ソフトウェア工学 38－1
（1984. 9. 26）

プログラムの制御フローに基づくテスト・ケース設計法

大場 充（日本アイ・ビー・エム(株) サイエンス・インスティチュート）

1．はじめに

プログラムが大規模になると，その制御構造が複雑になるため，全てのプログラム・パス（経路）を網らするテスト・ケースを実行することは，時間の面で現実的でなくなる．実際に，一定以上のプログラム・パスを網らし終わると，それ以後のテスト・ケースの実行によるエラーの発生（発見）確率（エラー発見率）は急激に低下する．そのため，累積発見エラー数の成長曲線は，図1のように通常の指数型成長曲線とは，似て非なるものとなる．テスト・ケースの単位実行当りのエラー発見率が一定であれば，観測されるエラー発見の成長曲線は，指数型でなければならない．

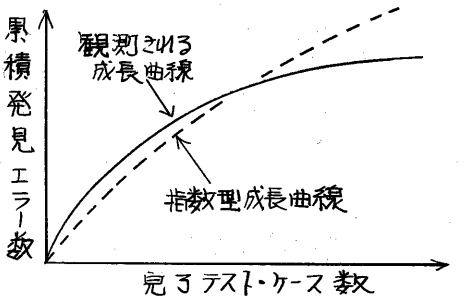


図1．テストケースとエラー発見数

従って，テスト・ケースの設計に際しては，どのようなパスを通るテスト・ケースを選択するかが問題となる．本報告では，実験計画法でよく用いられている直交表を応用することによって，プログラムの制御構造をもとに，効率のよいテスト・ケースを設計する方法について考察する．

2．プログラムの制御構造とテスト・ケースの関係

例として，図2に示されるような，木構造型制御フローをもつプログラムと，図3に示されるような，鎖構造型制御フローをもつプログラムを考える．

```
begin
  :
  if P1
    then if P2
           then if P3
                  then S1;
                  else S2;
           else if P4
                  then S3;
                  else S4;
    else if P5
           then if P6
                  then S5;
                  :
end
```

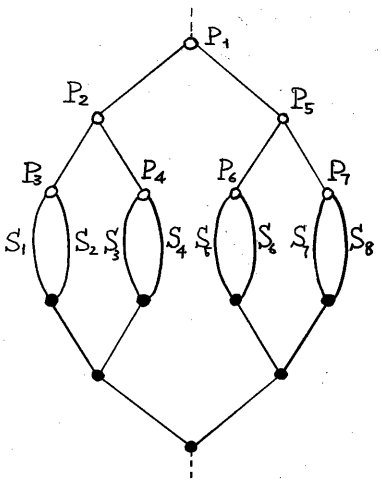


図2．木構造型制御フローの例

```
begin
  :
  if P1
    then  S1;
    else  S2;
  if P2
    then  S3;
    else  S4;
  if P3
    then  S5;
    else  S6;
  if P4
    then  S7;
    else  S8;
  :
end
```

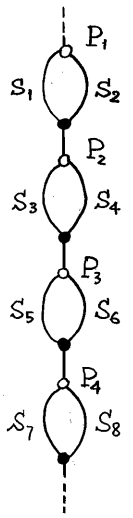


図3．鎖構造型制御フローの例

図2のような木構造型制御フローをもつプログラムでは，7個の判定があり，全てのパスを網らするためには，8通りのテスト・ケースが必要である．一般的には，木構造型制御フローの場合，n個の判定に対して，n＋1個のテスト・ケースが必要となる．

図3のような鎖構造型制御フローをもつプログラムでは，4個の判定があり，全てのパスを網らするためには，以下の16通りのテスト・ケースが必要となる：

1) $S_1 \cdot S_3 \cdot S_5 \cdot S_7$,
2) $S_1 \cdot S_3 \cdot S_5 \cdot S_8$,
3) $S_1 \cdot S_3 \cdot S_6 \cdot S_7$,
4) $S_1 \cdot S_3 \cdot S_6 \cdot S_8$,
5) $S_1 \cdot S_4 \cdot S_5 \cdot S_7$,
6) $S_1 \cdot S_4 \cdot S_5 \cdot S_8$,
7) $S_1 \cdot S_4 \cdot S_6 \cdot S_7$,
8) $S_1 \cdot S_4 \cdot S_6 \cdot S_8$,
9) $S_2 \cdot S_3 \cdot S_5 \cdot S_7$,
10) $S_2 \cdot S_3 \cdot S_5 \cdot S_8$,
11) $S_2 \cdot S_3 \cdot S_6 \cdot S_7$,
12) $S_2 \cdot S_3 \cdot S_6 \cdot S_8$,
13) $S_2 \cdot S_4 \cdot S_5 \cdot S_7$,
14) $S_2 \cdot S_4 \cdot S_5 \cdot S_8$,
15) $S_2 \cdot S_4 \cdot S_6 \cdot S_7$,
16) $S_2 \cdot S_4 \cdot S_6 \cdot S_8$.

一般的には，鎖構造制御フローの場合，n個の判定に対して，2^n 個のテスト・ケースが必要となる．

図2のような木構造型制御フローをもつプログラムでは，8通りのテスト・ケース全てを実行完了するまで，プログラムの品質については何も言えない．未実行のパスにエラーが存在する可能性が大きいからである．例として，図2のプログラムの，S_1 から S_8 までの全てのパスにエラーが存在すると考える．ここで，S_1 から S_8 を通るように設計されたテスト・ケースを順次実行すると，1番目から8番目までのテスト・ケースで除去されるエラーの累積数は，図4のようになる．

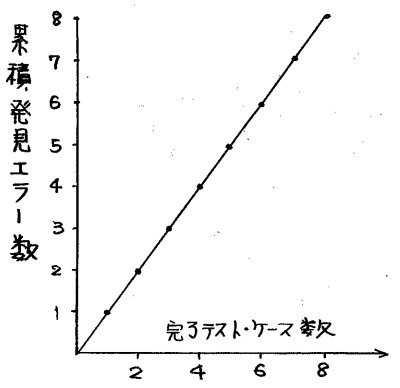


図4．木構造型制御フローの例

図3のような鎖構造型制御フローをもつプログラムでは，テスト・ケースの設計法によって，最少の場合では，2通りのテスト・ケースを完了した時点で，プログラムの品質についての予測ができる．以下のように，S_1 から S_8 までの全てのステートメントを最少の回数で網らすればよいからである．

1) $S_1 \cdot S_3 \cdot S_5 \cdot S_7$,
2) $S_2 \cdot S_4 \cdot S_6 \cdot S_8$.

例として，図3のプログラムの，S_1 から S_8 の全てのステートメントにエラーが存在すると考える．ここで，前述の16通りのテスト・ケースと，最少の2通りのテスト・ケースを順

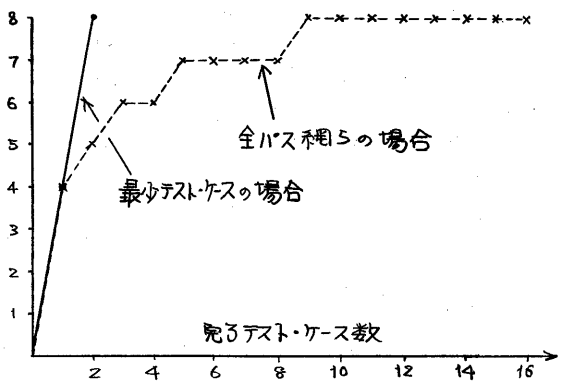


図5．鎖構造型制御フローの例

次に実行したとすれば，除去されるエラーの累積数は，図5のようになる．全てのパスを網らするテスト・ケースのエラー発見率の低下が，図5によく示されている．

上述の2つの例では，エッジのカバレージが100％になった時点で，全てのエラーが発見されることを仮定している．鎖構造型制御フローをもつプログラムでは，プログラム・パスを網らするのに必要なテスト・ケース数と，エッジを網らするのに必要最少なテスト・ケース数との間に大きな差がある．

3．テスト・ケースの設計

木構造型制御フローをもつプログラムでは，プログラム・パスを網らするテスト・ケースが必要である．木構造型制御フローをもつプログラムに存在する，プログラム・パスの総数は，一般に判定文の数をnとするとき，n+1個である．この場合，プログラム・パスが網らされないうちは，エッジも網らされないことが特徴である．図6に示される制御フローをもつプログラムのテストに必要な，最少のテスト・ケースの集合は，図7の4つのパスを通るものでなければならない．

鎖構造型制御フローをもつプログラムでは，エッジを網らするテスト・ケースが，最少テスト・ケースであり，パスを網らするテスト・ケースが，最大テスト

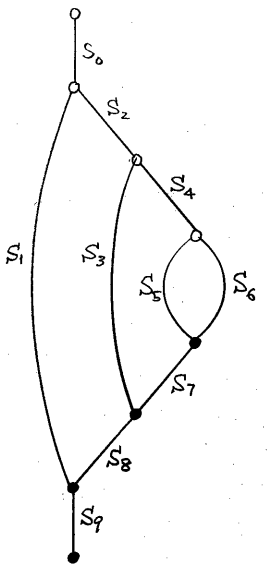
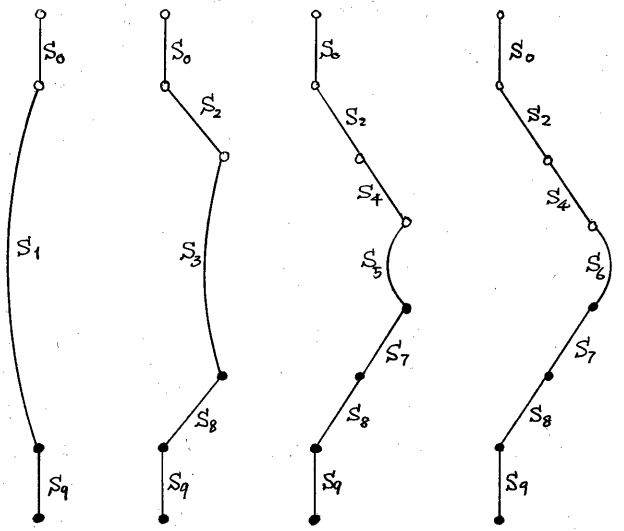


図6．例題プログラム1



ケース1　ケース2　ケース3　ケース4

図7．例題プログラム1のテスト・ケース

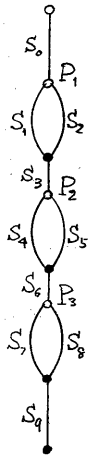
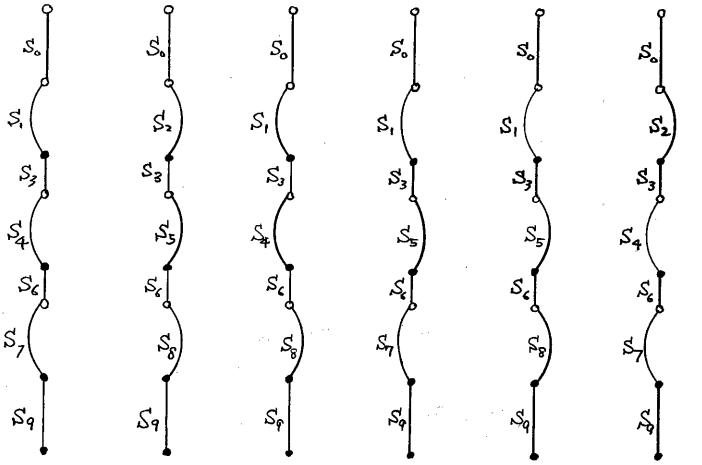


図8．例題プログラム2



ケースである．エッジの網らのためには，互いに素なテストケース（ケース1とケース2，ケース3とケース8，ケース4とケース7，ケース5とケース6）の組を用いれば十分である．ところが，鎖構造型制御フローをもつプログラムでは，エッジ間に相互作用をもつ可能性がある．例えば，エッジS_1で設定した値によって，エッジS_8での演算が変化することがある（エッジS_1の計算が行列のインデクスを決定し，S_8でその要素への代入が実行されるなど）．従って，鎖構造型制御フローをもつプログラムの場合，エッジの網らだけでは十分ではない．

ここで，鎖構造型制御フローをもつプログラムで，任意の2つのエッジ間の相互作用で発生するエラーを，発見するのに必要十分な最少テスト・ケースを考える．前述のように，図9のケース1と2，ケース3と8，ケース4と7，ケース5と6は，互いに相補的な関係にある．従って，これらの組の一方をとれば，他方は冗長となる．すなわち，4つのテストケースで十分となる．さらに，上記の各組でのエッジの出現（テスト）回数をバランスするようにとると，図10の4つのテスト・ケースが最良の組合せの1つとなる．図10の4つのテスト・ケースでは，S_1とS_2，S_4とS_5，S_7とS_8の出現回数がバランスしている．この場合，図9の8つのケースのうち，ケース1，5，7，と8の4つが選択されている．

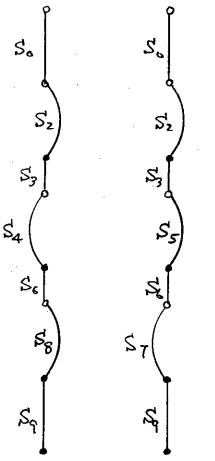


図9．テスト・ケース

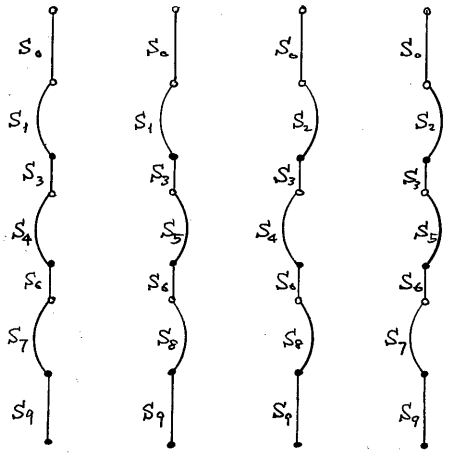


図10．相互作用を考慮したテスト・ケース

ところで，図10で与えられるテスト・ケースは，一般に，直交表で与えられる組合せの解と同じである．図8の判定文，P_1，P_2，P_3において，P_1が真の場合をS_1，P_2が真の場合をS_4，そしてP_3が真の場合をS_7とする．L_4の直交表（表1）の1を真（T），2を偽（F）とすれば，表2のテスト・ケース・マトリクスが得られる．

表1．直交表L_4

	1	2	3
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

表2．テスト・ケース・マトリクス

	P_1	P_2	P_3
1	T (S_1)	T (S_4)	T (S_7)
2	T (S_1)	F (S_5)	F (S_8)
3	F (S_2)	T (S_4)	F (S_8)
4	F (S_2)	F (S_5)	T (S_7)

図10のテスト・ケース，または表2のように直交表を用いて設計されたテスト・ケースでは，P_1とP_2，P_2とP_3，P_1とP_3の任意の組合せに対して，全ての場合が網らされている．すなわち，図8の例題の場合，表3，表4，表5のとうりである．

表3．$P_1 \times P_2$

	P_1	P_2
1	T (S_1)	T (S_4)
2	T (S_1)	F (S_5)
3	F (S_2)	T (S_4)
4	F (S_2)	F (S_5)

表4．$P_2 \times P_3$

	P_2	P_3
1	T (S_4)	T (S_7)
3	T (S_4)	F (S_8)
4	F (S_5)	T (S_7)
2	F (S_5)	F (S_8)

表5．$P_1 \times P_3$

	P_1	P_3
1	T (S_1)	T (S_7)
2	T (S_1)	F (S_8)
4	F (S_2)	T (S_7)
3	F (S_2)	F (S_8)

表3，4，5より明らかなように，図8の2つのエッジの組合せによって発生するエラーがあっても，図10の4つのテスト・ケースが実行されれば，そのエラーは発見できる．ただし，図10または表2の4つのテスト・ケースが実行されたとしても，3つのエッジの組合せでしか発生しないエラーがあれば，そのエラーを発見することは保証されない．例えば，図9のケース2のパスでのみ発生するエラーがあるとすれば，そのエラーは，図10の4つのテスト・ケースでは発見不能である．

4．直交表によるテスト・ケースの性質

直交表を用いて設計されたテスト・ケースでは，一般に以下の関係が成立する．

$$n + 1 \leq 2^m . \qquad (4.1)$$

ここで，nは鎖構造型制御フローでの判定文の個数，mは自然数である．(4.1)式は，2^m通りのテストで，n個の判定文をもつ鎖構造型制御フローをもつプログラムをテストすることができる．このとき，直交表を用いることにより，(4.1)式を満足する最少のmを求めることができる．すなわち，3個の判定文であれば，テスト・ケース数は，$m = 2$で，4となる．また，7個の判定文があれば，テス

ト・ケース数は，m＝3で，8となる．表6に，直交表L_8を示す．直交表L_8では，m＝3で，3つのエッジの相互作用によって発生する，全てのエラーを発見する能力のあるテスト・ケースを設計できる．ただし，4つ以上のエッジの相互作用によって発生するエラーの発見は保証されない．

表6．直交表L_8

ケース	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	1	1	1	2	2	2	2
3	1	2	2	1	1	2	2
4	1	2	2	2	2	1	1
5	2	1	2	1	2	1	2
6	2	1	2	2	1	2	1
7	2	2	1	1	2	2	1
8	2	2	1	2	1	1	2

直交表を用いて設計されたテスト・ケースは，最良の場合，（4.1）式の関係より，

$$n+1 = 2^m, \qquad (4.2)$$

が成立する．このとき，テスト・ケースの最少数は，制御フローの構造によらず，n＋1となる．しかし，最悪の場合でも，

$$n = 2^{m-1}, \qquad (4.3)$$

が成立するので，テスト・ケースの最少数の上限は，2nである．例えば，nが4のとき，テスト・ケースの最少数は，8となる．従って，一般的にn個の判定文をもつプログラムに必要な最少のテスト・ケース数Nは，

$$n+1 \leqq N \leqq 2n, \qquad (4.4)$$

となる．

ところで，パスを網らするテスト・ケースの数は，鎖構造型制御フローをもつプログラムでは，2^nとなるので，直交表によるテスト・ケースの設計により，必要テスト・ケース数を，2^{n-m}分の1にすることができる．例えば，n＝8の場合，m＝4であるから，テスト・ケース数は，16分の1（256ケースに対して16ケース）でよいことになる．

5．複合型制御フローをもつプログラムの場合

現実のテスト対象プログラムは，木構造型制御フローまたは鎖構造型制御フローだけから成るわけではなく，2つの構造の複合したものや，ループ構造をもつものが多い．ここでは，そのような複合構造をもつプログラムに対して最良のテスト・ケース集合を与える設計法を考える．

図11に，ループ構造の内部に，鎖構造をもつ，複合構造のプログラム例を示す．ループ構造内に部分構造を含む場合（この場合，WHILE型かまたは，UNTIL型のループを考える），そのプログラムをテストするためには，2回以上ループの内側を通らなければ，エッジ間の相互作用をテストすることはできない．従って，図11の例のような構造をもつプログラムの場合，図12に示されるような，構造的には図11のプログラムの部分に等価な構造をもつプログラムの，最良のテスト・ケースを設計すればよいことになる．

図11のループ内に含まれている，鎖構造をもつ判定文の個数を3とすれば，図12の等価構造をもつプログラムの判定文の個数は7個である．従って，テストに必要なテスト・ケースの総数は，直交表L_8より，8ケースとなる．これに対して，全パス網らのテスト・ケースを考えると，必要となるテスト・ケースの総数

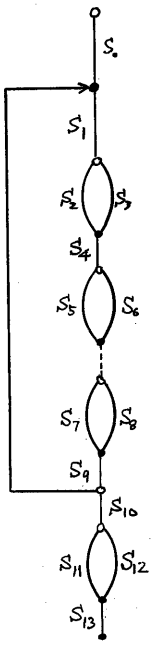


図11．ループの例

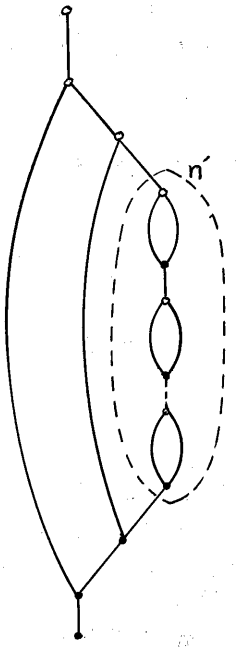


図13．複合構造の例1

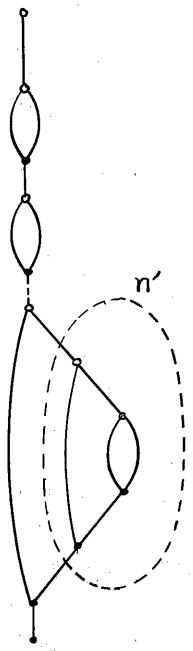


図14．複合構造の例2

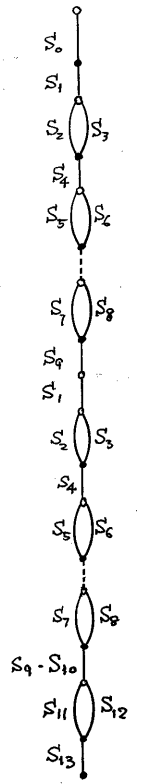


図12．等価構造

は，128ケースとなる．従って，直交表を用いたことにより，テスト・ケースの総数は，16分の1となる．

図13に，木構造型制御フローの内部に，鎖構造型制御フローを含む，複合構造プログラムの例を示す．この例の場合，全体の判定文の総数をn，鎖構造型制御フローをもつ部分の判定文の総数n'とすれば，パスを網らするのに必要なテスト・ケースの総数は，

$$n - n' + 2^{n'}, \qquad (5.1)$$

で与えられる．図13で，n'を3とすれば，テスト・ケースの総数は，10（5－3＋2^3）となる．

これに対して，直交表を用いた場合のテスト・ケース数は，

$$n - n' + 2^m, \quad (m = \min_m \{n'+1 \leq 2^m\}), \qquad (5.2)$$

となる．図13で，n'を3とするとき，テスト・ケースの総数は，mが2（3+1 = 2^2）であることから，6（5-3+2^2）となる．

実際のテスト・ケースの設計は，鎖構造部分を除いた木構造で考え，n－n'個のテスト・ケースを生成する．次に，残りの鎖構造部分に対するテスト・ケースを直交表を用いて設計して，木構造の最後のパスと結合すればよい．

図14に，鎖構造型制御フローの内部に，木構造型制御フローを含む，複合構造プログラムの例を示す．この例の場合，全

ての判定文の数をn，木構造型制御フローをもつ部分の判定文の総数をn'とすれば，パスを網らするのに必要なテスト・ケースの総数は，

$$2^{n-n'-1} \times \{(n'+1)+1\}, \qquad (5.3)$$

で与えられる．図14で，nを5，n'を2とすれば，テスト・ケースの総数は，16（$2^{5-2-1} \times \{(2+1)+1\}$）となる．

これに対して，直交表を用いた場合のテスト・ケース数は，

$$2^m + 2^{m-1} \times (n'+1) = 2^{m-1}(n'+3), \qquad (5.4)$$

で与えられる．図14で，nを5，n'を2とすれば，mが2（$3+1=2^2$）であることから，テスト・ケース数は，10（$2^{2-1}(2+3)=10$）となる．

6．まとめ

n個の判定文をもつ鎖構造型制御フローは，直交表を用いることにより，最少でn+1，最大でも2n個のテスト・ケースでテストできることが示された．

鎖構造と木構造の違いによる，必要テスト・ケース数の差（2^n と n+1）は，エッジの網ら度とパスの網ら度との関係による．逆に言えば，木構造を主体としたプログラムの場合には，エッジ相互の副作用が少ないので，エッジの網らで十分な評価ができる．これに対して，鎖構造を主体としたプログラムの場合には，エッジ間の相互作用のため，エッジの網らでは，十分な品質評価は不可能となっている．

このことは，プログラムの複雑性とも深い関係があると考えられる．構造によっては，判定文の個数だけでは複雑性の評価はできない．テスト・ケース数との対比で考えれば，鎖構造は木構造の少くとも2倍，複雑であると言える．

参考文献

1) 大場「プログラムの制御フローに基づくテスト・ケース設計法」第28回全国大会 4k-4．
2) 広田他「制御変数に着目したデータフロー解析とその応用」第26回全国大会 1J-5．
3) 黒田他「プログラムのパスに基づくテスト法の有効性評価」第27回全国大会 4C-8．