2003－SE－142 (2)
2003／5／30

フローの分析によるプログラム残存エラー数の推定

若杉忠男
元静岡産業大学国際情報学部

筆者はプログラムをフローグラフで表し，そのパスの数を使った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、これをパスベクトル法と名づけた[2]．本論文では，新たに試験項目による試験ずみパス数という概念を定義し，それを使ってプログラムの試験過程をシミュレートする方法を提案する．まずリンク当たりエラー発見率を適当に選べば試験ずみパス数と発見エラー数とは比例し，試験ずみパス数からプログラムのエラー総数を推定できることを理論的に示す．この方法が成立するためのプログラム条件や試験条件を考察し，ついで学生に書かせたＣ言語プログラムの例を使って，パスベクトルの求め方，エラー総数の推定方法を具体的に述べ，本方法の有効性を示す．

Estimation of number of residual programming errors using flow analysis method

Tadao WAKASUGI

In another paper, the author discussed a method estimating the complexity of programs by using a new concept, a ‘path vector’ of flowgraph, and called the method ‘the path vector method’. This paper proposes a method that defines a new concept ‘tested paths number’. By selecting the proper rate of error detection, proportional relations between the numbers of detected errors and the numbers of tested paths are gained, and number of errors can be estimated. The paper gives the necessary program conditions for the method, and the concrete process of application of the method to the program developed by beginners of C language. Developing method of path vector, estimation of the total numbers of program errors, and the results of the analysis are discussed.

１． はじめに

筆者は以前にフローグラフをパスの集合に分解し分析する手法を提案し，パスベクトル法と名づけた[2]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[2]では完全な網羅試験を有限個の試験項目で近似することをこころみる．すなわち，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，無限個の試験項目を有限個の試験項目で推定する．

本論文では，論文[2][3]に続きその手法を説明し，エラー総数の上限の求め方と，その実験結果を紹介する．

〒251-0033,神奈川県藤沢市片瀬山3-11-1,
電話0466-23-4832,
E-ﾒｰﾙ:wakasugi8@jcom.home.ne.jp
3-11-1 Katase-yama,Fujisawa-city,Kanagawa pre., 251-0033 Japan

２． 計算モデル

プログラムモデルについて，まず次の定義を行う．

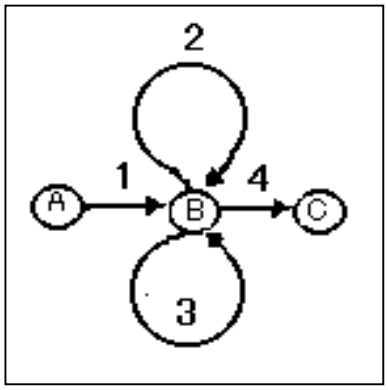
定義１ フローグラフ

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，ノードはステートメントの連結点を表す[4]．

Ｃ言語的な自動販売機プログラムの例を示す．

```
A  コーヒーI杯か紅茶J杯かの指示      (1)
B  {IF（I＞０？）  コーヒーをI杯出す  (2)
     ELSE  IF（J>0？）紅茶をJ杯出す(3)
     ELSE 何もせず              }
C  後処理                          (4)
```

フローグラフは図１の上図のようになる．矢印 1234はリンクを，ABCはノードを表す．フローチャートと違い，フローグラフではリンクがステートメントを表しその内容は省略される．図１の下の連結行列はこのフローグラフの連結状態を表す．行列の２行２列目の要素が２ということは，ノードBから出てBに戻るリンクが二つあること，すなわちループが２つあることを示す．フローグラフについての詳細は文献[4]などを参照されたい．



連結行列

	A	B	C
A	0	1	0
B	0	2	1
C	0	0	0

パスベクトル

$\{P_L\} = \{4, 9, 18, \cdots, 2.25\times 2^L, \cdots\}$

$\{P_{L+1} / P_L\} = \{2.25, 2, 2, 2, \cdots\}$

図１　ループが二つあるフローグラフその連結行列とパスベクトルの例

Fig.1 Sample of connection matrix and flowgraph that has 2 loops.

フローグラフから次のパスベクトルが導かれる．

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．

ここで，プログラム全体を表すフローグラフのパスベクトルは大文字で$\{P_L\}$，それを試験項目でカバーした部分集合を小文字で$\{p_L\}$と記述する．パスの各要素の値はプログラムにループやブランチが多いと大きくなる．よってパスベクトルの各要素はプログラムの複雑度を表す指標と考えられる[2]．

数え方は次のようにする．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（1，2，4）の３個，長さ２のパスは（12，22，24）の３個である．22というパスは2個あるが重ねては数えない．同様に長さ３は（122，222，224）の3個，長さ４は（1222，2224）の２個，長さ５は（12224）の１個である．したがって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を大文字を使って$\{P_L\}$と表すと，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$\{P_L\}$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．P_Lは連結行列をL乗してその要素の値を合計して求められる．

Ｌが大になるとき，パスベクトルの増加率をP_{L+1}/P_Lとおくと，一般に簡単な周期性が現れる[2]．図１の例では，{2.25，２，２，２，・・・}となる．

パスベクトル法の考えでは，プログラムを試験するとは，プログラム全体のパスベクトル$\{P_L\}$の中の試験項目のパス数が増えるように試験項目パスベクトル$\{p_L\}$を選択することと考える．またフローグラフを水の流れにたとえ，水源池がエラーで汚染された場合，河川の形状が複雑で入り組んでいれば，すなわち$\{P_L\}$が大ならば汚染の拡散は防ぎにくい．またレベルの高い技術者がいれば汚染を早期に発見して消毒剤をまき拡散を防ぐことができるが，レベルの低い技術者，すなわちエラー発見率ｒが小さい場合は見逃して被害を大きくしてしまうだろう．このモデルを“汚染モデル”と呼ぶ．ｒの厳密な定義は後述する．

プログラムの前提条件は次の通りである．

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)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，本論文ではすべてのパスを試験できるものと仮定する．

(3) 各リンクに含まれるエラーは同程度とする.

また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．

本論文ではステートメントの２重定義などは，実害がなければエラーとは考えない．

前提２　試験方法

厳密には，試験は試験項目をひとつづつ走らせて，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に進む．またプログラムの形を変更した場合には，始めの試験項目から試験をやり直す．

３．基本式

パスベクトル法の基本式は次のようなものである．

定理１　パスベクトル法の基本式

リンク i 内にあるエラーの個数を f i個とする．また試験時にエラーがリンク内で発見される割合をエラー発見率と呼び r i と表す．f iと r iを互いに独立な確率変数とし，その平均値をそれぞれ f と r とする．試験項目のパスベクトルを $\{p_i\}$ とすれば，その試験項目による発見エラー数の期待値は次の式で表される．

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

証明

リンク１に含まれるエラーの数を f_1 個，エラー発見率を r_1 とすると，リンク１でのエラーの発見数は $f_1 \times r_1$ となる．そのリンクで見逃したエラーが下流の次のリンクで発見される期待値は，エラーを１回見逃した後に発見する値であるから，$f_1 \times r_2 \times (1-r_1)$ となる．以下同様にして，リンク１を先頭にもつ長さＬのパスの発見エラー数の期待値は，それぞれ０，１，２，・・・Ｌ－１回見逃してから発見する確率であるから，$f_1 \times (1-r_1) \times (1-r_2) \times \cdots \times (1-r_{L-1}) \times r_L$ となる．f_i は，平均値が f で互いに独立な確率変数で，r_i も，平均値 r の独立な確率とすると，これらの積も確率変数で，その期待値は，

$$f \times r \times (1-r)^{L-1} \qquad (2)$$

となる．

試験項目全体のエラー発見数の期待値は，試験項目に含まれるすべてのパスのすべてのリンクについて，(2)式を求めればよい．したがって試験項目全体のパスベクトルを $\{p_i\}$ とすれば，(1)式が成立する．

ＱＥＤ．

このモデルでは，エラーの数は一定ではなく同一プログラムでも試験のやり方によって変動すると考える．図１の例では，２のリンクを先に試験するか３のリンクを先にするかによって変わりうる．また $\{P_L\}$ の各要素の値が大きくてエラー発見率 r が小さい場合，すなわちプログラムが複雑で試験実施者のレベルが低い場合，エラーの数は増加し，いくらデバッグしてもエラーがなくならず，エラー数が発散することがありうる．発散条件は資料[3]を参照のこと．またエラーの数が有限であっても，有限回ですべてを発見できることを保証しない．実際，理論的には無限ループのエラーは，有限回では見つからない．

ここで試験ずみパス数を次のように定義する．

定義４　試験ずみパス数

(1)式で f ×を除いた項　$\Sigma \{p_i \times r \times (1-r)^{i-1}\}$ は，プログラム全体のパスベクトルのパスのうち試験項目がどれだけ試験したかを表すので，“（重み r による）試験ずみパス数　Number of tested paths (modified with r)”，縮めて“試験ずみパス数”と呼

ぶ．これをｃ（ｒ）とすると，

試験ずみパス数ｃ（ｒ）＝

$$\sum_{i=1}^{L} \{ p_I \times r \times (1-r)^{i-1} \} \qquad (3)$$

となる．

ここで(3)式のΣは，試験項目に含まれるすべてパスを，重複せず，見落としなく合計しなければならない．

図１の例で説明すると，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，この試験項目のパスベクトルは｛３，３，３，２，１｝であるから，この試験項目のエラー数発見期待値は，(1)式から

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

となる．

４．ｒの求め方

リンク当たりエラー発見率ｒを求める方法を述べる．試験項目１，２，３・・の順に実施して，発見したエラーの個数をe_1, e_2, e_3,・・とする．一方それに対する各試験ずみパス数の値は，ｒの関数だからc_1（ｒ）,c_2（ｒ）,c_3（ｒ），・・・と記述する．

エラー数f_iもエラー発見率r_iもプログラム内に偏りなく分布しているとすれば，発見エラー数と試験ずみパス数の値は比例すると考えられる．したがって，Ｌ個の試験項目の発見エラー数と試験ずみパス数の比をK_Lとして(5)式のように記せば，K_LはＬによらずほぼ一定にすることができると考えられる．すなわち，L=１，２，３・・について

$$K_L(r) = \sum_{i=1}^{L} e_i / \sum_{i=1}^{L} c_i(r) \qquad (5)$$

(5)式を，Σc_i（ｒ）を横軸に，Σe_iを縦軸にしたグラフにすると，これらの点は勾配がほぼ一定なのでだいたい直線上に並ぶ．また試験開始前は試験ずみパス数も発見エラー数も０だから原点を通る．

この直線の勾配はＬとｒに依存して変るから“エラー／パス比”としてK_L（ｒ）と記述し，最小自乗法と同じ考えで(6)式を最小にするものとして近似的に求められる．(6)式はｒに関して非線形であるが，０＜ｒ＜１と変化する範囲が限定されているので，後で述べる実例では，ｒを0.01きざみで変えその中から(6)式の値をもっとも小さくするものとしてｒとK_L（ｒ）を選んだ．Ｌは試験項目数である．

$$\sum_{i=1}^{L} (e_i / c_i(r) - K_i(r))^2 \geqq 0 \qquad (6)$$

５．エラー数の上限

ここでエラー総数を考察する．

｛P_L｝をプログラム全体のパスベクトルとすれば，(1)式のp_LをP_Lに置きかえたものは，試験が必要な全パス数であるから，ダイクストラのいう網羅試験（exhaustive test）のパス数である．これをC_∞と表すと次式のようになる．

$$C_\infty = \sum_{i=1}^{\infty} P_i \times r \times (1-r)^{i-1} \qquad (7)$$

｛P_i｝はプログラムの複雑度を表し，これが大きいと(7)式は発散する．したがってC_∞の値が大となる．一方エラー発見率ｒは０＜ｒ＜１であるが，これが１に近いと(7)式は収束し，C_∞は小さくなる．

(7)式のｒは(6)式を最小にするものとして求められ，｛P_L｝は連結行列を使って近似的に求めることができる．(7)式が収束すれば，エラー総数は次式で求められる．

$$\text{エラー総数} = C_\infty \times K(r) \qquad (8)$$

ただし，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は通らないパスもあるので，(8)式は過大な見積もりとなる．したがって(8)式はエラー総数の上限値であり，次の式が成り立つ．

$$\text{エラー総数} \leqq C_\infty \times K(r) \qquad (9)$$

実際の求め方は次の章の実例で示す.

６．パスベクトル法の実験例

６．１　実験課題

本方法により得られたエラー総数の見積がどのていど正確かは，エラー数と試験ずみパス数とがどの程度比例するかに依存する．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，コンパイルにパスした時点で，３個の試験項目を順番に与える．前の出力結果が間違っていればそのプログラムの試験は打ちきり，正しければ次の試験項目を与える．

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を図２に示す．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
  ＊　＊＊＊
  ＊＊　＊＊
  ＊＊＊　＊
```

図２　課題の入力データと出力データ

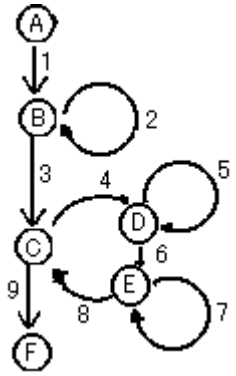
Fig.2 Input and output data of the problem.

この課題に対する模範プログラムを図３に，そのフローグラフを図４に示す.

```
include<stdio.h>
        int  main(void)
    {
        int  a,i,j;
①                   do{
②   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);
    }
```

図3　課題の模範プログラム例

Fig. 3　Model program of the theme.



	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４　模範プログラムのフローグラフとその連結行列

Fig. 4　Flowgraph and connected matrix of model program

６．２　分析手順

(1) フローグラフ作成

まず、プログラムのフローグラフを描く．フローグラフの描き方はプログラム言語によっても異なり一定のルールはないが，ここではC言語１ステートメントがなるべく１リンクになるように，次のようなルールにしたがった．

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，
②分岐文の終わり，
③リンクの合流点
④return文の次
とする．

(2)　リンクの番号づけ

上で定めた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．実例は図３，図４参照．

(3)　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．この表は，たとえば試験項目２は，リンクの１番２番３番４番の順に通ることを示す．同じ番号があるのは，同じリンクをくりかえすことを示す．

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

(4) パスベクトル表の作成

試験ずみパス数を(3)式で求めるために，試験項目全体のパスを一つの表にまとめる．そしてパスを長さ別に分類し，重複するパスがあれば一つを残して削除する．試験項目１は１と２のリンクからなるから，長さ１のパスはリンク１とリンク２の二個，長さ２のパスはリンク１と２をつなげたものが一個で，試験項目１に対するパスベクトル表は｛２，１｝となる．同様な処理を３つの試験項目にほどこす．この処理は人手では大変であるが，小規模なプログラムをパソコンで作成して用いた．使用言語はCやMATLABである．得られたパスベクトル表は次のとおりである．

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｝

(5) 出力結果の評価

出力結果を次のように分類した．出力には無限にループして止まらないということもありうるが，この実験ではなかった．

試験項目１：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．
試験項目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．
試験項目３：試験項目３を入力したが，正しい表示をしない．
合格：すべてのテストケースを正しく出力した．

(6) 発見エラー数の補正

この実験は２４人の学生が同じ問題を試験した結果の分析であるが，２４人の実力は等しくエラー生成数 f_i とエラー発見能力 r_i の平均値の f，r が等しいと仮定して補正した．

まず第１の試験項目でエラーとなったプログラムが3個あったので，２４人の平均で３／２４個のエラーがあったとした．第２の試験項目では，第１の試験項目にパスした２１個のうち１４個がエラーとなったので，１４／２１個のエラーがあったとした．

第３の試験項目については，同様に，第２試験項目をパスした７個のうち４つがエラーとなったので４／７個のエラーがあったとした．以上の結果を表１にまとめた．

表１　試験結果：テストずみパス数とエラー発見数

Table.1　Test results: test cases and no. of detected of errors

試験項目番号	失敗プログラム件数	一人当たりエラー数	一人当たり累積発見エラー数 e_L	累積試験ずみパス数 (r=0.44)
0	0	0	0	0
1	3	0.125=(3/24)	0.125	1.13
2	14	.667=(14/21)	0.792	7.73
3	4	0.571=(4/7)	1.363	12.13
計	21	1.363		

(7) エラー発見率 r を求める

累積試験ずみパス数と累積エラー発見数が比例するように，原点および試験で得られた３点の近くを通る直線から（6）式を最も小さくするｒとK(r)を求めた．求め方は０＜ｒ＜１で小数点以下２桁まで変化させて試行錯誤で求めた．ｒ=0.44，K(r)=0.1095で図５に示すようにほぼ直線となった．

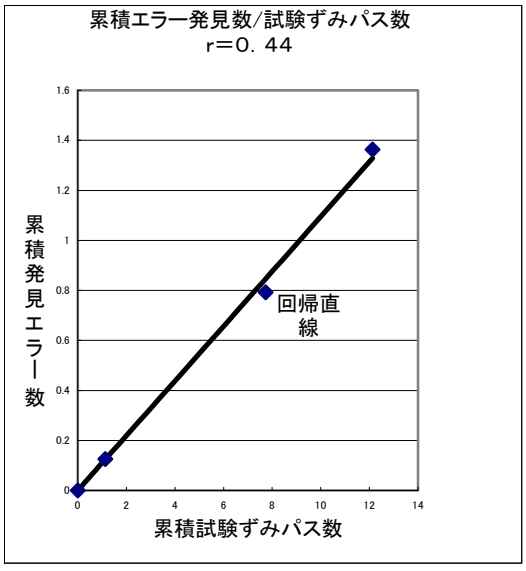


図５　発見エラー数と試験ずみパス数（エラー発見率ｒ=0.44）

Fig.5　No. of tested path vs. detected errors (error detective rates 0.44)

ｒをこれより大きめにすれば曲線は上に凸に反り，小さめにすると下に凸に反る．r=0.44で，試験ずみパス数とエラー数がほぼ比例することが分かる．このときのF検定の値は863.70，自由度は３で，確実度の係数 r^2=0.9965であった．

Ｘ軸は累積エラー数で，このデータは実際に試験をしないと分からない．しかし本理論では，(1)式とr=0.44という値を使えば，累積エラー数とＹ軸の累積試験ずみパス数が比例しているから，実際に試験しなくても，どういう試験項目を使えばどの程度の数のエラーが見つかるかということが推定できる．

リンク当たりエラー数の平均値ｆは，この実験例では0.112となった．リンク当たりエラー発見0.44とは，そのうち４４％が試験データを一回流すことにより発見されることを示す．

(8) エラー総数の上限値を求める

まずｒ=0.44を使って（7）式から全試験パス数C_∞を求める．パスベクトルは図４の連結行列を使い，長さＬのパスの数は，連結行列をＬ乗してその要素の値を合計したものである[2]．これをＬ＝６まで求めると，表２に示すようにP_Lの増加率はほぼ1.75に収束する．よって，(7)式において，$P_L=9\times1.75^L$と近似し，ｒ=0.44とすると，この級数は初項が９×ｒで項比が1.75×(1-0.44)=0.98＜１となるので収束し，C_∞=198となる．一般にP_Lの増加率は一定か振動をする[2]．振動した場合，たとえば振動のサイクルが３でa_1，a_2，a_3となる場合には，増加率として３個の調和平均である（$a_1\times a_2\times a_3$）$^{1/3}$を使う．

表２　図５のパスベクトル｛P_L｝

Table 2 Path vector of flowgraph of fig. 5

L	1	2	3	4	5	6
P_L	9	16	28	48	84	148
P_L増加率		1.78	1.75	1.71	1.75	1.76

エラー／パス比K(r)は，r=0.44と(6)式から0.112となり，したがってエラー総数の上限は(9)式から，r=0.44のとき21.6個となる．

実際に見つかったエラーは表１に示すように12.1個だからまだいくつかエラーが残って

いる可能性がある．初めてプログラムを書いた学生のものの中からテストできるレベルのものを選んだ結果である．

参考として，ｒをいろいろ変えたグラフを図６に示す．このグラフを見ると，ｒ＝0.44という値がもう少し低いとエラー数は発散する．これは分析の対象となったプログラムの質が悪いことを示している．

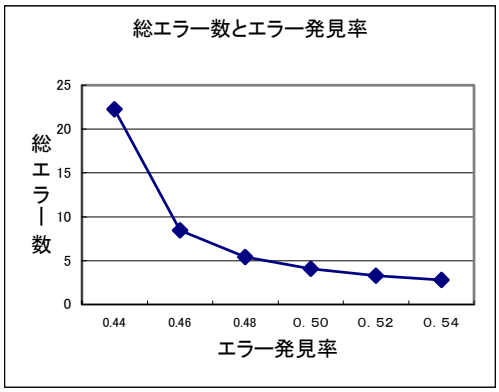


図６　エラー発見率とエラー総数の関係
Fig.6　Relations of error detective rates and no. of total errors

６．　まとめと今後の計画

本論文では試験ずみパス数に基づくエラー数の推定の理論を述べ，学生の作ったプログラムによってエラー数の予測を行った．

従来からのエラー数の予測モデルは，主にエラー発見率がプログラム内に残っている残存エラー数に比例すると仮定して，信頼度成長曲線に適当な曲線を当てはめるという方法で作られている．しかし発見エラー数はプログラム内におけるエラーの分布状態や試験項目の適用順序などに大きく依存すると考えられるから，試験の初期に得られたデータからプログラム全体に当てはまる曲線を見つけるという考えは当りはずれが大きい．「バグのないことは証明できない」というダイクストラの言葉があまりにも有名で，残存エラー数を理論的に評価しようという研究の意欲がそがれているように思われる．

本方法のように試験ずみパス数と発見エラー数との比例関係に着目してエラー数を推定すれば，よい精度でエラー数を評価できると思われる．しかし本論文で実験に使ったプログラムは小さいので十分な評価ができない．もっと多くのプログラムに適用してこの本手法の評価をしたい．小規模のプログラムなら，パスベクトルを求めるツールなどもできているので，比較的容易に解析可能である．そのために適当な例題を求めているので，ご協力をお願いしたい．

本理論は，エラーの数は技術者のレベルや試験のし方によって変動し，エラーの数が発散することもありうるという従来の考えとは変ったものであり，なかなか認めてもらえない．今後その有効性を実例によって示す必要があると思っている．その後，手法の実用化や簡略化にすすみたい．

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(4)－信頼度成長曲線－，電子情報通信学会研究報告,1998-11-5,pp124-4.

[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 Com- posite Design, Marson/Charter Publishers（197-5）