 つが得られた。この 4 つのパラメータ値を構成要素として最小組抽出を行うと、最も少ない要素数で最小組抽出を満たすパラメータ組として $\{F_3a, F_3b, F_3c\}$ と $\{F_3a, F_3b, Pv_i\}$ の 2 つが得られた。この 2 つのパラメータ組をここではこれ以上区別できないので、これらを一つのセットとして返却した。結果、最終的な返却として、

$$\{F_1a, F_1b, F_1c\}, \quad \{F_2a, F_2b, F_2c\}, \quad \{\{F_3a, F_3b, F_3c\} \; or \; \{F_3a, F_3b, Pv_i\}\} \tag{3.5}$$

が得られた。すなわち、SSCC 法は不具合誘発パラメータ組の特定結果として、

$$\{F_1a, F_1b, F_1c\} \text{ と } \{F_2a, F_2b, F_2c\} \text{ と } \{F_3a, F_3b, F_3c\} \tag{3.6}$$

$$\{F_1a, F_1b, F_1c\} \text{ と } \{F_2a, F_2b, F_2c\} \text{ と } \{F_3a, F_3b, Pv_i\} \tag{3.7}$$

という 2 通りの可能性を提示しているのである。ただし、(3.7) のパラメータ組のセットは全てが正しい不具合誘発パラメータ組ではない。

ここで正解検査を行う。(3.6) のパラメータ組のセットのいずれかを含んでいるテストケースを調べると、それらは全て失敗しており、また全ての失敗したテストケースはこれらのパラメータ組いずれかを含んでいる。すなわち、このパラメータ組の存在はテストの失敗に全射している。このことが確認できた場合、前者の不具合誘発パラメータは正解検査に合格したと言う。次に (3.7) のパラメータ組のセットを調べると、$\{F_3a, F_3b, Pv_i\}$ が含まれているが失敗していないテストケース、または失敗しているがどのパラメータ組も含まれないテストケース、あるいはその両方が発見され、(3.7) のパラメータ組のセットはテストの失敗に全射していないことが

判明した。このような場合、後者の不具合誘発パラメータ組は正解検査に合格しなかったとし、解として出力されない。結果として、出力された解は

$$\{F_1a, F_1b, F_1c\} \text{ と } \{F_2a, F_2b, F_2c\} \text{ と } \{F_3a, F_3b, F_3c\} \tag{3.8}$$

のみとなり、正しい結果であることが確認できる。

(3.6) と (3.7) の両方ともが正解検査に合格する場合もある。このときは入力となったテスト結果集合、つまり元を辿れば設計されたテストスイートにおいて、2 つの組み合わせ $\{F_3a, F_3b, F_3c\}$ と $\{F_3a, F_3b, Pv_i\}$ は常に同じテストケースにしか含まれていなかったためにこの 2 つを区別することが出来ないことが原因である。正解検査を実施しても複数の結果が出力された場合、利用者は不具合誘発パラメータ組のセットとしてこれら複数の可能性を考える必要がある。またこうした複数の可能性が考えられるのはテスト結果の情報不足が原因であるので、2.3 節の例題システムに対する FIL で見たように、候補を絞り込むのに有効なテストを追加で行うことで不具合誘発パラメータ組を一意に特定できる場合がある。

3.4 改良型

前節で述べた失敗要因への対応案を基に、初期実装の改良を行った。また以下に改良型の SSCC 法のアルゴリズムの説明を示す。

3.4.1 改良型 SSCC 法のアルゴリズム

- 組み合わせテストを行ったテストスイートと各テストケースのテスト結果を対応付けたテスト結果集合を入力とする。
- 関数 $SSCC$ は渡されたテスト結果集合を T_0 とし、処理を開始する。
- $[\mathbf{Search}_1]T_0$ における「全ての失敗したテストケースに共通するパラメータ値」を調べる。
  - 全ての失敗したテストケースに共通するパラメータ値が存在すれば、それらのパラメータ値を構成要素として最小組抽出（3.3.2 節を参照）を行う。
    - 最小組抽出が成功すればその結果を返す。

    - 最小組抽出が失敗すれば、検出されたパラメータ値を重複要素（3.3.4 節を参照）として登録し、処理を続行する。
  - 全ての失敗したテストケースに共通するパラメータ値が存在しないなら、処理を続行する。

- [**Select**] 各パラメータ値の失敗率スコア（3.3.1 節を参照）を算出し、失敗率スコアが最も高いパラメータ値を Pv_{max} とする。
  - 重複要素が登録されているなら、その重複要素であるパラメータ値は順位に含めない。

- [$\mathbf{Cut}_1$] T_0 から Pv_{max} を含むテストケースを除外した新たなテスト結果集合 T_1 を作成する。
- [**Call**] 関数 $SSCC$ に T_1 を渡し、パラメータ組のセットである返り値 R を受け取る。
- [$\mathbf{Cut}_2$] T_0 から R のうちいずれかでも含むテストケースを除外した新たなテスト結果集合 T_2 を作成する。
  - 返り値 R にパラメータ組のセットが含まれている場合、そのうちどれか 1 つのパラメータ組でも含んでいるテストケースを除外する。
  - T_2 における失敗したテストケースの数が 0 なら、下層からの返り値 R をそのまま返却する。

- [$\mathbf{Search}_2$] T_2 における全ての失敗したテストケースに共通するパラメータ値を調べる。
  - 全ての失敗したテストケースに共通するパラメータ値が存在すれば、それらのパラメータ値を構成要素として最小組抽出を行う。
    - 最小組抽出が成功すれば、その結果であるパラメータ組を R に加えたものを返す。
    - 最小組抽出が失敗したなら、全ての失敗テストケースに共通していたパラメータ値を重複要素として登録し、処理を続行する。
  - 全ての失敗したテストケースに共通するパラメータ値が存在しないなら、処理を続行する。

- **Search**$_2$ で不具合誘発パラメータ組を発見できなければ、その時点を **Search**$_1$ 終了時と同様に見なし、**Select** から **Search**$_2$ までをもう一度行う。
- 最終的な返却に対して正解検査（3.3.6 節を参照）を行い、合格した解のみを出力する。合格した解がなければ失敗とする。

3.5 制約への対応

現実にテストされるソフトウェアでは、例えば「パラメータ X が a のときはパラメータ Y は必ず b になる」などパラメータ間に何らかの制約があることが多く、組み合わせテストの設計時にも制約仕様は考慮される。前節までの SSCC 法のアルゴリズムは制約の存在に対応しておらず、結果の信頼性が損なわれたり処理が冗長になる場合がある。制約仕様を入力に加え、制約の存在を考慮したアルゴリズム改良を行うことで、パラメータ間に制約のあるソフトウェアに対しても SSCC 法を適用させることができる。

3.5.1 入力とする制約仕様

制約仕様は、開発者により「パラメータ値 a とパラメータ値 b は同時にテストケースに含まれない（非両立制約）」、「パラメータ値 a が含まれるときは常にパラメータ値 b が含まれる（条件制約）」など表現方法が異なるが、非両立制約と条件制約は互いに変換できるため、入力とする制約仕様は条件制約に統一する。また、「パラメータ a の値が 1 のときは常にパラメータ b の値は 1 になる」という条件制約式を

$$P_a.1 \rightarrow P_b.1 \tag{3.9}$$

と表し、$P_a.1$ を左辺、$P_b.1$ を右辺と呼称する。テスト結果集合に加え、対象の SUT モデルに存在するこうした条件制約式のリストを SSCC 法への入力とする。

3.5.2 スーパーパラメータの導入

$P_a.1 \rightarrow P_b.1$ という条件制約と $P_b.1 \rightarrow P_a.1$ が同時に存在すれば、パラメータ値 $P_a.1$ とパラメータ値 $P_b.1$ は常に同じテストケースに含まれることになる。このとき、不具合誘発パラメータの特定という立場からは $P_a.1$ と $P_b.1$ は区別できない。なのでこ

ういった制約仕様によって区別のできないパラメータ値のセットがあることが明らかな場合、それらを一つのスーパーパラメータとして扱う。条件制約式のリストからスーパーパラメータが存在すると分かった場合、テスト結果集合においてそのスーパーパラメータに含まれるパラメータ値を全てスーパーパラメータに置き換える。

$P_a.1$ と $P_b.1$ がスーパーパラメータ S_1 であるとき、SSCC 法を適用した結果として、

$$\{S_1, Pv_x, Pv_y\} \tag{3.10}$$

という 1 組の不具合誘発パラメータ組が得られたなら、それは実際の不具合誘発パラメータ組として

$$\{P_a.1, Pv_x, Pv_y\} \text{ もしくは } \{P_b.1, Pv_x, Pv_y\} \tag{3.11}$$

という 2 通りの可能性があることを表している。この可能性はテストケースの不足ではなく制約によるものなので、どのような追加のテストを行ってもどちらかの可能性が消えることはない。

3.5.3 制約を用いた Search 結果の補正

「$Pv_a \rightarrow Pv_b, Pv_c$（$Pv_a$ が含まれるテストケースには常に Pv_b と Pv_c が含まれる）」という制約 C があり、いま **Search** の対象としているテスト結果集合におけるテストの失敗を引き起こしているのは不具合誘発パラメータ組 F：$\{Pv_a, Pv_x, Pv_y\}$ のみであるとする。このとき失敗したテストケースに共通するパラメータ値を調べると、F の構成要素である Pv_a, Pv_x, Pv_y の他に、制約 C の作用で Pv_b, Pv_c も見つかってしまう。こうした場合、条件制約リストを順に調べて条件制約式の右辺と左辺の要素が全て **Search** 結果に含まれているような条件制約式が見つかれば、その右辺の要素を全て **Search** 結果から除外することで結果を補正できる。

3.5.4 制約を用いた Select 結果の補正

再び「$Pv_a \rightarrow Pv_b, Pv_c$」という制約 C があり、いま **Select** の対象としているテスト結果集合におけるテストの失敗を引き起こしている不具合誘発パラメータ組の一つに、ただ一つのパラメータ値からなる F：$\{Pv_b\}$ があるとする。このとき Pv_b を含むテストケースは全て失敗することから Pv_b の失敗率は 100%になるが、制約 C の

作用により Pv_a を含むテストケースには常に Pv_b が含まれるため、Pv_a を含むテストケースは常に失敗することになり、Pv_a の失敗率も 100%となる。こうして Pv_a と Pv_b の失敗率が同時に 100%となったとき、Pv_a の失敗率が高いのはそれが不具合誘発パラメータ組の構成要素であるからではなく、制約 C の作用であることが判断できる。

失敗率が 100%であるパラメータ値が複数見つかった時、失敗率スコアを算出する前に条件制約リストを順に調べ、条件制約式の左辺の要素が失敗率 100%であり右辺の要素にも失敗率 100%のものが含まれるような条件制約式が見つかれば、その左辺の要素を **Select** で使用する順位に含めないようにすることで結果を補正できる。

4. SSCC 法の適用例

本章では、制約への対応を含めた改良後の SSCC 法の動作へのより具体性ある理解を導くため、実際のシステムを想定した組み合わせテスト結果を用いた SSCC 法による FIL の様子を示す。

4.1 適用対象

手法の適用は、表 4.1 に示す SUT モデルで表される「SPIN Verifier」(SPINV) の 3-way テスト結果を対象として行った。SPIN(Simple Promela INterpreter) はオートマトンに基づくモデル検査ツールで、検査ツールとしてもシミュレータとしても使用でき、検査ツールとして使用する際の SUT モデルが SPINV となる。SPINV は 55 のパラメータを持ち、そのうち 42 のパラメータが 2 つの値を、2 つのパラメータが 3 つの値を、11 のパラメータが 4 つの値を持つ。またパラメータ間には 49 の非両立制約があり、それによって 4 つのスーパーパラメータと 41 の条件制約式が得られる (表 4.2)。3-way テストのテストスイートは組み合わせテスト生成ツール PICT によって生成され、332 のテストケースからなる。

テスト対象の SPINV には不具合があり、不具合を誘発するパラメータ組として

$$F_1 = \{p_{16}.2, p_{25}.2, p_{35}.2\} \tag{4.1}$$

$$F_2 = \{p_5.1, p_{41}.2, p_{53}.2\} \tag{4.2}$$

$$F_3 = \{p_7.2, p_{23}.1, p_{35}.2\} \tag{4.3}$$

$$F_4 = \{p_{16}.1, p_{40}.1\} \tag{4.4}$$

の 4 つが存在しており、このうちいずれか 1 組でも含むテストケースはテストが失敗する。テストの結果、332 のテストケースのうち 147 のテストケースが失敗した。

4.2 適用動作例

SSCC 法にこのテスト結果集合と条件制約式のリストを入力する。まず、スーパーパラメータが 4 種類あることがわかるので、テスト結果集合中の $p_{10}.1, p_{16}.1$ を S_1 に、

表 4.1　SPINV の SUT モデル

システム名	パラメータ数	モデルサイズ	制約サイズ
SPINV	55	$2^{42}3^{2}4^{11}$	$2^{47}3^{2}$

表 4.2　SPINV の条件制約式リスト

No.	スーパーパラメータ / 条件制約式	No.	条件制約式	No.	条件制約式
S_1	$p_{10}.1, p_{16}.1$	12	$p_{13}.2 \rightarrow p_{12}.1$	27	$p_{45}.4 \rightarrow p_{46}.1$
S_2	$p_{10}.2, p_{16}.2$	13	$p_{14}.2 \rightarrow p_{15}.1, p_{11}.1, p_{19}.1$	28	$p_{46}.2 \rightarrow p_{45}.1$
S_3	$p_{17}.1, p_{18}.1$	14	$p_{15}.1 \rightarrow p_{11}.1, p_{19}.1$	29	$p_{46}.3 \rightarrow p_{45}.1$
S_4	$p_{17}.2, p_{18}.2$	15	$p_{15}.2 \rightarrow p_{14}.1$	30	$p_{46}.4 \rightarrow p_{45}.1$
1	$p_2.2 \rightarrow p_{43}.1, p_{44}.1$	16	$p_{19}.2 \rightarrow p_{15}.2, p_{14}.1$	31	$p_{47}.2 \rightarrow p_7.1$
2	$p_3.2 \rightarrow p_4.1, p_5.1, p_6.1$	17	$p_{21}.2 \rightarrow p_{13}.2, p_{12}.1$	32	$p_{47}.3 \rightarrow p_7.1$
3	$p_4.2 \rightarrow p_3.1, p_5.1, p_6.1$	18	$p_{22}.2 \rightarrow p_{13}.2, p_{12}.1$	33	$p_{47}.4 \rightarrow p_7.1$
4	$p_5.2 \rightarrow p_3.1, p_4.1, p_6.1$	19	$p_{23}.2 \rightarrow p_{13}.2, p_{12}.1$	34	$p_{48}.1 \rightarrow p_{24}.1$
5	$p_6.2 \rightarrow p_3.1, p_4.1, p_5.1$	20	$p_{26}.2 \rightarrow p_{13}.2, p_{12}.1$	35	$p_{50}.1 \rightarrow p_{49}.1, p_{25}.1$
6	$p_7.2 \rightarrow p_{47}.1, p_8.1$	21	$p_{43}.2 \rightarrow p_{44}.1, p_2.1$	36	$p_{S_1}, p_8.2 \rightarrow p_{11}.2, p_9.1, p_{15}.2, p_{14}.1$
7	$p_8.2 \rightarrow p_7.1$	22	$p_{43}.3 \rightarrow p_{44}.1, p_2.1$	37	$p_{S_1}, p_{11}.1 \rightarrow p_8.1$
8	$p_9.2 \rightarrow S_1, p_{11}.1, p_8.1$	23	$p_{44}.2 \rightarrow p_{43}.1, p_2.1$	38	$p_{11}.1, p_8.2 \rightarrow S2, p_9.1$
9	$p_{11}.2 \rightarrow S_1, p_9.1, p_{15}.2, p_{14}.1$	24	$p_{44}.3 \rightarrow p_{43}.1, p_2.1$	39	$p_{13}.2, p_{22}.1 \rightarrow p_{27}.1$
10	$p_{12}.2 \rightarrow p_{13}.1, p_{22}.1, p_{23}.1, p_{21}.1, p_{26}.1$	25	$p_{45}.2 \rightarrow p_{46}.1$	40	$p_{13}.2, p_{27}.2 \rightarrow p_{22}.2, p_{12}.1$
11	$p_{13}.1 \rightarrow p_{22}.1, p_{23}.1, p_{21}.1, p_{26}.1$	26	$p_{45}.3 \rightarrow p_{46}.1$	41	$p_{22}.1, p_{27}.2 \rightarrow p_{13}.1, p_{23}.1, p_{21}.1, p_{26}.1$

$p_{10}.2, p_{16}.2$ を S_2 に、$p_{17}.1, p_{18}.1$ を S_3 に、$p_{17}.2, p_{18}.2$ を S_4 にそれぞれ変換しておく。(図)

一層目: T_0 において失敗したテストケースに共通するパラメータ値は存在しなかった。続けて各パラメータ値の失敗率を調べたところ 100%のものはなかったので、失敗率スコアの最も高いパラメータ値である $p_{40}.1$ を選択し、これを含むテストケースを T_0 から除外した T_1 を作成し、SSCC 関数に渡した。$p_{40}.1$ は F_4 の構成要素であるので、T_1 において F_4 が引き起こしたテストの失敗は存在しない。

二層目: T_0' では失敗したテストケースは 35 つだった。T_0' においても失敗したテストケースに共通するパラメータ値は存在しなかった。失敗率が 100%となるパラメータ値はなかったので、最も失敗率スコアの高い $p_{35}.2$ を選択し、これを含むテストケースを T_0' から除いた T_1' を SSCC 関数に渡した。$p_{35}.2$ は F_1 および F_3 の構成要素であるので、T_1' においてテストの失敗を引き起こしているのは F_2 だけである。

三層目: T_0'' では失敗したテストケースはわずか 2 つだった。T_0'' の失敗したテストケースに共通するパラメータを調べると、

$$p_2.1, p_4.1, p_5.1, S_1, p_{12}.1, p_{13}.2, S_4, p_{22}.2, p_{27}.2, p_{30}.1, p_{32}.2, p_{33}.2, p_{34}.2,$$

$$p_{35}.1, p_{37}.2, p_{38}.1, p_{40}.2, p_{41}.2, p_{43}.1, p_{44}.2, p_{43}.1, p_{44}.2, p_{46}.1, p_{53}.2 \tag{4.5}$$

の 24 のパラメータ値が共通していた。このうち、

$$p_{44}.2 \rightarrow p_2.1, p_{43}.1 \tag{4.6}$$

$$p_{22}.2 \rightarrow p_{13}.2, p_{12}.1 \tag{4.7}$$

という条件制約式の存在によって、$p_2.1, p_{13}.2, p_{12}.1, p_{43}.1$ の 4 つのパラメータ値は構成要素候補から除外される。残りを構成要素として最小組抽出を行うと、

$$\{S_1, p_{44}.2, p_{53}.2\}, \{p_{41}.2, p_{44}.2, p_{53}.2\}, \{p_5.1, p_{41}.2, p_{53}.2\} \tag{4.8}$$

の 3 つのパラメータ組が発見されたので、これらのセットを返却した。この 3 つの可能性のうち最後のパラメータ組は不具合誘発パラメータ組の 1 つである F_2 であるが、ここでは他の 2 つのパラメータ組と区別できない。

二層目: 三層目から返された 3 つのパラメータ組の選択肢を受け取り、それらのいずれか 1 組でも含まれるテストケースを T_0' から除外した T_2' を作成する。T_2' には

F_2 によって失敗したテストケースは存在しないことになる。T_2' のうち失敗したテストケースに共通しているパラメータ値として

$$p_9.1,\ p_{35}.2,\ p_{40}.2 \tag{4.9}$$

の 3 つが見つかった。これらを構成要素として最小組抽出を行ったところ、最小組抽出を満たす組み合わせは存在しなかった。なので固定パラメータではない $p_9.1,\ p_{35}.2,\ p_9.1,\ p_{35}.2,\ p_{40}.2$ の 3 つを重複要素とし、$Search_1$ より後の処理を繰り返す。$p_35.2$ は実際に F_1 と F_3 の重複要素だが $p_9.1$ はたまたま失敗したテストケースに共通しているだけのパラメータ値である。

T_2' において重複要素である $p_9.1,\ p_{35}.2$ を除いたパラメータ値の失敗率スコアを算出し最も高いパラメータ値である $p_7.2$ を含むテストケースを T_2' から除外した T_3' を作成し、$SSCC$ 関数に渡す。$p_7.2$ は F_3 の構成要素であるので、T_3' には F_1 が引き起こすテストの失敗しか存在しないことになる。

三層目: 再び三層目、T_0'' 中に失敗したテストケースは 17 ある。T_0'' の失敗したテストケースに共通するパラメータを調べると、

$$p_3.1,\ p_7.1,\ p_9.1,\ p_{11}.1,\ S_2,\ p_{25}.2,\ p_{35}.2,\ p_{40}.2 \tag{4.10}$$

の 8 つが共通していた。これらを構成要素として最小組抽出を行ったところ、ただ一つのパラメータ組 $\{S_2, p_{25}.2, p_{35}.2\}$ が見つかったのでこれを返却する。$\{S_2, p_{25}.2, p_{35}.2\}$ は不具合誘発パラメータ組 F_1 の $p_{16}.2$ をスーパーパラメータ S_2 に置き換えたものであることが確認できる。

二層目: 三層目からの返却と前回の返却に加え、現在保有している不具合誘発パラメータ組の候補は

$$\{\{S_1, p_{44}.2, p_{53}.2\}, \{p_{41}.2, p_{44}.2, p_{53}.2\}, \{p_5.1, p_{41}.2, p_{53}.2\}\}, \{S_2, p_{25}.2, p_{35}.2\} \tag{4.11}$$

という、3 つの組み合わせの可能性と、1 つの組み合わせである。これらの組み合わせいずれかでも含むテストケースを T_0' から除外した T_4' を作成し、T_4' の失敗したテストケースに共通するパラメータ値を調べると、

$$p_3.1,\ p_4.1,\ p_7.2,\ p_8.1,\ p_{23}.1,\ p_{35}.2,\ p_{47}.1 \tag{4.12}$$

の 7 つのパラメータ値が共通していた。ただし

$$p_7.2 \rightarrow p_{47}.1, p_8.1 \tag{4.13}$$

という条件制約式の存在により $p_{47}.1, p_8.1$ は除外されるので、残りの

$$p_3.1, p_4.1, p_7.2, p_{23}.1, p_{35}.2 \tag{4.14}$$

の 5 つのパラメータ値を構成要素として最小組抽出を行ったところ、$\{p_7.2, p_{23}.1, p_{35}.2\}$ というただ一つのパラメータ組を発見した。従ってこれまで発見済みの不具合誘発パラメータ組候補にこれを加え返却した。ここで $\{p_7.2, p_{23}.1, p_{35}.2\}$ は実際の不具合誘発パラメータ組 F_3 である。

一層目: 二層目から返却された発見済み不具合誘発パラメータ組候補 R は、

$$\{\{S_1, p_{44}.2, p_{53}.2\}, \{p_{41}.2, p_{44}.2, p_{53}.2\}, \{p_5.1, p_{41}.2, p_{53}.2\}\},$$

$$\{S_2, p_{25}.2, p_{35}.2\}, \{p_7.2, p_{23}.1, p_{35}.2\} \tag{4.15}$$

という、3 つの組み合わせの可能性と、2 つの組み合わせである。これらの組み合わせのうちいずれかでも含むテストケースを T_0 から除外した T_2 を作成し、T_2 の失敗したテストケースに共通するパラメータ値を調べると、

$$S_1, p_{40}.1 \tag{4.16}$$

の 2 つのパラメータ値が共通していた。補正条件を満たす条件制約式が無いことを確認後、これらのパラメータ値を構成要素として最小組抽出を行うと、$\{S_1, p_{40}.1\}$ というただ一つのパラメータ組を発見したので、これを R に加えて返却した。ここで、$\{S_1, p_{40}.1\}$ は実際の不具合誘発パラメータ組 F_4 である。

一層目からの最終的な返却は、

$$\{\{S_1, p_{44}.2, p_{53}.2\}, \{p_{41}.2, p_{44}.2, p_{53}.2\}, \{p_5.1, p_{41}.2, p_{53}.2\}\},$$

$$\{S_2, p_{25}.2, p_{35}.2\}, \{p_7.2, p_{23}.1, p_{35}.2\}, \{S_1, p_{40}.1\} \tag{4.17}$$

であり、すなわち対象のソフトウェアに存在する不具合誘発パラメータ組のセットとして

$$\{S_1, p_{44}.2, p_{53}.2\}, \{S_2, p_{25}.2, p_{35}.2\}, \{p_7.2, p_{23}.1, p_{35}.2\}, \{S_1, p_{40}.1\} \tag{4.18}$$

$$\{p_{41}.2, p_{44}.2, p_{53}.2\}, \{S_2, p_{25}.2, p_{35}.2\}, \{p_7.2, p_{23}.1, p_{35}.2\}, \{S_1, p_{40}.1\} \tag{4.19}$$

$$\{p_5.1, p_{41}.2, p_{53}.2\}, \{S_2, p_{25}.2, p_{35}.2\}, \{p_7.2, p_{23}.1, p_{35}.2\}, \{S_1, p_{40}.1\} \tag{4.20}$$

という 3 通りの可能性が存在するが、これらに対して正解検査を行ったところ、

$$\{p_5.1, p_{41}.2, p_{53}.2\}, \{S_2, p_{25}.2, p_{35}.2\}, \{p_7.2, p_{23}.1, p_{35}.2\}, \{S_1, p_{40}.1\} \tag{4.21}$$

の 1 組だけが正解検査に合格した。結果として、SSCC 法は唯一の解を提示することに成功した。厳密に言えばこの解はスーパーパラメータ $S_1 = \{p_{10}.1, p_{16}.1\}, S_2 = \{p_{10}.2, p_{16}.2\}$ を含むので、実際に存在している不具合誘発パラメータ組のセットとしては

$$\{p_5.1, p_{41}.2, p_{53}.2\}, \{P_{10}.2, p_{25}.2, p_{35}.2\}, \{p_7.2, p_{23}.1, p_{35}.2\}, \{p_{10}.1, p_{40}.1\} \tag{4.22}$$

$$\{p_5.1, p_{41}.2, p_{53}.2\}, \{P_{10}.2, p_{25}.2, p_{35}.2\}, \{p_7.2, p_{23}.1, p_{35}.2\}, \{p_{16}.1, p_{40}.1\} \tag{4.23}$$

$$\{p_5.1, p_{41}.2, p_{53}.2\}, \{P_{16}.2, p_{25}.2, p_{35}.2\}, \{p_7.2, p_{23}.1, p_{35}.2\}, \{p_{10}.1, p_{40}.1\} \tag{4.24}$$

$$\{p_5.1, p_{41}.2, p_{53}.2\}, \{P_{16}.2, p_{25}.2, p_{35}.2\}, \{p_7.2, p_{23}.1, p_{35}.2\}, \{p_{16}.1, p_{40}.1\} \tag{4.25}$$

という 4 通りの可能性があるが、これらを区別する手段は無い。そして、実際に存在している不具合誘発パラメータ組のセット F_1, F_2, F_3, F_4 と SSCC 法による特定結果は一致しており、SSCC 法による FIL 結果の正しさが示された。

5. 実証実験

本章では、実証実験を行い提案手法の有効性を評価する。

5.1 研究設問

評価したい事柄を明確にするため、以下のように研究設問を設定した。

RQ1 提案手法が FIL にかかる時間はどのくらいか？

RQ2 提案手法が正しい不具合誘発パラメータ組を特定する精度はどのくらいか？

RQ3 提案手法は従来手法と比べて優れているのか？

従って、これらの研究設問に回答するためには提案手法による FIL にかかる処理時間と精度を計測し、従来手法と比較すればよいことがわかる。

5.2 実験準備

5.2.1 実験対象

4 章の手法適用例で例示したように、FIL を行う対象はあるシステムにおける組み合わせテストの結果である。本実験では実在する規模の異なる 5 つの OSS（オープンソースソフトウェア）の SUT モデルから生成された t-way テストと、ランダムに決定された不具合誘発パラメータ組によって得られた組み合わせテストの結果を対象として FIL を行う。

使用するシステム名とその SUT モデルを表 5.1 に示す。TCAS (注 7) と SPINS (注 8) は小規模なシステムを代表し、TCAS は 12 のパラメータを、SPINS は 18 のパラメータを持つ。また SPINV と Bugzilla (注 9) は中規模なシステムを代表し、SPINV は 55 のパラメータを、Bugzilla は 52 のパラメータを持つ。さらに GCC (注 10) は大規模なシステムを代表し、199 のパラメータを持つ。TCAS 以外の SUT モデルにはパラメータ間の制約が存在し、SPINV のみがスーパーパラメータを持つ。またそれぞれの SUT モ

(注 7)：Traffic Collision Avoid System、航空機衝突防止システム
(注 8)：SPIN Simulator、前述の SPIN をシミュレータとして使用する場合
(注 9)：Mozilla 製のバグ管理ツール
(注 10)：GNU Compilar Collection、コンパイラ用のツールセット

デルが持つ因子数 2 および因子数 3 のパラメータ組の総数を 2-tuple 数および 3-tuple 数として示す。これらのシステムモデルから生成された t-way テストの持つテストケースの数を表 5.2 に示す。実験には TCAS と SPINS は 2∼5-way テスト、SPINV と Bugzilla は 2∼4-way テスト、GCC は 2∼3-way テストを使用する。すなわち、実験には 16 種類のテストスイートが使用される。t-way テストの生成には Microsoft 社の組み合わせテスト生成ツールである PICT を使用した。

5.2.2 実験内容

実験は次の手順で行われた。

まず、それぞれのシステムモデルにおいて不具合を誘発するパラメータ組を設定し、その不具合誘発パラメータ組に対して生成された t-way テストスイートを実行させた結果を得る。得られた組み合わせテストの結果に対し SSCC 法による FIL を実施し、その処理時間を計測する。また FIL によって出力されたパラメータ組が設定した不具合誘発パラメータ組と一致しているかを確かめる。実験において用いる SSCC 法は 3.4 節で述べた改良型に 3.5 節で述べた制約への対応を有効化させたものを使用する。

設定する不具合誘発パラメータ組は、まずその個数が 1 個から 4 個の間でランダムに n 個決定され、次にそれぞれの組み合わせ因子数を 2 から t の間でランダムに決定し、その数だけランダムにパラメータ値を選んで組み合わせることで n 個のパラメータ組を決定する。ただし組み合わせ因子数の上限 t は対象とする t-way テストの網羅因子数 t である。また各不具合誘発パラメータ組は、それ以外の不具合誘発パラメータ組を含まないようなテストケースが少なくとも 1 つ存在するように選ばれる。

用意した 16 種類のテストスイートについて、1 つのテストスイートに対しランダムな不具合誘発パラメータ組を設定し、SSCC 法によって FIL を行うことを最大で 10,000 回繰り返す。一回あたりの FIL にかかった時間と FIL の出力結果を記録し、処理時間の統計量と、設定した不具合誘発パラメータ組と FIL 出力結果との一致度ごとの割合を求める。設定した不具合誘発パラメータ組と FIL 出力結果の一致度は次のように分類される。

表 5.1　実験対象システムの SUT モデル

SUT 名	パラメータ数	モデルサイズ	制約サイズ	2-tuple 数	3-tuple 数
TCAS	12	$2^6 4^3 5^1 10^2$	–	879	9,911
SPINS	18	$2^{13} 4^5$	2^{13}	979	12,835
SPINV	55	$2^{42} 3^2 4^{11}$	$2^{47} 3^2$	8,741	369,976
Bugzilla	52	$2^{49} 3^1 4^2$	$2^4 3^1$	5,818	202,683
GCC	199	$2^{189} 3^{10}$	$2^{37} 3^3$	82,770	11,131,894

表 5.2　実験対象テストスイートに含まれるテストケース数

SUT 名	2-way テスト	3-way テスト	4-way テスト	5-way テスト
TCAS	100	402	1493	4795
SPINS	28	116	420	1432
SPINV	67	332	1796	-
Bugzilla	19	72	229	-
GCC	31	137	-	-

A **完全一致**

設定された不具合誘発パラメータ組と FIL 出力結果が完全に一致した場合。スーパーパラメータとして完全に一致した場合も含む。

B **合体完全一致**

設定された不具合誘発パラメータ組のペアがより少ない個数でテストの失敗を説明できることが明らかであり、その説明可能な最小個数のパラメータ組と FIL の出力結果が完全に一致した場合。

C **可能性一致**

FIL の出力結果が唯一のものでなく複数の可能性として表されるものであり、その中に設定不具合と完全一致あるいは合体完全一致なものがある場合。

D **代替一致**

部分的に一致しないが、FIL 出力における設定不具合と一致していない組み合わせが設定したものより少ない個数あるいは少ない因子数で失敗を説明出来る場合。また FIL 結果が複数の可能性を表しその中に上記のものが存在する場合。

E **包含一致**

FIL の出力結果の組み合わせセットが設定した不具合誘発パラメータ組み合わせのセットを包含し、かつ余分な組み合わせが存在する場合。また FIL 結果が複数の可能性を表しその中に上記のものが存在する場合。

F **部分一致**

完全には一致しないが部分的に一致した場合。また FIL 結果が複数の可能性を表しその中に上記のものが存在する場合。

G **不一致**

全く一致しなかった場合。

Z **失敗終了**

FIL が組み合わせを特定できずに終了した場合。

実験は、Macbook Pro(Retina 13-inch, Late 2013)、2.8 GHz Intel Core i7、16 GB 1600 MHz DDR3 で Python 3.5.1 を用いて行った.

5.2.3 比較対象

比較する従来手法には、提案手法と同じ事後分析型 FIL 手法であり全てのテスト結果を用いて厳密な特定を行う Ghandihari らの手法「BEN」(注 11) と Cheng らの手法「comFIL」を用いる。提案手法である SSCC 法では処理時間と特定精度（設定した不具合誘発パラメータ組と出力結果の一致度）を測定したが、BEN および comFIL は厳密に全ての不具合誘発パラメータ組を特定する手法であるため特定精度は必ず 100%となることが分かっているので、処理時間の測定のみを行う。

BEN と comFIL の処理はどちらも「不具合を誘発している可能性のあるパラメータ組を求める」という前段階と「求めたパラメータ組それぞれについて有効な追加テストを作成し実行させることで実際の不具合誘発パラメータ組かどうかを判断する」という後段階の処理からなっている (注 12)。またテストの大規模化による組み合わせ数の膨大化によって処理時間が指数関数的に増大するのは母数の大きい前段階の処理であることが予想される。実験の簡単化のため、本実験では BEN と comFIL の二つの従来手法について前段階の「不具合を誘発している可能性のあるパラメータ組を求める」という処理のみを行い、それにかかった処理時間と、求まった不具合誘発の可能性のあるパラメータ組の個数を記録する。これにより、もしも前段階処理での処理時間が提案手法を上回ればその時点で提案手法の時間効率性の優位が示される。また求められた不具合誘発の可能性を持つパラメータ組の数は追加テストの実行回数に直結するため処理時間全体の目安にもなる。

BEN と comFIL における前段階の処理は同じ方法によって行えるため、計測は一度のみ行う。提案手法の評価実験と同じ方法で不具合誘発パラメータ組を設定し、全ての失敗したテストケースに存在する因子数 t 以下の全ての組み合わせを検査して不具合誘発の可能性を持つパラメータ組を求め、候補として残った組み合わせ数と検査対象となった全ての組み合わせ数および処理時間を計測する。1 つのテストスイートに対してこの計測を 100 回繰り返し、各組み合わせ数と処理時間について統計量を求める。また実験環境は提案手法のものと同一である。

(注 11)：Ghandihari らは論文内ではこの FIL 手法に名付けを行わなかったが、後にこの手法を利用した FIL ツールを BEN と名付けて公開している。
(注 12)：これはちょうど 2.3 節で示した例題モデルに対する FIL と同様の手順である。

5.3 実験結果

従来手法について、表 5.3 に処理時間の測定結果を、表 5.4 に検査対象となった組み合わせ数と検査の結果不具合誘発の可能性があると判定された組み合わせの数を示す。表 5.3 では処理時間の統計量のほか、一定時間を超過した回数の割合も記載している。表 5.4 の出力された組み合わせ数が従来手法の後段階で追加テストによって確かめられる組み合わせの個数であり、BEN では必ずその回数、comFIL では最大でその回数だけ追加のテストケース作成と実行が行われる。「推定値」カラムにチェックの入っている TCAS の 5-way テスト、SPINS の 5-way テスト、SPINV の 4-way テスト、Bugzilla の 4-way テスト、GCC の 3-way テストは、1 回あたりの FIL にかかる時間が長く十分な回数を実測することが困難であったため、まず検査すべき組み合わせの数を求め、次にその中から 1000 個を検査し、検査にかかった時間の 1000 分の 1 の時間を検査すべき組み合わせの数に掛けた時間を総処理時間の推定値として求めた。そのためこれらについては出力された組み合わせ数は分からない。

BEN や comFIL のアルゴリズムでは前述のようにテストスイートの網羅組み合わせ因子数が上がると前段階処理で検査すべき組み合わせ数が爆発的に増え、それに伴って処理時間も爆発的に長くなっていることが確認できる。例えば TCAS や SPINS の 5-way テストを対象とした FIL には平均で 10 分程度の時間を要している。またソフトウェアの持つテストパラメータ数が多くなるにつれて網羅因子数を上げたときの処理時間の増加が激しくなることも確認できる。

表 5.5 に、提案手法である SSCC 法の処理時間に関する測定結果を示す。測定は 1 つのテストスイートに対して 10,000 回を基本として行ったが、SPINV の 4-way テスト、Bugzilla の 4-way テスト、GCC の 3-way テストでは FIL に 100 秒を超えるような長時間を要した頻度が比較的多く、十分な回数の実測を行うことに支障が生じたため、SPINV の 4-way テストでは 5,000 回、Bugzilla の 4-way テストおよび GCC の 3-way テストでは 1,000 回分のデータを使用した。

図 5.1 および図 5.2 に、それぞれのテストスイートについて、従来手法適用時の処理時間と SSCC 法適用時の処理時間を比較した箱ひげ図を示す。

また表 5.6 と図 5.3 に、SSCC 法を用いた FIL の特定精度に関する測定結果を示す。

表 5.3　実験結果：従来手法（前段階部分）の処理時間

SUT	t-way	テストケース	サンプル数	推定値	処理時間 (s)				超過割合 (%)		
					Ave.	Med.	Max.	Min.	≥1s	≥10s	≥100s
TCAS	2	100	100		0.056	0.055	0.101	0.008	0%	0%	0%
	3	402	100		2.210	2.129	4.820	0.195	85%	0%	0%
	4	1,493	100		45.300	40.740	172.910	3.446	100%	91%	3%
	5	4,795	100	✓	641.800	654.060	1,565.930	10.371	100%	99%	94%
SPINS	2	28	100		0.025	0.023	0.590	0.004	0%	0%	0%
	3	116	100		1.034	0.899	3.479	0.068	45%	0%	0%
	4	420	100		29.591	26.559	99.894	3.875	100%	93%	0%
	5	1,432	100	✓	589.597	591.969	1,221.157	94.670	100%	100%	99%
SPINV	2	67	100		0.455	0.366	5.030	0.067	4%	0%	0%
	3	332	100		119.365	69.956	2,280.582	15.234	100%	100%	20%
	4	1,796	100	✓	173,116.244	33,252.984	816,870.136	2,905.147	100%	100%	100%
Bugzilla	2	19	100		0.557	0.292	2.630	0.067	18%	0%	0%
	3	72	100		27.305	16.700	204.563	4.046	100%	89%	4%
	4	229	100	✓	1,683.623	1,659.557	3,455.088	381.928	100%	100%	100%
GCC	2	31	100		21.637	4.159	209.261	1.210	100%	37%	6%
	3	137	100	✓	11,490.310	6,661.501	142,756.702	754.410	100%	100%	100%

表 5.4　実験結果：従来手法の扱う組み合わせ量

SUT	t-way	テストケース	サンプル数	推定値	検査した全ての組み合わせ (個)				出力された組み合わせ (個)			
					Ave.	Med.	Max.	Min.	Ave.	Med.	Max.	Min.
TCAS	2	100	100		607.3	633	853	78	62.2	51	185	3
	3	402	100		6,304.5	6,721	9,698	740	414.2	360	1,304	16
	4	1,493	100		39,109.1	40,468	74,636	4,007	1,693.9	1,244	6,909	42
	5	4,795	100	✓	173,671.9	180,237	370,168	1,585	-	-	-	-
SPINS	2	28	100		757.9	802	980	171	96.4	86	269	6
	3	116	100		9,253.0	10,270	13,103	987	406.7	350	1,302	8
	4	420	100		86,167.4	96,389	124,167	15,325	1,670.9	1,262	6,469	22
	5	1,432	100	✓	500,754.2	509,812	841,860	85,470	-	-	-	-
SPINV	2	67	100		7,507.3	8198	8,858	1,540	326.6	199	2,071	11
	3	332	100		331,160.0	356,861	378,352	88,704	3,048.5	1,394	30,267	20
	4	1,796	100	✓	10,316,140.7	11,143,890	11,779,433	2,446,996	-	-	-	-
Bugzilla	2	19	100		5,062.4	5,308	5,899	2,260	554.6	462	1,486	17
	3	72	100		181,742.3	194,437	206,936	60,171	2,564.4	1,888	11,405	21
	4	229	100	✓	4,772,650.2	5,053,085	5,377,584	2,016,457	-	-	-	-
GCC	2	31	100		78,106.9	80,935	83,062	41,454	3,448.3	1,833	15,535	44
	3	137	100	✓	10,577,633.0	11,113,053	11,209,469	4,921,343	-	-	-	-

表 5.5　実験結果：SSCC 法の処理時間

SUT	t-way	テストケース	サンプル数	処理時間 (s)				超過割合 (%)		
				Ave	Med	Max	Min	≥1s	≥10s	≥100s
TCAS	2	100	10,000	0.014	0.009	0.321	0.001	0%	0%	0%
	3	402	10,000	0.042	0.023	0.717	0.001	0%	0%	0%
	4	1,493	10,000	0.186	0.086	4.904	0.009	4.41%	0%	0%
	5	4,795	10,000	0.802	0.366	27.611	0.015	9.10%	2.08%	0%
SPINS	2	28	10,000	0.022	0.010	2.272	0.000	0.05%	0%	0%
	3	116	10,000	0.068	0.016	8.980	0.001	0.22%	0%	0%
	4	420	10,000	0.296	0.054	58.620	0.002	7.09%	0.01%	0%
	5	1,432	10,000	1.055	0.204	86.976	0.004	11.31%	2.29%	0%
SPINV	2	67	10,000	0.596	0.042	538.540	0.001	4.23%	0.81%	0.08%
	3	332	10,000	1.514	0.472	2,877.334	0.002	10.59%	1.12%	0.09%
	4	1,796	5,000	7.268	0.228	5,053.825	0.006	9.48%	4.06%	1.72%
Bugzilla	2	19	10,000	0.049	0.016	1.132	0.000	0.04%	0%	0%
	3	72	10,000	0.900	0.085	331.971	0.001	22.74%	0.34%	0.03%
	4	229	1,000	14.883	0.043	659.240	0.002	29.73%	15.98%	3.99%
GCC	2	31	10,000	0.781	0.147	182.984	0.002	27.40%	0.74%	0.02%
	3	137	1,000	79.683	0.498	1,239.769	0.003	46.60%	35.40%	23.40%

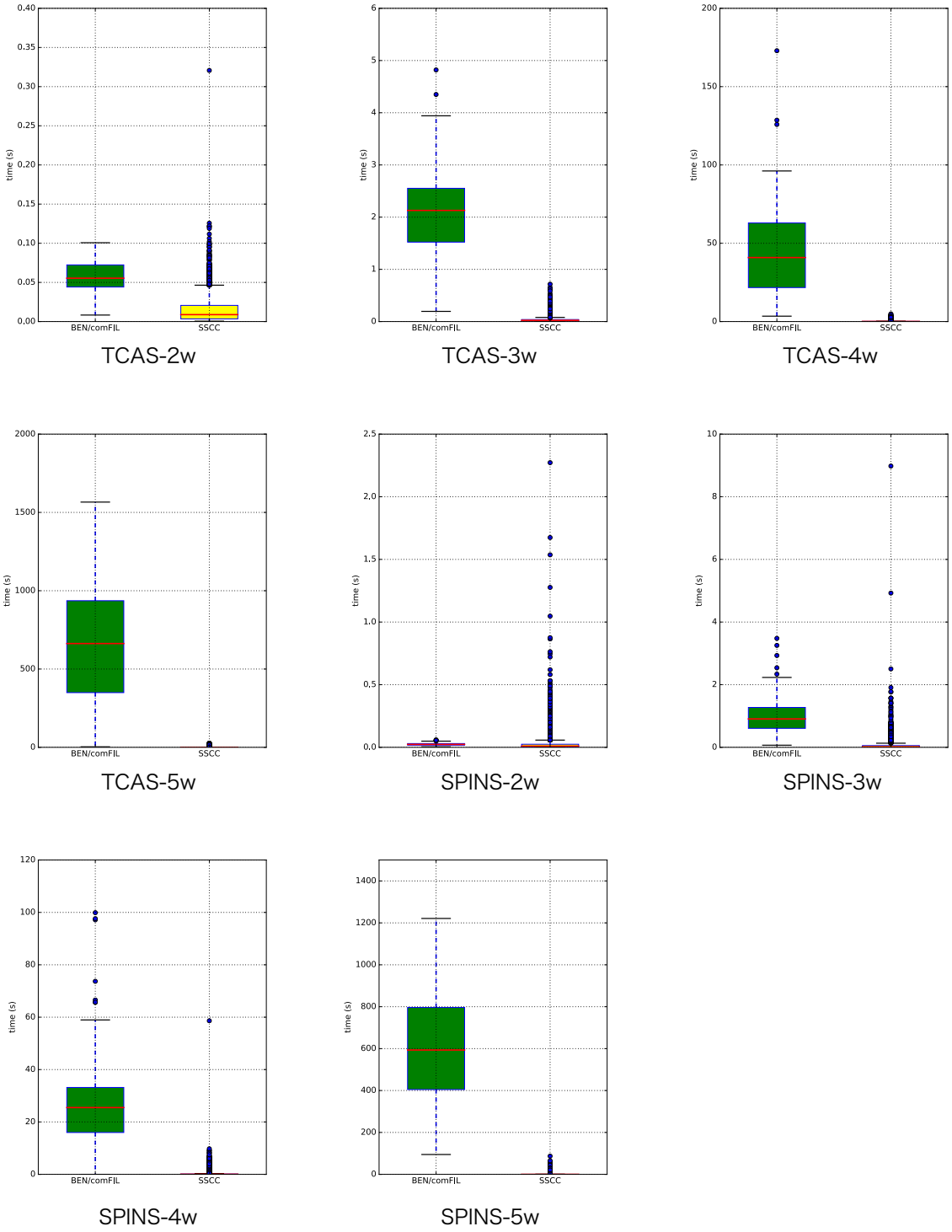

図 5.1　TCAS, SPINS の FIL 処理時間分布比較

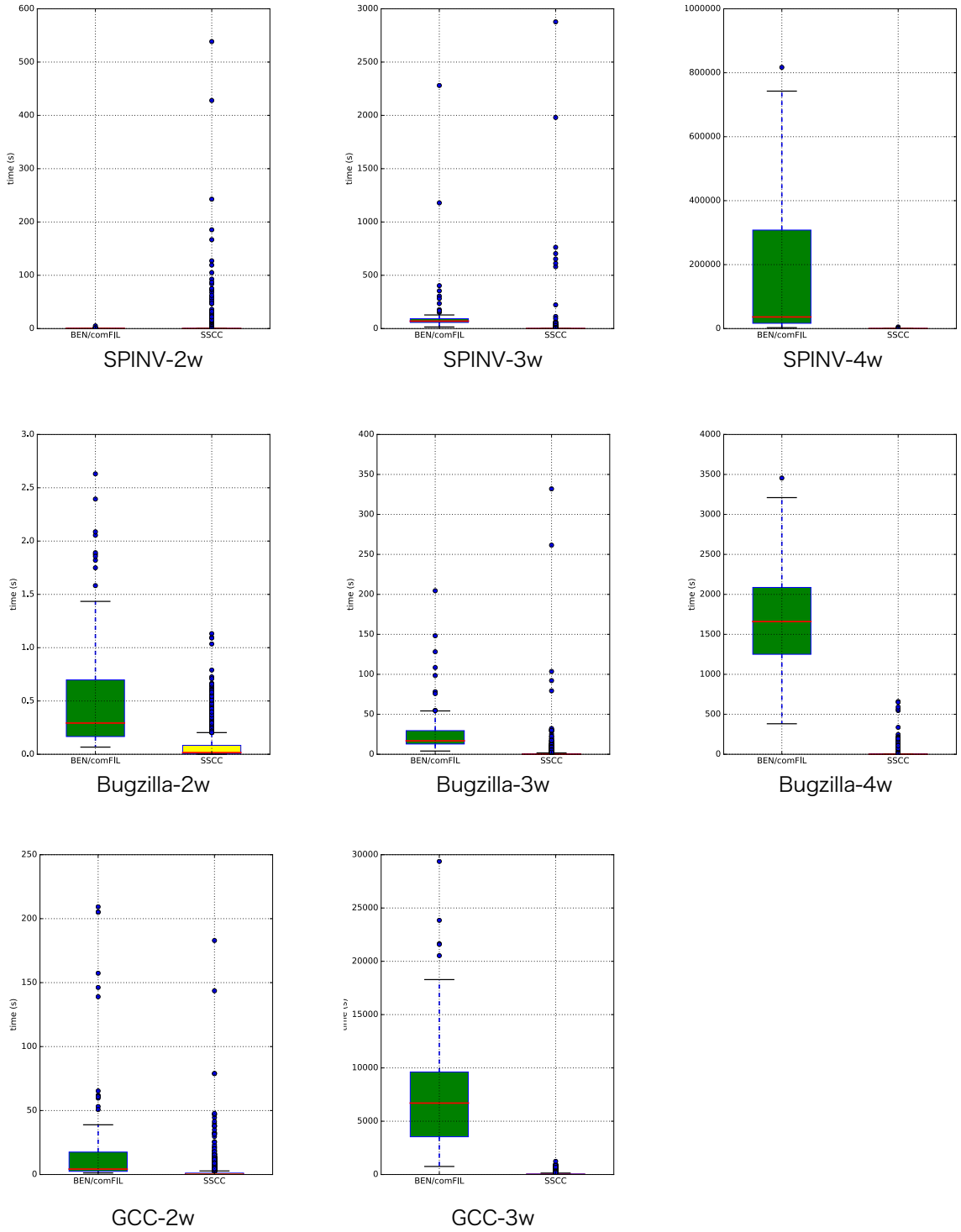

図 5.2　SPINV, Bugzilla, GCC の FIL 処理時間分布比較

表 5.6　実験結果：SSCC 法の出力結果の一致度割合

SUT	t-way	テストケース	サンプル数	一致度							
				A	B	C	D	E	F	G	Z
TCAS	2	100	10000	0.7705	0.0066	0.2042	0.0086	0.0001	0.0027	0.0001	0.0072
	3	402	10000	0.9755	-	0.0245	-	-	-	-	-
	4	1493	10000	0.9277	0.0004	0.0499	0.0194	0.0005	0.0021	-	-
	5	4795	10000	0.9378	0.0004	0.0366	0.0221	-	0.0031	-	-
SPINS	2	28	10000	0.2695	0.0137	0.3558	0.1108	0.0005	0.0030	0.0468	0.1999
	3	116	10000	0.5935	0.0002	0.2354	0.0927	0.0002	0.0553	0.0045	0.0182
	4	420	10000	0.7458	0.0003	0.0809	0.1432	0.0019	0.0255	0.0021	0.0003
	5	1432	10000	0.7498	0.0004	0.0326	0.2050	0.0004	0.0107	0.0010	0.0001
SPINV	2	67	10000	0.4603	0.0029	0.1673	0.0560	0.0043	0.0961	0.0233	0.1898
	3	332	10000	0.8076	0.0004	0.0414	0.0553	0.0078	0.0701	0.0035	0.0139
	4	1796	5000	0.8674	0.0002	0.0104	0.0881	0.0047	0.0266	0.0010	0.0016
Bugzilla	2	19	10000	0.1490	0.0276	0.2280	0.2410	-	0.0004	0.0494	0.3046
	3	72	10000	0.4940	0.0005	0.1193	0.0210	0.0029	0.0533	0.0023	0.3067
	4	229	1000	0.8350	0.0010	0.0670	0.0210	0.0090	0.0150	-	0.0520
GCC	2	31	10000	0.2230	0.0110	0.1380	0.1730	-	0.0020	0.0630	0.3900
	3	137	1000	0.6420	-	0.0290	0.0008	0.0001	0.0140	-	0.3141
平均				0.6530	0.0041	0.1138	0.078	0.0020	0.0237	0.0123	0.1124

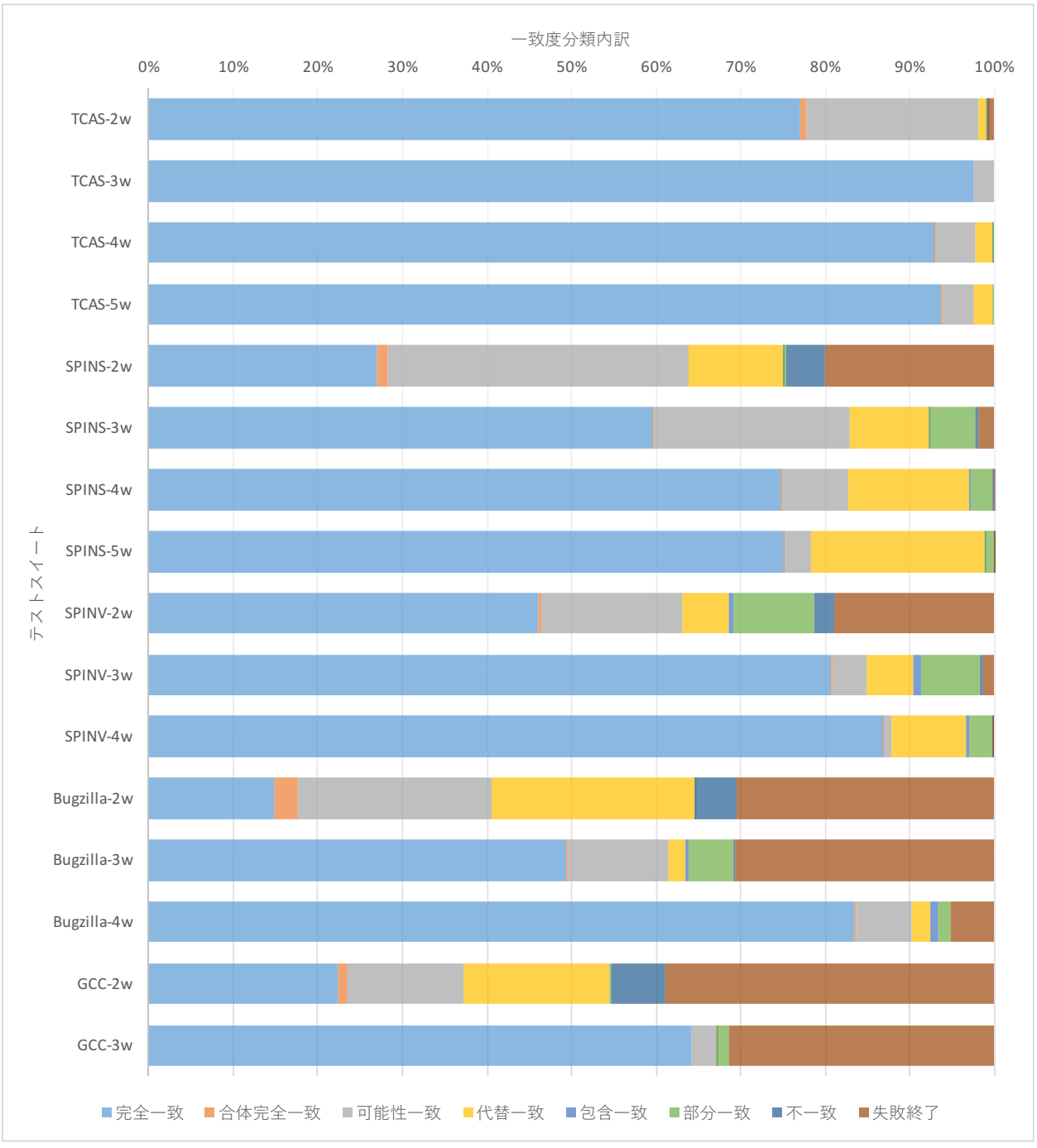

図 5.3　SSCC 法による FIL 結果の一致度分類内訳

6. 考察

6.1 研究設問への回答

研究設問に沿って実験結果に対する考察を示す。

RQ1 提案手法が FIL にかかる時間はどのくらいか？

提案手法の処理時間が示された表 5.5 と図 5.1、図 5.2 の提案手法部分を見ていく。まず注目するのは、特に網羅因子数の大きいテストスイートを対象とした場合に、処理時間の平均値と中央値が大きく乖離していることである。これはグラフを見ても分かるように、大部分の値が同程度の値域に集中している一方で少数の極端に大きく外れた外れ値が存在していることを示している。このためテストスイート毎の議論には平均値ではなく中央値を用いる。

各テストスイートの処理時間の中央値を見ると、全てのテストスイートで 1 秒未満であることが確認できる。また 1 秒以上の処理時間で完了した試行の割合は大部分のテストスイートで 1 割以下、最も多い GCC の 3-way テストでも 5 割以下の割合と低い。このことは、提案手法がどのような規模の組み合わせテスト結果を対象とした場合でも多くの場合で 1 秒未満というごく短い処理時間で特定処理を完了できると見込めることを示している。

次に注目するのは度々発生する巨大な値を持つ外れ値の存在である。例えば SPINV の 2-way テストでは中央値の 0.042 秒に対して最大値が 538 秒、3-way テストでは中央値の 0.472 秒に対して最大値が 2,877 秒、4-way テストに至っては中央値の 0.228 秒に対して最大値が 5,053 秒といずれも極端に大きな値になっている。また外れ値自体はどのテストスイートにもあるが、こうした極端に巨大な値を持つのは SPINV、Bugzilla、GCC などのパラメータ数の多いシステムのテストスイートであることが分かる。

極端に長い処理時間を要した試行のログを見てみると、処理時間が極端に長くなるのには 2 種類の原因があることが分かった。1 つは SSCC 法のアルゴリズムにおける最小組抽出に時間がかかる場合で、これは GCC の 3-way テストや SPINV と Bugzilla の 4-way テストなど、そのテストスイートが扱う組み合わせの総数が多い実験対象が該当する。最小組抽出の仕様として、最小組抽出を行う段階で失敗して

いるテストケースが 1 つしかない場合にはそのテストケースに含まれる全てのパラメータ値が最小組抽出の構成要素となるため、実質的に従来手法と同じ処理を行うことになる。例えば GCC の 3-way テストに従来手法を適用した場合の最小値は推定値で 754 秒なので、SSCC 法の適用に際しても最悪の場合これと同程度の処理時間が必要となることは不思議ではない。また失敗したテストケースが 2 つや 3 つの場合でも、テストケースのパラメータ数が多ければ多いほどそこに偶然共通するパラメータ値の数は多くなり得るので、その場合にも組み合わせ数が爆発的に増加し処理時間が極端に長くなる。199 種類の入力パラメータを持つ GCC の 3-way テストの提案種法での処理時間が 23.4%もの割合で 100 秒を超えているのはこうしたことが原因であると思われる。

もう 1 つは SSCC 法のアルゴリズムにおける正解検査に時間がかかる場合である。SSCC 法では複数の選択肢の形で特定した不具合誘発パラメータ組を返却することがあるが、この選択肢の数が時として膨大になる場合があるため、その全てに対して正解検査を行う際に膨大な時間がかかってしまう。出力結果として複数の選択肢が発生するのは一意な特定に足るだけのテストケースの数が不足していることが理由なので、SPINV や Bugzilla の 3-way テストなどの、最悪の場合でも組み合わせがそこまで大きく爆発しないようなケースで極端に長い処理時間が発生している場合はこちらが原因だと考えられる。

一方で、TCAS はどの網羅因子数のテストスイートでも極端に膨大な処理時間が発生したケースはなく安定して短い時間で処理を終えられている。これはパラメータ数が 12 と小さいがゆえに最小組抽出にかかる最大処理時間が小さく、また各パラメータのとる値の数に偏りがあるため t-way テストの網羅に必要なテストケースが多くなり、膨大な選択肢を発生させないために正解検査にも時間がかからないことが理由であると考えられる。また SPINS に対してもこれに準じたことが言える。

RQ1 への回答をまとめると、提案手法は多くの場合で適用対象によらずごく短い時間で処理を終えることができる一方で、仕様上、稀に膨大な処理時間を要することもあり、その発生は適用対象の性質に依存するものであるということが言える。

RQ2 提案手法が正しい不具合誘発パラメータ組を特定する精度はどのくらいか？

提案手法の出力結果の一致度が示された表 5.6 と図 5.3 から図**??**にかけてを見て

いく。

事前に分類した一致度のうち、どこまでの一致度を正しい結果として評価するかは議論の分かれる部分である。そこで、完全一致のみを正しい結果とする**評価基準 1**、完全一致・合体完全一致・可能性一致を正しい結果とする**評価基準 2**、評価基準 2 に加えて代替一致を正しい結果とする**評価基準 3**、不一致と失敗終了を除くすべての一致度を正しい結果とする**評価基準 4** という 4 つの評価基準を設け、それぞれの評価基準で評価した場合について論じる。評価基準の理由はそれぞれ後述する。各評価基準で正しい結果となる出力の割合を表 6.1 と図 6.1 に示す。

評価基準 1 （完全一致のみ）

完全一致 (A) は設定した不具合誘発パラメータ組のセットと提案手法によって出力された結果が完全に一致した場合であり、FIL の結果として理想的だと言える。従来手法では常に完全一致である出力結果が得られるが、提案手法の結果を見ると全ての場合で完全一致が果たせてはいないことが分かる。実験に用いた全種類のテストスイートを総合して見ると、65%の割合で完全一致となる FIL 結果が得られていることが分かる。これは他のどの一致度の平均よりも格段に高い割合である。最大では TCAS の 3-way テストを対象にした場合の 97.6%、最小では Bugzilla の 2-way テストを対象にした場合の 14.9%と対象によってかなりの開きがある。SUT 毎に見ると TCAS は網羅因子数に関わらず完全一致となった割合が高い。他の SUT では 2-way テストだと完全一致の割合が低く、網羅因子数が上がるにつれて完全一致の割合が高くなることが確認できる。

評価基準 2 （完全一致・合体完全一致・可能性一致）

全ての不具合誘発パラメータ組を特定するのに有効と思われる出力結果を評価基準 2 に含めた。合体完全一致 (B) に相当する 1 つの不具合誘発パラメータ組で説明できる複数の不具合誘発パラメータ組は、実質的にその 1 つの不具合箇所を指していると考えられる。また可能性一致 (C) は追加テストによって一意な不具合誘発パラメータの絞り込みが見込める。FIL の目的を考えると、評価基準 2 による精度の評価が最も妥当かつ重要なものと位置付けられる。

表 6.1　4 評価基準による SSCC 法の精度評価

SUT	t-way	評価基準 1	評価基準 2	評価基準 3	評価基準 4
TCAS	2	0.7705	0.9813	0.9899	0.9927
	3	0.9755	1.0000	1.0000	1.0000
	4	0.9277	0.9780	0.9974	1.0000
	5	0.9378	0.9748	0.9969	1.0000
SPINS	2	0.2695	0.6390	0.7498	0.7533
	3	0.5935	0.8291	0.9218	0.9773
	4	0.7458	0.8270	0.9702	0.9976
	5	0.7498	0.7828	0.9878	0.9989
SPINV	2	0.4603	0.6305	0.6865	0.7869
	3	0.8076	0.8494	0.9047	0.9826
	4	0.8674	0.8780	0.9661	0.9974
Bugzilla	2	0.1490	0.4046	0.6456	0.6460
	3	0.4940	0.6138	0.6348	0.6910
	4	0.8350	0.9030	0.9240	0.9480
GCC	2	0.2230	0.3720	0.5450	0.5470
	3	0.6420	0.6710	0.6718	0.6859
平均		0.6530	0.7709	0.8495	0.8753

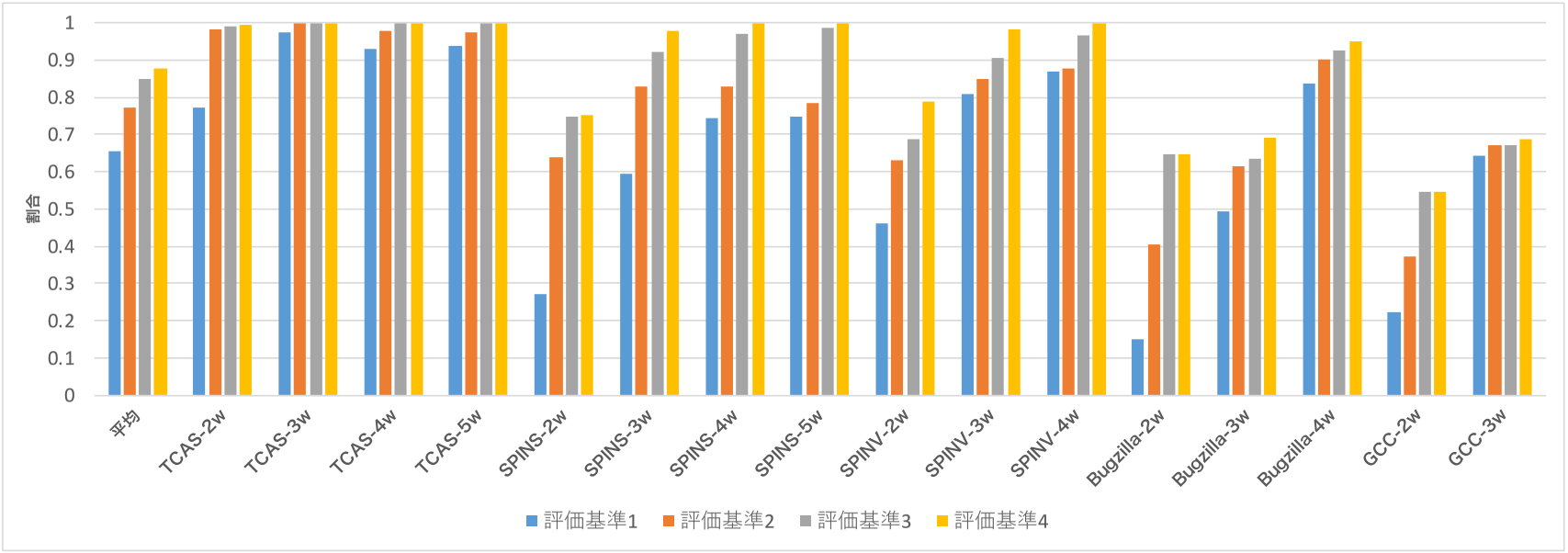

図 6.1　4 評価基準による SSCC 法の精度評価

この観点では、平均で77%の割合で不具合修正に有効なFIL結果が得られていることが分かる。テストスイート別に見ると、どのSUTに関しても2-wayテストで評価基準1と比べたときの上がり幅が大きい。ここから、2-wayテストを対象とする場合には完全一致でなく複数の可能性として得られる場合が多いことが言える。また3-wayテスト以上の網羅因子数のテストのみの平均を取ると85%となり、高い網羅因子数の組み合わせテストに対しては高い精度で有効なFILが行えることを示唆している。またTCASではほぼ100%の割合で有効なFIL結果に得られている一方で、GCCの2-wayテストでは37%、3-wayテストでも67%と精度が低く、対象システムの規模によっても提案手法を適用した際の不具合修正への貢献度は大きく異なると予想できる。

評価基準3 （完全一致・合体完全一致・可能性一致・代替一致）

提案手法の目的である、全てのテスト失敗を説明できる最小のパラメータ組の特定に成功した場合の出力結果を評価基準3に含めた。代替一致(D)は全てのテスト失敗を説明できる最小のパラメータ組を特定できてはいるが、それが不具合誘発パラメータ組と一致していないため不具合修正に効力を持たない出力結果であると言える。

この観点からの平均精度は84.95%となり、この数値は本論文で提案したSSCC法のアルゴリズムが実際に「全てのテスト失敗を説明できる最小のパラメータ組」を特定することのできる能力を表していると言える。この数値が100%ではないことは提示したアルゴリズムが不完全であることを示しているため、アルゴリズムに対して改善の余地があると結論づけることができる。

評価基準3の観点で失敗した場合について見ていく。特に多いのは失敗終了した場合である。最大のものでGCCの2-wayテストが39%の割合で失敗終了している。続けて多いのはGCCの3-wayテスト、Bugzillaの3-wayテストと2-wayテスト、SPINSの2-wayテスト、SPINVの3-wayテストと続く。これは端的に言えば、組み合わせテストの網羅因子数が低く、対象システムの持つパラメータ数が大きいほど失敗する割合が大きいことを示している。また現行のアルゴリズムでは失敗終了となる条件は、正解検査で正解の組み合わせセットが存在しなかった場合のみとなっている。失敗終了となった試行のログを見てみると、おそらく**Search**で複数の組み合わせの

候補が見つかった場合の **Cut** に関する部分が原因となっていると思われるが、詳しい解明と修正については今後の課題とする。失敗終了以外の失敗要因についても同様に今後の課題とする。

評価基準 4 （完全一致・合体完全一致・可能性一致・代替一致・包含一致・部分一致）

評価基準 3 の観点からだと包含一致 (E)、部分一致 (F)、不一致 (G)、失敗終了 (H) の 4 つに関しては明らかな動作失敗の事例であるものの、包含一致は不具合誘発パラメータ組を全て、代替一致と部分一致は部分的に FIL 結果の中に含んでいるので、その結果を不具合修正に活かすことは不可能ではない。部分的にでも不具合を修正することができればテスト結果は変化するので、修正後に組み合わせテストを再度実施し提案手法による FIL を行うことで別の不具合を特定し……というように連鎖的に全ての不具合を修正できる可能性はある。こうした観点を許容した場合の精度を評価基準 4 とした。

この観点での精度の平均値は 88%となるが、特定が失敗終了とならず何らかの結果が出力された場合に限定すれば 99%と極めて高い精度となる。しかし、不具合誘発パラメータ組の情報を元に不具合修正を行う場合にはそのパラメータ組が必ず不具合を誘発していることが言えなければ適切な修正点の発見に繋がるのは難しいと思われるため、この観点での評価はあくまで参考程度とする。

RQ2 への回答をまとめると、提案手法は特に網羅因子数が低くパラメータ数が大きい組み合わせテストに対しては特定失敗の割合が多くあまり精度の高い特定結果は得られないが、そうでない適用対象に対しては 85%程度の割合で不具合修正に有効と思われる不具合誘発パラメータ組を特定することが可能であると言える。また現状、SSCC 法のアルゴリズムは「全てのテスト失敗を説明できる最小のパラメータ組」の確実な特定が不完全であるので改善の余地がある。

RQ3 提案手法は従来手法と比べて優れているのか？

6.2 妥当性の検証

6.2.1 内的妥当性

実験環境について 本論文で行った実証実験は個人使用レベルの計算機をリソースとし、また使用言語も比較的遅いとされるスクリプト型言語である Python を使用しているが、実際のソフトウェア開発現場では演算用の高性能な計算機や C などの比較的高速な言語が使用されているなど環境が異なる場合があるので、処理時間の評価値に差が出る可能性がある。ただし、本実験よりも処理速度が遅い環境であることも同様に考えられる。

比較対象について 比較対象である BEN および comFIL の不具合誘発の可能性がある組み合わせを求めるプログラムは、論文の記述を基に筆者が作成したものである。考えうる限りの高速化は施したが、さらなる高速化が可能である場合、従来手法の処理時間の評価値がさらに短くなる可能性がある。

推定値について 従来手法の処理時間測定値のうち現実的な実測が困難なものに対しては推定値を用いた。ここでは 1000 回分の処理時間を記録してその平均値を処理回数に掛けることで 1 回あたりの処理時間における誤差の抑制を施したが、処理回数が極端に多い場合は僅かな誤差が総処理時間に影響を及ぼす可能性もあり、推定値が実際の値から大きく誤っている可能性もある。このことを考慮し、本論文では従来手法の最大値や最小値に関する言及を行わず、中央値で比較している。

6.2.2 外的妥当性

対象システムの種類について 本実験では 5 種類のシステムを対象とし、その SUT モデルから組み合わせテストのテストスイートを作成して利用した。この対象システムについては小規模なものから大規模なものまで、パラメータ数の大小やパラメータ値の制約の強さに幅を持たせられるよう選択した。また最も大きなものでは 199 のパラメータを持つ GCC を使用し、これは OSS として上限レベルの規模を持つと言える。これらのことから、対象システムの種類についての外的妥当性は高いと言えるだろう。

組み合わせテストの種類について 本実験では対象のシステムに対し、PICT を使用して 2-way テストから最大で 5-way テストまでのテストスイートを作成し利用

した。それぞれ扱った最大の網羅因子数より大きな網羅因子数のテストスイートを対象とした適用結果は未知であるので、本実験の結論と異なる結果が得られる可能性はある。またPICT以外の組み合わせ生成ツールを使用して組み合わせテストを行う場合にも異なる結論が得られる可能性も考えられる。

6.3 今後の課題

提案手法について 本論文ではSSCC法の基礎的なアルゴリズムの発案と改良を行ったが、のちに行った実証実験によってアルゴリズムにさらに改善の余地があることが明らかになった。具体的にはアルゴリズムが結果を返さずに失敗として終了する事例を無くすこと、アルゴリズムが「全失敗を説明できる最小の組み合わせ」を必ず返すようにすること、また極端に長い処理時間を要する処理を発生させないようにすることの3点が挙げられる。またFIL結果を不具合修正に活用させる際に、アルゴリズムによって求められた「全失敗を説明できる最小の組み合わせ」が一意に定まらないこと、あるいは実際の不具合誘発パラメータ組ではない別のパラメータ組を指すことへの対策も求められる。このために有効な追加テストケースを作成し実行させるような機能を追加することも視野に入れる必要がある。また今回は組み合わせテストの結果を入力とするような対象ソフトウェアに対して独立なツールを作成したが、実際のソフトウェアに作用してインタラクティブにテストとFILを行う発展的なツールを作成することも考えられる。こうしたツールが実現すれば実際の開発現場への投入も見込めると考える。

評価実験について 短期的には、内的・外的妥当性をさらに高めるために別の実験環境における再実験や、PICT以外で生成した別のテストスイートを用いた再実験などを行うことが考えられる。本実験では略式とした従来手法であるBENとcomFILについて最後まで一貫した実験を行い、厳密な比較を行うことで実験結果を補強することもできる。また今回の実験では不具合誘発パラメータ組をランダムに設定したが、実際に不具合のあるソフトウェアを用いてより現実的な不具合設定を行うことが望ましい。これはミューテーション解析に用いられる、人工的な不具合を含むモジュールであるミュータントへの差し替えを行うことで実現できると考えられる。またさらに、実際の開発者を対象として、組み合わせテストとFILの結

果によって実際に不具合のある箇所を特定し修正できるかを調べる実地的な実験を行うことにも大きな価値があると考える。

7. 結言

本研究では、組み合わせ爆発を抑制しつつ組み合わせテストの結果から不具合を誘発するパラメータ値の組み合わせを特定する手法として、構成要素の推定とテストスイートの圧縮および失敗テストケースの共通パラメータの検出を利用した再帰処理ベースのアルゴリズムである SSCC 法を提案した。

実際に提案手法を実在するソフトウェアの組み合わせテスト結果に適用させることで、提案手法が考え得る全ての組み合わせパターンを網羅的に調べることなく不具合を誘発するパラメータの組み合わせを特定する能力を持つことを示せた。

また複数のテスト結果を対象とした提案手法の適用結果の分析と従来手法との比較を行い、多くの場合で提案手法が高速に処理を完了できること、それなりに高い精度で設定した不具合誘発パラメータ組を検出できること、またこれらの傾向は大規模な組み合わせテストを対象とするときほど高くなり有効な手法となり得ることを示せた。

今後の課題として、精度の向上や一部の場合で起こる長時間処理への対策を含めたアルゴリズムの更なる改善や、実際にバグを含むソフトウェアを使用し提案手法の適用によって適切な修正が可能であるかといった実地的な評価を行う必要があると考えられる。

謝辞

本研究を行うにあたり，研究課題の設定や研究に対する姿勢，本報告書の作成に至るまで，全ての面で丁寧なご指導を頂きました，本学情報工学・人間科学系 水野修教授、加えて産業総合技術研究所情報技術部門ソフトウェアアナリティクス研究グループ 崔銀惠研究員に厚く御礼申し上げます．また本報告書執筆にあたり貴重な助言を多数頂きました，本学情報工学専攻 黒田翔太君、田中健太郎君、中川要さん、原田禎之君、小林勇揮君、近藤将成君、洪浚通君、本学情報工学課程 植村桂治君、北村紗也加さん、中村勝一君、廣瀬早都希さん、渡辺大輝君、応用生物課程 國領正真君、本研究室水槽管理ロボット兼マスコット あくあたん、および学生生活を通じて筆者の支えとなった家族や友人に深く感謝致します。