ソフトウェア工学 114－8
（1997．5．23）

残存エラー数の推定が可能なソフトウェア試験法について（2）

－パス分解法と構造化プログラミングとの関係－

若杉忠男
若杉情報技術コンサルタントオフィス　　電話0466(23)4832

筆者は，前にフローグラフをパスに分解し，長さ別のパスの個数の多少がプログラムの複雑度を表すとしてテストの理論的分析を試みた．今回はフローグラフの形と連結行列との関係を分析し，プログラムの形がパスの長さと個数にどう影響するかを考察する．まず，プログラムの基本要素であるＩＦ－ＴＨＥＮ－ＥＬＳＥやＲＥＰＥＡＴ－ＵＮＴＩＬを連結行列で表現し，これらの基本要素を合成して作られた連結行列の性質を考察する．この連結行列をべき乗した場合の各要素の値から，パスの長さと個数の関係を求める．これにより，パス数が増えにくいプログラムの構造はどのようなものかを考察し，パス分解法と構造化プログラミング法とを比較する．

On a software test method capable of estimating the number of programming errors(2)

Tadao Wakasugi
Wakasugi Information technology consultant office

The author tried, decomposing flowgraphs into paths, to analyse the quality of path coverage testing by using the numbers of paths which show the complexity of programs. Here, the relations between flowgraphs and connection matrices are studied, and how the structure of program affects the numbers and lengths of paths are discussed. From the results, the conditions of program structure for depressing the numbers of paths are made clear, and the idea of path decomposition is compared with the structured programming method.

1　はじめに

前回[1][2][3][4]の研究に続いてパス分解法に基づくフローグラフの評価を試みる．パス分解法では，プログラムのフローグラフをパスに分解し，長さＬのパスの個数をP_Lとしてパスのベクトル｛P_L｝を考え，この｛P_L｝によってプログラムの複雑度を表現し，またこれを試験によるエラーの個数の減少傾向などの推定に使う．

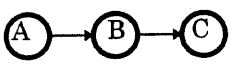
本論文では，フローグラフの基本要素への分解とそれらの合成について考察し，フローグラフの構造と｛P_L｝の関係を調べ，パス分解法の評価に基づくよいプログラムとはどういうものかを述べ，それと構造化プログラミング[5]の思想とを比較をする．

2　フローグラフの基本要素

フローグラフとは，図１に示すように，プログラムの制御の流れの骨格を図示したもので，フローチャートからセンテンスを省きリンクとノードだけで表現したものと見ることができる．ここにリンクはプログラムステートメントを，ノードはそのつなぎ目を表す．

フローグラフ　　連結行列　　2乗　　3乗　　4乗

1．プロセス



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

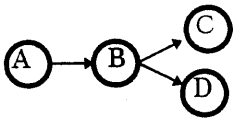
	A	B	C
A	0	0	1
B	0	0	0
C	0	0	0

	A	B	C
A	0	0	0
B	0	0	0
C	0	0	0

	A	B	C
A	0	0	0
B	0	0	0
C	0	0	0

べき乗すると，連結行列の非0要素は右上へと移動し，n次元マトリックスならばn乗すると0になる．

2．IF-THEN-ELSE



	A	B	C	D
A	0	1	0	0
B	0	0	1	1
C	0	0	0	0
D	0	0	0	0

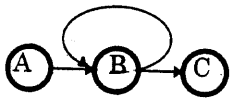
	A	B	C	D
A	0	0	1	1
B	0	0	0	0
C	0	0	0	0
D	0	0	0	0

	A	B	C	D
A	0	0	0	0
B	0	0	0	0
C	0	0	0	0
D	0	0	0	0

	A	B	C	D
A	0	0	0	0
B	0	0	0	0
C	0	0	0	0
D	0	0	0	0

プロセスの場合と同様に，非0要素は右上に移動して消えて行く．

3．ループ
DO－WHILE



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

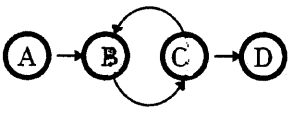
	A	B	C
A	0	1	1
B	0	1	1
C	0	0	0

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

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

べき乗しても，まん中の1は変化せず，その影響は右上に伝わってゆく．

4．ループ
REPEAT-UNTIL



	A	B	C	D
A	0	1	0	0
B	0	0	1	0
C	0	1	0	1
D	0	0	0	0

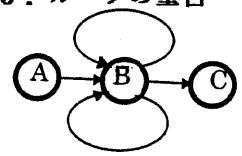
	A	B	C	D
A	0	0	1	0
B	0	1	0	1
C	0	0	1	0
D	0	0	0	0

	A	B	C	D
A	0	1	0	1
B	0	0	1	0
C	0	1	0	1
D	0	0	0	0

	A	B	C	D
A	0	0	1	0
B	0	1	0	1
C	0	0	1	0
D	0	0	0	0

まん中の4要素が周期2で変化し，一つおきに見ると，DO－WHILEと同じく，まん中の値は変化せず，その影響が右上に伝わる．

5．ループの重合



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

	A	B	C
A	0	2	1
B	0	4	2
C	0	0	0

	A	B	C
A	0	4	2
B	0	8	4
C	0	0	0

	A	B	C
A	0	8	4
B	0	16	8
C	0	0	0

重合とは一つのノードに二つ以上のループがついたものである．一般にN個のループが一つのノードについている場合には，ノードにあたる要素の値はNのべき乗で増加し，その影響は右上に伝わる．

図1　フローグラフの基本要素とその連結マトリックスのべき乗

フローグラフはまた連結行列でも表現できる．連結行列は，図１の例に示すようなノード数をｑとすると，ｑ×ｑの正方マトリックスで，要素ｉ，ｊの値をノードｉとノードｊとの間のリンク数としたものである．またこのマトリックスをＬ乗しその要素の値を合計すると，長さＬのパスの個数P_Lとなる．

パス分解法では，パスの個数P_Lが少ないほどフローグラフは簡単であると評価する．ここでプログラムの基本要素について，フローグラフとその連結行列を図１に示す．基本要素としては，構造化プログラミングでいう，プロセス（コンカチネーション），ブランチ（ＩＦ－ＴＨＥＮ－ＥＬＳＥ），ループ（ＤＯ－ＷＨＩＬＥとＲＥＰＥＡＴ－ＵＮＴＩＬ）の他に“ループの重合”を選ぶ．

ループについては，パスの数に関して，ＤＯ－ＷＨＩＬＥとＲＥＰＥＡＴ－ＵＮＴＩＬは同じである．図１の例で見当がつくように，ループにＮ個のノードを含む場合にはＮ乗するごとにサイクリックに元の状態に戻り，サイクルの途中で対角要素に１がＮ個並んだ状態が生ずる．ならせば，パスの数では対角要素に１が１個あるのと同じことになる．

ループの重合については，これはブランチとループを一緒にしたもので，これを基本要素として選んだ理由は，｛P_L｝を急激に増加させ，パス数への影響が大だからである．

図１を見ると，これらの要素の連結マトリックスは，次のように分類できる．

①　０になる要素：マトリックスをべき乗するといつかはすべての要素が０になるもの．フローグラフでいうとループのないもので，図１の１と２である．

②　一定の要素：べき乗すると，要素の値が一定か，またはサイクリックに変化して元に戻るもの．単純なループを含むフローで，フローグラフでいうと図１の３と４である．

③　増加する要素：べき乗すると要素の値が増加するもの．ループの重合がこれで図１の５である．

図１の添え書きに示すように，これらの基本要素には次のような共通の性質がある．すなわち，“これらのマトリックスをべき乗した場合，非０要素の影響は右上方向の要素に伝わるが，左下側は何の影響も与えない．したがって，左下側にある０要素はマトリックスをべき乗しても０のままである”．

３　基本要素の合成

ここでは，フローグラフのうち，２節の基本要素を下記の４つの方法で合成したものを考察する．一般のプログラムはこのようにして作成され，またこのように作成されるのが望ましいと考える．

①並列：二つの基本要素を，ＩＦ－ＴＨＥＮ－ＥＬＳＥでつなぐこと．

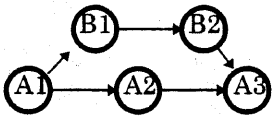
②直列：一つの基本要素の終点を，もう一つの基本要素の始点につなげること．

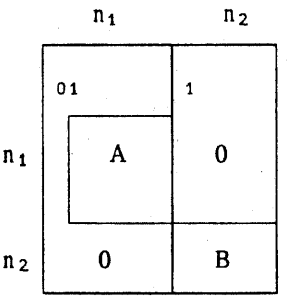
③置き換え：基本要素のリンクの一つを他の基本要素と置き換えること．

④附加：一つのノードに他の基本要素の入口と出口をつなげること．附加されたノードにはループが生じることになる．

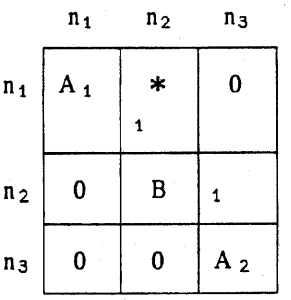
ここで基本要素のマトリックスをＡ，Ｂ，Ｃなどと表し，それらのマトリックスの次元を，n_1，n_2，n_3・・とすると，並列，直列，置き換え，附加はそれぞれ図２のように表される．

①並列

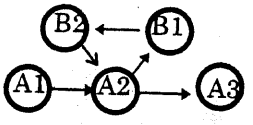


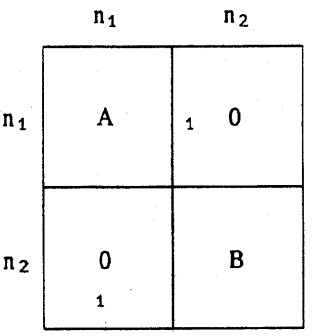


②直列と置き換え



③附加





附加の場合の左下のマトリックスは，右上のマトリックスを横倒しにしたもの

図2　連結マトリックスの合成

4　合成マトリックスの性質

基本要素の“並列”，“直列”，“置き換え”で作られた合成マトリックスは，基本要素のマトリックスA，B・・が対角線上に現れ，1がその右上側に現れるが左下は0だけである．“附加”によって作られた合成マトリックスについては，右上だけでなく左下にも1が一つ増える．これらのマトリックスについては一般に次のことが言える．

定理1

図1の基本要素を，“並列”“直列”“置き換え”“附加”で作成したフローグラフのループの数は，合成マトリックスの対角要素およびその左下の要素の値の和となる．

証明

対角要素とその左下の要素は，ＤＯ－ＷＨＩＬＥ，ＲＥＰＥＡＴ－ＵＮＴＩＬでは1，重合では重合したループの数となり，それ以外では0である．またそれらを合成した場合

で左下要素が1となるのは附加の場合だけで，附加の場合にはループが一つ増える．したがってループの数は対角要素と左下要素の合計と一致する．　　ＱＥＤ

補助定理1

Ａ，Ｂ，Ｃ・・を正方マトリックスとし，それらを対角線上に並べ，それらの左下の要素はすべて0であるマトリックスは，べき乗しても，それらの左下部分は0のままである．

証明

このマトリックスを図3左のように表す．Ａ，Ｂ，Ｃ・・はそれぞれ，n_1，n_2，n_3，・・次元の正方行列，0は0要素ばかりの部分，＊は適当に数値が入っている部分を表す．このマトリックスを自乗すると，図3右のようにn_1，n_2などの値は変わらず，対角要素の下のマトリックスはすべて0のままである．また右のマトリックスと左のマトリックスをかけたものも同じく0のままである．したがって補助定理1が成立する．　　ＱＥＤ

	n_1	n_2	n_3	・
n_1	A	*	*	*
n_2	0	B	*	*
n_3	0	0	C	*
・	0	0	0	・

元のマトリックス

	n_1	n_2	n_3	・
n_1	A^2	A*+B*	A*+*+C*	・
n_2	0	B^2	B*+C*	・
n_3	0	0	C^2	・
・	0	0	0	・

左のマトリックスの2乗

ここに・は省略，*は任意の要素であることを示す．

図3　合成したマトリックスのべき乗による0要素の状態

定理2

2節で定義した基本要素のマトリックスを，“並列”，“直列”で合成したマトリックスは，べき乗しても左下の0要素は0のままである．

証明

基本要素を，“並列”，“直列”で合成したマトリックスは，対角線上に正方行列が並び，右上に1が現れ，左下は0ばかりで，補助定理1の条件が満たされる．したがって左下の0は変わらない．　　ＱＥＤ

定理2の合成マトリックスをＭとし，図4に示すようにループと重合ばかりからなるマトリックスM_1と，それ以外の部分からなるマトリックスM_2との和に分解する．すなわち図4に示すように，$M = M_1 + M_2$で，Ａ，Ｂ，Ｃ，・・はループと重合または0，また＊は任意の数，残りの空白部分は0とする．この場合に次の定理が成立する．

定理3

定理2の条件のもとで作成した連結マトリックスについて，元のフローグラフが，

（1）プロセスとＩＦ文だけからなる場合には，ノードの数だけべき乗すると0となる．すなわちノードの数より長いパスはない．

（２）プロセスとＩＦ文の他にＤＯ－ＷＨＩＬＥ，ＲＥＰＥＡＴ－ＵＮＴＩＬを含み重合を含まない場合には，Ｌ乗すると要素の和，すなわちP_LはＬ－１次多項式で増加する．

（３）ループの重合を含む場合には，P_LはＬのべき乗で増加する．

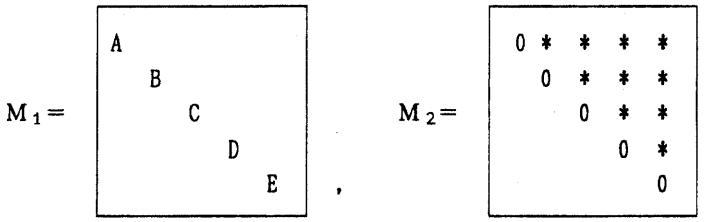


図４　要素マトリックスからなる合成マトリックスの分解

証明

フローグラフのノードの数，すなわち連結マトリックスの行数（＝列数）をｑ，べき乗の回数をＬ，このマトリックスの対角要素に含まれる数値の最大値をＫとすると，マトリックスをべき乗したときの要素の和がパス数P_Lであるから，P_Lの大きさは

$$M^L = (M_1+M_2)^L = \sum_{N=0}^{L} (M_1^{L-N} \times {}_LC_N \times M_2^N)$$

$$= M_1^L + \sum_{N=1}^{q-1}(M_1^{L-N} \times {}_LC_N \times M_2^N) + \sum_{N=q}^{L}(M_1^{L-N} \times {}_LC_N \times M_2^N) \quad (1)$$

↓　　↓　　↓

要素の和：（ＫのＬ乗）＋（Ｋのｌ-1乗以下）×（Ｌのｑ-1次以下の多項式）＋ ０

によって求められる．

(1)式の第１項はM_1のＬ乗であり，M_1の対角要素にＫがあるとK^Lというオーダーになる．すなわちループの重合があると第１項はＬのべき乗で増加する．重合がなく，ＤＯ－ＷＨＩＬＥなどの場合には$1^L=1$で一定である．

(1)式の第２項以下は，次の３つの項目のかけ算である．

①　M_2^N：これはＮ＝ｑ以上のときに０となる．

②　${}_LC_N$：これは＝Ｌ！／｛Ｎ！×（Ｌ－Ｎ）！｝で，①の条件によりＮ＝ｑ以上になると無視できるので，Ｌの次数の最も大きな項目は，

$${}_LC_{q-1} = L!/\{(q-1)! \times (L-q+1)!\}$$

$$= L(L-1)(L-2)\cdots(L-q+2)/(q-1)!$$

＝Ｌの（ｑ－１）次多項式／（ｑ－１）！

である．

③　M_1^{L-N}：ＫのＬ－１乗以下の数値となる．

ゆえに，プロセスとＩＦ文だけからなる場合には，(1)式はＬ＝ｑ以上では０となる．また重合していないループがあるが重合のない場合には，Ｌの（ｑ－１）次以下の多項式となる．したがって証明された．　　ＱＥＤ

基本要素の直列によるパスの変化の例としてループを直列につなげた場合を図５に示す．ループのある２つのノードＡとＣの影響は右と上とに伝わり，その合流点となる要素（図５の１行３列目）は０，１，２，３とＬの一次式で増加していることが分かる．

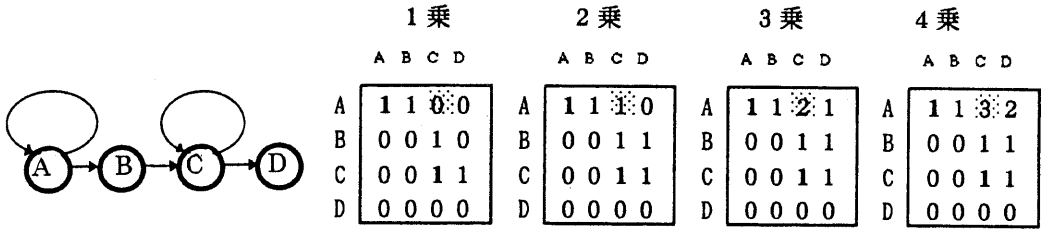


図５　二つのループを直列に合成したマトリックスの例

５　パス分解法から見たよいプログラム

パス分解法ではパスベクトル｛P_L｝の小さいプログラムを単純でよいプログラムとするので，その考えにしたがって次のようにプログラムすることを推薦する．

（１）フローに戻りがない，すなわちループがないようにする．

これは構造化プログラミングの思想と一致するが，ループがなければ，すべてのパスは有限の長さとなり，１００％のカバレジを実現できる．ノードの数以上に長いパスはない．

ＧＯ－ＴＯ文については，後戻りするＧＯ－ＴＯ文は連結行列の要素を増加させる．先に進むＧＯ－ＴＯ文はパス数にはあまり影響しないので禁止はしない．

（２）ブランチは少なくする．

１つのブランチから二つのリンクが出るのでパスの数が２倍になる．

（３）ループの並列，直列，重合の順にパスの数が急激に増加する．すなわち

並列ではパスの数が加算される．

直列では，パスの長さの（ループの数－１）乗でパスの数が増加する．

ループの重合は，パスの長さのべき乗でパスが増加する．

したがってループの重合の使用は避ける．

６　構造化プログラミングとの比較

構造化プログラミングの思想には，人によっていろいろな主張があるが，プログラムのフローに後戻りがないようにするということが主眼で[5]，基本要素として次の４つを使用するように提案している．

①プロセス　　②ＩＦ－ＴＨＥＮ－ＥＬＳＥ

③ＲＥＰＥＡＴ－ＵＮＴＩＬ　　④ＤＯ－ＷＨＩＬＥ

ループの重合については特に考慮していないようで使用してもよい．一方，ＧＯＴＯの使用はよくないとする．

特に次の４つへのＧＯＴＯを禁止している．[6]

①ループへの飛び込み　　②ループからの飛び出し

③ブランチの判定へのブランチ　　④ブランチの判定結果からのブランチ

一方，パス分解法では，基本要素として，プロセス，ＩＦ－ＴＨＥＮ－ＥＬＳＥ，ループ，ループの重合を考える．ただしパスの数が少ないほどよいとするので，重合は望まし

くない．またパス分解法では，ＧＯＴＯ文についてはパスが増加しないならば使ってもよい．たとえば構造化プログラミングでは図６の左のようなフローは禁止され右の図はよいとするが，パス分解法ではどちらにしてもパスの数は同じなので左のフローを禁止する理由はない．

構造化プログラムでは禁止される　　禁止されない
しかしパスの数はどちらも同じ

図６　構造化プログラミングとパス分解法の比較

７　まとめ

フローグラフのパスの数でプログラムの複雑度が評価できるというパス分解法の考えに基づき，フローグラフの構成とパス数との関係を考察した．

まずプログラムを基本要素について，パスの長さＬが増加すると，パス数が

①無くなるもの　　②一定であるもの　　③Ｌのべき乗で増加するもの

があることを示した．

次に，これらを合成してプログラムを作ると考え，パスの長さＬが増加するとパスの個数が

①０になる場合　　②Ｌの多項式で増加する場合　　③Ｌのべき乗で増加する場合

があることを示した．

またパス分解法から見たよいプログラム，すなわち複雑度の少ないプログラムの条件を検討し，それと構造化プログラミングとの相違点を示した．

これにより，パス分解法は，複雑度を数値で表し，プログラムの構造を数学的に扱うことができるすぐれたアイデアであると言える．

基本要素の合成の結果については，並列と直列についてしか考察していないので，今後の課題として，置き換えと附加について検討する予定である．

参考文献

[1]　若杉忠男：“状態遷移図の同定問題の一解法”，情報処理学会マルチメディア通信と分散処理研究会，(1996-10)．

[2]若杉忠男：“ＯＳＩ適合性試験スイートの評価法－マルチトランジション試験”，日本電子情報通信学会論文誌，VOL.J72-BI No.4（1996-4）．

[3] 若杉忠男:“有限状態マシン(FSM)で表されるシステムの複雑度の評価について”,情報処理学会マルチメディア通信と分散処理研究会(1995-9)．

[4] 若杉忠男:“ISOで開発したトランスポートプロトコル適合性試験スイートの質の評価”，情報処理学会論文誌，Vol.37 No.3（1996-3）．

[5] E.W.Dijkstra and other:"Structured programming",構造化プログラミング，野下他訳，サイエンス社（1975-5）．

[6] McCabe,T.J.:"A complexity measure",IEEE Transaction on Software Engineering 2:308-320(1976).