Avaliação do Impacto da Minimização de Conjuntos de Casos de Teste no Custo e Eficácia do Critério Análise de Mutantes

SIMONE DO ROCIO SENGER DE SOUZA[1]
JOSÉ CARLOS MALDONADO[2]

[1] Universidade Estadual de Ponta Grossa - UEPG
Praça Santos Andrade, S/N,
CEP: 84010-330, CX.P. 992/3,
Ponta Grossa - PR, Brasil
rocio@icmsc.sc.usp.br

[2] Instituto de Ciências Matemáticas de São Carlos - ICMSC/USP
R: Dr. Carlos Botelho, 1466,
CEP: 13560-970, CX.P. 668,
São Carlos - SP, Brasil
jcmaldon@icmsc.sc.usp.br

Resumo

Estratégias de minimização de conjuntos de casos de teste são relevantes para as fases de teste e manutenção do software. Muitos estudos empíricos têm sido conduzidos para avaliar o impacto da minimização no custo e eficácia dos critérios de teste. Este artigo apresenta a avaliação de uma estratégia de minimização de conjuntos de casos de teste desenvolvida a partir do algoritmo de minimização proposto por Harrold [HAR93]. O módulo de minimização está integrado à *Proteum* – uma ferramenta de apoio ao critério Análise de Mutantes para o teste de programas na linguagem C –. A avaliação foi realizada através de dois estudos empíricos, onde os resultados indicam uma redução significativa no custo e uma pequena diminuição na eficácia em revelar erros do critério Análise de Mutantes.

Palavras Chaves: Minimização de conjuntos de casos de teste, Critério Análise de Mutantes, Ferramentas de teste, Teste de software.

Abstract

Test case set minimization strategies play a relevant rule in software testing and maintenance as well. Many empirical studies have been conducted to evaluate its impact on testing cost and effectiveness. In this paper it is presented and evaluate a minimization strategy based on Harrold et al's work [HAR93]. The minimization module has been integrated to *Proteum*, a testing tool that suports the Mutation Analisys criterion for testing C programs. The evaluation was carried out by two empirical studies. The results indicate a significant reduction on the cost and a small reduction on the effectiveness of the Mutation Analisys application.

Keywords: Test case set minimization, Mutation Analisys criterion, Testing tool, Software testing.

1. Introdução

Teste de software é uma das atividades de garantia de qualidade mais utilizada, que tem por objetivo assegurar o desenvolvimento do software de acordo com o esperado, com o mínimo de erros possível. Para a condução dos testes, critérios devem ser utilizados visando a sistematizar essa atividade como também a reduzir os seus custos.

Durante a condução dos testes são construídos casos de teste procurando-se observar os requisitos estabelecidos por um determinado critério. Entretanto, dependendo do critério e da maneira com que os casos de teste são gerados (por exemplo, geração aleatória), o conjunto obtido pode tornar-se bastante grande. Além disso, com a evolução do software, decorrente da atividade de manutenção, muitos dos casos de teste podem tornar-se obsoletos, não fazendo mais sentido sua existência no conjunto.

Vários autores têm proposto estratégias que visam a reduzir os custos dos testes de regressão – testes realizados após algum tipo de manutenção com o objetivo de revalidar o software evidenciando a presença de possíveis defeitos introduzidos com as alterações – [CHU87, OST88, LEU91, HAR93, WON97]. As estratégias propostas procuram selecionar do conjunto de casos de teste um subconjunto, que seja eficaz para revelar os erros e permaneça adequado ao critério de teste empregado.

Assim, a minimização de conjuntos de casos de teste tem o objetivo de, a partir de um conjunto de casos de teste **T**, gerar um conjunto $\mathbf{T}_{min}$ com o mesmo grau de adequação de **T**, geralmente com um tamanho menor. A redução do tamanho do conjunto **T** é obtida pela eliminação de casos de teste redundantes e obsoletos. A minimização procura beneficiar os testes de regressão, como também fornecer um mecanismo mais real para a avaliação do custo de aplicação de um critério. Wong realizou um estudo nesse sentido, avaliando o efeito da minimização na eficácia em revelar erros do critério baseado em Análise de Fluxo de Dados Todos-Usos [WON94a]; Wong observou que a redução no tamanho dos conjuntos não afeta a eficácia desses conjuntos, onde para poucos casos houve uma redução na eficácia, considerada pouco significativa.

Um outro critério que tem sido estudado intensivamente é o critério Análise de Mutantes [MAT93, MAT94, OFF96a, OFF96b, WON93, WON94b, WON95, WON97]. A Análise de Mutantes é um critério de teste baseado em erros que, para avaliar a adequação de um conjunto de casos de teste **T** em relação a um programa **P**, utiliza um conjunto de programas, ligeiramente diferentes de **P**, chamados **mutantes**. O objetivo é obter um conjunto com casos de teste que consiga revelar as diferenças de comportamento existentes entre **P** e seus mutantes [DEM80]. Esse critério apresenta-se bastante eficaz para revelar erros [MAT94, OFF96a]; entretanto o seu alto custo devido ao número de mutantes gerados e que precisam ser executados pelo conjunto de casos de teste é um fator que limita sua aplicação. Pesquisas procuram obter meios de diminuir o custo sem perder a eficácia desse critério; uma delas propõe a utilização de **mutação alternativa** ou **seletiva** a qual seleciona um conjunto de operadores de mutação para geração dos mutantes, diminuindo o número de mutantes gerados [MAT93, OFF96b, WON95].

Várias estratégias para a minimização de conjuntos de casos de teste têm sido investigadas [CHU87, HAR93]. O problema de obter um conjunto que seja minimal é *np-completo*; para contornar esse aspecto, *"guidelines"* e heurísticas são utilizadas para a proposição dessas estratégias.

No contexto do critério Análise de Mutantes não existe disponível a implementação dessas estratégias. Para atingir os objetivos deste trabalho, ou seja, para avaliar o efeito da minimização de conjuntos de casos de teste no custo e eficácia do critério Análise de Mutantes

fez-se necessário implementar um módulo que realizasse essa minimização e integrá-lo à ferramenta *Proteum* - ferramenta que apoia a aplicação do critério Análise de Mutantes para o teste de programas na linguagem C [DEL96]. Esse módulo foi implementado baseando-se no algoritmo de minimização proposto por Harrold [HAR93], o qual é apresentado na Seção 3.

Dessa forma, este artigo está organizado da seguinte maneira: na Seção 2 são apresentados alguns conceitos básicos a respeito da minimização de conjuntos de casos de teste e estudos empíricos. Na Seção 3 são apresentadas a especificação e os aspectos de implementação do módulo de minimização de conjuntos de casos de teste, denominado *Minimize*. A Seção 4 apresenta o experimento realizado para avaliar o impacto da minimização no custo e na eficácia do critério Análise de Mutantes. Finalmente, na Seção 5 têm-se as conclusões deste trabalho.

2. Conceitos Básicos e Terminologias

O objetivo de minimizar um conjunto de casos de teste é reduzir os custos da atividade de teste e de manutenção, os quais podem ser mensurados de várias maneiras. Uma maneira considera o tempo necessário para executar cada caso de teste; uma outra considera o tempo gasto para construir os casos de teste. Nesse sentido, a redução do número de casos de teste de um conjunto possibilita que, durante os testes de regressão, o tempo de execução seja menor.

Os critérios de adequação fornecem mecanismos para sistematicamente selecionar um subconjunto do domínio de entrada do programa em teste e ainda assim ser eficaz para revelar os erros existentes. Para um particular critério de adequação existem infinitos subconjuntos de casos de teste adequados em relação a um programa. Além disso, para um dado conjunto de casos de teste T_a adequado a um critério **C** em relação a um programa **P**, qualquer que seja $T_1 \in D$ (domínio de entrada do programa), $T = T_a \cup T_1$ também é adequado a **C**. Assim, dado um conjunto de casos de teste **T** esse conjunto pode ser reduzido, eliminando-se os casos de teste que exercitam os mesmos requisitos de teste estabelecidos pelo critério **C**.

Para um determinado conjunto de casos de teste **T** adequado a **C** a minimização desse conjunto pode resultar em vários, não somente um, conjunto T_{min} com cardinalidade (número de casos de teste) menor que **T** e que sejam ainda adequados a **C**.

Definição: **T** é um conjunto de casos de teste para um programa **P**. |T| denota o número de elementos em **T**. **C** é um critério de adequação. C(T) denota a cobertura de **T** para **P** em relação a **C**. T_m é o subconjunto minimal de **T** em termos do número de casos de teste se $C(T_m) = C(T)$ e $\forall\ T' \subseteq T$ onde $C(T') = C(T)$, $|T_m| \leq |T'|$.

As estratégias para redução de casos de teste obtêm um subconjunto com casos de teste representativos para o critério de adequação; entretanto a redução máxima do conjunto é uma atividade difícil de se atingir na prática e com isso, as estratégias existentes procuram soluções para aproximar-se o máximo possível desse objetivo, visto que o problema de encontrar um subconjunto minimal é *NP-Completo* [GAR79].

Para avaliação dos critérios de teste, três fatores comparativos são utilizados [WON93]: custo, eficácia e *strength* (dificuldade de satisfação). Custo refere-se ao esforço necessário para que o critério seja usado, o qual pode ser medido pelo número de casos de teste necessários para satisfazer o critério, ou por outras métricas dependentes do critério, tais como: o tempo necessário para executar todos os mutantes gerados (para o critério Análise de Mutantes) ou o tempo gasto para identificar mutantes equivalentes, caminhos e associações não executáveis, construir manualmente os casos de teste e aprender a utilizar as ferramentas de teste. A eficácia refere-se à capacidade que um critério possui de detectar um maior número

de erros em relação a outro. O fator *strength* refere-se à probabilidade de satisfazer-se um critério tendo satisfeito outro critério. Neste trabalho os fatores custo e eficácia são utilizados para avaliar o impacto da minimização dos conjuntos adequados ao critério Análise de Mutantes.

3. Especificação e Implementação do Módulo *Minimize*

Devido à não disponibilidade de mecanismos de minimização de conjuntos de casos de teste para o critério Análise de Mutantes, em particular para a linguagem C, fez-se necessário implementar essa funcionalidade. Essa funcionalidade está agregada a *Proteum* (*Program Testing Using Mutants*), uma ferramenta de apoio ao critério Análise de Mutantes para programas escritos na linguagem C [DEL93]. A implementação do módulo de minimização baseou-se no algoritmo proposto por Harrold [HAR93], discutido a seguir. Optou-se por esse algoritmo devido o mesmo ser independente do critério de teste, diferente do algoritmo de Chusho, que foi desenvolvido para o teste de caminhos [CHU87]. Além disso, considerando o critério Análise de Mutantes e a ferramenta *Proteum*, são necessárias menos comparações para obter o conjunto mínimo com o algoritmo de Harrold do que o algoritmo de Chusho.

O módulo de minimização, denominado *Minimize*, tem a função de obter um subconjunto de casos de teste a partir de um conjunto de casos de teste adequado ao critério Análise de Mutantes, pertencente a uma sessão de teste desenvolvida na ferramenta *Proteum* [DEL96]. O módulo recebe como entrada uma sessão de teste com seu conjunto de casos de teste **T** e tem como saída um subconjunto de casos de teste de **T** com o mesmo grau de adequação. Os seguintes requisitos foram observados:

- possibilitar manter no conjunto mínimo um determinado caso de teste ou um subconjunto de casos de teste. Às vezes é necessário manter os casos de teste gerados para um particular critério de teste (por exemplo, para um critério funcional), ou manter casos de teste que revelam erros, ou ainda casos de teste que exploram características específicas do software;
- seguir o mesmo padrão de desenvolvimento dos demais módulos da ferramenta *Proteum*, baseando-se na sua estrutura e utilizando, quando possível, as bibliotecas desenvolvidas para os módulos existentes; e
- permitir a execução do módulo *Minimize* via *shell scripts* da mesma forma que os demais módulos da ferramenta *Proteum 1.2* [DEL96].

• Algoritmo de Minimização

A estratégia de minimização proposta por Harrold [HAR93] tem a característica de ser independente do critério de adequação empregado, ou seja, a técnica de redução requer associações entre os casos de teste e os requisitos de teste a serem satisfeitos. Esses requisitos podem ser: a especificação do sistema (para o teste funcional), os componentes do programa, como caminhos (para o teste estrutural) ou os mutantes gerados para um programa (no caso para Análise de Mutantes). Naturalmente, à medida que modificações são feitas no programa os requisitos de teste irão modificar-se também, sendo acrescentados novos requisitos ou eliminados os que já existem.

O problema de selecionar um conjunto de casos de teste representativo que possui o mesmo grau de cobertura do conjunto inicial é dado da seguinte forma:

Dados: Um conjunto de casos de teste **TS**, um conjunto de requisitos de teste $r_1, r_2, \ldots, r_n$ que devem ser satisfeitos para fornecer o grau de cobertura desejado para o

programa e os subconjuntos de **TS:** $\mathbf{T_1}$, $\mathbf{T_2}$, ..., $\mathbf{T_n}$, os quais estão associados aos $\mathbf{r_i}$'s onde qualquer $\mathbf{t_j}$ pertencente a $\mathbf{T_i}$ pode ser usado para testar $\mathbf{r_i}$.

Problema: Encontrar um conjunto de casos de teste representativo (**RS**) a partir de **TS** que satisfaz todos os requisitos $\mathbf{r_i}$.

Para obter um conjunto mínimo que satisfaça todos os requisitos $\mathbf{r_i}$'s são necessárias heurísticas para selecionar quais casos de teste de **TS** devem fazer parte do conjunto **RS.** O conjunto **RS** representa o conjunto mínimo de casos de teste para o programa em teste, onde a redução máxima do conjunto é obtida quando encontra-se o menor conjunto representativo de casos de teste a partir de **TS**. Como a obtenção de **RS** mínimo é difícil na prática, as heurísticas procuram fornecer soluções que aproximam de um conjunto de cardinalidade mínima [HAR93]. As heurísticas do algoritmo proposto por Harrold são:

1. obter para cada requisito de teste r_i seu conjunto T_i composto de todos os casos de teste que satisfazem esse requisito;

2. adicionar no conjunto RS todos os casos de teste t_i pertencentes a conjuntos T_j unários:

$RS = \{ t_i \mid t_i \in T_j \wedge Card(T_j) = 1 \}$, onde $Card(T_j)$ = número de elementos de T_j.

3. marcar todos os conjuntos T_j que possuam os casos de testes inseridos em RS:

$Marked[T_j] = true \mid \forall t_i \in T_j,\ t_i \in RS$

4. para todos os conjuntos não marcados T_j 's de tamanho n (n = 1, 2, 3, ...) selecionar um caso de teste t_i que ocorre no maior número de T_j 's não marcados de cardinalidade n e adicionar em RS;

$RS = RS \cup \{t_i\} \mid t_i \in T_j \wedge \{Marked[T_j] = false \wedge Card(T_j) = n \wedge count[t_i] = max\_ocor$

onde max_ocor = maior número de ocorrências dos casos de teste nos conjuntos T_j's.

Durante o passo 4 pode ocorrer que vários casos de teste, ocorram o mesmo número de vezes nos conjuntos T_j's de cardinalidade n. Quando isso ocorre, é necessário examinar os conjuntos T_j's não marcados de cardinalidade n + 1 para os casos de teste envolvidos no empate. Caso o empate continue e a cardinalidade já é a máxima, então escolhe-se aleatoriamente um dos casos de teste para ser adicionado ao conjunto RS.

5. repetir os passos 3 e 4 enquanto existirem conjuntos não marcados.

Esse algoritmo foi adaptado para possibilitar que o primeiro requisito definido para o módulo pudesse ser atendido. Esse requisito estabelece que o módulo de minimização permita que alguns casos de teste sejam mantidos no conjunto mínimo, de forma a realizar uma minimização restrita. Na Figura 1, tem-se o algoritmo de Harrold com as extensões realizadas.

```
Algoritmo ReduceTestSuite
input  T1, T2,..Tn: conj. de c. de t. e associados com r1, r2, .., rn respectivamente, contendo c. de t. de t1, t2, ..tot.
       R1, R2,..Rn: conjuntos de casos de teste requeridos
output RS: um conjunto representativo de T1, T2,..Tn.
declare MAX_CARD, CUR_CARD: 1...nt
       LIST: lista de ti's
       NEXT_TEST: um dos casos de teste t1, t2, ..tot
       MARKED: vetor [1..n] de boleanos, inicializado com falso
       MAY_REDUCE: boleano
       Max( ): função que retorna o valor máximo do conjunto
       Card( ): função que retorna a cardinalidade (número de elementos) do conjunto
begin
/* Passo 1: Inicialização */
       RS := ∪Ri                    /* recebe todos os casos de teste requeridos */
       foreach Ti onde Ti ∩ RS ≠ ∅ do MARKED[I] := true  /* marca todos os Ti que possuem
                                                          casos de teste requeridos */
       MAX_CARD := Max(Card(Tj))    /* maior cardinalidade de Ti 's */
       RS := ∪Ti, Card(Ti) = 1      /* recebe todos os elementos únicos de Ti 's */
       foreach Ti onde Ti ∩ RS ≠ ∅ do MARKED[I] := true  /* marca todos os Ti que têm elem. em RS */
       CUR_CARD := 1
/* Passo 2: Calcula RS de acordo com heurísticas definidas */
       loop
         CUR_CARD := CUR_CARD + 1
         while existe Ti onde Card(Ti) = CUR_CARD and not MARKED[ i ] do
              LIST := todos tj ∈ Ti onde Card(Ti) = CUR_CARD and not MARKED[ i ]
              NEXT_TEST := SelectTest(CUR_CARD, LIST)
              RS := RS ∪ {NEXT_TEST}        /* adiciona NEXT_TEST em RS */
              MAY_REDUCE := false
              foreach Ti onde NEXT_TEST ∈ Ti do
                   MARKED[ i ] := true      /* marca Ti que contém NEXT_TEST */
                   if Card(Ti) = MAX_CARD then MAY_REDUCE := 1
              endfor
              if MAY_REDUCE then
                   MAX_CARD := Max(Card(Ti)), para todo i onde MARKED[ i ] = false
         endwhile
       until CUR_CARD = MAX_CARD
end ReduceTestSuite.
.....
function SelectTest(SIZE, LIST)  /* seleciona o próximo ti para incluir em RS */
delcare COUNT: array[1..nt]
begin
       foreach ti em LIST do calcule COUNT[ i ], o número de não marcados Ti 's de cardinalidade SIZE
contendo ti
       Construir TESTLIST contendo os casos de teste de LIST onde COUNT[I] é máximo
       if Card(TESTLIST) = 1 then return(o caso de teste em TESTLIST)
       elseif SIZE = MAX_CARD then return(qualquer caso de teste de TESTLIST)
       else return(SelectTest(SIZE+1, TESTLIST)
end SelectTest.
```

Figura 1. Algoritmo de Harrold Estendido.

- **O Módulo *Minimize* para a Ferramenta *Proteum***

O módulo *Minimize* se relaciona com os demais módulos da ferramenta *Proteum* a partir das bases de dados atualizadas e mantidas por esses módulos. Na Figura 2 tem-se o Diagrama de Fluxo de Dados da ferramenta *Proteum*, onde o módulo implementado apresenta-se destacado.

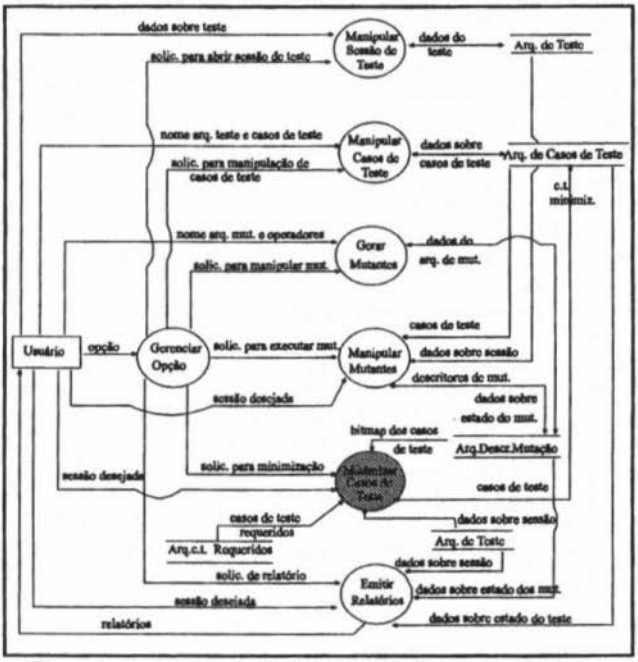

Figura 2. Diagrama de Fluxo de Dados da Ferramenta *Proteum*.

A minimização é realizada sobre um sessão de teste já existente, composta de um conjunto de casos de teste adequado ao critério Análise de Mutantes. Para utilizar o módulo *Minimize* os seguintes parâmetros estão disponíveis, onde os parâmetros entre [ ] são opcionais:

minimize [-DR directory] [-R filename] [-DM directory] [-M filename] [-t n] [-f m] [-v] [-D directory] test-name

[-DR directory] - diretório onde encontra-se o arquivo com *required test cases*. O diretório corrente é utilizado como *default*.

[-R filename] - fornece ao módulo o nome do arquivo que contêm os casos de teste requeridos (*required test cases*). Esse parâmetro está de acordo com o primeiro requisito estipulado para o módulo.

[-DM directory] - fornece ao módulo em qual diretório o arquivo de casos de teste mínimo deve ser gravado. O diretório corrente é utilizado como *default*.

[-M filename] - possibilita que seja fornecido um nome para o arquivo de casos de teste mínimo, sem extensão pois a mesma é fixa e fornecida pelo módulo. Se esse parâmetro for omitido, o nome do arquivo será o mesmo da sessão de teste.

[-t n] [-f m] - permitem estabelecer um intervalo de casos de teste a ser considerado na minimização. Dessa forma, *n* indica o número do caso de teste inicial e *m* o número do último caso de teste considerado para minimização. O valor *default* para *n* é 0 e para *m* é o tamanho máximo do conjunto de casos de teste. Da mesma forma que a opção -R esses parâmetros atendem ao primeiro requisito definido para o módulo.

[-v] - permite que o usuário visualize se a execução está ou não em andamento. Dependendo do número de casos de teste e de mutantes existentes na sessão de teste, a minimização pode consumir um certo tempo, podendo levar o usuário a imaginar que o sistema está com algum problema.

[-D directory] - fornece ao módulo o diretório onde se encontra a sessão de teste. O diretório corrente é utilizado como *default.*

test-name - nome da sessão de teste associada ao conjunto a ser minimizado, sendo esse o único parâmetro que deve ser obrigatoriamente fornecido ao módulo.

Um problema que deve ser considerado cuidadosamente na minimização de conjuntos de casos de teste é a eficácia em revelar erros desses conjuntos, porque durante a minimização podem ser eliminados casos de teste capazes de revelar os erros existentes no programa. Wong realizou um experimento empírico neste sentido, visando a avaliar a eficácia dos conjuntos minimizados para o critério Todos-Usos [WON93, WON94a]. Como resultados desse estudo, tem-se que a redução do tamanho dos conjuntos de casos de teste variou em torno de 1.19% a 45%. Com relação à eficácia, o autor concluiu que, para os programas utilizados, a redução da eficácia dos conjuntos mínimos é muito pequena, nula na maioria dos casos; a redução da eficácia variou no intervalo de 0% a 1.45%. Ressalta-se que esses resultados foram obtidos somente para os critérios de Fluxo de Dados. A seguir apresenta-se um estudo realizado visando validar o módulo implementado e avaliar o efeito da minimização sobre o custo e a eficácia em revelar erros para o critério Análise de Mutantes, a exemplo do trabalho de Wong [WON94a].

4. Avaliação Empírica do Impacto da Minimização no Custo e Eficácia

Para a validação do módulo *Minimize* utilizou-se um *benchmark* composto de 27 programas escritos na linguagem C, os quais fazem parte de um experimento previamente desenvolvido por Souza para avaliar o custo de aplicação e *strength* dos critérios Análise de Mutantes e Potenciais-Usos [SOU97]. Os conjuntos AM-adequados dos programas (adequados ao critério Análise de Mutantes) obtidos por Souza foram minimizados, conseguindo-se os resultados apresentados na Tabela 1. Os resultados obtidos demonstram uma significativa redução no número de casos de teste para os conjuntos. A redução no tamanho dos conjuntos variou no intervalo de 0% a 87.2%, o que constata sua importância para diminuir os custos associados, onde, na média, a diminuição no tamanho dos conjuntos foi de cerca de 50%. Os resultados demonstram a validade e a importância de estratégias de minimização de conjuntos de casos de teste para redução dos custos na atividade de teste e motivaram a realização do experimento para avaliação do impacto da minimização na eficácia desse critério.

Tabela 1. Resultado da Minimização de Conjuntos AM-Adequados.

Programas	Tamanho dos Conjuntos		% Minimização
	AM-Adequados	AM-Minimizados	
Append	17	7	58.8
Archive	15	13	13.3
Change	11	5	54.5
Ckglob	14	6	57.1
Cmp	16	9	43.7
Command	88	34	61.4
Compare	13	4	69.2
Compress	16	6	62.5
Dodash	60	17	71.7
Edit	47	6	87.2
Entab	14	2	80.0
Expand	18	6	66.7
Getcmd	15	15	0.0
Getdef	40	12	70.0
Getfn	21	9	57.1
Getfns	13	9	30.8
Getlist	30	8	73.3
Getnum	14	10	28.6
Getone	26	10	61.5
Gtext	7	4	42.9
Makepat	49	20	59.2
Omatch	50	23	54.0
Optpat	15	10	33.3
Spread	18	12	33.3
Subst	46	13	71.7
Translit	64	12	81.3
Unrotate	8	2	75.0
Média	27.6	10.5	55.5

Para avaliar o impacto da minimização na eficácia utilizaram-se cinco programas com erros, extraídos do trabalho de Wong [WON93]. Abaixo tem-se a especificação dos programas utilizados juntamente com o tipo de erro existente para cada um:

Find: recebe como parâmetros um vetor de inteiros *a* e um índice *f*. Os elementos de *a* são permutados de maneira que, depois de permutados, os elementos à direita da posição *f* são maiores que ou iguais a *a[f]* e os elementos à esquerda da posição *f* são menores que ou iguais a *a[f]*.

Erro: ocorre quando o número de elementos de *a* é menor que o valor de *f*.

Mysort: ordena um vetor de entrada *a* em ordem decrescente.

Erro: quando o elemento comparado *a[x]* é maior que os elementos nas posições seguintes de *a[x]* um valor antigo é atribuído a uma das posições do vetor *a*. Esse erro origina um vetor com valores repetidos, desaparecendo alguns valores de entrada.

Position: recebe como parâmetros um vetor de inteiros *a* e um valor *max*. Os elementos de *a* são somados até que a soma seja igual ou maior que *max*. Quando isso ocorre é retornado a posição do vetor; caso contrário, o valor retornado é 0.

Erro: irá ocorrer quando a soma de todos os elementos de *a* é igual a *max*, e quando a soma ultrapassa *max* a posição retornada é do elemento anterior à posição onde a soma excede *max*.

Stat: A partir de um vetor de inteiros *a* calcula a soma, valor mínimo e máximo do vetor.

Erro: é detectado quando o valor de *a[1]* é mínimo e diferente de 0 ou quando o valor mínimo de *a* é maior que 0.

Strmatch: a partir de um texto e um padrão de entrada de 0 ou mais caracteres verifica se o padrão aparece no texto. Se isso ocorre, a primeira posição da ocorrência do padrão é retornada; caso contrário, retorna o valor 0.

Erro: será detectado caso ocorra uma das seguintes situações: 1) o texto tem mais que 10 caracteres, 2) o padrão tem mais que 3 caracteres ou 3) o texto e o padrão são vazios.

A partir desses programas foram realizados dez experimentos diferentes, variando-se o domínio de entrada dos programas (Tabela 2).

Tabela 2. Conjunto de Experimentos.

Experimento	Programa	Domínio de Entrada*
FIND	FIND	• $-5 \leq$ tamanho vetor ≤ 10 • $-10 \leq$ elementos do vetor ≤ 100 • se tamanho do vetor > 0 então $1 \leq$ índice $\leq$ tamanho do vetor senão $-3 \leq$ índice ≤ 2
POS1	POSITION	• tamanho do vetor = 5 • $-5 \leq$ elementos do vetor ≤ 10 • $max \in \{x \mid x = 5$ ou $35 \leq x \leq 55\}$
POS2	POSITION	• tamanho do vetor = 5 • elementos do vetor $\in$ {0,1,2,3,4,5,6,-5,8} • $max \in$ {-5, 0, 1, 4, 8}
SORT1	MYSORT	• $-1 \leq$ tamanho vetor ≤ 10 • $-10 \leq$ elementos do vetor ≤ 100
SORT2	MYSORT	• $-1 \leq$ tamanho vetor ≤ 3 • $-2 \leq$ elementos do vetor ≤ 2
STAT1	STAT	• tamanho vetor = 5 • $-20 \leq$ elementos do vetor ≤ 20
STAT2	STAT	• tamanho vetor = 5 • $-20 \leq$ elementos do vetor ≤ 8
STRM1	STRMATCH	• $0 \leq$ tamanho do texto ≤ 15 • $0 \leq$ tamanho do padrão ≤ 6 • elementos do texto $\in$ {a,b,#} • elementos do padrão $\in$ {a,b,#}
STRM2	STRMATCH	• $0 \leq$ tamanho do texto ≤ 12 • $0 \leq$ tamanho do padrão ≤ 4 • elementos do texto $\in$ {a,b,#} • elementos do padrão $\in$ {a,b,#}
STRM3	STRMATCH	• $0 \leq$ tamanho do texto ≤ 11 • $0 \leq$ tamanho do padrão ≤ 4 • elementos do texto $\in$ {a,b,#} • elementos do padrão $\in$ {a,b,#}

*Todas as entradas são inteiros.

Para cada experimento foram gerados aleatoriamente dez conjuntos, contendo cada um 200 casos de teste. A geração de dados de teste aleatória possui a vantagem de gerar grandes

conjuntos de teste de uma forma mais econômica e, principalmente, elimina qualquer influência do testador em criar casos de teste de acordo com o conhecimento do programa em teste, o que poderia "viciar" os resultados obtidos. A utilização de vários conjuntos de casos de teste é necessária porque existem infinitos conjuntos de casos de teste que podem satisfazer um determinado critério e, utilizando somente um conjunto os resultados obtidos podem levar a falsas conclusões [WON93].

Basicamente, o experimento foi conduzido de acordo com as seguintes etapas:

1. Seleção e preparação dos programas;
2. Seleção das ferramentas de teste;
3. Geração dos conjuntos inicias de casos de teste (geração aleatória);
4. Criação e execução dos programas *scripts* para cada experimento;
5. Análise dos mutantes vivos para determinar equivalência;
6. Geração dos conjuntos AM-adequados;
7. Análise da eficácia dos conjuntos AM-adequados;
8. Minimização dos conjuntos AM-adequados;
9. Análise da redução no tamanho dos conjuntos de casos de teste;
10. Análise da eficácia dos conjuntos AM-adequados mínimos.

Para geração dos mutantes utilizou-se a estratégia de **mutação restrita** (*constrained mutation*). Nessa estratégia são selecionados operadores de mutação específicos para gerar os mutantes. O objetivo é diminuir o número de mutantes gerados sem reduzir a eficácia do critério. Na Tabela 3 são apresentadas as seis classes de mutação restrita utilizadas no experimento, as quais foram definidas por Wong [WON94b].

Tabela 3. Classes de Mutação Restrita.

Classes	Operadores de Mutação
MUT-A	*olln, olng, orrn*
MUT-B	*olln, olng, orrn, ocng, orln, olrn, olan, oaln*
MUT-C	*vdtr, vtwd*
MUT-D	*strp*
MUT-E	*olln, olng,orrn, vdtr, vtwd*
MUT-F	*olln, olng, orrn, vdtr, vtwd, strp*

4.1. Análise do Custo

Para calcular a redução de tamanho do conjunto após realizada a minimização utiliza-se a equação abaixo, extraída do trabalho de Wong [WON93]. Na equação $|T|$ representa o tamanho do conjunto de casos de teste e $|T_{min}|$ o tamanho do conjunto após a minimização.

$$(1) \qquad \frac{|T| - |T_{min}|}{|T|} * 100$$

O Gráfico 1 apresenta o tamanho médio dos conjuntos de casos de teste adequados às classes de mutação restrita (AM-adequados) antes e depois da minimização. A Tabela 4 apresenta a porcentagem de minimização para cada experimento, onde os dados foram obtidos utilizando-se a Equação 1. Para cada classe tem-se a porcentagem média de redução e o respectivo desvio padrão. A partir dos resultados apresentados pode-se observar que:

- A classe de mutação restrita MUT-D foi a que apresentou menor porcentagem de minimização (cerca de 25% na média).

• A classe MUT-F foi a que apresentou maior porcentagem de minimização (cerca de 40% na média).

• O experimento POS1 apresentou as maiores porcentagens de minimização para todas as classes de mutação restrita, ao contrário do experimento STAT2 que apresentou, para a maioria das classes de mutação, as menores porcentagens.

• Somente para os conjuntos AM-adequados da classe MUT-B do experimento STAT2 não ocorreu nenhuma redução no tamanho dos conjuntos após a minimização.

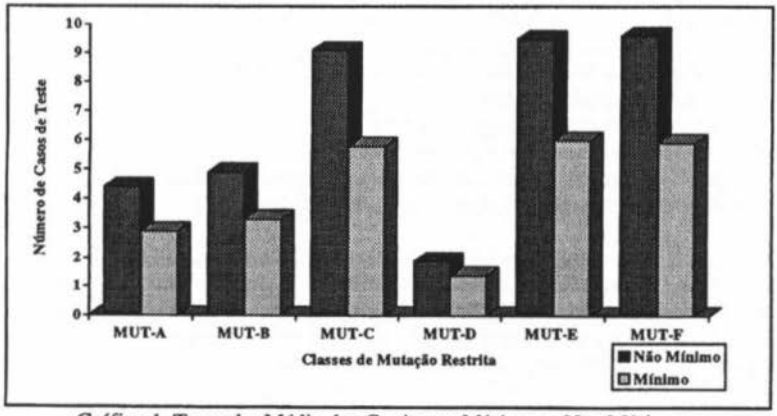

Gráfico 1. Tamanho Médio dos Conjuntos Mínimos e Não Mínimos.

Tabela 4. Porcentagem de Minimização Obtida.

Experimentos	Porcentagem de Minimização					
	MUT-A	MUT-B	MUT-C	MUT-D	MUT-E	MUT-F
FIND	36.4	30.8	41.5	34.4	45.8	44.4
POS1	52.9	45.8	55.8	47.4	57.7	59.5
POS2	46.4	42.2	37.1	9.1	40.6	40.6
SORT1	25.6	28.9	34.0	16.7	38.8	36.3
SORT2	18.7	32.0	31.1	33.3	30.4	36.6
STAT1	13.0	9.1	40.7	23.1	42.4	42.6
STAT2	9.1	0.0	40.4	16.7	43.1	46.5
STRM1	42.9	32.2	28.1	16.7	22.5	36.7
STRM2	41.9	42.3	38.0	31.0	29.1	27.8
STRM3	40.0	42.0	24.2	23.1	23.6	28.2
Média	32.7	30.6	37.1	25.1	37.4	39.9
Desvio Padrão	15.066	15.051	8.735	11.360	10.968	9.268

Esses resultados demonstram que a minimização é diretamente proporcional ao tamanho dos conjuntos de casos de teste, ou seja, quanto maior o conjunto maior a porcentagem de minimização obtida. Por exemplo, a classe MUT-F é a que possui um maior número de casos de teste (9.6 na média) e apresentou uma maior porcentagem de minimização (39.9%); por outro lado, a classe MUT-D possui menor número de casos de teste (1.5 casos de teste na média) e, em geral, obteve porcentagens de minimização mais baixas que o restante das classes de mutação restrita (25.1%).

Os resultados obtidos demonstram uma significativa redução no número de casos de teste para os conjuntos. A redução variou no intervalo de 25.1% a 39.9%, o que constata sua importância para diminuir os custos associados.

4.2. Análise da Eficácia

A eficácia de um critério de teste é obtida através dos casos de teste capazes de revelar os erros existentes. Assim, dado um programa P, uma especificação S e um conjunto de casos de teste T adequado para um critério C, tem-se que T é capaz de revelar os erros de P se existe pelo menos um caso de teste $t \in T$ o qual faz com que P tenha um comportamento diferente de S. Se T é capaz de detectar no mínimo um erro de P tem-se que T é "um conjunto de casos de teste revelador de erros/defeitos (*fault revealing*)". Para avaliar a eficácia dos conjuntos AM-adequados utilizou-se a Equação 2, extraída do experimento de Wong [WON93].

$$(2) \qquad \frac{\#\text{ de conjuntos } T \text{ que revelam no mínimo um erro/defeito}}{\#\text{ de conjuntos } T \text{ gerados}} * 100\%$$

Tendo-se os conjuntos mínimos e utilizando-se a Equação 2, obteve-se a porcentagem de eficácia dos conjuntos antes e depois da minimização. Esses resultados são apresentados no Gráfico 2. Sintetizando esses dados, a Tabela 5 apresenta a porcentagem de redução da eficácia dos conjuntos mínimos em relação aos conjuntos não mínimos.

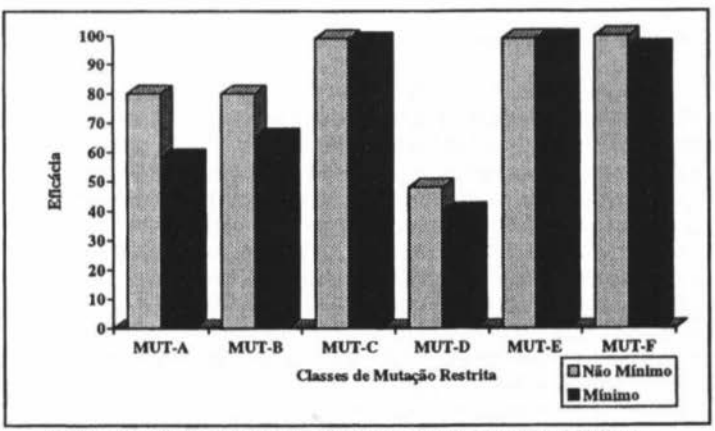

Gráfico 2. Eficácia Média dos Conjuntos Mínimos e Não Mínimos.

A partir dos resultados apresentados pode-se observar que:

- Somente o experimento STAT2 para a classe de mutação MUT-D não teve nenhum conjunto mínimo com casos de teste que revelassem o erro. Observe-se que o conjunto não mínimo também não tinha nenhum caso de teste capaz de revelar o erro existente;
- Para o programa FIND não houve redução da eficácia, ou seja, as porcentagens obtidas tanto para os conjuntos mínimos como não mínimos foram iguais para todas as classes de mutação restrita;

• Para a classe de mutação MUT-E não houve redução da eficácia dos conjuntos após realizada a minimização.

Observa-se na Tabela 5 que as classes MUT-A e MUT-B apresentaram uma maior redução na eficácia (29.2% e 20.9%, respectivamente). Para as demais classes a redução na eficácia foi bastante pequena e pouco significativa. Para a classe MUT-E, por exemplo, não houve nenhuma redução na capacidade de revelar os erros existentes após realizada a minimização.

Tabela 5. Redução da Eficácia nos Conjuntos Minimizados (%).

Experimentos	Porcentagem de Redução da Eficácia após Minimização					
	MUT-A	MUT-B	MUT-C	MUT-D	MUT-E	MUT-F
FIND	0.0	0.0	0.0	0.0	0.0	0.0
POS1	10.0	10.0	0.0	0.0	0.0	10.0
POS2	20.0	20.0	0.0	0.0	0.0	0.0
SORT1	14.3	11.1	0.0	0.0	0.0	0.0
SORT2	12.5	14.3	30.0	0.0	0.0	0.0
STAT1	0.0	0.0	0.0	40.0	0.0	10.0
STAT2	0.0	0.0	10.0	0.0	0.0	30.0
STRM1	80.0	0.0	0.0	0.0	0.0	0.0
STRM2	88.9	87.5	0.0	25.0	0.0	0.0
STRM3	66.7	66.7	0.0	60.0	0.0	0.0
Média	29.2	20.9	4.0	12.5	0.0	5.0
Desvio Padrão	35.060	30.803	9.661	21.763	0.0	9.719

5. Conclusões

Neste trabalho foram apresentadas a especificação e os aspectos de implementação do módulo de minimização de casos de teste – *Minimize* – desenvolvido para a ferramenta *Proteum*. Utilizou-se a estratégia de minimização proposta por Harrold a qual tem a vantagem de ser aplicada para qualquer critério de teste que contenha associação entre os requisitos de teste e os casos de teste. A minimização foi aplicada para um conjunto de programas e os resultados demonstram que a estratégia viabiliza a diminuição de custos relacionados ao número de casos de teste, obtendo-se uma redução significativa no tamanho do conjunto de casos de teste, principalmente para os conjuntos com maior cardinalidade.

Com relação à eficácia dos conjuntos minimizados, os resultados obtidos a partir do experimento, demonstraram, em alguns casos, uma pequena redução na eficácia. A estratégia de minimização restrita permite melhorar esse resultado. Tendo-se conhecimento de quais casos de teste revelam os erros existentes, pode-se mantê-los no conjunto final, garantido, desta forma, a eficácia já existente para esses conjuntos.

É importante salientar que os resultados registrados aqui foram obtidos a partir de programas bastante simples e pequenos. É importante a condução de outros estudos com programas mais complexos o que é alvo desta linha de pesquisa, considerando-se a relevância desses estudos do ponto de vista prático. É também necessário analisar estratégias alternativas para obtenção do conjunto mínimo, como por exemplo: preservar casos de teste específicos, aplicar a minimização por classes de operadores e aplicar a minimização incrementalmente de acordo com a estratégia de teste, dentre outras.

Com a minimização de conjuntos de casos de teste obtém-se uma melhor estimativa e uma redução do custo associado à utilização do critério visto que, na maioria das vezes, o custo é estimado pelo tamanho do conjunto de casos de testes necessário para satisfazer um critério. Além disso, os testes de regressão, realizados após a manutenção do software, também terão seus custos diminuídos pois utilizam o conjunto de casos de teste obtido durante a fase de desenvolvimento como base para a revalidação do software.

Referências Bibliográficas

[CHU87] Chusho, T., "Test Data Selection and Quality Estimation Based on Concept of Essential Branches for Path Testing", *IEEE Trans. on Software Eng.*, v. 13, n. 5, Maio 1987.

[DEL93] Delamaro, M.E., "Proteum - Um Ambiente de Teste Baseado na Análise de Mutantes", *Dissertação de Mestrado*, ICMSC/USP - São Carlos, SP, Brasil, Outubro, 1993.

[DEL96] Delamaro, M.E., Maldonado, J.C., Mathur, A.P., "Proteum - A Tool for the Assesment of Test Adequacy for C Programs: User's Guide", Technical Report SERC-TR-168-P, Software Engineering Research Center, Purdue University, W.Lafayette, IN, Abril, 1996.

[DEM80] Demillo, R.A., "Mutation Analysis as a Tool for Software Quality Assurance", in *Proc. of COMPSAC80*, Chicago - IL, Outubro, 1980.

[GAR79] Garey, M.R., Johnson, D.S., *Computers and Intractability, A Guide to the Teory of NP-Completeness*, V.Klee, Ed. Freeman, New York, 1979.

[HAR93] Harrold, M.J., Soffa, M.L., Gupta, R., "A Methodology for Controlling the Size of a Test Suite", *ACM Transaction on Software Engineering and Methodology*, v. 2, n.3, pp. 270-285, Julho, 1993.

[LEU91] Leung, H.K.L., White, L., "A Cost Model to Compare Regression Test Strategies", in *Proceedings Conference on Software Maintenance*, pp. 201-208, Outubro, 1991.

[MAT93] Mathur, A.P., Wong, W.E., "Evaluation of The Cost Alternate Mutation Strategies", *VII Simpósio Brasileiro de Engenharia de Software*, Rio de Janeiro, 1993.

[MAT94] Mathur, A.P., Wong, W.E., "An Empirical Comparison of Data Flow and Mutation-Based Test Adequacy Criteria", *Software Testing, Verification and Reliability*, vol.4, pp. 9-31, 1994.

[OFF96a] Offut, A.J., Pan, J., Tewary, K., Zhang, T., "An Experimental Evaluation of Data Flow and Mutation Testing", *Software Pratice and Experience*, 26(2), pp.165-176, Fevereiro, 1996.

[OFF96b] Offut, A.J., Lee, A., Rothermel, G., Untch, R.H., Zapf, C., "An Experimental Determination of Sufficient Mutant Operators", *ACM Transaction on Software Engineering Methodology*, 5(2), pp. 99-118, Abril, 1996.

[OST88] Ostrand, T.J., Weyuker, E.J., "Using Data Flow Analysis for Regression Testing", in *Sixth Annual Pacific Northwest Software Quality Conference*, Ortland-Oregon, Setembro, 1988.

[SOU97] Souza, S.R.S., Maldonado, J.C., Vergílio, S.R., "Análise de Mutantes e Potenciais-Usos: Uma Avaliação Empírica", *VIII CITS - Conferência Internacional de Tecnologia de Software*, pp. 225-235, Curitiba, Junho, 1997.

[WON93] Wong, W.E., "On Mutation and Data Flow", *Tese de Doutorado*, Software Engineering Research Center - Purdue University, West Lafayette, Indiana, Dezembro, 1993.

[WON94a] Wong, W.E., London, S., Mathur, A., "Effect of Test Set Minimization on the Fault Detection Effectiveness of the All-Uses Criterion", *Technical Report SERC-TR-152-P*, Software Eng. Research Center, Purdue University, Abril, 1994.

[WON94b] Wong, W.E., *et al.*, "Mutation versus All-uses: An Empirical Evaluation of Cost, Strength, and Effectiveness", *Software Quality and Productivity - Theory, practice, education and training*, Hong Kong, Dezembro, 1994.

[WON95] Wong, W.E., Mathur, A., "Reducing the Cost of Mutation Testing: An Empirical Study", *Journal of Systems Software*, v.31, pp. 185-196, 1995.

[WON97] Wong, W.E., Maldonado, J.C., Delamaro, M.E., "Reducing the Cost of Regression Testing by Using Selective Mutation", *VIII CITS - Conferência Internacional de Tecnologia de Software*, pp. 93-109, Curitiba, Junho, 1997.