Uma Contribuição para a Determinação de um Conjunto Essencial de Operadores de Mutação no Teste de Programas C

Ellen Francine Barbosa
Auri Marcelo Rizzo Vincenzi
José Carlos Maldonado

Universidade de São Paulo – USP
Instituto de Ciências Matemáticas e de Computação – ICMC
{francine, auri, jcmaldon}@icmc.sc.usp.br
São Carlos, SP, Brasil

Resumo

Estudos empíricos têm mostrado que a Análise de Mutantes – um dos critérios baseados em erros – é bastante eficaz em revelar a presença de erros. Entretanto, seu alto custo de aplicação, decorrente principalmente do grande número de mutantes gerados, tem motivado a proposição de diversos critérios alternativos para a sua aplicação. Um deles visa a redução do número de mutantes gerados através da diminuição dos operadores de mutação utilizados. Um estudo relevante nesse sentido resultou na determinação de um conjunto essencial de operadores de mutação para a linguagem Fortran, mostrando-se que é possível reduzir o custo de aplicação do critério, preservando um alto escore de mutação em relação à Análise de Mutantes. Este trabalho tem como objetivo investigar alternativas pragmáticas para a aplicação do critério Análise de Mutantes e, nesse contexto, é proposto um procedimento para a determinação de um conjunto essencial de operadores de mutação para a linguagem C.

Palavras-chave: teste de software, análise de mutantes, conjunto essencial de operadores de mutação, ferramentas de teste.

Abstract

Mutation Analysis – one of the error based criteria – has been found to be effective on revealing faults. However, its high cost of application, due to the high number of mutants created, has motivated the proposition of many alternative criteria for its application. One of them aims to reduce the number of generated mutants through a reduction on the number of mutant operators. In this perspective, a relevant study resulted on the determination of an essential mutant operator set for Fortran, indicating that it is possible to have a large cost reduction of mutation testing, preserving a high Mutation Analysis score. This work aims to investigate pragmatic alternatives for mutation analysis application and, in this context, a procedure for the determination of an essential mutant operators set for C is proposed.

Keywords: software testing, mutation analysis, essential mutant operators set, testing tools.

1. Introdução

Atividades agregadas sob o nome de Garantia de Qualidade de Software têm sido introduzidas ao longo de todo o processo de desenvolvimento de software, entre elas atividades de VV&T – Verificação, Validação e Teste, com o objetivo de minimizar a ocorrência de erros[1] e riscos associados ao desenvolvimento de produtos de software. A **atividade de teste** consiste em uma análise dinâmica do produto, sendo relevante para a identificação e eliminação de erros que persistem, constituindo um dos elementos para fornecer evidências da confiabilidade do software, em complemento a outras atividades de VV&T. O principal objetivo do teste de software é revelar a presença de erros no produto. Portanto, o teste bem sucedido é aquele que consegue determinar casos de teste para os quais o programa em teste falhe.

Do ponto de vista de qualidade do processo, o teste sistemático é uma atividade fundamental para a ascensão ao Nível 3 do Modelo CMM do SEI [PAU93]. Ainda, o conjunto de informação oriundo da atividade de teste é significativo para as atividades de depuração, manutenção e estimativa de confiabilidade de software.

Salienta-se, entretanto, que a atividade de teste tem sido apontada entre as mais onerosas no desenvolvimento de software. Na tentativa de reduzir os custos associados ao teste, faz-se necessária a aplicação de técnicas e critérios que dêem indicações de como testar o software, quando parar o teste e que, se possível, forneçam uma medida objetiva do nível de confiança e de qualidade alcançados com os testes realizados.

As técnicas mais utilizadas para se conduzir a atividade de teste são as técnicas de teste funcional, estrutural e baseada em erros. Essas técnicas diferenciam-se, basicamente, pela origem da informação utilizada para avaliar ou construir conjuntos de casos de teste [MAL91]. Em geral, nenhuma delas é suficiente para garantir a qualidade da atividade de teste, devendo ser aplicadas em conjunto para se tentar assegurar um teste de qualidade.

Considerando a diversidade de técnicas e critérios de teste existentes bem como o seu aspecto complementar, vários estudos teóricos e empíricos têm sido conduzidos com o objetivo de estabelecer uma estratégia de teste eficaz e de baixo custo, que combine as vantagens dessas técnicas e critérios [BUD80, WEY90, MAT93, OFF93, WON93, WON94, WON95b, OFF96, SOU96, WON97a]. Tanto a abordagem teórica quanto a empírica buscam comparar a adequação de um critério de teste através de três fatores básicos: **custo** – esforço necessário para que o critério seja utilizado –, **eficácia** – capacidade de um critério em revelar a presença de um maior número de erros em relação a outro – e **dificuldade de satisfação** ou ***strength*** – probabilidade de satisfazer-se um critério tendo satisfeito outro. Do ponto de vista teórico, procuram-se estabelecer propriedades e características dos critérios de teste, tais como sua complexidade (número máximo de casos de teste requeridos no pior caso) ou uma relação de hierarquia entre os mesmos. Do ponto de vista empírico, dados e estatísticas são coletados, registrando, por exemplo, o custo e a freqüência com que diferentes estratégias de teste revelam a presença de erros em um determinado conjunto de programas.

Estudos empíricos têm mostrado que a Análise de Mutantes [DEM78] é um critério bastante eficaz em revelar a presença de erros [BUD80, MAT93, WON93, SOU96, WON97a]. Em

---

[1] O padrão IEEE 610.12-1990 [I390] diferencia os termos: defeito (*fault*) – passo, processo ou definição de dados incorreto; engano (*mistake*) – ação humana que produz um resultado incorreto; erro (*error*) – diferença entre o valor obtido e o valor esperado; e falha (*failure*) – produção de uma saída incorreta com relação à especificação. Neste texto, os termos engano, defeito e erro serão referenciados como erro (causa) e o termo falha (conseqüência) a um comportamento incorreto do programa.

relação aos critérios baseados em Análise de Fluxo de Dados (técnica estrutural) [HER76, RAP85, MAL91], considerados rigorosos e eficazes, pode-se dizer, de um modo geral, que a Análise de Mutantes, embora necessite de mais casos de teste, revela a presença de um maior número de erros [WON95b, SOU96]. Entretanto, seu alto custo tem motivado a proposição de diversas alternativas para a sua aplicação [ACR79, MAT91, OFF93, OFF96]. Uma dessas alternativas procura reduzir o custo de aplicação do critério através da determinação de um conjunto essencial de operadores de mutação [OFF96].

Este trabalho tem como objetivo investigar alternativas pragmáticas para a aplicação do critério Análise de Mutantes e, nesse contexto, é proposto um procedimento para a determinação de um conjunto essencial de operadores de mutação para a linguagem C, com base nos operadores disponíveis na ferramenta *Proteum* [DEL93].

O artigo está organizado da seguinte forma: na Seção 2 são apresentados os conceitos do critério Análise de Mutantes e abordagens alternativas com menor custo de aplicação. A Seção 3 discute alguns pontos relevantes a serem considerados durante a caracterização de um conjunto essencial de operadores de mutação. O procedimento para a determinação de um conjunto essencial de operadores é descrito na Seção 4. A aplicação do procedimento a fim de determinar o conjunto essencial de operadores para linguagem C é apresentada na Seção 5. Na Seção 6 o conjunto essencial obtido é comparado a outras abordagens e critérios. A Seção 7 sintetiza os principais resultados e desdobramentos deste trabalho.

2. O Critério Análise de Mutantes

Um dos primeiros artigos que descrevem a idéia de teste de mutantes foi publicado em 1978 [DEM78]. A idéia básica da técnica apresentada por DeMillo, conhecida como hipótese do programador competente (*competent programmer hypothesis*), assume que programadores experientes escrevem programas corretos ou muito próximos do correto. Assumindo a validade desta hipótese, pode-se afirmar que erros são introduzidos no programa através de pequenos desvios sintáticos que alteram sua semântica e, conseqüentemente, o programa pode apresentar um comportamento incorreto. O testador deve construir casos de testes que mostrem que tais transformações levam a um programa incorreto [AGR89].

Outra hipótese explorada na aplicação do critério Análise de Mutantes é o efeito de acoplamento (*coupling effect*) [DEM78], a qual assume que erros complexos estão relacionados a erros simples. Assim sendo, espera-se, e estudos empíricos já confirmaram essa hipótese [ACR79, BUD80], que conjuntos de casos de teste capazes de revelar a presença de erros simples são também capazes de revelar a presença de erros complexos.

Partindo-se da hipótese do programador competente e do efeito de acoplamento, a princípio, o testador deve fornecer um programa P a ser testado e um conjunto de casos de teste T cuja adequação deseja-se avaliar. O programa P é executado com T e se apresentar resultados incorretos então um erro foi revelado e o teste termina. Caso contrário, P ainda pode conter erros que o conjunto T não conseguiu revelar. O programa P sofre então pequenas alterações, dando origem aos programas P_1, P_2, ..., P_n, denominados **mutantes** de P, diferindo de P apenas pela ocorrência de erros simples, ou seja, aplica-se uma mutação (transformação sintática) de cada vez no programa P em teste. As transformações sintáticas aplicadas em P são definidas através de **operadores de mutação** (*mutant operators*). Os operadores de mutação são construídos para satisfazer a um entre dois propósitos: 1) induzir mudanças sintáticas simples com base nos erros típicos cometidos pelos programadores (como trocar o nome de uma variável); ou 2) forçar determinados objetivos de teste (como executar cada arco do programa) [OFF96].

Gerados os mutantes, estes são executados com o mesmo conjunto de casos de teste *T*. O objetivo é obter um conjunto de casos de teste *T* que resulte apenas em mutantes mortos (para algum caso de teste o resultado do mutante e do programa original diferem entre si) e equivalentes (o mutante e o programa original apresentam sempre o mesmo resultado, para qualquer item de dado pertencente ao domínio de entrada); neste caso, *T* é adequado ao teste de *P*, no sentido de que, ou *P* está correto ou possui erros pouco prováveis de ocorrerem [DEM78]. É preciso ressaltar que, em geral, a equivalência entre programas é uma questão indecidível e requer a intervenção do testador; alguns métodos e heurísticas foram propostos para determinar a equivalência entre programas em uma grande parte dos casos de interesse [BUD81].

Um ponto importante a ser destacado é que a Análise de Mutantes fornece uma medida objetiva do nível de confiança da adequação dos casos de teste analisados através da definição de um **escore de mutação** (*mutation score*), o qual relaciona o número de mutantes mortos com o número de mutantes não equivalentes gerados. Assim, através do escore de mutação, é possível quantificar e avaliar a qualidade da atividade de teste.

Um ponto fundamental no uso de critérios de teste é a utilização de ferramentas que automatizem sua aplicação. No experimento descrito neste artigo utiliza-se a ferramenta *Proteum* (*Program Testing Using Mutants*) [DEL93], desenvolvida no Instituto de Ciências Matemáticas e de Computação – ICMC/USP. A *Proteum* conta com 71 operadores de mutação divididos em quatro classes: mutação de comandos (*statement mutations*), mutação de operadores (*operator mutations*), mutação de variáveis (*variable mutations*) e mutação de constantes (*constant mutations*). A ferramenta também disponibiliza ao testador recursos para selecionar os operadores a serem utilizados, bem como a porcentagem de aplicação de cada um deles, o que viabiliza o estudo e avaliação de critérios alternativos da Análise de Mutantes [WON94, SOU96, WON97a].

2.1. Problemas e Alternativas para a Aplicação do Critério Análise de Mutantes

Várias alternativas têm sido propostas com o intuito de contornar os aspectos de custo na aplicação da Análise de Mutantes, entre elas: Mutação Aleatória (*Randomly Selected X% Mutation*), Mutação Restrita (*Constrained Mutation*) e Mutação Seletiva (*Selective Mutation*).

Na Mutação Aleatória, embora sejam utilizados todos os operadores de mutação, apenas uma porcentagem de cada um deles (*x%*) é aplicada aleatoriamente, limitando a análise aos mutantes gerados a partir dessa porcentagem [ACR79]. Segundo DeMillo *et al.* [DEM88], em geral, mesmo com uma pequena amostragem do total de mutantes é possível construir bons casos de teste, aspecto este investigado em vários experimentos [MAT93, WON97a].

Na Mutação Restrita é selecionado um subconjunto de operadores de mutação para ser utilizado na geração dos mutantes [MAT91]. O objetivo é reduzir sensivelmente o número de mutantes gerados sem diminuir de maneira significativa a capacidade em revelar erros do critério [WON95a].

Offutt *et al.* propuseram a Mutação Seletiva (*Selective Mutation*), na qual os operadores de mutação responsáveis pelo maior número de mutantes não são aplicados [OFF93]. Desse modo, a Mutação *N*-Seletiva é aquela que deixa de gerar mutantes para *N* operadores de mutação, exatamente aqueles que geram mais mutantes.

Estudos empíricos têm sido conduzidos a fim de verificar o efeito da aplicação dessas alternativas no teste de programas [MAT93, WON94, WON95a, SOU96]; os resultados

demonstram que é possível reduzir o custo da Análise de Mutantes sem uma redução significativa da eficácia.

Concretamente, o que se almeja é a determinação de um conjunto de mutações de forma que, conseguindo-se um conjunto de casos de teste T capaz de distinguir essas mutações, T seja capaz de distinguir todos os mutantes não equivalentes gerados por todos os operadores.

Nessa perspectiva, a partir dos operadores de mutação da ferramenta *Mothra* [DEM88], que apóia a aplicação do critério Análise de Mutantes para programas escritos em Fortran, Offutt *et al.* determinaram um conjunto essencial de operadores para essa linguagem [OFF96]. A ferramenta *Mothra* possui 22 operadores de mutação divididos em três categorias: operadores de troca de operandos (*replacement-of-operand*), operadores de modificação de expressões (*expression modification*) e operadores de modificação de comandos (*statement modification*). Com base nessas categorias, Offutt *et al.* definiram quatro classes de mutação seletiva: *ES-selective*, *RS-selective*, *RE-selective* e *E-selective*, que utilizam operadores *expression/statement*, *replacement/statement*, *replacement/expression* e *expression*, respectivamente, na criação dos mutantes.

Os resultados obtidos indicaram que a utilização de apenas cinco dos 22 operadores de mutação da ferramenta *Mothra* (exatamente os pertencentes à classe *E-selective*) proporcionou uma redução significativa de custo em termos do número de mutantes gerados (77.56%), preservando um alto índice de cobertura (escores de mutação acima de 98%) em relação à Análise de Mutantes. A Tabela 1 apresenta a descrição desses operadores.

Além disso, um estudo preliminar realizado por Wong *et al.* [WON97a], comparando a Mutação Restrita no contexto das linguagens C e Fortran, forneceu indícios de que os operadores da ferramenta *Proteum* utilizados naquele experimento possuem uma forte relação com os operadores essenciais obtidos por Offutt *et al.* [OFF96] para a linguagem Fortran. Os operadores obtidos por Wong *et al.* foram selecionados com base na experiência dos autores e os resultados motivaram a condução deste trabalho.

Tabela 1. Operadores Essenciais da Linguagem Fortran [OFF96]

Operador	Descrição
ABS	Força cada expressão aritmética a assumir um valor zero, um valor positivo e um valor negativo.
AOR	Substitui cada operador aritmético por todos os outros operadores sintaticamente permitidos.
LCR	Substitui cada conectivo lógico (AND e OR) por outros tipos de conectivos lógicos.
ROR	Substitui operadores relacionais por outros operadores relacionais.
UOI	Insere operadores unários na frente das expressões.

3. Considerações para a Determinação de um Conjunto Essencial de Operadores

Considere CM_1, CM_2, ..., CM_n conjuntos que representam classes de operadores de mutação. O critério Análise de Mutantes (AM) utiliza, na sua concepção original, o conjunto formado por todos os operadores de mutação definido por $OP = CM_1 \cup CM_2 \cup ... \cup CM_n$. Qualquer subconjunto de operadores $SC \in 2^{|OP|}$ constitui um critério de mutação restrito.

Dado um critério de mutação restrito SC, diz-se que um conjunto de casos de teste T é SC-adequado se T obtiver um escore de mutação igual a 1 em relação aos mutantes gerados pelos operadores de SC; ou seja, se T for capaz de revelar as diferenças de comportamento existentes entre P (programa em teste) e os mutantes não equivalentes gerados pelos operadores de SC. Quando SC for composto por um único operador de mutação op ($SC = \{op\}$), em alguns casos, por simplicidade de notação, será utilizado op.

Na prática, por limitações de tempo e custo, obter-se um escore de mutação próximo de 1 pode ser satisfatório. Seja *ms* um escore de mutação definido pelo testador. Dados um critério C e um conjunto de casos de teste T, diz-se que T é empiricamente adequado a C (T é C-adequado*) se T obtiver um escore de mutação igual ou superior a *ms*.

Uma característica importante utilizada na comparação de critérios de teste é a relação de inclusão, definida por Rapps e Weyuker [RAP85]. Dados dois critérios C_1 e C_2, diz-se que C_1 inclui C_2 ($C_1 \Rightarrow C_2$) se para todo conjunto de casos de teste T C_1-adequado, T é C_2-adequado e existe um T_2 C_2-adequado que não é C_1-adequado; C_1 e C_2 são equivalentes se para qualquer T C_1-adequado, T é C_2-adequado e vice-versa.

Além disso, diz-se que C_1 inclui empiricamente C_2 com um determinado escore *ms* ($C_1 \overset{ms}{\Rightarrow}{}^{*} C_2$) se para todo conjunto de casos de teste T C_1-adequado, T é C_2-adequado* para um dado *ms* e existe um T_2 C_2-adequado que não é C_1-adequado*; C_1 e C_2 são equivalentes* se para qualquer T C_1-adequado, T é C_2-adequado* e vice-versa.

Desse modo, determinar um **conjunto essencial de operadores de mutação** consiste basicamente em selecionar um subconjunto $CE \in 2^{(OP)}$ a partir do conjunto de operadores de mutação OP definido para a linguagem alvo, de forma que se um conjunto de casos de teste T for CE-adequado, T seja também AM-adequado*. Em outras palavras, se T obtiver um escore de mutação igual a 1 em relação ao conjunto essencial de operadores (CE), T obterá um escore de mutação maior ou igual a *ms* em relação à Análise de Mutantes (OP). Dados dois critérios de teste C_1 e C_2, diz que C_1 determina um alto escore de mutação em relação a C_2 se todo conjunto T C_1-adequado obtiver um alto escore de mutação em relação a C_2, ou seja, se T for capaz de distinguir a maioria dos mutantes gerados pelos operadores de C_2.

Para a determinação dos operadores de mutação que irão compor o conjunto essencial algumas diretrizes devem ser consideradas:

i) **Selecionar operadores *op* cujos casos de teste *op*-adequados determinem alto escore de mutação em relação à Análise de Mutantes**: A fim de garantir que o conjunto essencial de operadores determine um alto escore de mutação em relação à Análise de Mutantes, devem ser selecionados, a princípio, os operadores *op* tal que conjuntos de casos de teste *op*-adequados sejam capazes de distinguir a maioria dos mutantes gerados pelo conjunto total de operadores.

ii) **Procurar selecionar um operador de cada classe de mutação**: Cada classe de mutação modela erros específicos em certos elementos do programa (por exemplo: comandos, operadores, variáveis e constantes). Assim, é desejável que o conjunto essencial contenha ao menos um operador representativo de cada classe.

iii) **Avaliar inclusão empírica entre operadores**: Deve-se remover os operadores que são incluídos empiricamente por outros operadores do conjunto essencial, visto que tais operadores elevam o custo de aplicação do conjunto, em termos do número de mutantes gerados, e não contribuem efetivamente para a melhoria da atividade de teste.

iv) **Estabelecer uma estratégia incremental de aplicação**: Dado o custo de aplicação e os requisitos de teste específicos que cada classe determina, é interessante que seja estabelecida uma estratégia incremental de aplicação entre os operadores do conjunto essencial. A idéia é aplicar, primeiramente, os operadores relevantes a determinados requisitos mínimos de teste (por exemplo, cobertura de todos os nós e todos os arcos) e, em seguida, em função do grau de adequação que se quer atingir e da disponibilidade de

orçamento e tempo, aplicar os demais operadores relacionados a outros conceitos e requisitos.

v) **Selecionar operadores que proporcionem incremento no escore de mutação**: Sabe-se que, em geral, independentemente da qualidade dos conjuntos de casos de teste utilizados, 80% dos mutantes gerados morrem na primeira execução [BUD80]. Considerando então que somente em torno de 20% dos mutantes gerados contribuem efetivamente para a melhoria do conjunto de casos de teste, um incremento de 1% no escore de mutação representa 5% do número de mutantes significativos. Além disso, são esses mutantes que contribuem efetivamente para a eficácia do critério. Desse modo, deve-se investigar entre os operadores não selecionados se existe um ou mais operadores que, se adicionados ao conjunto essencial, proporcionem incremento no escore de mutação.

vi) **Selecionar operadores de alto *strength***: Outros operadores a serem considerados durante a determinação do conjunto essencial são os que apresentam alto *strength* em relação ao conjunto total de operadores. A principal característica dos operadores de alto *strength* é que, em média, eles são pouco incluídos pelos demais, fator que pode ser importante no que diz respeito à eficácia do conjunto essencial.

Deve-se observar que, dado um *ms*, não existe um conjunto essencial de operadores único. Pelo contrário, pode-se determinar diferentes conjuntos tal que o escore de mutação obtido esteja próximo ao especificado pelo testador. No entanto, o custo computacional para a determinação de todos os possíveis conjuntos essenciais candidatos pode ser muito alto, o que torna fundamental a definição de abordagens pragmáticas e de baixo custo que levem à determinação de tal conjunto. Além disso, o conjunto essencial pode ser dependente do domínio de aplicação e do conjunto de programas utilizados, ou seja, as características específicas de cada programa (ou conjunto de programas) influenciam na seleção dos operadores que irão compor o conjunto; a determinação do conjunto essencial para um certo domínio de aplicação pode ser refinada à medida que se acumulem dados históricos de teste no domínio específico. O ideal seria estabelecer, com base nessas informações, um conjunto essencial de operadores independente do domínio de aplicação, o qual seria refinado em função dos objetivos da atividade de teste.

4. Definição de um Procedimento para a Determinação de um Conjunto Essencial de Operadores de Mutação

Nesta seção é apresentado um procedimento para a determinação de um conjunto essencial de operadores de mutação, composto de seis passos, definido com base nas diretrizes da Seção 3 e refinado pela condução do próprio experimento, descrita na Seção 5. Deve-se observar que esses passos podem ser aplicados modular e incrementalmente, de acordo com os objetivos do teste. Para a definição do procedimento foi utilizada a terminologia apresentada na Tabela 2.

Tabela 2. Terminologia Utilizada no Procedimento

Termo	Descrição/Definição
$\overset{n}{\underset{i=1}{CM_i}}$	classes de operadores de mutação (para a ferramenta *Proteum*, n = 4 classes: comandos, operadores, variáveis e constantes)
OP	conjunto de todos os operadores de mutação $OP = CM_1 \cup CM_2 \cup ... \cup CM_n$
CE	conjunto essencial de operadores de mutação ($CE \subseteq OP$)
CE_{pre}	conjunto essencial preliminar de operadores de mutação ($CE_{pre} \subseteq OP$)
$\overline{CE}_{pre}$	complemento de CE_{pre} ($\overline{CE}_{pre} = OP - CE_{pre}$)

Continuação da Tabela 2

Termo	Descrição/Definição
CandElim	conjunto de operadores de CE_{pre} candidatos a serem eliminados em virtude da relação de inclusão empírica ($CandElim \subseteq CE_{pre}$)
CandIns	conjunto de operadores de $\overline{CE}_{pre}$, pertencentes a determinada classe CM_i, candidatos a serem inseridos em CE_{pre} ($CandIns \subseteq (\overline{CE}_{pre} \cap CM_i)$)
CAS	conjunto de operadores de alto *strength* ($CAS \subseteq \overline{CE}_{pre}$)
x	número de operadores de cada classe CM_i que se deseja incluir em CE_{pre} (definido pelo testador)
ms	escore de mutação (definido pelo testador)
IMES	índice médio de escore (definido pelo testador)
IIM	índice de incremento mínimo (definido pelo testador)
IMS	índice médio de *strength* (definido pelo testador)
$f_{MS}(C_1, C_2)$	função que retorna o escore de mutação determinado por C_1 em relação a C_2
$f_{MSM}(op)$	função que retorna a média dos escores que *op* determina em relação a cada operador de *OP* $f_{MSM}(op) = ((\sum_{j=1}^{\|OP\|} f_{MS}(\{op\}, \{op_j\}) \mid op_j \in OP) / \|OP\|)$
$f_{STR}(C_1, C_2)$	função que retorna o *strengths* de C_1 em relação a C_2 $f_{STR}(C_1, C_2) = 1 - f_{MS}(C_2, C_1)$
$f_{STRM}(op)$	função que retorna a média dos *strength* de *op* em relação a cada operador de *OP* $f_{STRM}(op) = 1 - ((\sum_{j=1}^{\|OP\|} f_{MS}(\{op_j\}, \{op\}) \mid op_j \in OP) / \|OP\|)$
$f_{INCR}(op, C_1)$	função que retorna o índice de incremento que *op* proporciona em relação a Análise de Mutantes se for adicionado a C_1 (este índice é truncado após o primeiro dígito significativo) $f_{INCR}(op, C_1) = f_{MS}(C_1 \cup \{op\}, OP) - f_{MS}(C_1, OP)$

Passo 1: Selecionar operadores *op* cujos casos de teste *op*-adequados determinem alto escore de mutação em relação à Análise de Mutantes

De acordo com a Diretriz *i* da Seção 3, neste passo procura-se selecionar os operadores que determinam alto escore de mutação. Uma informação relevante nesse sentido é o escore que cada operador de mutação determina em relação ao conjunto total de operadores. A seleção dos operadores que determinam alto escore é feita a partir de um índice médio de escore (*IMES*), que pode ser dependente do conjunto de programas utilizados e, em geral, deve ser definido em função das características do domínio de aplicação. Desse modo, CE_{pre} é composto pelos operadores que, em média, determinem um escore de mutação em relação à Análise de Mutantes igual ou superior a *IMES*.

$$CE_{pre} \leftarrow \{op_j \in OP \mid f_{MSM}(op_j) \geq IMES\}$$

Passo 2: Procurar selecionar um operador de cada classe de mutação

O conjunto CE_{pre} obtido com a aplicação do Passo 1, embora seja formado por operadores que determinem alto escore, pode não conter operador de alguma classe de mutação. Conforme a Diretriz *ii* da Seção 3, neste passo procura-se garantir que ao menos um operador de cada classe de mutação esteja presente em CE_{pre}. Assim, para cada classe de mutação CM_i que não estiver representada em CE_{pre}, procura-se selecionar o operador $op \in CM_i$ que determine o maior escore e não seja incluído empiricamente pelos operadores de CE_{pre}.

para *i* de 1 até *n* faça
  se $CM_i \cap CE_{pre} = \emptyset$ então
    se $\exists\, op_j \in CM_i \mid CE_{pre} \overset{mr}{\Rightarrow}^* \{op_j\} \wedge f_{MSM}(op_j) \geq f_{MSM}(op_k), \forall\, op_k \in \{op \in CM_i \mid CE_{pre} \overset{mr}{\Rightarrow}^* \{op\}\}$ então
      $CE_{pre} \leftarrow CE_{pre} \cup \{op_j\}$
fim para

Passo 3: Avaliar inclusão empírica entre operadores

Com base na Diretriz *iii* da Seção 3, neste passo é realizada uma análise com respeito à relação de inclusão empírica entre os operadores de CE_{pre} e aqueles que forem incluídos empiricamente são selecionados para compor o conjunto de operadores candidatos a serem eliminados de CE_{pre} (*CandElim*). A partir de *CandElim* elimina-se o operador *op* que for mais incluído empiricamente pelos demais operadores de CE_{pre}, mesmo que este seja o único operador representativo de uma determinada classe de mutação CM_i.

Sempre que um operador é eliminado o conjunto *CandElim* deve ser gerado novamente. Este processo é repetido enquanto existirem operadores em CE_{pre} que sejam incluídos empiricamente pelos demais operadores de CE_{pre}.

$CandElim \leftarrow \{\forall\ op_j \in CE_{pre} \mid (CE_{pre} - \{op_j\}) \stackrel{ins}{\Rightarrow} * \{op_j\}\}$
enquanto $CandElim \neq \emptyset$ faça
  $CE_{pre} \leftarrow CE_{pre} - \{op\} \mid op \in CandElim \wedge f_{MS}(CE_{pre} - \{op\}, \{op\}) > f_{MS}(CE_{pre} - \{op_k\}, \{op_k\}),\ \forall\ op_k \in CandElim$
  $CandElim \leftarrow \{\forall\ op_j \in CE_{pre} \mid (CE_{pre} - \{op_j\}) \stackrel{ins}{\Rightarrow} * \{op_j\}\}$
fim enquanto

Passo 4: Estabelecer uma estratégia incremental de aplicação

O objetivo deste passo é estabelecer uma estratégia incremental de aplicação entre os operadores de CE_{pre}. É importante ressaltar que a ordenação aqui proposta é fundamentada na Diretriz *iv* da Seção 3, podendo ser alterada conforme as características específicas dos programas utilizados, ou mesmo de acordo com os tipos de erros que se deseja enfatizar, com base na eficácia dos operadores. Optou-se em aplicar, inicialmente, os operadores das classes de mutação de comandos e operadores, garantindo-se que certos requisitos mínimos de teste sejam satisfeitos: cobertura de comandos e decisões. A seguir, foram aplicados os operadores pertencentes às classes de mutação de variáveis e constantes, respectivamente. Dentro de uma mesma classe de mutação os operadores foram ordenados segundo seu custo. Definida a ordem em que os operadores de CE_{pre} serão aplicados, esta deve ser obedecida sempre que um operador for inserido no conjunto.

Passo 5: Selecionar operadores que proporcionem incremento no escore de mutação

Tendo em vista a Diretriz *v* da Seção 3, neste passo procura-se adicionar a CE_{pre} *x* operadores de cada classe de mutação, desde que tais operadores proporcionem um incremento no escore igual ou superior a um índice de incremento mínimo (*IIM*) preestabelecido e não sejam incluídos empiricamente pelos operadores de CE_{pre}. Para isso, a partir dos operadores com índice de incremento igual ou superior a *IIM*, são selecionados para fazer parte do conjunto de operadores candidatos a serem inseridos em CE_{pre} (*CandIns*) o operador *op* da classe CM_i que determine o melhor índice de incremento e demais operadores de CM_i, cujos índices de incremento encontrem-se na mesma faixa do índice de *op*. Dos operadores de *CandIns*, adiciona-se a CE_{pre} o operador de maior *strength* em relação a CE_{pre}. A inserção dos operadores em CE_{pre} é feita de forma intercalada, respeitando a ordem de classes estabelecida no Passo 4, até que *x* operadores de cada classe de mutação tenham sido adicionados a CE_{pre} ou quando não houver operadores com índice de incremento igual ou superior a *IIM*.

É importante ressaltar que a inserção de um operador em CE_{pre} faz com que o número de operadores que proporcionem índice de incremento igual ou superior a *IIM* diminua, garantindo, assim, o término deste passo.

$cont\_oper \leftarrow 1$
enquanto $cont\_oper \leq x$ faça
  para i de 1 até n faça
    $CandIns \leftarrow \{op \in \overline{CE}_{pre} \cap CM_i \mid f_{INCR}(op, CE_{pre}) \geq IIM$
      $\wedge (\forall\ op_j \in \overline{CE}_{pre} \cap CM_i, f_{INCR}(op_j, CE_{pre}) \leq f_{INCR}(op, CE_{pre}))\}$
    se $CandIns \neq \emptyset$ então
      $CandIns \leftarrow CandIns \cup \{\forall\ op_k \in \overline{CE}_{pre} \cap CM_i \mid op_k \neq op \wedge f_{INCR}(op, CE_{pre}) - f_{INCR}(op_k, CE_{pre}) = 0\}$
      $CE_{pre} \leftarrow CE_{pre} \cup \{op\} \mid op \in CandIns \wedge CE_{pre} \overset{mr}{\Rightarrow}^* \{op\} \wedge f_{STR}(\{op\}, CE_{pre}) \geq f_{STR}(\{op_j\}, CE_{pre}),$
        $\forall\ op_j \in CandIns$
    fim se
  fim para
  $cont\_oper \leftarrow cont\_oper + 1$
fim enquanto

Passo 6: Selecionar operadores de alto *strength*

Tendo em vista a Diretriz *vi* da Seção 3, este passo determina quais operadores de alto *strength* devem ser inseridos em CE_{pre}. Assim como para os operadores que determinam alto escore, a seleção dos operadores de alto *strength* é feita a partir de um índice médio de *strength* (*IMS*) que pode ser dependente do conjunto de programas utilizados e, em geral, deve ser definido em função das características do domínio de aplicação.

Embora um operador *op* possa apresentar, em média, um alto *strength*, *op* pode ser incluído empiricamente por um ou mais operadores específicos. Se esses operadores específicos já fizerem parte de CE_{pre}, o operador *op* não é considerado de alto *strength* em relação a CE_{pre}.

Os operadores de $\overline{CE}_{pre}$ que apresentarem *strength*, em relação à Análise de Mutantes, igual ou superior a *IMS* e não forem incluídos empiricamente pelos operadores de CE_{pre} são selecionados para fazer parte do conjunto de operadores de alto *strength* (*CAS*). Dos operadores de *CAS*, adiciona-se a CE_{pre} o operador de maior *strength* em relação a CE_{pre}.

Sempre que um novo operador é adicionado a CE_{pre}, o conjunto *CAS* deve ser gerado novamente. Este passo termina quando todos os operadores de alto *strength* que não são incluídos empiricamente por CE_{pre} tenham sido adicionados ao conjunto.

$CAS \leftarrow \{\forall\ op_j \in \overline{CE}_{pre} \mid f_{STRM}(op_j) \geq IMS \wedge CE_{pre} \overset{mr}{\Rightarrow}^* \{op_j\}\}$
enquanto $CAS \neq \emptyset$ faça
  $CE_{pre} \leftarrow CE_{pre} \cup \{op\} \mid op \in CAS \wedge f_{STR}(\{op\}, CE_{pre}) \geq f_{STR}(\{op_k\}, CE_{pre}), \forall\ op_k \in CAS$
  $CAS \leftarrow \{\forall\ op_j \in \overline{CE}_{pre} \mid f_{STRM}(op_j) \geq IMS \wedge CE_{pre} \overset{mr}{\Rightarrow}^* \{op_j\}\}$
fim enquanto
$CE \leftarrow CE_{pre}$

5. Descrição do Experimento

Na seção anterior foi apresentado um procedimento para a determinação de um conjunto essencial de operadores de mutação. Com base nos passos definidos foi conduzido um experimento a fim de determinar um conjunto essencial de operadores para a linguagem C, a partir dos operadores de mutação implementados na ferramenta *Proteum*. A estrutura e as etapas deste experimento são descritas a seguir.

5.1. Seleção dos Programas

Foi selecionado um conjunto de 27 programas de pequeno porte, os quais compõem um editor de texto simplificado. Tais programas foram escritos em Pascal e convertidos para C, fazendo parte de um *benchmark* usado inicialmente por Weyuker [WEY90] e posteriormente por Maldonado [MAL91] e Souza [SOU96].

5.2. Geração dos Conjuntos AM-Adequados

Os conjuntos de casos de teste iniciais (*ad hoc*) foram os mesmos definidos por Maldonado [MAL91] e utilizados em experimentos conduzidos por Souza [SOU96]. A geração de tais conjuntos foi baseada na especificação dos programas; no entanto, nenhum critério funcional específico foi considerado. Verificada a adequação dos conjuntos de teste em relação ao critério Análise de Mutantes, novos casos de teste foram adicionados aos conjuntos até serem obtidos conjuntos AM-adequados. Para cada um dos 27 programas foi gerado um conjunto de casos de teste adequado. Deve-se observar que os conjuntos AM-adequados contém apenas casos de teste efetivos, isto é, que mataram ao menos um mutante. Em um outro experimento, utilizando-se programas Unix [BAR98], foram gerados para cada programa 11 conjuntos AM-adequados, sendo obtidos resultados similares aos deste experimento.

5.3. Aplicação do Benchmark

Obtidos os conjuntos AM-adequados, verificou-se, para cada programa, o escore de mutação de conjuntos adequados a cada operador *op* em relação ao conjunto total de operadores, ou seja, a capacidade de um conjunto de casos de teste *op*-adequado em distinguir os mutantes de cada operador de *OP*. As linhas da Tabela 3 apresentam uma visão parcial de tal informação para o programa *Omatch*. Por exemplo, considerando o operador Ccsr (linha) tem-se que o escore de mutação do conjunto Ccsr-adequado em relação a ORRN (coluna) é de 0.940, indicando que o conjunto Ccsr-adequado é capaz de distinguir 94% dos mutantes de ORRN.

Tabela 3. Escore por Operador para o Programa *Omatch*

Op \ Op	Cccr	Ccsr	CRCR	ORRN	SSDL	SSWM	VTWD	...
Cccr	1.000	0.710	0.730	0.940	0.880	1.000	0.800	...
Ccsr	0.880	1.000	1.000	0.940	0.960	1.000	1.000	...
CRCR	0.880	1.000	1.000	0.940	0.960	1.000	1.000	...
ORRN	0.820	0.690	0.710	1.000	0.840	0.860	0.700	...
SSDL	0.760	0.710	0.720	0.910	1.000	1.000	0.680	...
SSWM	0.590	0.690	0.690	0.780	0.760	1.000	0.610	...
VTWD	0.880	0.920	0.950	0.940	0.960	1.000	1.000	...
...	...	...	...	...	...	...	...	...

Outra informação que pode ser extraída da Tabela 3 diz respeito ao *strength* dos operadores de mutação. Analisando-se uma determinada coluna obtém-se o escore de mutação de conjuntos adequados a cada operador em relação a um determinado operador *op*, ou seja, a capacidade de conjuntos de casos de teste adequados a cada operador de *OP* em distinguir os mutantes de *op*. A partir desta informação é calculado o *strength* de *op*: *1 - escore de mutação em relação a op*. Por exemplo, considerando o operador CRCR (coluna) tem-se que o *strength* deste operador em relação a Ccsr (linha) é 0.000 (1 - 1.000), indicando que o conjunto Ccsr-adequado é também CRCR-adequado; já em relação ao operador ORRN, o *strength* de CRCR é de 0.290 (1 - 0.710), ou seja, o conjunto ORRN-adequado não é capaz de distinguir 29% dos mutantes de CRCR.

Esse processo foi repetido para os todos os programas do experimento; os resultados obtidos estão sintetizados na Tabela 4.

Tabela 4. Escore de cada Operador para os 27 Programas

Op \ Op	Cccr	Ccsr	CRCR	ORRN	SSDL	SSWM	VTWD	...	Média Geral[2]
Cccr	1.000	0.887	0.904	0.921	0.936	0.865	0.923	...	0.922
Ccsr	0.902	1.000	0.992	0.938	0.908	0.835	0.974	...	0.948
CRCR	0.901	0.982	1.000	0.915	0.910	0.835	0.973	...	0.937
ORRN	0.900	0.863	0.870	1.000	0.921	0.465	0.865	...	0.897
SSDL	0.909	0.853	0.881	0.887	1.000	1.000	0.868	...	0.885
SSWM	0.470	0.580	0.500	0.890	0.840	1.000	0.495	...	0.594
VTWD	0.902	0.950	0.969	0.916	0.905	0.800	1.000	...	0.928
...	...	...	...	...	...	...	...	...	...
Média Geral[2]	0.732	0.719	0.730	0.767	0.786	0.544	0.730	...	–

Analisando-se a Tabela 4 tem-se que:

- em geral, o escore de mutação que o operador ORRN determina em relação a Ccsr é de 0.863 e o *strength* de Ccsr em relação a ORRN é de 0.137 (1 - 0.863);
- em média, conjuntos de casos de teste ORRN-adequados determinam um escore de mutação de 0.897 em relação a Análise de Mutantes; e
- o *strength* de um operador *op* em relação a Análise de Mutantes é calculado a partir da média geral da coluna referente a *op*. Em média, tem-se que o *strength* do operador Ccsr é de 0.281 (1 - 0.719).

Os operadores da Tabela 4 foram ordenados segundo a média geral de escore e *strength*, dando origem às tabelas 5(a) e 5(b), respectivamente. Além do escore de mutação e do *strength*, outro aspecto relevante para a determinação do conjunto essencial é o custo associado a cada operador; essa informação também foi coletada e é apresentada Tabela 5(c).

Tabela 5. Operadores Mais Significativos: (a) Escore (b) *Strength* (c) Custo

(a)

Operador	Escore
Ccsr	0.948
Vsrr	0.948
CRCR	0.937
VTWD	0.928
Cccr	0.922
ORRN	0.897
VDTR	0.891
SSDL	0.885
ORSN	0.873
ORAN	0.872
SRSR	0.866
ORBN	0.856

(b)

Operador	*Strength*
SSWM	0.456
SWDD	0.364
OABN	0.353
SMTC	0.339
OLSN	0.338
Vprr	0.323
OASN	0.310
OLLN	0.291
OARN	0.283
Ccsr	0.281
CRCR	0.270
VTWD	0.270

(c)

Operador	Custo
Vsrr	2620
CRCR	1631
Ccsr	1559
VDTR	1437
Cccr	1219
SRSR	1193
OEAA	1010
VTWD	958
ORAN	830
ORRN	830
STRP	677
SSDL	676

5.4. Coleta e Análise dos Resultados

Nesta seção cada passo do procedimento é aplicado ao conjunto de 27 programas e os dados coletados são apresentados. Além disso, ao final de cada passo é apresentado o conjunto

[2] Os valores de Média Geral referem-se a todos os operadores utilizados.

essencial preliminar obtido, culminando na determinação do conjunto essencial de operadores para a linguagem C.

Passo 1

Para a aplicação deste passo utilizou-se *IMES* = 0.900 ± 0.005. Considerando esse índice, observa-se que os seis primeiros operadores da Tabela 5(a) são selecionados para compor CE_{pre}. Ao final do Passo 1, CE_{pre} = {Ccsr, Vsrr, CRCR, VTWD, Cccr, ORRN}.

Passo 2

Observe que o conjunto CE_{pre}, obtido com a aplicação do Passo 1, não apresenta operador da classe de mutação de comandos (*statement mutations*). De acordo com a Tabela 5(a), tem-se que o operador de mutação de comandos que determina o maior escore geral é SSDL; como SSDL não é incluído empiricamente[3] por CE_{pre} ($CE_{pre} \overset{0.99}{\Rightarrow} *\{SSDL\}$), ele é adicionado ao conjunto. Após a aplicação do Passo 2, CE_{pre} = {Ccsr, Vsrr, CRCR, VTWD, Cccr, ORRN, SSDL}.

Passo 3

Para o conjunto CE_{pre} obtido no Passo 2, as seguintes relações de inclusão empírica foram observadas: {Vsrr, CRCR, VTWD, Cccr, ORRN, SSDL} $\overset{0.997}{\Rightarrow} *$ {Ccsr}; {Ccsr, CRCR, VTWD, Cccr, ORRN, SSDL} $\overset{0.998}{\Rightarrow} *$ {Vsrr}; e {Ccsr, Vsrr, VTWD, Cccr, ORRN, SSDL} $\overset{0.998}{\Rightarrow} *$ {CRCR}; desse modo, *CandElim* = {Ccsr, Vsrr, CRCR}. Dos operadores de *CandElim*, CRCR foi o mais incluído e, por esse motivo, foi eliminado de CE_{pre}.

Após a remoção de CRCR, CE_{pre} foi avaliado novamente. Observou-se que o operador Ccsr deixou de ser incluído empiricamente pelos demais operadores de CE_{pre}, o mesmo não ocorrendo para o operador Vsrr ({Ccsr, VTWD, Cccr, ORRN, SSDL} $\overset{0.990}{\Rightarrow} *$ {Vsrr}) que, desse modo, também foi removido de CE_{pre}. Ao final do Passo 3, CE_{pre} = {Ccsr, VTWD, Cccr, ORRN, SSDL}.

Passo 4

De acordo com a classificação estabelecida, os operadores do conjunto CE_{pre}, obtidos no Passo 3, devem ser aplicados na seguinte ordem: SSDL, ORRN, VTWD, Cccr e Ccsr. Considerando somente a classe de mutação de constantes, Cccr (1219 mutantes) vem antes que Ccsr (1559 mutantes) por apresentar menor custo. As informações sobre o custo de cada operador foram retiradas da Tabela 5(c). Após a aplicação do Passo 4, CE_{pre} = {SSDL, ORRN, VTWD, Cccr, Ccsr}.

Passo 5

Para a aplicação deste passo considerou-se x = 1, ou seja, espera-se adicionar a CE_{pre}, no máximo, um operador de cada classe de mutação. Além disso, definiu-se *IIM* = 0.001, o que representa adicionar em CE_{pre} operadores que permitam distinguir pelo menos 0.5% dos mutantes significativos.

A Tabela 6(a) contém informações a respeito do índice de incremento que cada operador de $\overline{CE}_{pre}$ proporciona na primeira iteração deste passo. Observa-se que os nove primeiros

---

[3] Neste passo ms = 0.99, ou seja, considera-se que o critério C_1 inclui empiricamente o critério C_2 se e somente se o escore de mutação que C_1 determina em relação a C_2 for igual ou superior a 0.99.

operadores proporcionam incremento igual ou superior a *IIM*. De acordo com a ordenação proposta no Passo 4, deve-se procurar adicionar, primeiramente, um operador da classe de mutação de comandos. O operador desta classe que proporciona o maior índice de incremento é SMTC (0.003875); dentro desta faixa de incremento (0.003) também encontra-se o operador SMTT (0.003386). Desse modo, o conjunto *CandIns* é formado pelos operadores SMTC e SMTT. Analisando tais operadores em relação ao conjunto CE_{pre}, tem-se que nenhum deles é incluído empiricamente e o *strength* de SMTC e SMTT é de 0.117 e 0.075, respectivamente. Por apresentar o maior *strength*, o operador SMTC é inserido em CE_{pre}. Observe que o incremento de 0.003875 no escore de mutação obtido com a inserção de SMTC em CE_{pre} representa mais de 1.5% dos mutantes significativos.

Tabela 6. Incremento Determinado pelos Operadores: (a) 1ª Iteração (b) 2ª Iteração

(a)

Operador	Índice de Incremento	Escore CE_{pre} + Incremento
SMTC	0.003875	0.992269
Varr	0.003407	0.991801
SMTT	0.003386	0.99178
VDTR	0.002425	0.990819
Vsrr	0.001525	0.989918
OLAN	0.001286	0.98968
OLRN	0.001273	0.989667
OLBN	0.001231	0.989624
ORAN	0.001116	0.98951
OEAA	0.000869	0.989263
ORBN	0.000803	0.989197
...	...	...
OABN	0.000118	0.988512
Vprr	0.000026	0.98842

(b)

Operador	Índice de Incremento	Escore CE_{pre} + Incremento
VDTR	0.002425	0.994695
Vsrr	0.001428	0.993697
OLAN	0.001286	0.993555
OLRN	0.001273	0.993542
OLBN	0.001231	0.993499
ORAN	0.001116	0.993386
OEAA	0.000869	0.993138
ORBN	0.000803	0.993072
STRI	0.000763	0.993032
ORSN	0.000668	0.992938
STRP	0.000484	0.992753
...	...	...
SMTT	0.000052	0.992321
Vprr	0.000026	0.992295

Após a inserção de SMTC, a tabela que contém o índice de incremento proporcionado pelos operadores de $\overline{CE}_{pre}$ deve ser gerada novamente (Tabela 6(b)) na tentativa de adicionar a CE_{pre} um operador da classe de mutação de operadores. De acordo com a Tabela 6(b) tem-se que, para a classe de mutação de operadores, o conjunto *CandIns* é formado pelos operadores OLAN, OLRN, OLBN e ORAN, todos proporcionando índice de incremento na faixa de 0.001. Tais operadores não foram incluídos empiricamente pelos operadores de CE_{pre}, apresentando *strengths* de 0.023, 0.020, 0.040 e 0.014, respectivamente. Como o operador OLBN é o de maior *strength*, este deve ser adicionado a CE_{pre}.

Este processo foi repetido para as classes de mutação de variáveis e de constantes, nesta ordem. Da classe de mutação de variáveis foi adicionado o operador VDTR; em relação à classe de mutação de constantes nenhum operador foi adicionado, visto que nenhum deles proporcionou índice de incremento igual ou superior à *IIM*. Ao final do Passo 5, CE_{pre} = {SMTC, SSDL, OLBN, ORRN, VTWD, VDTR, Cccr, Ccsr}.

Passo 6

Para a aplicação deste passo definiu-se $IMS = 0.300 \pm 0.005$. Analisando a Tabela 5(b), tem-se que os operadores SSWM, SWDD, OABN, SMTC, OLSN, Vprr e OASN apresentam alto *strength*. Destes, observa-se que o único operador que não foi incluído empiricamente[4] por

[4] Neste passo, $ms = 0.98$, ou seja, considera-se que o critério C_1 inclui empiricamente o critério C_2 se e somente se o escore de mutação que C_1 determina em relação a C_2 for igual ou superior a 0.98.

CE_{pre} foi SWDD ($CE_{pre} \overset{0.98}{\Rightarrow} *\{SWDD\}$), sendo selecionado para compor o conjunto *CAS* (Tabela 7). Como somente o operador SSWM faz parte de *CAS*, este deve ser adicionado a CE_{pre}. É importante relembrar que, se existissem dois ou mais operadores em *CAS*, deveria ser inserido em CE_{pre} o de maior *strength* e, além disso, *CAS* deveria ser gerado novamente. O operador SMTC, embora apresente alto *strength*, não foi analisado visto que ele não pertence a $\overline{CE}_{pre}$ (foi inserido em CE_{pre} no Passo 5).

Tabela 7. Escore de CE_{pre} em Relação aos Operadores de Alto *Strength*

Operador	Escore de Mutação
SWDD	0.892
OASN	0.981
Vprr	0.993
OABN	0.996
OLSN	1.000
SSWM	1.000

Ao término do Passo 6 obtém-se o conjunto essencial final *CE* = {SWDD, SMTC, SSDL, OLBN, ORRN, VTWD, VDTR, Cccr, Ccsr}. A Tabela 8 ilustra os conjuntos essenciais preliminares (CE_{pre}) e o conjunto essencial (*CE*) obtidos com a aplicação do procedimento, bem como o escore de mutação e o custo desses conjuntos em relação à Análise de Mutantes.

Tabela 8. Operadores Obtidos a cada Passo do Procedimento

Passo	Operadores	Escore	Custo
1	{Ccsr, Vsrr, CRCR, VTWD, Cccr, ORRN }	0.98087	0.40543
2	{Ccsr, Vsrr, CRCR, VTWD, Cccr, ORRN, SSDL }	0.99014	0.44431
3	{Ccsr, VTWD, Cccr, ORRN, SSDL }	0.98839	0.26072
4	{SSDL, ORRN, VTWD, Cccr, Ccsr}	0.98839	0.26072
5	{SMTC, SSDL, OLBN, ORRN, VTWD, VDTR, Cccr, Ccsr}	0.99592	0.34009
6	{SWDD, SMTC, SSDL, OLBN, ORRN, VTWD, VDTR, Cccr, Ccsr}	0.99616	0.34139

Os operadores de mutação que compõem o conjunto essencial são descritos na Tabela 9.

Tabela 9. Descrição dos Operadores Essenciais da Linguagem C

Operador	Descrição
SWDD	Substitui o comando *while* por *do-while*.
SMTC	Interrompe a execução do laço após duas execuções.
SSDL	Retira um comando de cada vez do programa.
OLBN	Substitui operador lógico por operador *bitwise*.
ORRN	Substitui operador relacional por operador relacional.
VTWD	Substitui referência escalar pelo seu valor sucessor e predecessor.
VDTR	Requer os valores negativo, positivo e zero para cada referência escalar.
Cccr	Substitui constante por constante.
Ccsr	Substitui referências escalares por constantes.

6. Comparação com Outras Abordagens e Outros Critérios

Nesta seção os aspectos de custo e escore de mutação associados à aplicação do conjunto essencial de operadores de mutação (*CE*) são apresentados e comparados à aplicação das abordagens de Offutt *et al.* e Wong *et al.* no contexto da linguagem C. Considerou-se também o conjunto essencial restrito de operadores (*CER*) formado por um único operador (o de maior escore) de cada classe de mutação. Ainda, esses dados são comparados à Mutação Aleatória (10, 20, 30 e 40%).

A partir da Tabela 10 pode-se observar que o conjunto *CE*, embora esteja entre os de maior custo, é o que proporciona o maior escore de mutação. Observe-se ainda que aplicando *CER* obtém-se a mesma faixa de custo que as abordagens de Offutt *et al.* e de Wong *et al.*, com um escore de mutação maior. Note-se que a classe de mutação selecionada por Offutt *et al.* é a de constante e, por outro lado, o conjunto proposto por Wong *et al.* não possui nenhum operador desta classe de mutação; na verdade, nenhuma dessas abordagens tem como requisito explícito procurar manter no conjunto essencial ao menos um operador de cada classe de mutação.

Tabela 10. Mutação Restrita e Aleatória em Relação à Análise de Mutantes

Mutação Restrita			*Mutação Aleatória*		
Critério	**Escore**	**Custo**	**Critério**	**Escore**	**Custo**
Conjunto Essencial Restrito[5] (*CER*)	0.98505	0.20118	Mutação 10% Aleatória	0.97160	0.09814
Conjunto Essencial (*CE*)	0.99616	0.34139	Mutação 20% Aleatória	0.98791	0.20564
Conjunto Essencial (Offutt *et al.*)[6]	0.97143	0.21061	Mutação 30% Aleatória	0.99120	0.31202
Conjunto Essencial (Wong *et al.*)[7]	0.98107	0.27152	Mutação 40% Aleatória	0.99420	0.40714

Outro aspecto relevante é a eficácia do critério. Tem sido observado que a Mutação Restrita é mais eficaz que a Mutação Aleatória. Nesse sentido, dado que os custos de aplicação de *CE* e *CER* são menores que o da Mutação Aleatória, restaria investigar o aspecto de eficácia da abordagem proposta neste trabalho com as abordagens de Offutt *et al.* e de Wong *et al.*

Um outro aspecto muito importante é que a determinação do conjunto essencial de operadores pode ser feita e aplicada de forma incremental, determinando-se inicialmente o operador que proporciona o mais alto escore de cada classe de mutação, no caso *CER*, e refinando esse conjunto em função da criticalidade da aplicação, tempo e orçamento, considerando que *CER* determina um escore de mutação maior que as outras abordagens de Mutação Restrita de mesmo custo.

O procedimento descrito neste trabalho e as comparações realizadas também foram conduzidas para um outro conjunto de programas (utilitários do Unix, largamente utilizados em estudos anteriores [WON93, WON97b]), tendo sido obtidos essencialmente os mesmos resultados [BAR98].

7. Conclusão

Ressaltou-se neste artigo a necessidade de determinação de abordagens pragmáticas para a aplicação da Análise de Mutantes na produção de software em ambientes industriais, dado que esse critério apresenta limitações de custo para sua aplicação mas, por outro lado, tem-se mostrado bastante eficaz em revelar a presença de erros. As questões de produtividade e qualidade pertinentes à Engenharia de Software são também afetas à atividade de teste, e a redução de custo e aumento de eficácia e produtividade das atividades de teste constituem aspectos fundamentais para o desenvolvimento de software.

A proposição de um procedimento para a determinação de um conjunto essencial de operadores de mutação (*CE*) para a linguagem C, no contexto da ferramenta *Proteum*, é uma contribuição relevante para a redução do custo de aplicação do critério Análise de Mutantes para o teste de programas escritos nessa linguagem.

---

[5] Conjunto Essencial Restrito = {SSDL, ORRN, VTWD, Ccsr}

[6] Conjunto Essencial (Offutt *et al.*) = {Cccr, Ccsr, CRCR}

[7] Conjunto Essencial (Wong *et al.*) = {VDTR, VTWD, ORRN, OLLN, OLNG, OCNG, ORLN, OLRN, OLAN, OALN, STRP}

Além disso, comparou-se a abordagem proposta neste artigo com outras abordagens descritas na literatura e observou-se que o escore de mutação determinado por *CE* em relação ao critério Análise de Mutantes é maior que o escore de mutação determinado pelas demais abordagens de Mutação Restrita. O conjunto essencial obtido proporciona uma redução superior a 65%, preservando um alto escore de mutação em relação à Análise de Mutantes.

Os resultados obtidos neste trabalho motivam a continuidade desta linha de pesquisa. A curto prazo pretende-se investigar esses aspectos para outros programas e outros domínios de aplicação, inclusive aspectos de eficácia das abordagens mencionadas neste artigo. Inicialmente um estudo de caso para investigar os aspectos de custo e eficácia está sendo projetado com o programa *Space*. Trata-se de uma aplicação real, desenvolvida pela Agência Espacial Européia, que permite ao usuário descrever o posicionamento de antenas através de uma linguagem de alto nível. Esse programa possui um histórico a respeito dos erros encontrados durante seu desenvolvimento, fornecendo, desse modo, um cenário de experimentação bastante próximo da realidade.

8. Bibliografia

[ACR79] ACREE, A.T. *et al.*; *Mutation Analysis*, Technical Report GIT-ICS-79/08, Georgia Institute of Technology, Atlanta, GA, Setembro, 1979.

[AGR89] AGRAWAL, H. *et al.*; *Design of Mutant Operators for the C Programming Language*, Technical Report SERC-TR-41-P, Software Engineering Research Center, Purdue University, West Lafayette, IN, Março, 1989.

[BAR98] BARBOSA, E. F.; *Uma Contribuição para a Determinação de um Conjunto Essencial de Operadores de Mutação no Teste de Programas C*, Mestrado em Andamento, ICMC/USP, São Carlos, SP.

[BUD80] BUDD, T.A. *et al.*; "Theoretical and Empirical Studies on Using Prog Mutation to Test the Functional Correctness of Prog.", *7th ACM Symposium on Principles of Programming Languages*, Janeiro, 1980.

[BUD81] BUDD, T.A.; *Mutation Analysis: Ideas, Examples, Problems and Prospects, Computer Program Testing*, North-Holand Publishing Company, 1981.

[DEL93] DELAMARO, M.E.; *Proteum – Um Ambiente de Teste Baseado na Análise de Mutantes*, Dissertação de Mestrado, ICMC/USP, São Carlos, SP, Outubro, 1993.

[DEM78] DEMILLO, R.A.; *Software Testing and Evaluation*, The Benjamim/Commings Publishing Company, Inc, 1978.

[DEM88] DEMILLO, R.A. *et al.*; "An Extended Overview of the Mothra Testing Environment", in *Proceedings of the Second Workshop on Software Testing, Verification and Analysis*, Banff, Canadá, 1988.

[HER76] HERMAN, P.M.; "A Data Flow Analysis Approach to Program Testing", *Australian Computer Journal*, v. 8, n. 3, Novembro, 1976.

[I390] *IEEE Standard Glossary of Software Engineering Terminology*, Padrão 610.12, IEEE, 1990.

[MAL91] MALDONADO, J.C.; *Critérios Potenciais Usos: Uma Contribuição ao Teste Estrutural de Software*, Tese de Doutorado, DCA/FEE/UNICAMP, Campinas, SP, Julho, 1991.

[MAT91] MATHUR, A.P.; "Performance, Effectiveness, and Reliability Issues in Software Testing", in *Proceedings of the Fifteenth Annual International Computer Software and Applications Conference*, Tóquio, Japão, 1991, pp. 604-605.

[MAT93] MATHUR, A.P.; WONG, W.E.; "Evaluation of the Cost of Alternate Mutation Strategies", *VII SBES – Simpósio Brasileiro de Engenharia de Software*, Rio de Janeiro, RJ, Outubro, 1993.

[OFF93] OFFUTT, A.J.; ROTHERMEL, G.; ZAPF, C.; "An Experimental Evaluation of Selective Mutation", in *Proceedings of the 15th International Conference on Software Engineering*, Baltimore, MD, Maio, 1993, pp. 100-107.

[OFF96] OFFUTT, A.J. *et al.*; "An Experimental Determination of Sufficient Mutant Operators", *ACM Transactions on Software Engineering Methodology*, v. 5, n. 2, Abril, 1996, pp. 99-118.

[PAU93] PAULK, M.C. *et al.*; *Capability Maturity Model for Software – versão 1.1*, SEI Technical Report CMU/SEI-93-TR-24, Fevereiro, 1993.

[RAP85] RAPPS, S.; WEYUKER, E.J; "Selecting Software Test Data Using Data Flow Information", *IEEE Transactions on Software Engineering*, SE-11(4), Abril, 1985.

[SOU96] SOUZA, S.R.S.; *Avaliação do Custo e Eficácia do Critério Análise de Mutantes na Atividade de Teste de Software*, Dissertação de Mestrado, ICMC/USP, São Carlos, SP, Junho, 1996.

[WEY90] WEYUKER, E.J.; "The Cost of Data Flow Testing: An Empirical Study", *IEEE Transactions on Software Engineering*, SE-16(2), Fevereiro, 1990, pp. 121-128.

[WON93] WONG, W.E.; *On Mutation and Data Flow*, Tese de Doutorado, Software Engineering Research Center – Purdue University, West Lafayette, IN, Dezembro, 1993.

[WON94] WONG, W.E. *et al.*; "Constrained Mutation in C Programs", *VIII Simpósio Brasileiro de Engenharia de Software*, Curitiba, PR, Outubro, 1994, pp. 439-452.

[WON95a] WONG, W.E.; MATHUR, A.; "Reducing the Cost of Mutation Testing: An Empirical Study", *J. Systems Software*, v.31, 1995, pp. 185-196.

[WON95b] WONG, W.E.; MATHUR, A.; "Fault Detection Effectiveness of Mutation and Data Flow Testing", *Software Quality Journal*, V. 4, 1995, pp.69-83.

[WON97a] WONG, W.E. *et al*; "A Comparison of Selective Mutation in C and Fortran", *Workshop do Projeto Validação e Teste de Sistemas de Operação*, Águas de Lindóia, SP, Janeiro, 1997, pp. 71-84.

[WON97b] WONG, W.E. *et al*; "Reducing the Cost of Regression Testing by Using Selective Mutation", *VIII CITS – International Conference on Software Technology* , Curitiba, PR, Junho, 1997, pp. 11-13.