不具合誘発パラメータ組み合わせ特定三手法の比較評価

渡辺　大輝
京都工芸繊維大学 大学院工芸科学研究科
博士前期課程 情報工学専攻
d-watanabe@se.is.kit.ac.jp

西浦　生成
京都工芸繊維大学 大学院工芸科学研究科
博士後期課程 設計工学専攻
k-nishiura@se.is.kit.ac.jp

水野 修
京都工芸繊維大学 情報工学・人間科学系
o-mizuno@kit.ac.jp

要旨

組み合わせテストによる不具合誘発パラメータ組み合わせの特定は，ソフトウェア開発者が不具合誘発の原因となる要因を特定する上で重要な役割を果たす．近年，様々な研究者によって組み合わせテストの手法が数多く提案されている．一方で，不具合の個数や不具合誘発条件の複雑さ，用いるシステムの規模などで示されるある特定の状況下において，実際どの手法を用いれば最も効率よく正確に不具合誘発パラメータ組み合わせを特定できるのかという疑問が抱かれる．本論文では，これまでに提案された 3 種類の従来手法を用いて，組み合わせテストにかかる処理時間，必要な追加テストケースとその実行回数，不具合特定成功率といった 3 つの観点を中心に比較評価を行った．実験の結果，用いたテストスイートの変化による同一手法内でのデータの変化や，同一のテストスイートにおける 3 種類の従来手法の実験結果の差異について収集することが出来た．また，得られたデータを元に比較を行い，3 種類の従来手法の有用性の差別化や，テストスイートの変化が引き起こす影響についての結論を示した．

1　はじめに

ソフトウェアの開発は，必ずしも最初から最後まで思い通りに進むとは限らない．開発を行っている最中，または世の中に出回った後に不具合が見つかる可能性も大いに存在する．この不具合が現代社会に及ぼす影響は大きいものであるがゆえに，その不具合を検出するテスト手法の一つである組み合わせテストは重要な役割を果たす．

ソフトウェアにおける不具合は，ある単一のものから誘発されるとは限らず，ある条件の組み合わせが存在することによって不具合が誘発されるということが起こりうる．ソフトウェアにおいて不具合が誘発されることが発覚したとき，我々はどの組み合わせが不具合を誘発させているのかどうかを特定する必要がある．近年，多くの研究者がパラメータ集合を元に，それらを全て正確に導き出し，不具合誘発パラメータ組み合わせとして出力する組み合わせテストの手法を様々に提案している．その手法は，元のテストスイートに大量にテストケースを追加していき不具合誘発パラメータ組み合わせを特定する手法から，たった一つのテストケースからそのパラメータを変更していき不具合誘発パラメータ組み合わせを特定する手法まで様々である．一般に，組み合わせテストによって不具合誘発パラメータ組み合わせを特定することを FIL（Fault Interaction Location）と呼ぶ．では，実際に FIL を行うとなったとき，一体どの手法を選択すればいいのだろうか．例えばあるテストスイート下において 100%正しく FIL を正確に行うことが出来る手法があったとしても，テストの実行にかかる時間が長ければ使用者のストレスの増加や作業の非効率化が予測される．逆にどんなに速やかに FIL を行うことが出来る手法があったとしても，その出力結果の正確さが欠けている

と不具合修正の容易さに悪影響を及ぼしてしまう．ゆえに，対象とするテストスイートの性質に対して最適な手法を選択することが出来れば，組み合わせテストの効率化を図ることが出来ることが期待される．

そこで本論文では，次の 3 つの従来手法の実装を行った: 2016 年に Zheng らによって提案された comFIL（complete Fault Interaction Location）[1]，2011 年に Zhang らによって提案された FIC（Faulty Interaction Characterization）[2]，2012 年に Ghandehari らによって提案された identifying inducing combinations [3]．また，20 種類のテストスイートを用意し，それぞれを処理時間，成功率，テストの実行回数の 3 つの観点から比較を行い，3 手法の有用性を調査した．

2 準備

2.1 組み合わせテスト

組み合わせテストは，パラメータが取る値の組み合わせに注目し，ある組み合わせ数のパラメータ間で取る値のパターンの全てを網羅するテストのことであり，組み合わせ数を t としたとき，t-way テストや t-wise テストと呼ぶ．以後，あるパラメータが取る値のことをパラメータ値と表記する．例えば，表 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．まず総テストケースなら，3 パターンのパラメータ値を取るパラメータが 1 つ，2 パターンのパラメータ値を取るパラメータが 2 つなので，テストケース数は $3\times2\times2=12$ となる．対して 2-way テストの組み合わせテストなら，表 2 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 つの組み合わせ (X,Y)，(Y,Z)，(Z,X) のパラメータ値を確認すると，確かに全て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 [4] が開発した PICT[1]，産業技術総合研究所の Choi ら [5] が開発した pricot，Bryce ら [6] が開発した DDA などがある．

2.2 FI(Fault Interaction)

FI (Fault Interaction) とは，テストケースにおける，不具合を誘発させる原因となる組み合わせのことである．以後，FI で統一する．FI は単一のパラメータによって構成されることもあれば，複数パラメータ (組み合わせ) によって構成されることもある．この論文では，FI を次のように表記する．

表 1. パラメータの仕様

parameter	value
X	1,2,3
Y	1,2
Z	1,2

表 2. 表 1 の 2-way 組み合わせテスト

テストケース	X	Y	Z
t_1	1	1	1
t_2	1	2	2
t_3	2	1	2
t_4	2	2	1
t_5	3	1	1
t_6	3	2	2

- FI が単一のパラメータであるとき，(変数名=パラメータ値)
- FI が複数のパラメータであるとき，(変数名=パラメータ値, 変数名=パラメータ値, ...)

またこれ以後，FI を含んでいるときの実行結果を Fail，FI を含んでいないときの実行結果を Pass と表記する．更に，この論文では「誘発」という言葉を「FI を含んでいることが原因となってテストの実行結果が Fail となること」という意味で定義している．

2.3 従来手法

2.3.1 comFIL(complete Fault Interaction Location)

comFIL(complete Fault Interaction Location) [1] は，2016 年に Zheng らによって提案された組み合わせテストの一種である．以後，comFIL で統一する．

この手法では，まずテストスイートを入力する．例えば，2 パターンのパラメータ値を取るパラメータが 2 つ，3 パターンのパラメータ値を取るパラメータが 1 つ，4 パターンのパラメータ値を取るパラメータが 1 つの 2-way テストの入力は表 3 のテストスイートになる．同時に，そのテストケースが FI を含んでいるかどうかを一つずつ判定していく．表 3 の実行結果は，FI が $(a=0,c=0)$，$(a=0,d=3)$ であったときを示している．

[1] https://github.com/Microsoft/pict

表 3. 2-way テストの実行結果

テストケース	a	b	c	d	実行結果
1	0	0	0	0	Fail
2	1	1	1	0	Pass
3	0	1	2	0	Pass
4	1	0	0	1	Pass
5	0	0	1	1	Pass
6	1	1	2	1	Pass
7	0	1	0	2	Fail
8	1	0	1	2	Pass
9	0	0	2	2	Pass
10	0	1	0	3	Fail
11	1	0	1	3	Pass
12	1	0	2	3	Pass

次に，Fail したテストケースの部分集合を考える．その中から，Pass した全てのテストケースの部分集合でないものが，FI の候補となる．例えば，$(a = 0, b = 0)$ は Pass したテストケ ースの部分集合になっているので FI の候補とはならないが，$(a = 0, d = 0)$ は Pass したテストケースの部分集合でないので FI の候補となる．この場合，全ての FI の候補は表 4 の通りになる．

次に，FI の候補を元にテストケースを追加していく．追加テストケースの条件は以下の通りである．

- 最初に入力したテストスイートとは被らないようにする．
- 追加テストケースに真の FI が追加されることはないものとする．例えば，表 4 の T_{11} には $a = 0$ は追加されない．
- 複雑化を防ぐため，追加テストケースを作成したときに新たな FI は生まれないものとする．

FI の候補を 1 つずつ取り出し，追加テストケースを作成したあとに実行する．Fail だった場合，その FI 候補を部分集合に持つ他の FI 候補を FI 候補から取り除く．例えば，表 4 の T_1 は実行後 Fail となり，T_1 を部分集合に持つ T_8 が FI 候補から取り除かれる．Pass だった場合，その FI 候補自身とその FI 候補自身の部分集合を FI 候補から取り除く．例えば，表 4 の T_2 は実行後 Pass となり，T_2 自身と T_2 の部分集合である T_6 が FI 候補から取り除かれる．なお，この T_6 や T_8 のように追加テストケースを作成する前に取り除かれる FI 候補には追加テストケースを作成しない．これを順番に繰り返し，最終的に取り除かれることなく残った FI 候補が真の FI として特定される．

表 4. 表 3 の FI 候補

FI 候補	a	b	c	d
t_1	0	0	0	-
t_2	0	0	-	0
t_3	0	-	0	0
t_4	-	0	0	0
t_5	0	-	0	-
t_6	-	0	-	0
t_7	-	-	0	0
t_8	0	0	0	0
t_9	0	1	0	2
t_{10}	0	1	0	-
t_{11}	-	1	0	-
t_{12}	0	-	0	2
t_{13}	-	-	0	2
t_{14}	-	1	0	2
t_{15}	-	1	-	2
t_{16}	0	1	-	2
t_{17}	0	1	0	3
t_{18}	0	-	0	3
t_{19}	-	-	0	3
t_{20}	0	-	-	3
t_{21}	-	1	-	3
t_{22}	-	1	0	3
t_{23}	0	1	-	3

2.3.2 FIC (Faulty Interaction Characterization)

FIC(Faulty Interaction Characterization) [2] は，2011 年に Zhang らによって提案された組み合わせテストの一種である．以後，FIC で統一する．

この手法では，最初に入力するテストケースはたった 1 つであり，テストケースのパラメータ値そのものを変更していくことによってその中に含まれる FI を特定する．他に，テストケースの要素数と，パラメータ値のパターン数を入力する．今回の例では，入力したテストケースを $[1, 2, 2, 1, 2, 2, 1, 1]$(このときテストケースの要素数は 8)，パラメータ値のパターン数は 8 個とも 2 パターンとする．また，FI は $(c = 2, f = 2)$，$(d = 1)$ とする．

次に，FI を特定するためにテストケースのパラメータ値を順番に変更していき，1 回ずつ実行していく．その実行結果が Fail のときは，変更したパラメータ値はそのままに次の要素のパラメータ値を変更していく．Pass のときはパラメータ値を変更する前に戻し，次の要素のパラメータ値を変更していく．今回の例での推移を表 5 に

表 5. FIC の推移例

テストケース								実行結果
a	b	c	d	e	f	g	h	
1	2	2	1	2	2	1	1	Fail(Skip)
2	2	2	1	2	2	1	1	Fail
2	1	2	1	2	2	1	1	Fail
2	1	1	1	2	2	1	1	Fail
2	1	1	2	2	2	1	1	Pass
2	1	1	1	1	2	1	1	Fail
2	1	1	1	1	1	1	1	Fail
2	1	1	1	1	1	2	1	Fail
2	1	1	1	1	1	2	2	Fail
1	2	2	1	2	2	1	1	Fail
2	2	2	2	2	2	1	1	Fail
2	1	2	2	2	2	1	1	Fail
2	1	1	2	2	2	1	1	Pass
2	1	2	2	1	2	1	1	Fail
2	1	2	2	1	1	1	1	Pass
2	1	2	2	1	2	2	1	Fail
2	1	2	2	1	2	2	2	Fail
1	2	1	2	2	1	1	1	Pass

示す．まず最初に a のパラメータ値を 2 に変更し，その後再度実行する．実行結果は Fail のため，a のパラメータ値は 2 のままに，次は b のパラメータ値を 1 に変更する．これを繰り返していくと，d を 2 に変更したときに初めて実行結果が Pass になる．よって，d のパラメータ値は 1 に戻され，再び e からパラメータ値を変更していく．その後は最後まで Fail であり続けるため，これにより一つ目の FI が $(d = 1)$ と特定される．

1 週目が終了した後，元のテストケースより FI である $(d = 1)$ を取り除くため，d のパラメータ値を 2 に変更し，テストケースを $[1, 2, 2, 2, 2, 2, 1, 1]$ とする．その後再度実行すると結果は Fail となるため，このテストケースにはまだ FI が存在しているということになる．よって，再びこのテストケースのパラメータ値を順番に変更していく．しかし，d のパラメータ値を 1 に変更してしまうと FI である $(d = 1)$ を含んでしまうため，d のパラメータ値の変更は行わない．すると表 5 に示された通り，c と f のパラメータ値を変更したときに実行結果が Pass になる．よって二つ目の FI が $(c = 2, f = 2)$ と特定される．

2 週目が終了した後，今度は $(c = 2, f = 2)$ を取り除くために c のパラメータ値を 1 に，f のパラメータ値を 1 にそれぞれ変更し，テストケースを $[1, 2, 1, 2, 2, 1, 1, 1]$ とする．その後再度実行すると結果は Pass となるため，このテストケースには FI が存在していないことになる．以上から，真の FI は $(c = 2, f = 2)$，$(d = 1)$ と特定される．

表 6. 表 3 の FI 候補 (IIC)

FI 候補	a	b	c	d
t_1	0	-	0	-
t_2	-	1	0	-
t_3	-	-	0	0
t_4	-	-	0	3
t_5	-	-	0	2
t_6	0	-	-	3
t_7	-	1	-	3
t_8	-	1	-	2
t_9	-	0	-	0

2.3.3 Identifying Inducing Combinations

Identifying Inducing Combinations [3] は，2012 年に Ghandehari らによって提案された組み合わせテストの一種である．以後，IIC で統一する．

この手法は，FI 候補となる組み合わせを，ある計算式によって真の FI である可能性をランク付けし，その可能性の高いものから順に追加テストケースを作成していくというものである．

まずテストスイートを入力する．今回は 2.3.1 節で使用したテストスイートと同じものを使用するものとし，FI も 2.3.1 節と同様に $(a = 0, c = 0)$，$(a = 0, d = 3)$ とする．このとき，テストスイートとそれぞれの実行結果は表 3 に示される．

次に，comFIL と同様に，Fail したテストケースの部分集合の中から，Pass した全てのテストケースの部分集合でないものを，FI の候補とする．しかし，comFIL は部分集合のパラメータ数に決まりがなかったのに対し，IIC では部分集合のパラメータ数は t-way テストの t の値のものに限る．つまり，今回の例ではパラメータ数が 2 である部分集合を考える．この場合，全ての FI の候補は表 6 の通りになる．

次に，FI 候補の構成パラメータ一つ一つを，後述の式 1 によって，その疑わしさを値として算出する．

$$\rho(o) = \frac{1}{3}(u(o) + v(o) + w(o)) \quad (1)$$

ここで，$u(o)$，$v(o)$，$w(o)$ は以下の式で算出される．

$$u(o) = \frac{|\{f \in F_i | r(f) = fail \wedge o \in f\}|}{|\{f \in F_i | r(f) = fail\}|} \quad (2)$$

$$v(o) = \frac{|\{f \in F_i | r(f) = fail \wedge o \in f\}|}{|\{f \in F_i | o \in f\}|} \quad (3)$$

$$\rho(o) = \frac{|\{c | o \in c \wedge c \in \pi\}|}{|\pi|} \quad (4)$$

$u(o)$ は，Fail したテストケースに含まれている総数の値から，Fail したテストケースの総数の値を除算する．$v(o)$ は，Fail したテストケースに含まれている総数の値から，全てのテストケースに含まれている総数の値を除算する．$w(o)$ は，FI 候補に含まっている総数の値から，FI 候補自体の総数の値を除算する．例えば，$\rho\ (c \to 0)$ は式 5 の通りに算出される．

$$\rho(c \leftarrow 0) = \frac{1}{3} * (1 + \frac{3}{4} + \frac{5}{9}) = 0.7685 \quad (5)$$

同様に，全ての FI 候補の構成パラメータについて式 1 の値を算出する．その結果を表 6 に示す．

次に，真の FI である可能性をランク付ける 2 つの値を算出する．それらは式 6 と式 7 で与えられる．

$$\rho_c(c) = \frac{1}{|c|} \sum_{\forall o \in c} \rho(o) \quad (6)$$

$$\rho_e(c) = Min(\sum_{o \in f \wedge o \notin c} \rho(o), \forall f \in F) \quad (7)$$

ρ_c は，ある FI 候補の構成パラメータ 1 つ 1 つの $\rho(o)$ の平均値を取る．例えば，$\rho_c(a \to 0, c \to 0)$ は，$\rho(a \to 0)$ と $\rho(c \to 0)$ の値の平均値を取る．また，ρ_e は，ある FI 候補の構成パラメータ以外で構成されている ρ_c の中の最小値を取る．例えば，$\rho_e(a \to 0, c \to 0)$ は $\rho_c(b \to 0, d \to 0)$，$\rho_c(b \to 1, d \to 2)$，$\rho_c(b \to 1, d \to 3)$ の中から最小の値を取る．これらを全ての FI 候補について算出し，ρ_c はその値の高い順に，ρ_e はその値の低い順にそれぞれランク付けを行う．以下，その結果をそれぞれ R_c，R_e と呼ぶこととする．

次に，R_c と R_e の値を加算した値を算出し，その値の低い順にそれぞれランク付けを行う．これを R と呼ぶこととする．この R の値の低い FI 候補が，真の FI である可能性が高いものとして疑われることになる．以上をまとめた結果を表 7 に示す．

表 7. IIC のランク付け

FI 候補	ρ_c	R_c	ρ_e	R_e	$R_c + R_e$	R
t_1	0.6713	1	0.2460	1	2	1
t_2	0.6176	2	0.4352	3	5	2
t_3	0.5324	4	0.3849	2	6	3
t_4	0.5509	3	0.5204	4	7	4
t_5	0.5324	4	0.5204	4	8	5
t_6	0.4537	5	0.6176	5	10	6
t_7	0.4000	6	0.6713	6	12	7
t_8	0.3815	7	0.6713	6	13	8
t_9	0.2460	8	0.6713	6	14	9

次に，R の値の低い FI 候補から順に追加テストケースを作成し，実行を行う．実行自体は全ての FI 候補に対して行う．追加テストケースの作成方法は以下の通りである．

- 最初に入力したテストスイートとは被らないようにする．
- 追加する部分のパラメータ値は，$\rho(o)$ の値が 1 番低いものを採用する．

これにより，真の FI を特定する．

3 実験

本節では，研究設問，実験環境の詳細な明記，テストスイートなどの実験対象の説明，実験手順や留意点，得られた実験結果について説明する．

3.1 研究設問

本実験を行うにあたって，以下の研究設問を設定した．

- RQ1:同一の手法のなかで，パラメータ数の大小とパラメータ値の種類数が与える影響は何か．
- RQ2:異なる手法のなかで，それぞれのテストスイートを用いるときの他の手法との差は何か．

これらを回答するために，まず 1 つの従来手法において様々なテストスイートとシステムモデルを用いて実行し，それと全く条件で他の 2 つの従来手法においても実行すればいいことが分かる．

今回はそれぞれのテストスイートにおいて，以下の 3 つの観点について測定を行い，これらから 3 種類の従来手法の有用性について差別化を行う．

- 処理時間
- 成功率
- 追加テストの実行回数

3.2 実験準備

3.2.1 実験環境

本実験で用いたプログラミング言語，開発環境，PC は以下の通りである．

- PC:MacBook Pro(Retina 13-inch，Late 2013)，Sierra(バージョン 10.12.6)，2.8 GHz Intel Core i7，8 GB 1600 MHz DDR3
- 言語:Python 3.6.3

3.2.2 実験対象

用いるテストスイートのパラメータ数，パラメータ値のパターン数の 2 つの要素の設定をシステムモデルと表記する．本実験では，4 種類のシステムモデルに対してそれぞれ 5 通りの方法で作成された計 20 種類のテストスイートを実験対象として用いる．

- ランダムテスト (100 個)
- ランダムテスト (500 個)
- 2-way テスト
- 3-way テスト
- 4-way テスト

ここで，ランダムテストとは，ランダムにパラメータの値を決定して作成された n 個のテストケースをテストスイートとしたものである．本実験では $n = 100$ の場合と $n = 500$ の場合を用いる．また，t-way テストの生成には Microsoft の組み合わせテスト生成ツールである PICT を用いた．t-way のそれぞれのテストケース数を表 8 に示す．

表 8. t-way テストのテストケース数

	2-way	3-way	4-way
システム A	13	43	127
システム B	24	111	455
システム C	19	69	233
システム D	31	163	795

表 9. 対象システムモデルの詳細

システム	パラメータ数	パラメータ値のパターン数
A	10	2,2,2,2,2,3,3,3,3,3
B	10	3,3,3,3,3,4,4,4,4,4
C	20	2,2,2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,3,3
D	20	3,3,3,3,3,3,3,3,3,3,4,4,4,4,4,4,4,4,4,4

組み合わせテストのテストスイート設計法の 1 つである直交表を利用したテストは，指定の大きさの組み合わせ因子を網羅しつつテストケース数をできるだけ少なくする点で t-way テストと同じであり，目的とする役割が t-way と同じであるため本実験では使用しなかった．

また，使用する対象システムモデルを表 9 に示す通りに設定する．このシステムは，本実験用に作成した架空のものである．例えば，システム A は 2 種類の値を取るパラメータを 5 個，3 種類の値を取るパラメータを 5 個，合計 10 個のパラメータを持つ．

3.2.3 実験手順

2 節で述べた通り，既存の FIL 手法である comFIL，FIC，IIC の 3 手法を用いて比較を行う．

まずプログラムを実行する前に，FI の数，FI の大きさ，FI のパラメータ値の設定を行う．FI の数は全てのテストにおいて 1 から 3 までの値でランダムに設定する．FI の大きさはランダムテストは 3，t-way テストは t で統一する．FI のパラメータ値は各対象システムモデルにおいて設定されているパラメータ値のパターン数の中からランダムに与える．ただし，少なくとも 1 つのテストケースは失敗するように選択する．

FI の設定後，それらに対して各手法を適用させて設定した FI を特定した．その後，処理時間，成功回数，追加テストケースの実行回数を測定した．これらの 1000 種類のテスト結果のサンプルを用意し，処理時間とテストの実行回数の平均値，最小値，第 1 四分位数，中央値，第 3 四分位数，最大値を算出した．成功率は百分率で値

を算出した．このテスト結果はランダムに設定された FI による作用である．

また，comFIL では本来その大きさに関わらず，Fail したテストケースに部分集合として含まれ，かつ Pass したテストケースには部分集合として含まれないパラメータの全ての組み合わせを FI 候補とするが，テストケースのパラメータ数が大きいほど組み合わせ数は膨大になり処理に大きな時間がかかると予想される．そこで本実験では実験と比較の簡易化のため，それぞれ対象とするテストスイートが網羅する組み合わせの大きさ以下の部分集合のみを考える．

また，FI の数は分かっていないことを前提とし，IIC においては全ての FI が特定出来たとしても追加テストケースを作成していない FI 候補が残っている場合は全て終わるまで実行を続けることとする．

3.3 実験結果

実験結果を表 10 に示す．ここで，ave(t)[s] は平均処理時間を，ave(n)[回] は追加テストの平均実行回数を表す．

成功率は，各従来手法の出力によって導き出された FI と，予めランダムで決定されて与えられた FI が完全に一致している場合を成功とし，その割合を算出する．テストケースの実行回数は初めにテストケースを Fail したものと Pass したものに分けるための実行は含めず，追加のテストケースを実行した回数を測定する．以下，実行回数と表記されているものも同様である．また，今回は 3 種類の従来手法全てにおいて，追加のテストケースを実行するためにかかる時間を 0.1 秒と仮定する．しかし，そのままソースコードに 0.1 秒待機する命令を加えてしまうと本実験に費やされる時間が膨大になることを予測し，本実験では効率化のため実行回数に 0.1 を乗算した値を測定された処理時間に加算し，その値を処理時間として算出した．

実験結果では，同じ従来手法でもテスト種類によって大きく異なる結果が得られた．また，それぞれの従来手法においても大きく異なる結果が得られた．

また，テストの成功率は全て 100%を記録していた．このことから，3 種類の手法は全て FI を正確に特定出来ていることが分かった．

4 考察

4.1 研究設問への回答

考察は表 10 の実験結果を元に行う．

RQ1：同一の手法のなかで，パラメータ数の大小とパラメータ値の種類数が与える影響は何か．

最初に，comFIL について考える．ランダムテストにおいては，4 種類全てのシステムにおいて，テストケースを多く用いる場合に処理時間と追加テストの実行回数が減少することが分かった．comFIL は実行前に全てのテストケースを Fail したテストケースと Pass したテストケースに分類し，その後 Fail したテストケースのみの部分集合になっているパラメータ値の組み合わせを FI 候補とするが，本実験では FI の個数は多くても 3 つであるため，テストケースの数が多くなると同時に Fail したテストケースと比べて Pass したテストケースの個数が相対的に増える幅が大きいことが予想される．これにより，FI 候補の数が減少することが必然的に予想され，その結果追加テストケースの個数が減少したことで処理時間と追加テストの実行回数が減少したと考えられる．

次に，t-way テストは処理時間についてはシステム B においてのみ，2-way テストが最も処理時間が大きい結果となった．それ以外のシステムにおいては t の値が大きくなるにつれて大きくなる傾向が見られた．また，追加テストの実行回数においては 4 種類全てのシステムにおいて t の値が大きくなるにつれて多くなる傾向が見られた．これは，組み合わせ数が大きい方が Pass したテストケースの部分集合になりにくくなり，FI 候補の数が増加するためであると考えられる．異なるシステム間においては，パラメータ数とパラメータ値のパターン数が多いほど処理時間と追加テストの実行回数が増加することが分かった．またシステム B と C を比較すると，パラメータ値のパターン数が与える影響よりもパラメータ数が与える影響の方が大きいことが分かった．これは，パラメータ値のパターン数が増加するよりもパラメータ数が増加する方が，考えなければならない部分集合の数が爆発的に多くなることからこのような結果になったと考えられる．

次に，FIC について考える．同一のシステム内においては 5 種類全てのテストにおいてほぼ全く同じ結果が得られた．また，パラメータ数が同一のシステム A，B とシステム C，D において，処理時間と追加テストの実行

表 10. 実験結果 (3 手法のシステム毎の処理時間と追加テストの実行回数の平均値)

システム	手法	テスト	ave(t)	ave(n)	手法	テスト	ave(t)	ave(n)	手法	テスト	ave(t)	ave(n)
		ran100	2.797	27.668		ran100	2.241	22.396		ran100	1.391	13.844
		ran500	1.723	15.869		ran500	2.228	22.242		ran500	0.274	2.062
A	comFIL	2-way	5.769	57.565	FIC	2-way	2.174	21.736	IIC	2-way	4.139	41.321
		3-way	7.710	76.839		3-way	2.204	22.033		3-way	4.094	40.828
		4-way	8.007	79.410		4-way	2.207	22.055		4-way	4.238	42.253
		ran100	8.878	88.428		ran100	2.089	20.867		ran100	4.267	42.520
		ran500	2.580	24.761		ran500	2.167	21.637		ran500	0.254	2.323
B	comFIL	2-way	11.310	56.503	FIC	2-way	2.235	22.341	IIC	2-way	3.663	36.574
		3-way	7.430	73.971		3-way	2.176	21.747		3-way	4.174	41.632
		4-way	8.285	81.512		4-way	2.238	22.341		4-way	4.385	43.698
		ran100	13.758	135.257		ran100	4.192	41.895		ran100	4.301	42.517
		ran500	4.203	31.051		ran500	4.148	41.412		ran500	0.338	2.039
C	comFIL	2-way	12.505	124.434	FIC	2-way	4.122	41.202	IIC	2-way	4.702	46.752
		3-way	12.742	195.257		3-way	4.086	40.845		3-way	4.490	44.143
		4-way	25.705	239.703		4-way	4.052	40.488		4-way	4.380	42.532
		ran100	67.331	664.086		ran100	3.904	39.018		ran100	4.324	35.938
		ran500	5.955	52.162		ran500	4.271	42.651		ran500	0.210	1.280
D	comFIL	2-way	15.776	157.012	FIC	2-way	4.214	42.126	IIC	2-way	4.887	48.501
		3-way	22.856	225.418		3-way	4.140	41.370		3-way	4.494	44.087
		4-way	31.988	275.066		4-way	4.258	42.483		4-way	4.202	40.510

回数の値も同様な結果が得られた．FIC はテストケースのパラメータ値を 1 つずつ順番に変えて実行をしていき，FI を 1 つ特定する手法なので，追加テストの実行回数はパラメータ数の値と FI の個数に完全に依存するものである．このことから，得られる結果の固定化を招いたと考えられる．また，システム A，B と比べてパラメータ数が 2 倍であるシステム C，D は処理時間と追加テストの実行回数もほぼ 2 倍になっていることから，処理時間と追加テストの実行回数はパラメータ数と比例していると考えられる．

最後に，IIC について考察する．comFIL と同じく，ランダムテストを対象とした場合には，4 種類全てのシステムにおいて，テストケースを多く用いる場合に処理時間と実行回数が減少することが分かった．これは，実験前に FI 候補を割り出す手順が comFIL と同じであるため，同様の理由でこのような結果が得られたと考えられる．t-way テストにおいては，comFIL と比べると t の値によって明白な差が生まれることはなかった．しかし，この論文の表中には示していないが，2-way テストでは処理時間と追加テストの実行回数のデータに大きなばらつきが見られた．特に，データの最大値が 3-way テストと 4-way テストに比べて高くなっていることが分かった．2-way テストは他のテストと比べてテストスイートの総テストケース数が少ないので，FI の個数が多いと Pass したテストケースの割合が他のテストに比べて減少する傾向にあると考えられる．これにより，FI 候補の数が多くなる可能性があることが予測される．IIC は comFIL と違い，最終的に FI 候補は全て追加テストケースを作成し実行する．よって，FI 候補の増加が必然的に処理時間と追加テストの実行回数に直結することになる．これが最大値を突出させた原因であると考えられる．異なるシステム間においても，処理時間と追加テストの実行回数に明白な差は生まれなかった．これにより，パラメータ数とパラメータ値のパターン数は生成される FI 候補の数には依存しないことが分かった．comFIL は本実験では FI の大きさの値以下の要素数を持つパラメータ組み合わせを全て考えている．対して，IIC は FI の大きさの値と同等の要素数を持つパラメータ組み合わせのみを考える．このことから，comFIL と比べて考える部分集合の数の伸びが少なくなることからこのような結果が得られたと考えられる．

RQ2：異なる手法のなかで，それぞれのテストスイートを用いるときの他の手法との差は何か．

本実験の結果から，ほぼ全てのテストスイートを対象とした場合において，処理時間と追加テストの実行回数が最も少なくなるのは FIC であることが分かった．先述

の通り，FIC の処理時間と追加テストの実行回数は追加で行うテストのテストケースのパラメータ数に依存し，comFIL と IIC は FI 候補の数に依存する．仮にパラメータ数が 1 増加したとすると，FIC の実行回数は 1×（FI の個数）回しか増加しない．これに対し，comFIL と IIC はパラメータ数が 1 増加するだけでも考えなければならない部分集合の数が膨大に増加するため，追加テストの実行回数も大きく増加する．このことから，多くの場合においてはテストケースをどんどん追加していく手法を用いるより，1 つのテストケースを用いて FIL を行う手法を用いる方が効率が良いことが分かった．しかし，テストケースが多量のランダムテストにおいてはその限りではないことが分かった．

次に，comFIL と IIC を比較し，その差別化について考察する．本実験の結果では，一般的に全てのテストにおいて，IIC の方が処理時間と追加テストの実行回数が少なかった．これは先述の通り，IIC が FI の大きさの値と同等の要素数を持つパラメータ組み合わせのみを考えるのに対し，comFIL は FI の大きさの値以下の要素数を持つパラメータ組み合わせを全て考えているので，考えなければならない部分集合の数が IIC の方が少なくなることからこのような結果となった．しかし，これは今回のように FI の大きさが事前に見当がつく場合に限り，現実のソフトウェア開発においては必ずしも FI の大きさが事前に見当がつくとは限らない．その場合においては，IIC とは違い全ての部分集合を考えることの出来る comFIL を用いる方が汎用的に優れていると考えられる．以上のことから，本実験から得られた 3 種類の従来手法の差別化は以下の通りに考えられる．

- IIC: FI の大きさに見当がつき，多量のランダムテストケースを扱うとき．
- comFIL: FI の大きさに見当がつかず，多量のランダムテストケースを扱うとき．
- FIC: t-way テストや少量のランダムテストケースを扱うとき．

また，成功率については 3 種類全ての手法で 20 種類全てのテストスイートにおいて 100%となった．このことから，得られる出力結果の精度についてはどの従来手法も同等であると考えられる．

4.2 処理時間と追加テストの実行回数

本実験の結果から，処理時間と追加テストの実行回数は正の相関関係があることが分かった．従って，追加テストの実行回数の削減が処理時間の削減に直結することが分かる．本実験では追加テストの実行時間を 0.1 秒と仮定しているが，もしこの実行時間が更に長くなったとき，テストにかかる総時間は追加テストの実行回数が多ければ多いほど大幅に長くなることが予測される．このような状況下で本実験で用いた 3 種類の従来手法から 1 つを選ぶときは，次の 2 つの判断基準が考えられる．

- FI の大きさに見当がつく場合，IIC を用いてランク付けを行い，順番に実行していく．FI を全て特定をし終えたら実行をやめる．そうすると残りを実行しなくて済むので，追加テストの実行回数の削減に繋がる．
- 本実験のほぼ全てのテストにおいて最も優れており，かつ処理時間がある程度予測できる FIC を用いる．

comFIL はどうしても追加テストの実行回数が多くなってしまうので，実行時間が長くなったときには不向きであると考えられる．

4.3 妥当性の検証

4.3.1 ランダム性

今回の実験では，FI の数，FI のパラメータ値を 1 回のループごとにランダムに与えている．FI の数が大きくなればなるほど，どの手法でも処理時間は長くなるので，与えられた FI の数に偏りが生じれば，テスト結果に少なからず影響が及ぶと考えられる．また，ランダムテストに用いたテストスイートにも偏りが生じている可能性も考えられる．総テストケースは 1 番少ないシステム A でも $2^5 \times 3^5 = 7776$ 通り存在する．1 番多いシステム D だとおよそ 619 億通りにも及ぶ．ランダムテストに用いたテストケース数の 100 や 500 という数はこれに比べるとかなり小さい値なので，テストスイートに偏りが生じる可能性は十分に存在する．

4.3.2 アルゴリズムの正当性

3 種類の従来手法の各アルゴリズムの実装は，他言語で記載されているアルゴリズムを参考に，Python 3.6.3

で書き換えた．論文に記載されているアルゴリズムは大まかに記載されており部分が少ないため，我々が実装したソースコードと異なってしまっている可能性がある．その場合も，得られる実験結果に誤差が生じる可能性が存在する．

4.3.3 comFIL の簡易化

本実験では実験の簡易化のため，comFIL で作成する FI 候補の要素数に FI の大きさの値までという制限をかけた．本来であれば，comFIL は FI の大きさに寄らずに考えられうる部分集合全てを元に FI 候補を作成するため，この制限により実験結果に誤差が生じる可能性が存在する．

4.3.4 出現する FI の大きさ

本実験では出現する FI の大きさをランダムテストでは 3，t-way テストでは t と固定した値を用いている．しかし，例えば 2-way テストにおいて FI の大きさが 3 以上のものが出現した場合，これは必ずしも特定できるとは限らない (テストスイートにその FI が含まっていない可能性があるため)．これはテストの成功率に影響を及ぼすと考えられる．

4.4 今後の課題

本実験における今後の課題として，与える FI の大きさなどといったランダム性の強い部分を出来るだけ平等になるような実装をすることが挙げられる．また，本実験では比較する要因として処理時間，成功率，テストの実行回数の 3 つを測定したが，この他にも従来手法の有用性を差別化できる要因を多く探し出すことが挙げられる．

5 まとめ

本論文では，これまでに発表された 3 種類の組み合わせテストによる不具合誘発パラメータ特定の従来手法について，処理時間，成功率，テストの実行回数の結果からの比較を行い，それぞれの有用性について差別化を行った．それぞれの従来手法のアルゴリズムの実装を行い，計 20 種類のテストスイートを用いて測定し，この 3 つの観点に関するデータを収集した．得られた結果から，それぞれの従来手法がどのような状況下において有用性が生まれるかについての結論の考察を行った．

考察の結果，comFIL は FI の大きさに見当がつかず多量のランダムテストケースを扱うとき，FIC は t-way テストや少量のランダムテストケースを扱うとき，IIC は FI の大きさに見当がついており多量のランダムテストケースを扱うときに有用性が生まれることが分かった．

今後の課題として，より正しいデータを得るために，与える FI などといったランダム性のある要因を消去出来るような実装や，有用性をより詳細に示すために他の比較すべき観点の探索が挙げられる．

参考文献

[1] D.H. Wei Zheng, Xiaoxue Wu, and Q. Zhu, “Locating minimal fault interaction in combinatorial testing,” Advances in Software Engineering, vol.2016, pp.1-10, 2016.

[2] Z. Zhang and J. Zhang, “Characterizing failure-causing parameter interactions by adaptive testing,” Proceedings of the 2011 International Symposium on Software Testing and Analysis, pp.331-341, ISSTA ’11, ACM, New York, NY, USA, 2011.

[3] L. S. G. Ghandehari, Y. Lei, T. Xie, R. Kuhn, and R. Kacker, “Identifying Failure-Inducing Combinations in s Combinatorial Test Set,” Proceedings of the IEEE International Conference on Software Testing, Verification and Validation (ICST), pp.370-379, 2012.

[4] J. Czerwonka, “Pairwise testing in the real world: Practical extensions to test case generators,” Microsoft Corporation Software Testing Technical Articles, 2008.

[5] E. Choi, T. Kitamura, C. Artho, A. Yamada, and Y. Oiwa. “Priority integration for weighted combinatorial testing,” Proc. of 39th Annual International Computer Software and Applications Conference, pp. 242-247, 2015.

[6] R. Bryce and C. Colbourn, “Prioritized intersction testing for pair-wise coverage with seeding and constraints,” Information and Software Technology, Vol. 48, No. 10, pp.960-970, 2006.