卒 業 研 究 報 告 書

題目　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

指導教員　水野　修　教授

京都工芸繊維大学　工芸科学部　情報工学課程

学生番号　14122064

氏名　渡辺　大輝

平成30年2月14日提出

不具合誘発パラメータ組み合わせ特定三手法

平成30年2月14日　　14122064 渡辺　大輝

概　　要

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，不具合の個数や大きさ，用いるシステム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．

目次

1. 緒言

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，いわゆ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ない．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のだ．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度が欠けていると世の中に出回った後の不具合発覚に繋がってしまう．ゆえに，テストスイートの種類ごとに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，2016年にWei Zhengらによって提案されたcomFIL（complete Fault Interaction Location）[1]，2011年にZhiqiang Zhang とJian Zhang によって提案されたFIC（Faulty Interaction Characterization）[2]，2012年にLaleh Shikh Gholamhossein Ghandehariらによって提案されたidentifying inducing combinations[3]，この3つの従来手法の実装を行った．そして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，それぞれの有用性

を調査した.

本論文の構成として，第2章では，組み合わせテストに行うにあたって前提となる事項の解説と，3種類の従来手法について，フローチャート図や簡単な例を用いて具体的に解説を行う．第3章では，使用したテストスイートのシステムモデルの解説や，実験結果を示す．第4章では，得られた実験結果を元に考察を行い，研究設問に回答する．また，処理時間，成功率，テストの実行回数の相関関係，実験結果の妥当性と今後の課題について述べる．そして最後に第5章で本論文のまとめを行う.

2. 準備

本章では，実験の前提となる事項と実験に用いた従来手法について説明する．

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を表記する．例えば，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．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3×2×2＝12となる．対して2-wayテストの組み合わせテストなら，表2.2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（注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．

- 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,…）

（注1）：https://github.com/Microsoft/pict

表2.1　パラメータの仕様

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

表2.2　表2.1の2-way組み合わせテスト

test	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

また以後，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．

2.3 従来手法

2.3.1 comFIL（complete Fault Interaction Location）

comFIL（complete Fault Interaction Location）[1]は，2016年にWei 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2.3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2.3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．

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2.4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2.4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2.4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

表2.3　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

表2.4　表2.3のFI候補（comFIL）

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

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2.4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．

最後に，簡単なフローチャート図を図2.1に示す．

2.2.2 FIC（Faulty Interaction Characterization）

FIC（Faulty Interaction Characterization ）[2]は，2011年にZhiqiang Zhang，Jian 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2.5に示す．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，

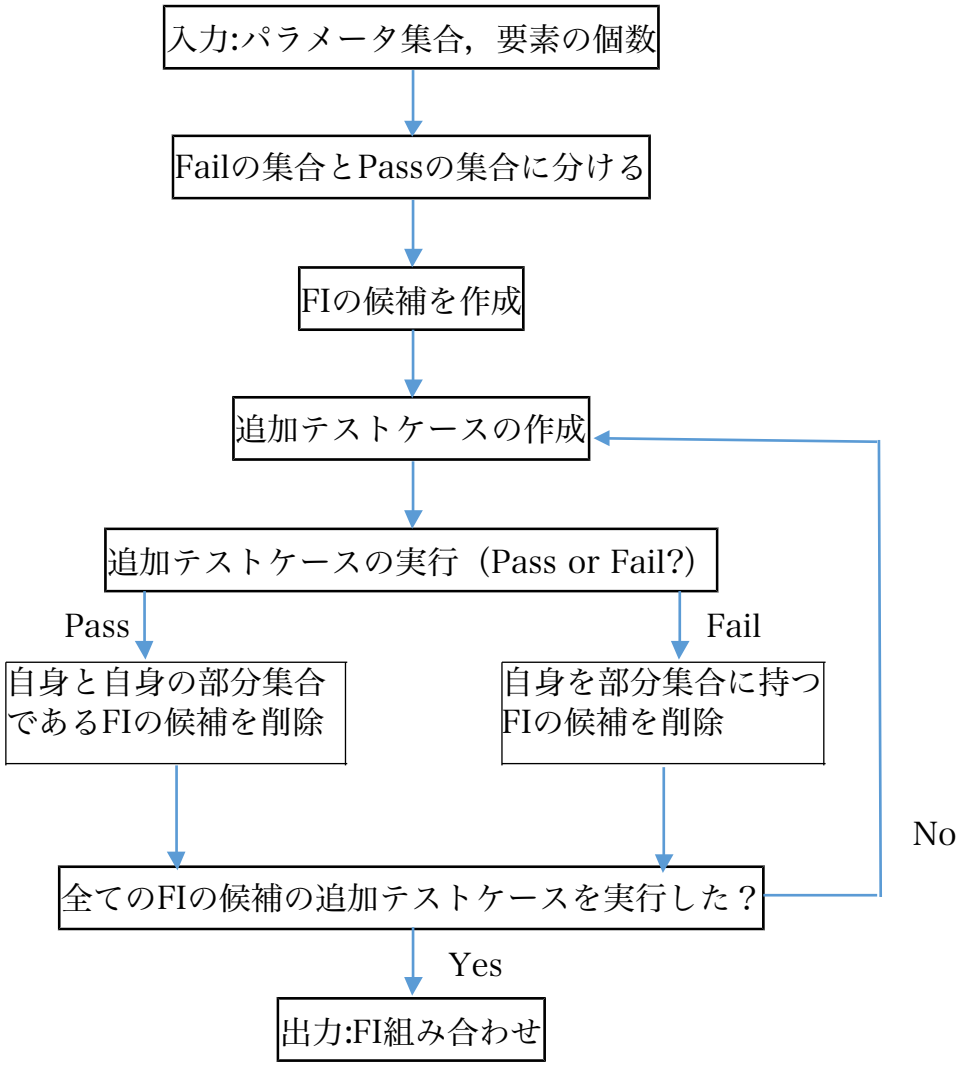


図2.1　comFILのフローチャート図

表2.5　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	2	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

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2.5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．

最後に，簡単なフローチャート図を図2.2に示す．

2.2.3 Identifying Inducing Combinations

Identifying Inducing Combinations[3] は，2012年にLaleh Shikh Gholamhossein 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2.2.1章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2.2.1章と同様に（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2.3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2.6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2.1によって，その疑わしさを値として算出する．

$$\rho(o) = \frac{1}{3}\big(u(o) + v(o) + w(o)\big) \tag{2.1}$$

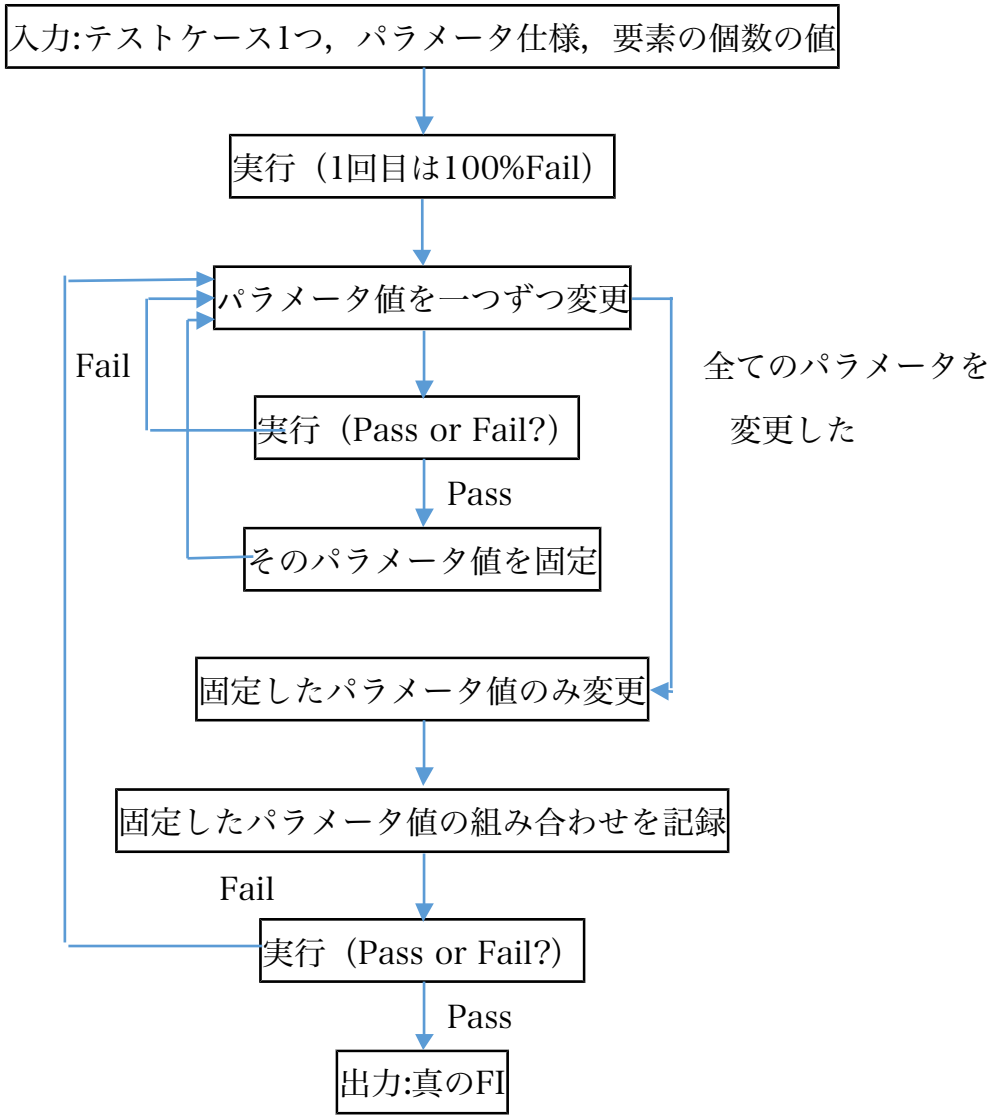


図2.2　FICのフローチャート図

表2.6　表2.3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

ここで，u（o），v（o），w（o）は以下の式で算出される．

$$u(o) = \frac{|\{f \in F_i | r(f) = fail \ \wedge o \in f\}|}{|\{f \in F_i | r(f) = fail\}|} \tag{2.2}$$

$$v(o) = \frac{|\{f \in F_i | r(f) = fail \ \wedge o \in f\}|}{|\{f \in F_i | o \in f\}|} \tag{2.3}$$

$$w(o) = \frac{|\{c | o \in c \ \wedge c \in \pi\}|}{|\pi|} \tag{2.4}$$

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，ρ（c→0）は式（2.5）の通りに算出される．

$$\rho(c \leftarrow 0) = \frac{1}{3} * \left(1 + \frac{3}{4} + \frac{5}{9}\right) = 0.7685 \tag{2.5}$$

同様に，全てのFI候補の構成パラメータについて式2.1の値を算出する．その結果を表2.6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2.6と式2.7で与えられる．

$$\rho_c(c) = \frac{1}{[c]} \sum_{\forall o \in c} \rho(o) \tag{2.6}$$

$$\rho_e(c) = \mathrm{Min}\left(\sum_{o \in f \wedge o \notin c} \rho(o), \forall f \in F \right) \tag{2.7}$$

ρ_cは，あるFI候補の構成パラメータ1つ1つのρ（o）の平均値を取る．例えば，ρ_c（a→0,c→0）は，ρ（a→0）とρ（c→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，ρ_e（a→0,c→0）はρ_c（b→0,d→0），ρ_c（b→1,d→2），ρ_c（b→1,d→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2.7に示す．

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．

表2.7　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

- 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，ρ（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．

最後に，簡単なフローチャート図を図2.3に示す．

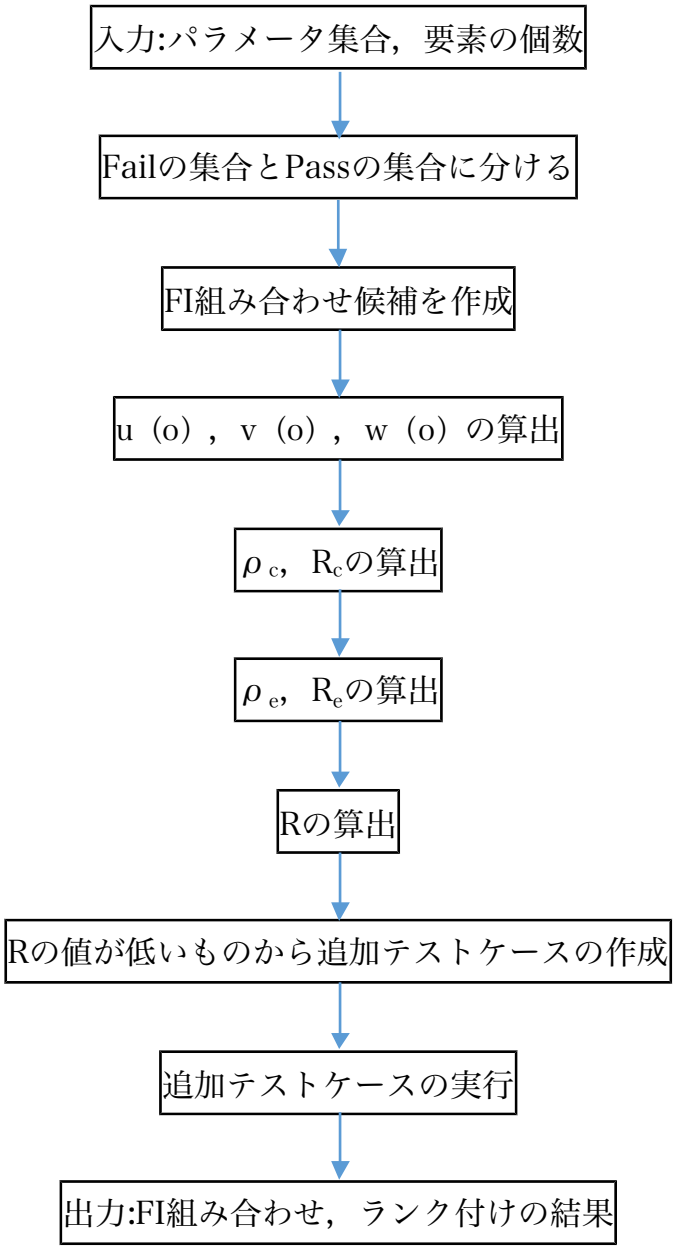


図2.3　FICのフローチャート図

3. 実験

本章では，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
- 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．

また，使用する対象システムモデルを表3.1に示す通りに設定する．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．システムBは3種類の値を取るパラメータを5個，4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．システムCは2種類の値を取るパラメータを10個，3種類の値を取るパラメータを10個，合計20個のパラメータを持つ．システムDは3種類の値を取るパラメータを10個，4種類の値を取るパラメータを10個，合計20個のパラメータを持つ．

3.2.3 実験手順

第2章で明記した通り，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．これらの実装は，それぞれの論文に記載されている他言語のアルゴリズムを元に，全てPython 3.6.3で行った．

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

表3.1　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

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．しかし，FIをランダムで与えた場合，ランダムテストではテストが全てPassする場合が考えられる．それを防ぐため，プログラムを実行する前にあらかじめランダムテストのテストスイートの中にFIが含まれているかどうかを確認し，もし1つも含まれていない場合はFIが含まれるまでFIをランダムに与え続ける．

FIの設定後，各手法のプログラムを実行し，処理時間，成功回数，追加テストケースの実行回数を測定する．これらの手順を1000回ループさせ，処理時間とテストの実行回数は平均値，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，成功率は百分率で値を算出する．

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

結果を表3.2から表3.10，図3.1から図3.6に示す．表3.2，表3.3，表3.4はそれぞれcomFIL の処理時間，実行回数，成功率を表す．表3.5，表3.6，表3.7はそれぞれFICの処理時間，実行回数，成功率を表す．表3.8，表3.9，表3.10はそれぞれIICの処理時間，実行回数，成功率を表す．また，図3.1は表3.2を，図3.2は表3.3を，図3.3は表3.5を，図3.4は表3.6を，図3.5は表3.8を，図3.6は表3.9をそれぞれ箱ひげ図にまとめたものである．

表の1列目には実験に用いたテスト種類，表の2列目にはシステムモデルを示す．例えば，表の2行目は100個のテストケースをテストスイートとしたランダムテストをシステムAを用いて実行した結果の出力である．表の3列目から8列目にはそれぞれ得られた1000個の値の最大値，第3四分位数，中央値，第1四分位数，最小値，平均値を示す．ただし，それぞれの値は小数点第4位を四捨五入した値で示している．また，処理時間の単位は秒とする．

図の横軸は使用したテストスイートの種類を簡易的に表している．例えば，ran100 Aとは100個のテストケースをテストスイートとしたランダムテストをシステムAを用いて実行したということである．

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．

実験結果から見られる通り，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全く結果が得られた．まず，FICについては図3.3と図3.4から分かる通り，ほぼ全てのテストにおいて同一の結果が得られた．comFILとIICについては値の大小は見られるが，図3.1，図3.2，図3.5，図3.6の箱ひげ図を比較して分かる通り，似たような挙動を示していることがわかる．ランダムテストにおいては，使用するテストケースの数が変わることによりcomFILとIICは大きく影響を受けているが，FICはほぼ全く影響を受けていないことが分かる．t-wayテストにおいては，comFILはtの値が大きくなるにつれてわずかに処理時間と実行回数が増加しており，FICはこれについてもほぼ全く影響を受けていない．また，図3.5，図3.6より2-wayテストではIICの分布の範囲が他のテストと比べて大きくなっていることが分かる．また3種類の手法の全てにおいて，実行回数が多ければ多いほど処理時間も大きくなっていることが分かる．また，成功率は全

て100%を記録していることから，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る．

表3.2　実験結果：comFILの処理時間

テスト種類	システム	最大値	第3四分位数	中央値	第1四分位数	最小値	平均値
ran100		8.089	3.431	2.637	1.992	0.602	2.797
ran500		2.665	1.963	1.806	1.473	1.127	1.723
2-way	A	25.007	7.522	5.109	3.206	0.800	5.769
3-way		25.316	10.324	7.120	4.516	0.801	7.710
4-way		24.294	10.681	7.460	4.830	0.904	8.007
ran100		31.165	12.073	8.135	5.215	0.903	8.878
ran500		3.442	2.928	2.710	2.181	1.209	2.580
2-way	B	34.847	15.016	10.911	6.603	2.201	11.310
3-way		21.219	9.743	7.025	4.624	1.003	7.430
4-way		24.709	11.067	7.916	5.280	1.317	8.285
ran100		66.228	17.822	11.964	7.403	1.113	13.758
ran500		8.199	4.753	4.234	3.571	2.769	4.203
2-way	C	70.866	16.521	10.734	6.718	1.202	12.505
3-way		91.478	25.983	17.387	10.496	1.110	12.742
4-way		100.596	34.190	22.784	13.281	1.492	25.705
ran100		246.341	92.285	60.538	35.857	6.838	67.331
ran500		8.269	6.629	6.051	5.243	2.250	5.955
2-way	D	61.548	21.104	14.753	9.235	2.005	15.776
3-way		71.011	30.117	21.369	14.135	2.427	22.856
4-way		81.639	42.945	31.533	19.328	2.973	31.988

表3.3　実験結果：comFILの実行回数

テスト種類	システム	最大値	第3四分位数	中央値	第1四分位数	最小値	平均値
ran100		80	34	26	19.75	6	27.668
ran500		19	18	17	14	11	15.869
2-way	A	249	75	51	32	8	57.565
3-way		252	103	71	45	8	76.839
4-way		241	106	74	48	9	79.410
ran100		310	120.25	81	52	9	88.428
ran500		33	28	26	21	12	24.761
2-way	B	174	75	54.5	33	11	56.503
3-way		211	97	70	46	10	73.971
4-way		243	109	78	52	13	81.512
ran100		651	176	117.5	73	11	135.257
ran500		34	33	32	29	26	31.051
2-way	C	700	164.25	107	67	12	124.434
3-way		899	257	172	104	11	195.257
4-way		945	318	212	124.75	14	239.703
ran100		2391	909.5	599	356	68	664.086
ran500		71	56	53	48	22	52.162
2-way	D	609	210	147	92	20	157.012
3-way		697	297	211	139.75	24	225.418
4-way		692	370	270	167	26	275.066

表3.4 実験結果：comFILの成功率

テスト種類	システム	成功率(%)
ran100	A	100
ran500		100
2-way		100
3-way		100
4-way		100
ran100	B	100
ran500		100
2-way		100
3-way		100
4-way		100
ran100	C	100
ran500		100
2-way		100
3-way		100
4-way		100
ran100	D	100
ran500		100
2-way		100
3-way		100
4-way		100

表3.5　実験結果：FICの処理時間

テスト種類	システム	最大値	第3四分位数	中央値	第1四分位数	最小値	平均値
ran100		3.304	3.301	2.201	1.101	1.100	2.241
ran500		3.310	3.304	2.204	1.103	1.102	2.228
2-way	A	3.303	3.301	2.200	1.100	1.100	2.174
3-way		3.304	3.301	2.201	1.101	1.100	2.204
4-way		3.304	3.301	2.201	1.101	1.101	2.207
ran100		3.329	3.301	2.201	1.101	1.101	2.089
ran500		3.310	3.304	2.203	1.103	1.102	2.167
2-way	B	3.304	3.301	2.201	1.100	1.100	2.235
3-way		3.309	3.301	2.201	1.101	1.101	2.176
4-way		3.310	3.304	2.203	1.103	1.102	2.238
ran100		6.307	6.303	4.202	2.102	2.101	4.192
ran500		6.317	6.306	4.206	2.106	2.104	4.148
2-way	C	6.313	6.302	4.201	2.101	2.101	4.122
3-way		6.307	6.302	4.202	2.101	2.101	4.086
4-way		6.309	6.304	4.203	2.103	2.102	4.052
ran100		6.313	4.208	4.202	2.102	2.101	3.904
ran500		6.322	6.306	4.206	2.105	2.104	4.271
2-way	D	6.311	6.302	4.202	2.101	2.101	4.214
3-way		6.311	6.303	4.203	2.103	2.102	4.140
4-way		6.325	6.309	4.209	2.109	2.106	4.258

表3.6　実験結果：FICの実行回数

テスト種類	システム	最大値	第3四分位数	中央値	第1四分位数	最小値	平均値
ran100		33	33	22	11	11	22.396
ran500		33	33	22	11	11	22.242
2-way	A	33	33	22	11	11	21.736
3-way		33	33	22	11	11	22.033
4-way		33	33	22	11	11	22.055
ran100		33	33	22	11	11	20.867
ran500		33	33	22	11	11	21.637
2-way	B	33	33	22	11	11	22.341
3-way		33	33	22	11	11	21.747
4-way		33	33	22	11	11	22.341
ran100		63	63	42	21	21	41.895
ran500		63	63	42	21	21	41.412
2-way	C	63	63	42	21	21	41.202
3-way		63	63	42	21	21	40.845
4-way		63	63	42	21	21	40.488
ran100		63	42	42	21	21	39.018
ran500		63	63	42	21	21	42.651
2-way	D	63	63	42	21	21	42.126
3-way		63	63	42	21	21	41.370
4-way		63	63	42	21	21	42.483

表3.7 実験結果：FICの成功率

テスト種類	システム	成功率(%)
ran100	A	100
ran500		100
2-way		100
3-way		100
4-way		100
ran100	B	100
ran500		100
2-way		100
3-way		100
4-way		100
ran100	C	100
ran500		100
2-way		100
3-way		100
4-way		100
ran100	D	100
ran500		100
2-way		100
3-way		100
4-way		100

表3.8　実験結果：IICの処理時間

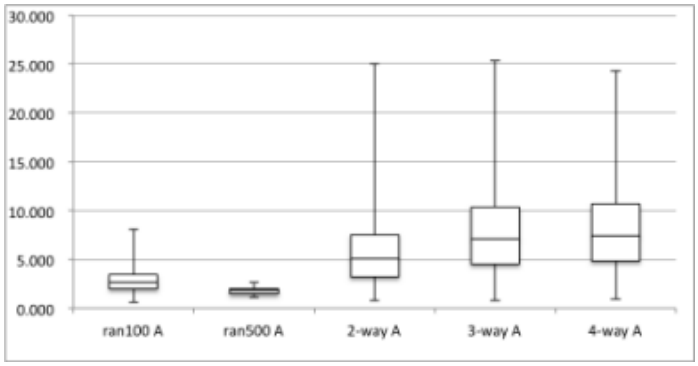
テスト種類	システム	最大値	第3四分位数	中央値	第1四分位数	最小値	平均値
ran100		5.061	1.907	1.204	0.701	0.100	1.391
ran500		2.525	0.389	0.262	0.138	0.107	0.274
2-way	A	14.438	5.853	4.105	2.301	0.100	4.139
3-way		6.212	5.211	4.810	3.003	0.200	4.094
4-way		6.412	5.224	4.826	3.303	0.300	4.238
ran100		6.321	5.213	4.813	3.603	0.200	4.267
ran500		0.746	0.331	0.223	0.115	0.101	0.254
2-way	B	6.913	5.321	3.804	2.003	0.300	3.663
3-way		6.412	5.307	4.807	3.105	0.400	4.174
4-way		6.520	5.318	4.930	3.543	0.200	4.385
ran100		6.428	5.214	4.713	3.746	0.101	4.301
ran500		0.774	0.474	0.323	0.184	0.116	0.338
2-way	C	9.352	5.710	5.212	4.092	0.300	4.702
3-way		6.113	5.124	4.707	4.077	0.501	4.490
4-way		6.056	5.079	4.543	3.804	0.501	4.380
ran100		10.040	5.319	4.320	3.217	0.769	4.324
ran500		0.830	0.228	0.168	0.148	0.110	0.210
2-way	D	6.919	5.714	5.124	4.383	0.601	4.887
3-way		6.217	5.132	4.628	3.977	0.301	4.494
4-way		6.232	5.085	4.251	3.491	0.962	4.202

表3.9　実験結果：IICの実行回数

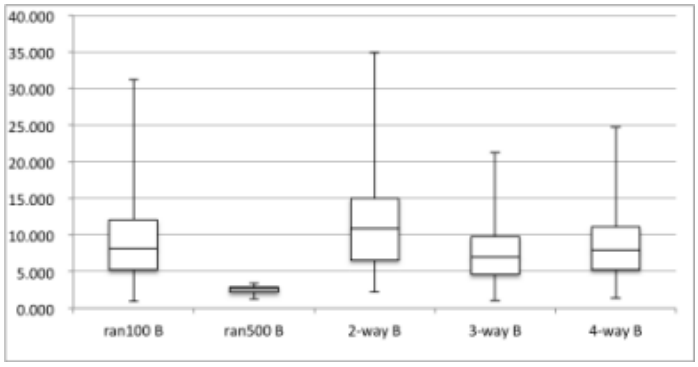
テスト種類	システム	最大値	第3四分位数	中央値	第1四分位数	最小値	平均値
ran100		50	19	12	7	1	13.844
ran500		23	3	2	1	1	2.062
2-way	A	144	58.25	41	23	1	41.321
3-way		62	52	48	30	2	40.828
4-way		64	52	48	33	3	42.253
ran100		63	52	48	36	2	42.520
ran500		7	3	2	1	1	2.323
2-way	B	69	53	38	20	3	36.574
3-way		64	53	48	31	4	41.632
4-way		65	53	49	35	2	43.698
ran100		64	52	46	37	1	42.517
ran500		5	3	2	1	1	2.039
2-way	C	93	57	52	40.75	3	46.752
3-way		61	51	46	40	5	44.143
4-way		60	50	44	36	5	42.532
ran100		100	48.25	35	22	1	35.938
ran500		5	1	1	1	1	1.280
2-way	D	69	57	51	43	6	48.501
3-way		62	51	45	39	3	44.087
4-way		62	50	41	33	6	40.510

表3.10 実験結果：IICの成功率

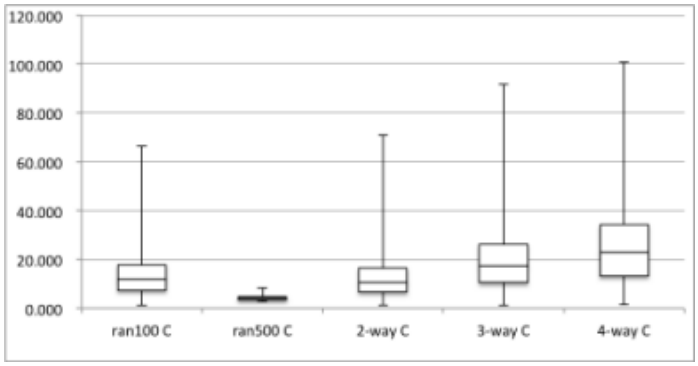
テスト種類	システム	成功率(%)
ran100	A	100
ran500		100
2-way		100
3-way		100
4-way		100
ran100	B	100
ran500		100
2-way		100
3-way		100
4-way		100
ran100	C	100
ran500		100
2-way		100
3-way		100
4-way		100
ran100	D	100
ran500		100
2-way		100
3-way		100
4-way		100



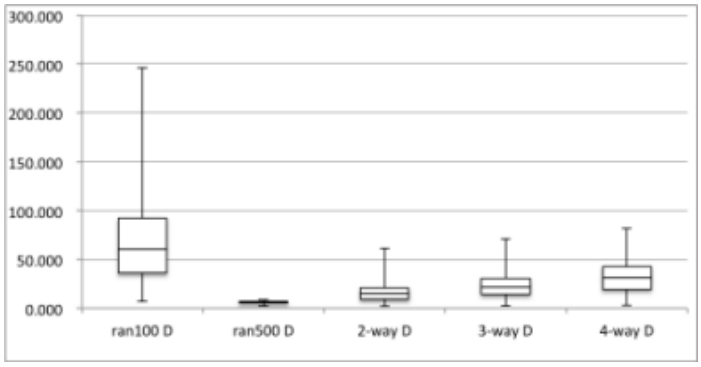
(a)　システムA



(b)　システムB

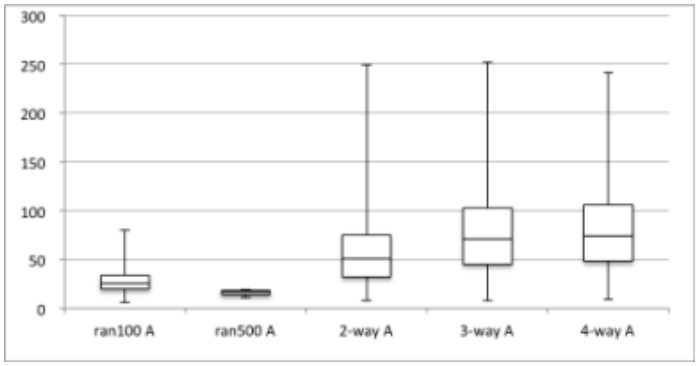


(c)　システムC

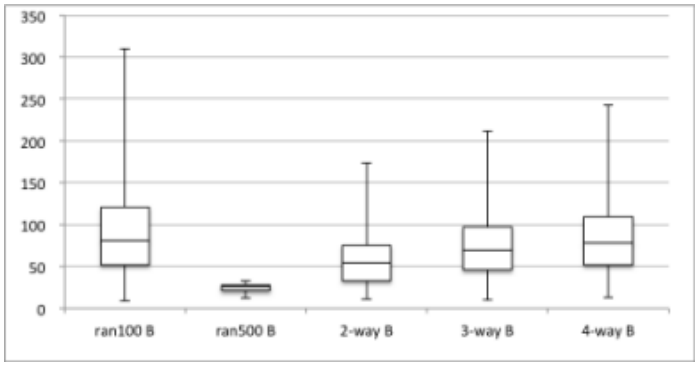


(d)　システムD

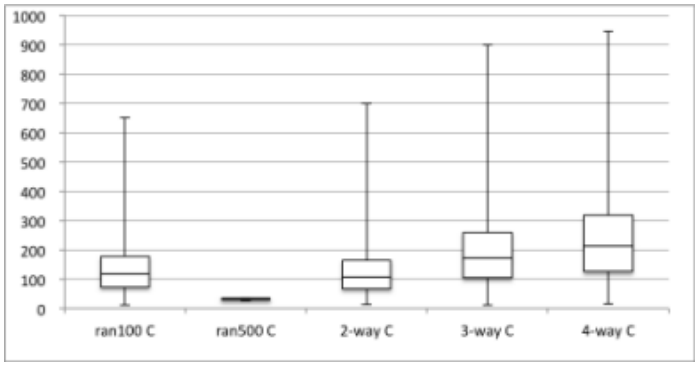
図3.1　実験結果：comFILの処理時間



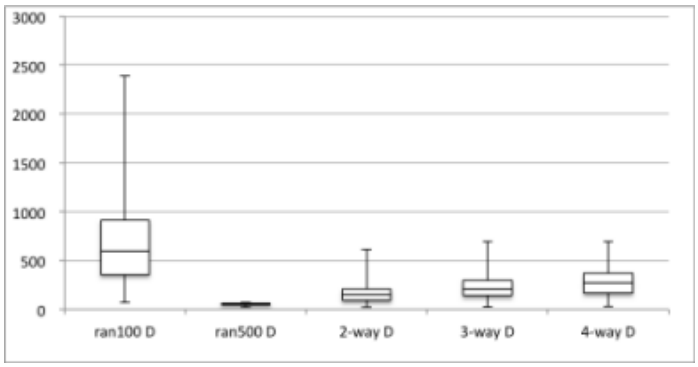
(a)　システムA



(b)　システムB

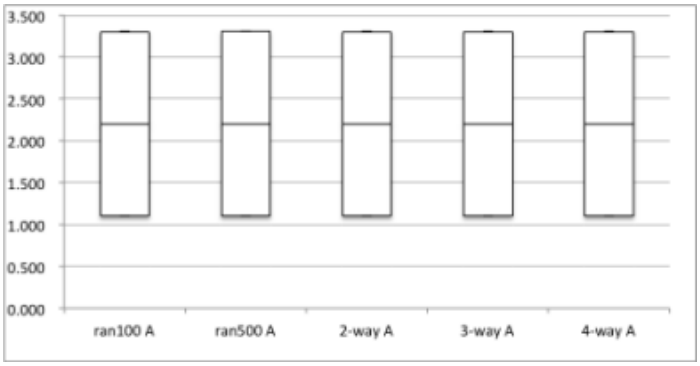


(c)　システムC

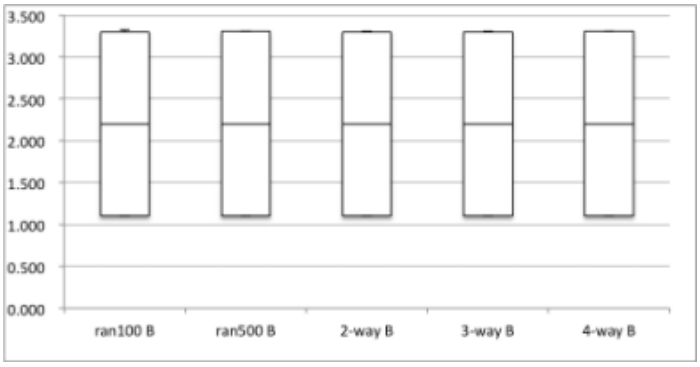


(d)　システムD

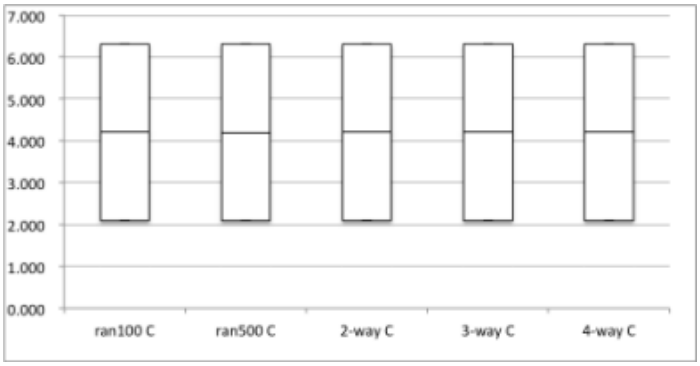
図3.2　実験結果：comFILの実行回数



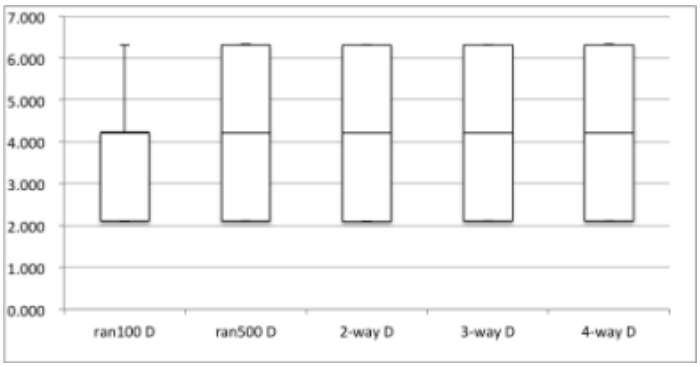
(a)　システムA



(b)　システムB

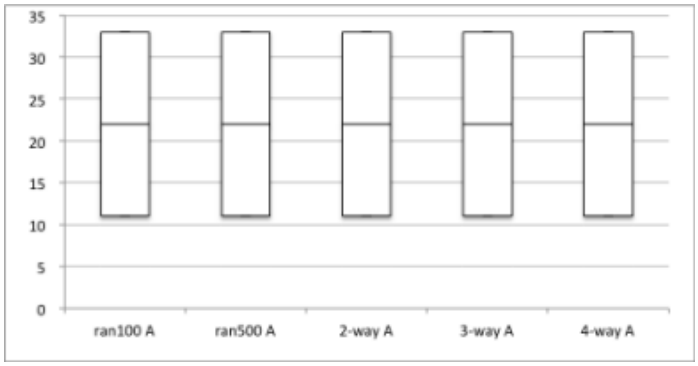


(c)　システムC

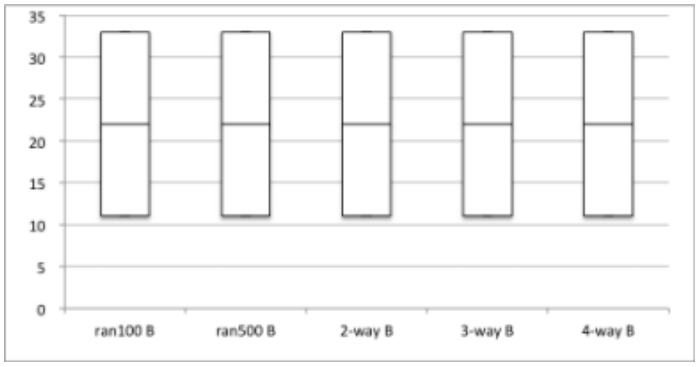


(d)　システムD

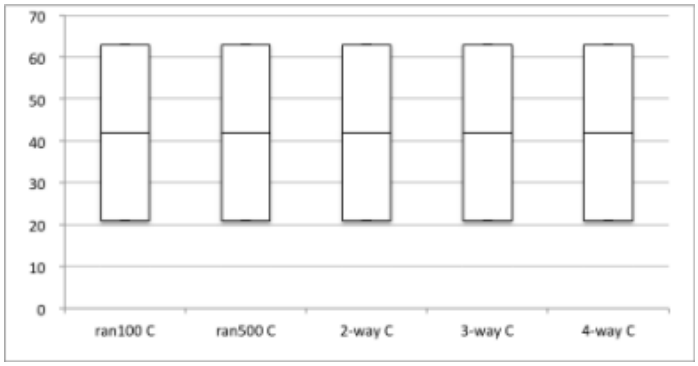
図3.3　実験結果：FICの処理時間



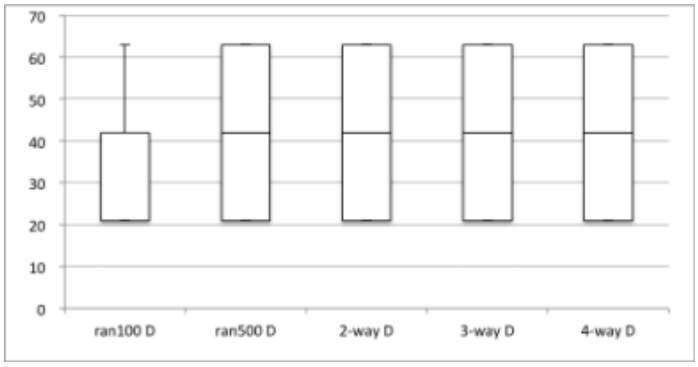
(a)　システムA



(b)　システムB

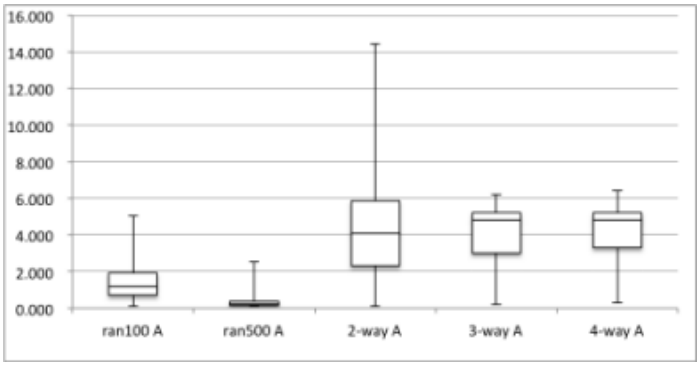


(c)　システムC

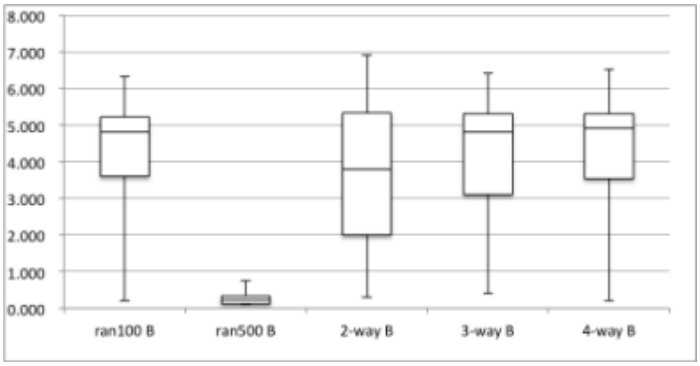


(d)　システムD

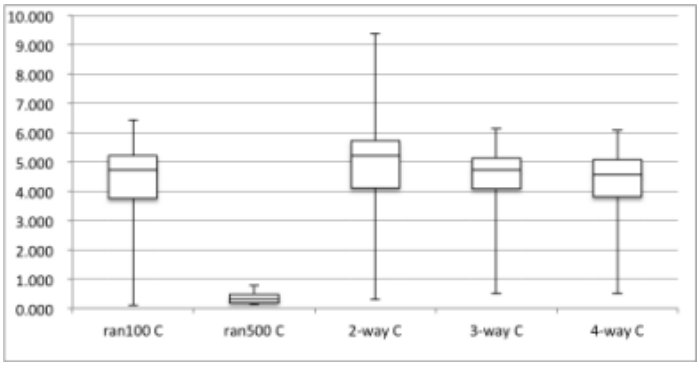
図3.4　実験結果：FICの実行回数



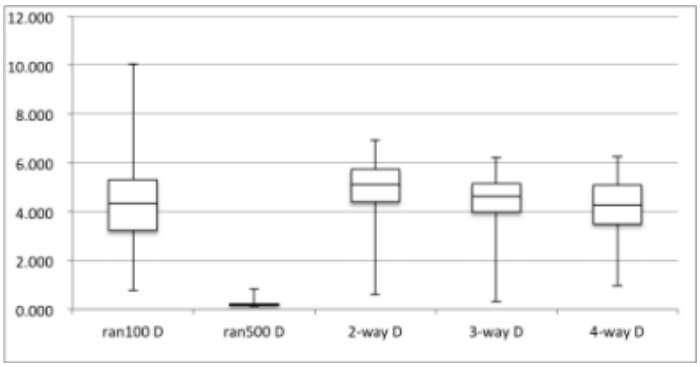
(a)　システムA



(b)　システムB

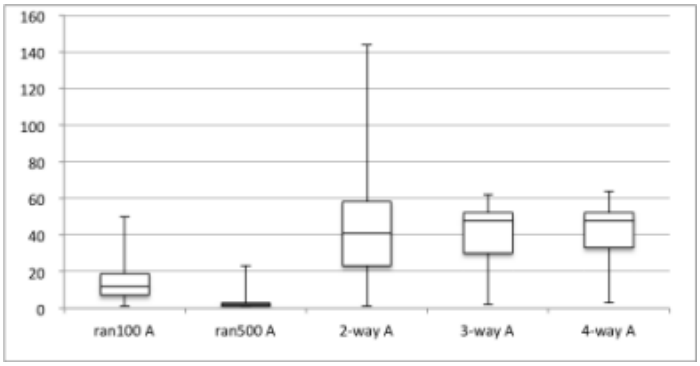


(c)　システムC

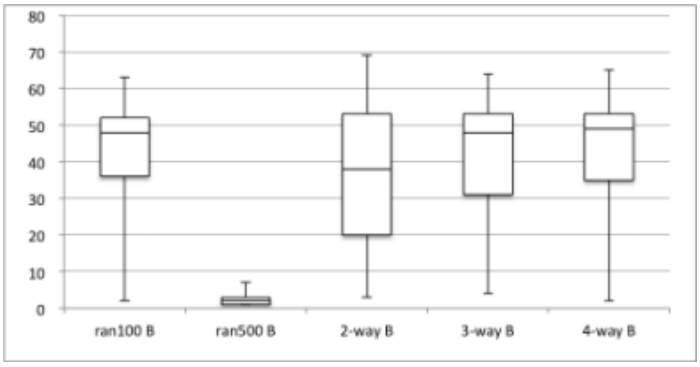


(d)　システムD

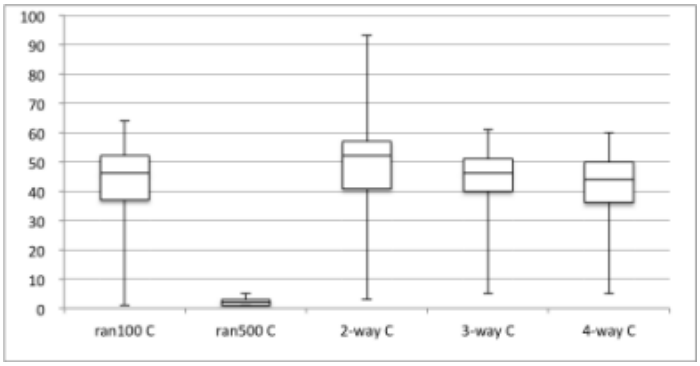
図3.5　実験結果：IICの処理時間



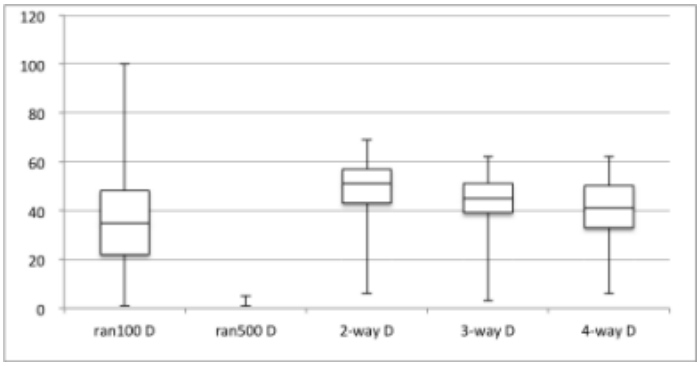
(a)　システムA



(b)　システムB



(c)　システムC



(d)　システムD

図3.6　実験結果：IICの実行回数

4. 考察

本章では設定した研究設問への回答，得られた結果の妥当性について考察する．

4.1 研究設問への回答

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．comFILの実験結果で表3.1，表3.2，表3.3，図3.1，図3.2において考察を行う．まず，ランダムテストについて考える．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テストについて考える．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とシステム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．FICの実験結果である表3.4，表3.5，表3.6，図3.3，図3.4において考察を行う．これらから分かる通り，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とシステムBと比べてパラメータ数が2倍であるシステムCとシステム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．IICの実験結果である表3.7，表3.8，表3.9，図3.5，図3.6を元に考察を行う．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る．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の大きさが事前に分かっている場合に限るため，現実のソフトウェア開発のように必ずしもFIの大きさが事前に分かっている場合では分からない．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の大きさが判明しており，多量のランダムテストケースを扱うとき
- comFIL→FIの大きさが判明しておらず，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の大きさが分かっているとき，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

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 新たなFIの出現

今回使用した3種類の従来手法は，全て追加のテストケースを実行する途中に新たなFIが発生しないことを前提に進めている．本実験では実装の時点で新たなFIが出現する可能性を0にしているが，もし新たなFIが出現する可能性が存在する場合，3種類の従来手法は正しくFIを特定出来なくなってしまう．これにより，正しいデータを測定することが不可能となってしまう．

4.3.3 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．

4.3.4 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得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．

4.4 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 結言

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の大きさが判明しておらず，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の大きさが判明し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．

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，加えて本学情報工学専攻 西浦生成先輩に厚く御礼申し上げます．本報告書執筆にあたり貴重な助言を多数頂きました，本学ソフトウェア工学研究室の皆さん，学生生活を通じて著者の支えとなった家族や友人に深く感謝致します．

參考文獻

[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] Laleh Shiki Gholamhossein Ghandehari, Yu Lei, Tao Xie, Richard Kuhn, Raghu Kacker, “Identifying Failure-Inducing Combinations in s Combinatorial Test Set” Proceedings of the IEEE International Conference on Software Testing, Verification and Validation (ICST), pp370-379, 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.