ソフトウェア工学 133－5
（2001．9．19）

残存エラー数の推定が可能なプログラム試験法 その実施例と分析

若杉忠男
静岡産業大学国際情報学部情報処理コース
会員 6415326
連絡先：〒426-8668 静岡県藤枝市駿河台4-1-1
電話 054-646-5406（ダイヤルイン）

筆者はプログラムをフローグラフで表し，そのパスベクトルという概念を定義してプログラムの複雑度を評価し，パスベクトル法と名付けたが，さらに試験項目による試験ずみパス数という概念を定義し，それでプログラムの信頼度を評価できることを理論的に示した．本論文ではその具体的な方法とその実験結果を紹介する．まずこの方法が成立するためのプログラムに必要な条件，試験条件などを考察し，ついで実験例に基づいて，データ処理方法，エラー総数の推定値の計算法とその推定精度を評価する．最後に学生に書かせたC言語プログラムを，本方法に基づいて評価する．

On a software test method capable of estimating the number of programming errors - An example and analysis -

Tadao WAKASUGI, Member

The author discussed the estimating method of complexity of program by using the new concept 'path vector' of flowgraph, and in the paper, reliability of program by using tested paths number. The paper shows the method concretely. First, necessary conditions of applying the method, estimation of the numbers of program error, and data handling technique are discussed. Then, an example of program testing by beginners of C language is presented, and estimation of number of error is analyzed.

1 はじめに

筆者はフローグラフをパスの集合に分解し分析するパスベクトルという手法を提案した．プログラムの試験については，“Non-exhaustive testing can be used to show the presence of bugs, but never to show their absence.”「完全な網羅試験でなければ，バグのあることは証明できても，バグのないことは証明できない」というダイクストラの有名な言葉があるが[1]，パスベクトル法では，完全な網羅試験を有限個の試験項目で近似し，エラーの数を確率論的に推定する．

試験対象には一般に無限ループが存在する．たとえばコーヒーの自動販売機の試験を考えると，1杯目のコーヒーが無事に出たとしても，2杯続くと紙コップが接触してひっくり返るとか，100杯も出せば原料が切れるとかという事故が考えられる．したがって事故がおきないことを確認するには，無限回のループの実行が必要である．本手法では，無限項の和，1＋1／2＋1／4＋・・が2になることが，はじめの数項の和から推定できるように，無限長で無限個の試験を有限長で有限個の試験項目から推定するものである．

本論文は，論文[2][3]に続きその手法を説明し，エラー総数の推定値の求め方と，その実験結果を紹介する．

2　計算モデル

2．1　プログラムモデル

まず次の定義を行う．

定義1　フローグラフ

フローグラフはノードとリンクからなり，リンクはプログラムのステートメントやプロセス，ノードはステートメントの適当な切れ目や分岐点を表す[4]．

フローチャートとは逆にリンクがプロセスやステートメントを表すことに注意されたい．図1の左にフローグラフの簡単な例を示す．ＡＢＣはノードを，1234はリンクを表す．リンク1は自動販売機の起動処理，2と3はそれぞれコーヒーと紅茶の出力，4は自動販売機のリセット処理である．右図はこのフローグラフの連結行列を表す[4]．このマトリックスのｉ行ｊ列の要素が2ということは，ノードｉからノードｊにリンクが2本あることを示す．図1のマトリックスの2行2列目の要素が2ということは，ノードBにループが二つあることを示す．フローグラフから次に説明するパスベクトルが導かれる．

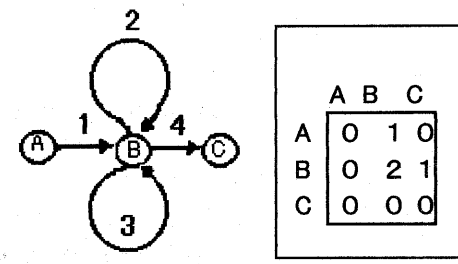


図1　ループが二つあるフローグラフとその連結行列

定義2　パスベクトル

一連のリンクをパスと呼ぶ．パスがＬ個のリンクからなるとき，長さＬのパスと定義する．フローグラフの長さＬのパスの個数をP_Lと記述し，またそれらを長さ順に並べたものをパスベクトルと呼ぶ．ただし同じパスがあるときは重複しては数えない．

なおここでは，プログラムのフローグラフは大文字で$\{P_L\}$，それを試験項目でカバーした部分集合を小文字で$\{p_L\}$などと記述する[2]．

たとえば図1でコーヒーを3杯出す試験項目として12224を考える．この数字は試験項目が通るリンクの番号で，リンクは1，2，2，2，4の5個からなる．長さ1のパスは，リンク2が重複するので省くと，1，2，3の3個からなる．また長さ2のパスは12，22，24の3個であり，以下同様にして長さ3のパスが3個，長さ4は2個，長さ5は1個であるから，試験項目のパスベクトル＝$\{p_L\}$＝{3，3，3，2，1}となる．

一方，図1のフローグラフ全体のパスベクトルでは，長さ1のパスは4個（1，2，3，4），長さ2のパスは9個（12，13，14，22，23，24，32，33，34），以下同様にしてフローグラフのパスベクトルは{4，9，18，・・，2.25×2^L，・・}で，Ｌ＞1なるＬについて2.25×2^L個となる．一般にＬが大になると$\{P_L\}$には簡単な規則性が現れる[2]．

本手法では，試験項目のパスベクトルでプログラム全体のパスベクトルの一部をカバーし，全体をカバーしたのに等しいエラー数を推定する．そのために，次のような前提条件を考える．

前提1　対象とするプログラムの条件

（1）プログラムはフローグラフで表される．

（2）プログラムには摩耗や劣化がなく無限に繰り返すことができ，リセットすれば完全に初期状態に戻るものとする．

（3）フローグラフのすべてのパスは実行でき，またループは無限回実行できるものとする．

実際には条件によっては通らないパスもありうるが，すべてのパスを試験するものとする．

2．2　エラーモデル

エラーについては次のように考える．

定義3　エラー

プログラムをテストした結果，生じた仕様書と一致しない現象をエラーと呼ぶ．また，作業者がプログラムの使いにくさなどもエラーと考えるならば，それはエラーとする．

前提2　エラー発見率rとエラー数f

あるリンク内にあるエラーが，試験時にそのリンク内で発見される割合をエラー発見率と呼びrと記す．rは定数とするが，平均値がrとなる確率変数と考えてもよい．またリンク内のエラーの数もリンクにかかわらず一定のf個とする．

2．3　試験モデル

前提3　試験の方法

一つの試験項目を走らせて，エラーを発見した場合にはそのエラーを除去し，再度同じ試験項目を走らせてエラーがなくなったことを確認してから次の試験項目を走らせる．

3　試験ずみパス数

前提3のもとで，図1の例を使って説明する．リンク1，2，4を通る試験項目を考えると，リンク4にあるエラーf個の発見期待値はf×rとなる．またリンク4の一つ前のリンク2で見逃してリンク4で見つかったエラーの期待値はf×(1－r)×rとなる．同様にリンク4の2個前のリンク1にあってリンク4で発見されるエラーは，f×(1－r)^2×rとなる．したがってこの例の場合，リンク4の発見エラー数の期待値はその合計でf×r×{1＋(1－r)＋(1－r)^2}となる．

一般に，長さLのパスではエラー発見期待値は

$$= f \times r \times \{1 + (1-r) + (1-r)^2 + (1-r)^3 + \cdots + (1-r)^{L-1}\}$$

$$= f \times r \times (1 - (1-r)^L) \qquad (1)$$

となる．(1－r)＜1であるから(1)式はL＝無限大で収束する．

また試験をパスの長さLで打ち切った場合，

$$\text{残存エラー数} = f \times (1-r)^L \qquad (2)$$

となる．

発見されるエラーの数は，リンクにあるエラーの数fと，試験項目がカバーするパスの長さLと，エラー発見率rとに依存し，その期待値は(1)式で，見逃したエラー数は(2)式で決まる．(1)式のfを除いた項はフローグラフのパスをカバーした程度を表すので，試験ずみパス数と呼ぶ．

試験項目によるエラー発見期待値は，試験項目のリンクすべてについて(1)式を重複部分を除いて加える．図1の例で，コーヒーを3杯出す試験項目の場合，リンクは1，2，2，2，4というパスで表せる．エラーはリンク1，2，4で発見されるから，エラー発見期待値は次のようになる．

リンク1で見つかるエラーの数＝f×r

リンク2で見つかるエラーの数＝f×r×

{1＋(1－r)＋(1－r)^2＋(1－r)^3}

リンク4で見つかるエラーの数＝f×r×

{1＋(1－r)＋(1－r)^2＋(1－r)^3

＋(1－r)^4}

したがって,
試験項目１２２２３のエラー発見期待値＝

$$f \times r \times \{3+2(1-r)+2(1-r)^2 +2(1-r)^3+(1-r)^4\} \quad (3)$$

(3) 式の f を除いた部分は試験項目がパスをカバーする程度を表すから，これを試験ずみパス数と呼ぶ．またこの試験項目のパスベクトルは {3，2，2，2，1} である．

一般に試験項目でカバーしたパスベクトルを $\{p_1, p_2, p_3, \cdots p_L\}$ とすると，試験ずみパス数は，

$$= \sum_{i=1}^{L} p_i \times r \times (1-r)^{i-1} \quad (4)$$

で求められる．

前提２によって，エラー f がリンクに平均的に分布している場合には，(3)式から分るように試験ずみパス数とエラー数は比例する．したがって試験ずみパス数を求めればエラーの数が推定できる．

4． r の求め方

エラー発見率 r を求める方法を説明する．試験項目１，２，３・・の順に実施して，発見したエラーの数を $e_1, e_2, e_3, \cdots$，各々の試験ずみパス数を c_1, c_2, c_3，・・・とすると，前提２から，e_i / c_i は試験項目にかかわらず一定になる．この定数をKとすると，

$$e_1/c_1 = e_2/c_2 = \cdots e_i/c_i =$$

$$\Sigma e_i / \Sigma c_i = K \quad (5)$$

が近似的に成り立つ．このKは試験ずみパス数あたり発見エラー数である．よって最小自乗法で，

$$\sum_{I=1}^{\text{試験項目数}} (e_i/c_i - K)^2 \geqq 0 \quad (6)$$

となる適当な r を０＜ r ＜１の範囲から選んで計算し，そのうちで上式の値をもっとも小さくする r を試行錯誤で求める．後で述べる実例では，(6)式にウエイト c_i を掛け，

$$\sum_{I=1}^{\text{試験項目数}} (e_i - c_i \times K)^2 \geqq 0 \quad (7)$$

$$K = \Sigma e_i / \Sigma c_i \quad (8)$$

とし，適当な r をいくつか選んで c_i を求め，まず(8)式でKを計算し，それを(7)式に代入してもっとも小さくなる r を選んだ．

5 エラー総数の推定

残存エラー数の求め方は，まず r を前章の方法で求める．プログラムの総試験ずみパス数を C_∞ とし，$\{P_L\}$ をそのパスベクトルとすると，次の式が成り立つ．

$$C_\infty = \sum_{L=1}^{\infty} P_L \times r \times (1-r)^{L-1} \quad (9)$$

(9) 式は無限の長さをもつが，収束条件が満されるならば，連結行列を使って近似的に求めることができる[2]．収束条件とは次のようなものである．

① P_L の増加率が小さいこと：P_L は一般にLが大になるほど大きくなる．特にループや分岐命令があると大きくなる．言いかえればプログラムが複雑なほど収束しにくい．
② r が大きく１に近いこと：技術者のレベルが低く，r が０に近いと収束しにくい．

従来からの考えでは，プログラム内のエラーの数は一定で，除去すれば減っていくとしているが，本理論では，エラーの数はデバッグ技術者の技術レベルによって増減し，場合によって

はエラーの数が発散することもあるとする.

(9)式で求めたC_∞を使えば,

エラー総数 : C_∞=

発見エラー数 : 試験ずみパス数　　(10)

となる. したがって,

エラー総数

=C_∞×発見エラー数／試験ずみパス数

=$C_\infty \times K$　　(11)

で求められる. ただし, 先にも述べたように, 連結行列ではすべてのリンクを試験項目が通ると仮定しているが, 実際のプログラムでは条件によって通らないこともあるので, 過大な見積もりとなる. よってC_∞はエラーの上限値であり, 次式が成り立つ.

残存エラー数≦$C_\infty \times K$

－現時点の発見エラー数　(12)

6. エラー総数の推定精度

次に(11)式のKの誤差を考える. エラーがフローグラフ内に偏って分布している場合, すなわち前提2を満たさないとどうなるかである.

エラーの分布の偏りを知るために, 発見エラー数／試験ずみパス数　e_i / c_i を考える. 共通のパスは除外してあるので互いに独立と考え, その平均を求める. 試験ずみパス数c_iには大小があるので(7)式の考えにそろえてウエイトc_iを掛けてから現時点での総試験ずみパス数で割ると, Kと一致する. すなわち,

$$\Sigma \{c_i \times (e_i / c_i)\} / \Sigma c_i$$

$$= \Sigma e_i / \Sigma c_i = K \qquad (13)$$

同様に (e_i / c_i) の分散は,

$$\Sigma \{(c_i \times e_i / c_i) - c_i \times K\}^2 / (\Sigma c_i)$$

$$= \Sigma \{(e_i - c_i \times K)^2\} / (\Sigma c_i) \qquad (14)$$

(14)式の分子は, 試験作業が順調に進行すればe_iもc_iも0に近づく. 一方, 分母は試験済みパス数の和で, 試験作業が進行するにつれて単調に増加する. したがって(14)は試験が順調に進行すると単調に小さくなり, Kの推定精度は上がってゆく.

7　パスベクトル法の実験例

本方法の実験例として, C言語をはじめて学習した２４人の学生が作成した２４個のプログラムを対象に, 本手法によってエラー数を推定した例を示す. 課題にしたがってプログラムを作成し, コンパイルがすんだ時点で, 3個の試験項目を順番に与えてその出力が正しければ合格とし次に進む.

7.1 課題

学生に与えた課題は次のようなものである.

「次の実行例のように段数を入力し, それに対応する長方形を「＊　＊」で作成するプログラムを作成せよ. ただし, 段数として正の整数が入力されるまで繰り返すようにすること. （ｄｏ－ｗｈｉｌｅ文を使用する）.

入力データと出力データは次のようなもので, 出力は太字で, 入力はアンダーラインで示してある.

＊試験項目①

何段ですか（正数入力）

0

＊試験項目②

何段ですか（正数入力）

```
  1
  * *
＊試験項目③
何段ですか（正数入力）
  3
  *  * * *
  * *  * *
  * * *  *                    」
```

7．2 模範プログラム例

```
    #include<stdio.h>
      int main(void)
①   {
        int a,i,j;
②               do{
    printf("何段ですか(正数入力):");
     scanf("%d",&a);
     } while(a<=0);
③     for(i=1; i<=a; i++){
④       for(j=1; j<=i; j++)
⑤               putchar('*');
                putchar(' ');
    }
⑥             for(j=a-i+1; j>=1; j--)
⑦               putchar('*');
⑧               putchar('¥n');
              }
⑨             return(0);
    }
```

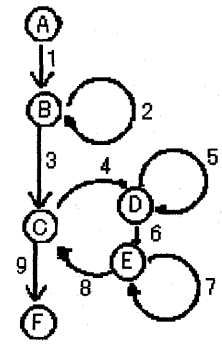
7．3 フローグラフの作成

(1)プログラムのフローグラフを描く．フローグラフはプログラムのすべての実行文からなる．C＋＋言語を例にとって説明すると，

各リンクの始点は，①メインプログラムの始点，②while文，if文，for文などの分岐文の次，③いくつかのリンクの合流点の次，④メンバ関数の呼び出し文，ポインタ変数の呼び出し文の次などである．したがってリンクの終わりは，①各プログラムの終わりかreturn文，②分岐文の終わり，③リンクの合流点，である．

(2) 次にリンクのすべてに通し番号をつける．つける順序は分かりやすければよい．

実例を図２に示す．



	A	B	C	D	E	F
A	0	1	0	0	0	0
B	0	1	1	0	0	0
C	0	0	0	1	0	1
D	0	0	0	1	1	0
E	0	0	1	0	1	0
F	0	0	0	0	0	0

図２　プログラムのフローグラフとその連結行列

7．4　試験項目の作成

試験項目１，２，３について，実施順に通過するリンクの番号を記述する．

項目１：1,2

項目２：1,2,3,4,5,6,7,8,9

項目３：1,2,3,4,5,6,7,7,7,8,4,5,5,6,7,7,8,
4,4,4,5,6,7,8,9

7．5　パスベクトルの作成

試験ずみパス数は(4)式から求める．試験項目全体のパスを一つの表にまとめ，重複するパスがあれば一つを残して削除する．そしてパスを長さ別に分類した表を作り，パスベクトルを求める．そのような処理は人手では大変であるが

小規模なものならパソコンで作れる.

試験項目１

｛２，１｝

試験項目１＋２

｛９，８，７，６，５，４，３，２，１｝

試験項目１＋２＋３では

｛９，１１，１４，２０，２０，２２，２１，２０，１７，１６，１５，１４，１３，１２，１１，１０，９，８，７，６，５，４，３，２，１｝

7.6 課題の出力結果

試験結果がエラーかどうかを次のように定めた.

結果１：試験項目１を実行させても何も出力しない. あるいは，データ入力指示のメッセージは出して，次に進まない. あるいは止まらない.

結果２：試験項目１は正しく出力したが，試験項目２を入力すると，正しい表示が出ない. あるいは止まらない.

結果３：試験項目３を入力したが，正しい表示をしないか，止まらない.

結果４：すべてのテストケースを正しく出力した.

7.7 発見エラー数の補正

この実験では多数の人が同じ問題を試験した結果の分析であるから，これを一人が実行したように補正した. その方法を述べる.

まずエラー数は失敗したプログラムの個数と等しいとした. また２４人の学生のエラー生成数ｆとエラー発見能力ｒがすべて同じと考え，１項目めの発見エラー数の期待値と発見したエラー数e_iに比例するとすれば，１項目めはプログラムが２４個だから，発見エラー数は一人当たりの２４倍と考えた. したがって

$$f \times r \times c_i = e_i / 24 \qquad (15)$$

となる.

２項目めについては，１項目めに３個のプログラムがパスできなかったので，残り２１個のプログラで試験を続けた. したがって，発見エラー数は２１で割った，同様に，３項目めのエラー数は７で割った. 以上の結果を表１にまとめてある.

7.8 エラー発見率 r と試験ずみパス数

試験ずみパス数とエラー発見数が比例するようなｒを求める，ｒをいくつか選び，(7)式が最も小さくなるｒを試行錯誤で小数点以下２桁まで求めた. 図３にｒを変化させた場合の試験ずみパス数と発見エラーとの関係のグラフを示す. この図ではｒ＝０．４でほぼ直線になり，試験ずみパス数とエラー数が比例している. この実験では，より精密に分析して，ｒ＝０．４４を採用した. r=０．４４とした場合の試験結果の分析データを表１に示す.

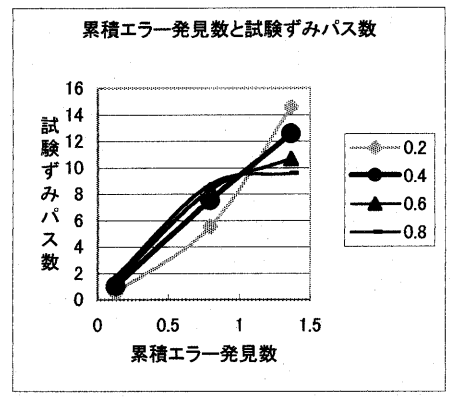


図３ 発見エラー数と試験ずみパス数 エラー発見率ｒを変えた場合

試験項目番号	累積試験ずみパス数	失敗プログラム件数	一人当たりエラー数	累積発見エラー数
1	1.13	3	0.125	0. 125
2	7.73	14	0.667	0. 792
3	12.13	4	0.571	1. 363
計		21	1.363	

表１ 実験結果のまとめ

7.9 エラー総数の推定

r＝0．44を使って (9) 式からC_∞を求めてみる．パスベクトルは図2の連結行列から求められる．すなわち，長さLのパスの数は，連結行列$^{L-1}$の要素の合計値である．これをL＝5まで求めると，表2のように，P_Lの増加率はほぼ1．75となり，$P_L=9\times1.75^{L-1}$と近似すると (9) 式は等比級数の和になるので，C_∞＝198と求められる．

L	1	2	3	4	5
P_L	9	16	28	48	84
P_Lの増加率		1.78	1.75	1.71	1.75

表2　パスベクトル $\{P_L\}$

エラー総数の上限は(11)式から22．2個となる．参考として，rをいろいろ変えたグラフを図4に示す．同じプログラムでも試験する人のレベルが低いとエラーが増加することを示している．極端にrが小さいと，エラーの数はいくら試験をしても減らないとことが言える．

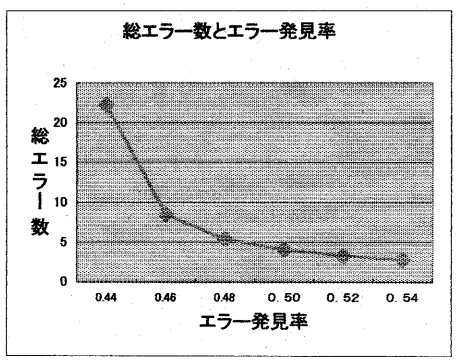


図4　エラー発見率と総エラー数の関係

8　まとめと今後の計画

パス分解法に基づくプログラム試験とエラー数の推定の理論を述べ，学生の作ったプログラム例の分析を行った．わずか3個の試験データの結果の分析であるが，本手法の有効性を示したと考えている．しかしエラー数の推定誤差や技術者のレベルによるエラー数に対する影響などを知るために，本方法をもっと多くの問題に使ってみる必要がある．エラーの数は技術者のレベルによって変わるとか，発散することがあるなどと，従来の常識に反する理論であるせいか，なかなか認めてもらえない．広く協力者を求めて，実験を続けたいと思っています．

参考文献

[1] Dijkstra, E.W.:On a Political Pamphlet from the Middle Ages, ACM SIGSOFT Software Engineering Notes 3-2,14-18,1978.

[2]若杉忠男：フローグラフや状態遷移図の特性のパス数による分析，情報処理学会論文誌，第40巻，第2号，pp.742-749(1999-2)

[3]若杉忠男：残存フォールト数の推定が可能なソフトウ ェア試験法について(5)ー信頼度成長曲線ー，情報処理学会ソフトウェア工学研究会報告,99-SE-124-4,(1999-10).

[4] Beizer,B.: Software Testing Techniques, P442, 小野間,山浦訳,日経ＢＰ出版センター（1994）.

[5] 石原辰雄，BASICによる統計，共立出版，1984.7

[6] Dijkstra:Goto Statement Considered Harmful, CACM 11-3, 147-148,1968.

[7] Myers, G. J.: Reliable Software through Composite Design, Marson/ Charter Publishers, 1975.