Effect of Test Set Minimization on Fault Detection Effectiveness

W.Eric Wong, Joseph R. Horgan,
Saul London, and Aditya P. Mathur

(17th ICSE)

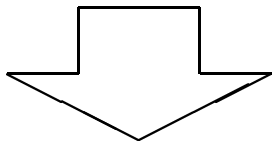
寺口 正義

井上研究室

概要

プログラム P がテスト集合 T のもとで実行されている

1. T のサイズ
2. T によって実行されるコードの割合
3. T のエラー発見効率



1．と 2．が 3．とどのような関係にあるのかを調べる

コードのカバー割合を一定とする

T のサイズを減少させた場合のエラーの発見効率の変化は?

基本概念と用語 (1 / 3)

テストケース 1 つの実行のプログラムに対する入力の系列

テスト集合 テストでプログラムが実行する 1 つもしくは複数の test case

Dataflow based adequacy criteria

基本ブロック 分岐を含まない文の列 $\rightarrow$ 1 文が実行されれば全て実行

テスト集合 T をブロック基準で評価

プログラムの総ブロック数に占める実行されるブロック数の割合

$s_i, s_j (1 \leq i, j \leq n)$: 2 つの文 (変数 x の定義と参照)
DU pair $(d_i(x), u_j(x))$

p-use 参照が条件文内となる DU pair の集合

c-use 参照が条件文以外 DU pair の集合

all-use p-use と c-use の和集合

テスト集合 T を all-use 基準で評価

プログラムの総 all-use 数に占めるカバーされる all-use の割合

基本概念と用語 (2 / 3)

Fault detection effectiveness

実験では

- (準備)
  10 個のプログラムを利用
  それぞれにエラー (の集合) を埋め込む
- (テスト)
  それぞれのテストケースに対して埋め込む前後の出力を比較
  違いが見つかればエラーが発見できたことになる
  テスト集合内のテストケースの内 1 つでもエラーを発見できればよい

特定のプログラムに対するテスト集合のエラー発見効率とは

埋め込んだエラー数に占める発見できたエラー数の割合

基本概念と用語 (3 / 3)

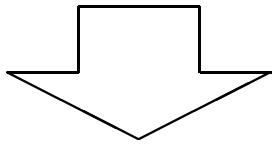
Test set minimization

テスト集合を最小化する目的

かかるコストを減少させる

コストとは

1. それぞれのテストケースにかかる時間の総和
2. テストケースの構築，分析に費やした時間の総和
3. テスト集合に含まれるテストケースの数 → 採用



all-use 基準においてもとの集合と同等の範囲をカバーする
最小テスト集合が見つかる

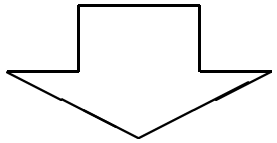
実験手法

1. プログラムの選択
2. テストケースプールの構築
3. テスト集合の生成
4. テスト集合の最適化
5. プログラムへのバグの埋め込み
6. 生成したテスト集合を用いたバグの発見

プログラムの選択

10 個の C プログラムを用意 (表 1)

全て日常で徹底的に利用されている



- エラーを埋め込んだ後のプログラムの振舞いを評価するのに信頼できる
- 自然と埋め込んだ以外のエラーが起こらないことが期待できる

テスト集合の生成と最適化

- プログラム仕様に従ってランダムに 1000 テストケースをそれぞれ生成 (表 2)

  2 つのテストケースが同じ実行トレースになるなら 1 つだけ採用

  → 自動生成のため完全なカバレッジは実現できない

- ブロックのカバレッジ割合を考慮してテスト集合をそれぞれ生成 (表 3)

  エラー発見実験前に全てのテスト集合を作成

  → ある種のエラーに特化したテスト集合の作成を防ぐ

- all-use カバレッジを減少させないようにテスト集合を最適化

  ブロックカバレッジによる最適化は行わない

  → 実際サイズの増大，エラー発見効率の現象を伴う

エラーの埋め込みと発見

- 院生 (C 経験 3 年以上) が構文的に正しくエラーを埋め込む (表 4)
- エラーの数 = プログラム数

  1 つのプログラムに複数のエラーがあるのが普通 (4 ~ 40 個/1000LOC)

  → Sort(842LOC) にだけ 1 つのプログラムに 25 個のエラーを埋め込む

  1 つのプログラムに複数のエラーが含まれる

  → バグを引き起こしたエラーを特定するのが困難

  → 実際の開発においてもデバグする時は 1 つのエラーに着目
- プール中のどのテストケースを用いても発見できないエラーは埋め込まない

  → 実際の現場で利用したいとわからないような難しいエラーを除く
- エラーを難易度によって分類

  プール中のテストケースの中でそのエラーを発見できる割合

エラー発見効率の減少

- (50-55)%

  → 見られない

- (60-65)%，(70-75)%，(80-85)%，(90-95)%

  → それぞれ 0.39%，0.23%，5.05%，6.55%以下である

- 10 個のプログラムの内 9 個は単調増加

  → *Comm* では 0.00%:(50-55)% - 4.65%:(90-95)%

テスト集合サイズの減少

- (80-85)%，(90-95)%

  → 全てで見られる

- (50-55)%，(60-65)%，(70-75)%，(80-85)%，(90-95)%

  → 最大でそれぞれ 6.55%，17.94%，31.17%，50.02%，68.23%である

高 (低) いブロックカバレッジ $\Longleftrightarrow$ 高 (低) いサイズ減少割合

例外：*Spline* の 7.38，31.17%:(70-75)%，2.10，3.33%:(80-85)%

(80-85)%をカバーする数個のテストケースからなるテスト集合は容易

(70-75)%をカバーする場合はテスト集合が大きくなる

→ テストケースがプログラムを覆う手法に依存

同一カバレッジ：サイズ減少が小 $\neq$ エラー発見効率の減少が小

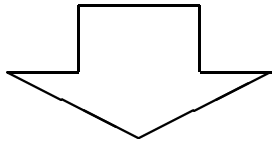
エラーの難易度とエラー発見効率の減少

T(テスト集合)，$\mathrm{T_m}$(最適化されたテスト集合)

エラー分類 $\mathrm{i}(1 \leq \mathrm{i} \leq 4)$ において T が $\mathrm{T_m}$ よりも多くのエラーを発見

→ エラー分類 i において T は最適化することによりエラー発見効率が減少
→ (他の分類 $\mathrm{i}'(\mathrm{i}' \neq \mathrm{i})$ においては成り立つとは限らない)

- エラー分類 4 では見られない
- エラー分類 1，2，3 では最大でそれぞれ 6.19%，6.67%，3.34% である



多くのテストケースによって発見されるエラー → 最適後も発見できる

少しのテストケースによって発見されるエラー → 最適後は発見できない

エラー発見効率の減少するテスト集合

例：プログラム Cal，エラー分類 2 において

$(60\text{-}65)\%$では $5.00\%(1$ 個$)$ のテスト集合で 1 つ以上のエラーを発見できない最適化する前のテスト集合で見つかったエラーの 0.83%を発見できない

表 3，表 7 からわかること

- $(50\text{-}55)\%$の 124 テスト集合
  $\rightarrow$ 減少しない
- 全 1022 テスト集合のうち，$48(4.70\%)$ テスト集合
  $\rightarrow$ 減少
- エラー分類 1，2，3
  $\rightarrow$ 全 1022 テストのうち，それぞれ 2.54%，2.25%，0.20%で減少
- エラー分類 4
  $\rightarrow$ 減少しない

さらなる考察 (1 / 2)

- 1 つのケーススタディを示したに過ぎない
  → 異なった特徴をもつプログラム，テストに対して適用していない
- プログラムと実験データが一緒にある環境においてのみ有効

エラー発見効率の減少

ブロックカバレッジに比例して増加

最適化による平均減少率

$(50\text{-}55)\% : 0.00\%$, $(60\text{-}65)\% : 0.03\%$, $(70\text{-}75)\% : 0.01\%$,
$(80\text{-}85)\% : 0.38\%$, $(90\text{-}95)\% : 1.45\%$

高いブロックカバレッジを持つテスト集合においても憂慮するほどではない

さらなる考察 (2 / 2)

テスト集合サイズの減少

ブロックカバレッジに比例して増加

最適化による平均減少率

(50-55)% : 1.19% , (60-65)% : 4.46% , (70-75)% : 7.78% ,
(80-85)% : 17.44% , (90-95)% : 44.23%

高いブロックカバレッジを持つテスト集合においても憂慮するほどではない

エラー分類によるエラー発見効率の減少

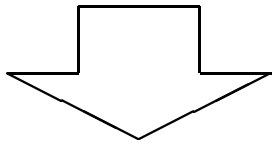
最適化による平均減少率

分類 1 : 0.39% , 分類 2 : 0.66% ,
分類 3 : 0.098% , 分類 4 : 0.00%

最も難しい分類 1 のエラーにおいても憂慮するほどではない

まとめ

- ある一定の all-use カバレッジを保ちつつテスト集合のサイズを減少
  → エラー発見効率はほとんど減少しない
- テスト集合にカバレッジを増やさないテストケースを追加
  → エラー発見において非効率



カバレッジを増やさないテストを削除することで，コストを削減可能