TIAGO NOBRE

UMA ABORDAGEM BASEADA EM ALGORITMOS DE OTIMIZAÇÃO MULTIOBJETIVOS PARA REDUZIR O CUSTO DO CRITÉRIO DE TESTE ANÁLISE DE MUTANTES

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientadora: Profa. Dra. Silvia Regina Vergilio

CURITIBA

2011

N754 Nobre, Tiago
Uma abordagem baseada em algoritmos de otimização multiobjetivos para reduzir o custo do critério de teste análise de mutantes / Tiago Nobre – Curitiba, 2011.
67f. : il., tabs.

Impresso.
Dissertação (mestrado) – Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática.
Orientadora: Silvia Regina Vergilio

1. Informática. 2. Software - Testes. I. Vergílio, Silvia Regina. II. Título.

CDD: 005.14

TIAGO NOBRE

UMA ABORDAGEM BASEADA EM ALGORITMOS DE OTIMIZAÇÃO MULTIOBJETIVOS PARA REDUZIR O CUSTO DO CRITÉRIO DE TESTE ANÁLISE DE MUTANTES

Dissertação aprovada como requisito parcial à obtenção do grau de Mestre no Programa de Pós-Graduação em Informática da Universidade Federal do Paraná, pela Comisso formada pelos professores:

Orientadora: Profa. Dra. Silvia Regina Vergilio
Departamento de Informática, UFPR

Profa. Dra. Aurora Ramirez Pozo
Departamento de Informática, UFPR

Profa. Dra. Ellen Francine Barbosa
Departamento de Ciências e de Computação, ICMC/USP

Curitiba, 08 de abril de 2011

Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, do aluno Tiago Nobre, avaliamos o trabalho intitulado, *"UMA ABORDAGEM BASEADA EM ALGORITMOS DE OTIMIZAÇÃO MULTIOBJETIVOS PARA REDUZIR O CUSTO DO CRITÉRIO DE TESTE ANÁLISE DE MUTANTES"*, cuja defesa foi realizada no dia 08 de abril de 2011, às 08:30 horas, na Sala de Video-Conferência do Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela aprovação do candidato.

Curitiba, 08 de abril de 2011.

Profª. Dra. Silvia Regina Vergilio
DINF/UFPR - Orientadora

Profª. Dra. Ellen Francine Barbosa
USP - Membro Externo

Profª. Dra. Aurora Trinidad Ramirez Pozo
DINF/UFPR - Membro Interno

MINISTÉRIO DA EDUCAÇÃO - UNIVERSIDADE FEDERAL DO PARANÁ - Setor de Ciências Exatas - Coordenação de Pós-Graduação em Informática

AGRADECIMENTOS

Agradeço a Deus, aos meus pais, ao meu irmão Hamilton pelo apoio financeiro, preocupação e incentivo, à minha irmã Bete pela segurança e apoio. Agradeço aos meus amigos que sempre me motivaram. Agradeço também às professoras Silvia e Aurora pela exigência e bom direcionamento do trabalho. Também sou grato ao CNPQ pelo auxílio financeiro.

SUMÁRIO

RESUMO

Na literatura o critério Análise de Mutantes tem se mostrado eficaz em revelar defeitos e o escore de mutação é uma medida bastante utilizada para avaliar a qualidade do teste. Entretanto, este critério apresenta um alto custo computacional devido ao número elevado de mutantes que precisam ser executados durante o teste. Estratégias para reduzir o custo de aplicação do critério foram propostas e geralmente visam a estabelecer um conjunto de operadores ditos essenciais para gerar um número de mutantes menor sem reduzir o escore global. Entretanto, um problema com estas estratégias é que elas são baseadas em procedimentos empíricos que ao final formam apenas um conjunto de operadores que geralmente visa a maximizar o escore com um número pequeno de mutantes. Elas não permitem a geração de mais de um conjunto de acordo com diferentes objetivos de teste, ou seja, não consideram o problema como multiobjetivo, para o qual não existe uma única solução ótima, mas sim diferentes boas soluções que representam uma relação de compromisso entre diferentes objetivos tais como: número de mutantes, número de casos de teste, defeitos encontrados, mutantes equivalentes, etc. Para dar um tratamento adequado ao problema foi proposta uma abordagem multiobjetivo, mas, entretanto, esta abordagem foi aplicada apenas no teste de unidade. Os resultados promissores obtidos são motivação para o presente trabalho que tem como objetivo explorar o uso da abordagem multiobjetivo no contexto do teste de integração. Três diferentes algoritmos foram explorados e avaliados em um experimento com programas reais. Os resultados obtidos com a estratégia considerando número de mutantes e escore são melhores quando comparados aos obtidos pelas estratégias tradicionais.

ABSTRACT

In the literature, the Mutant Analysis criterion has proved to be effective to reveal faults, and the mutation score is a measure largely employed to evaluate the quality of the test. Nonetheless, this criterion demands high computational costs due to the large number of mutants that must be executed during the test. Strategies to reduce the application costs of the criterion have been proposed. They generally aim at the establishment of a set of essential operators that generates a lower number of mutants without reducing the global score. However, a problem with these strategies is that they are based on empirical procedures, which produce only a set of operators, usually with the goal of maximizing score with a small number of mutants. They do not allow the generation of more than one set according to different test objectives, that is, they do not consider the problem as multi-objective to which there is no optimal solution, but different good solutions that represent a possible trade-off among different objectives, such as: number of mutants, number of test cases, found faults, equivalent mutants, etc. In order to handle this problem properly, a multi-objective approach was proposed, which has been nevertheless, applied only in the unit test. The promising results achieved at this level served as a motivation for the present work that explores the use of the multi-objective approach in the integration test context. Three different algorithms have been explored and evaluated in an experiment carried out with real programs. The results achieved by the strategy, considering the number of mutants and the score, are better, when compared to that ones obtained by traditional strategies.

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO 1

INTRODUÇÃO

De forma geral, o teste de software visa a encontrar defeitos [28]. Geralmente compreende algumas atividades tais como: planejamento de testes, projeto de casos de teste, execução dos casos de teste e avaliação dos resultados obtidos. Essas etapas devem ocorrer concomitantemente ao desenvolvimento do software e geralmente se repetem em diferentes fases: teste de unidade, de integração e de sistema [36]. A fase de teste de unidade visa a testar os módulos menores do software como funções e métodos. Já o teste de integração, abrange o teste da interface entre essas unidades do sistema. O teste de sistema analisa o software como um todo.

O primeiro ponto importante é encontrar formas de direcionar o teste, pois não é possível testar o software com todas as entradas possíveis, comparando com todas as saídas esperadas. Para isso foram propostos diferentes critérios de teste que direcionam a escolha dos casos de teste para aqueles que levam o sistema a falhar. Os critérios de teste também criam objetivos de teste para o software, fazendo com que os casos de teste sejam construídos de forma a atingir tais objetivos. Assim, conforme os objetivos vão sendo atingidos, é possível verificar o quanto o software foi testado de acordo com o critério dado.

Os critérios de teste foram derivados a partir de três técnicas básicas [36]. técnica funcional, técnica estrutural e técnica baseada em defeitos. A técnica funcional não considera a estrutura interna do software e preocupa-se somente em descobrir quais as funções do programa e quais os casos de teste necessários para testá-las. A técnica estrutural utiliza a estrutura interna do software e seus critérios em geral baseiam-se em caminhos que o fluxo do código pode seguir considerando estruturas condicionais e de repetição, no teste de definições e usos de variáveis em diferentes partes do programa, etc.

A técnica baseada em defeitos considera os enganos mais comuns cometidos por pro-

gramadores para derivar os dados de teste. Por exemplo, o critério baseado em defeito conhecido como Análise de Mutantes [15] aplica mudanças no código fonte gerando programas mutantes. Esses mutantes são gerados de acordo com uma categoria chamada operador de mutação que aplica um tipo específico de mudança no código. A Análise de Mutantes tem como objetivo gerar dados de teste para diferenciar a saída do programa original das saídas geradas pelos programas mutantes. Ao final, um escore de mutação é produzido, dado pelo número de mutantes diferenciados por conjuntos T de teste. O escore pode ser utilizado para avaliar a qualidade do conjunto de casos de teste T.

O critério Análise de Mutantes pode ser aplicado tanto no teste de unidade como no teste de integração, nível no qual é conhecido como mutação de interface [12]. A aplicação prática deste critério é possível devido a existência de ferramentas de suporte. Exemplos dessas ferramentas são: Mothra [9], Mujava [25], Proteum [13] e Proteum IM [14].

Em experimentos conduzidos para comparar critérios de teste, o critério Análise de Mutantes apresentou a maior eficácia em revelar defeitos, mas com alto custo computacional [41]. Por isso é importante desenvolver estratégias para a redução desses custos. Nesse sentido, diversas estratégias foram propostas. Dentre essas destacam-se:

A Mutação Aleatória $X\%$ [1] que seleciona aleatoriamente uma porcentagem X de mutantes a serem gerados para cada operador. A Mutação Restrita [27] e Seletiva [32] que visam a reduzir o número de operadores a serem aplicados e estabelecem maneiras para escolher apenas um subconjunto de operadores. Neste sentido alguns estudos foram efetuados para encontrar um conjunto essencial de operadores de mutação. Esses estudos consideram o nível de unidade [4] [5] [31] e também o de integração [5] [29] [43].

Essas estratégias têm seu ponto de partida na seguinte questão: se cada operador descreve um tipo de defeito, não seria possível identificar um conjunto pequeno de operadores de tal maneira que, revelando os defeitos descritos por esses operadores, também garantidamente seriam revelados os defeitos descritos pelo conjunto total de operadores? O objetivo dessas estratégias é gerar menos mutantes sem diminuir a qualidade dos testes. Isso é feito classificando alguns operadores que geram número reduzido de mutantes mantendo a qualidade do teste, formando um conjunto de operadores essenciais. Esse

problema é conhecido como suficiência de operadores.

Entretanto, as abordagens existentes para redução do custo do teste de mutação e determinação do conjunto essencial de operadores de mutação são baseadas em procedimentos empíricos que ao final fornecem apenas um conjunto de operadores que geralmente visa a maximizar escore com um número baixo de mutantes. Esses procedimentos e estratégias não permitem a criação de mais de um bom conjunto de caso de teste de acordo com os objetivos do teste que podem ser múltiplos, ou seja, o problema de escolher um conjunto para redução dos custos de critério Análise de Mutantes é de fato multiobjetivo, pois ele pode envolver diferentes aspectos além do alto escore e do baixo número de mutantes. Pode-se também querer minimizar o número de casos de teste e de mutantes equivalentes[1], maximizar número de defeitos revelados, etc.

Problemas multiobjetivos têm sido resolvidos na literatura por algoritmos de busca multiobjetivos. Exemplos destes algoritmos multiobjetivos são: os algoritmos genéticos multiobjetivo (NSGA-II) [11], o algoritmo de busca MTabu [6] [19] e algoritmos baseados em colônias de formiga (PACO) [18]. Esses algoritmos baseiam-se em conceitos de Dominância de Pareto [33] para encontrar conjuntos de soluções para um problema. Esses conjuntos representam um "trade-off", ou seja, uma solução de compromisso para os diferentes objetivos envolvidos, visto que na maioria das vezes uma solução única não é possível, pois diversas soluções podem ser consideradas boas.

Em Banzi et al [3] foi apresentada uma abordagem baseada em algoritmos de otimização multiobjetivo. Esse trabalho explorou o problema de suficiência de operadores em um experimento envolvendo seis programas escritos em linguagem C, utilizando a ferramenta Proteum. Ao final, devido à abordagem, foi possível ordenar os operadores da Proteum de acordo com sua relevância e contribuição para alto escore de mutação e baixo número de mutantes. Além disso, foi definido um conjunto essencial de operadores, cujos resultados superaram os resultados obtidos com as estratégias tradicionais.

Entretanto, a abordagem de Banzi et al [3] ainda não foi aplicada ao teste de integração e os resultados promissores obtidos no teste de unidade são uma motivação para o trabalho aqui apresentado.

[1]Mutantes equivalentes são programas que computam a mesma função que o programa P em teste e para os quais não existe caso de teste capaz de diferenciar seu comportamento.

1.1 Objetivos

Dado o contexto exposto na seção anterior, o objetivo do presente trabalho é explorar a abordagem baseada em algoritmos multiobjetivos proposta por Banzi et al [3] no teste de integração, mais especificamente considerando Mutação de Interface [12]. Para avaliar a proposta é utilizada a ferramenta Proteum IM [14] e seus operadores. Os resultados dos três algoritmos multiobjetivos mencionados na seção anterior são comparados, ampliando a confiabilidade do conjunto obtido. Com as soluções resultantes dos algoritmos é aplicado um procedimento que permite estabelecer conjuntos essenciais, de forma semelhante ao trabalho de Banzi et al [3]. Ao final, depois de obtido um dos possíveis conjuntos, seus resultados são comparados com os resultados das estratégias tradicionais.

1.2 Organização do Trabalho

No Capítulo 2 são abordados conceitos sobre a otimização multiobjetivo e descritos os principais algoritmos multiobjetivos implementados.

No Capítulo 3 é fornecida uma visão geral do Teste de Mutação aprofundando o critério Análise de Mutantes e focalizando as estratégias que visam a reduzir o custo de aplicação deste critério.

No Capítulo 4, depois de representar a população e detalhar a implementação dos algoritmos, são descritos os programas utilizados no experimento. Em seguida são mostrados resultados da aplicação da estratégia multiobjetivo no teste de integração utilizando-se os operadores da ferramenta Proteum IM [14] e três diferentes algoritmos de busca que são também comparados.

No Capítulo 5 é descrito um procedimento para se obter o conjunto essencial de operadores de mutação de interface e este procedimento é avaliado em comparação com as estratégias tradicionais. As conclusões deste trabalho são apresentadas no Capítulo 6.

O trabalho também contém um apêndice (Apêndice A) onde são mostrados resultados da comparação do conjunto obtido neste trabalho com as demais estratégias.

CAPÍTULO 2

METAHEURÍSTICAS E OTIMIZAÇÃO MULTIOBJETIVO

De acordo com Souza [40], um computador, que processa uma rota a cada 10^{-8} segundos ao executar o problema do caixeiro viajante considerando 20 cidades, precisa de 19 anos para encontrar a melhor rota possível. Independentemente da capacidade de processamento, verifica-se que não é viável realizar buscas exaustivas e por isso foram surgindo alternativas ao longo do tempo. Assim descobriu-se a importância da utilização de heurísticas que são justificadas porque muitas vezes não se pode garantir que uma solução para um problema seja ótima e porque uma solução um pouco melhor pode ser satisfatória. Sendo assim, o fato de encontrar soluções próximas às ótimas reduzem bastante o tempo de execução quando se utilizam heurísticas.

Segundo Souza [40], há dois tipos principais de heurísticas:

- Construtiva - a solução é construída elemento por elemento;
- de Refinamento - Técnicas de busca local baseadas na noção de vizinhança e de movimentos (trocas) entre os elementos constituintes de uma solução. Parte de uma solução inicial que pode ser obtida por uma heurística construtiva ou de forma aleatória. O refinamento das soluções ocorre ao percorrer a vizinhança e descartar as soluções inferiores e armazenar as soluções melhores.

Souza [40] também define as heurísticas clássicas de refinamento:

Best Improvement Method - Em uma iteração é escolhido o melhor vizinho da solução corrente;

First Improvement - É escolhido o vizinho que apresenta a primeira melhora com relação a solução corrente;

Random Descent/Uphill Method - Aceita o primeiro vizinho aleatório que apresenta melhora se for estritamente melhor que a solução corrente;

Random Not Descent/Uphill Method - Aceita o primeiro vizinho aleatório que apresenta melhora se for melhor ou igual a solução corrente;

Variable Neighborhood Descent (VND) - Troca de forma ordenada de estrutura de vizinhança durante a busca aceitando somente soluções de melhora com relação a solução corrente. Volta à primeira estrutura quando uma solução melhor é encontrada.

Pelo fato da busca heurística ser presa à resolução de um problema específico, as metaheurísticas foram propostas na década de 80. Elas foram criadas para oferecer uma estrutura teórica com caráter mais geral e mais flexível independente do tipo de problema sendo resolvido. Com uma estrutura teórica as metaheuríticas passaram a ser implementadas de acordo com as características definidas por seus idealizadores. Isso facilitou a disseminação e a implementação das metaheurísticas de uma forma geral. As metaheurísticas mais conhecidas são: Algoritmos Genéticos, Redes Neurais, Simulated Annealing, Busca Tabu, GRASP e Colônia de Formigas.

Glover et al [20] definem metaheurística como uma heurística que guia e modifica outra heurística. De acordo com [39] uma metaheurística possui uma heurística subordinada a ela e também possui mecanismos para retirar a busca do ótimo local, podendo eventualmente chegar a um ótimo global. Essa heurística precisa ser adapatada ao problema a ser resolvido.

Um metaheurística é caracterizada por:

- Possuir uma heurística subordinada;
- Por ter um caráter geral e ser independente do problema;
- Possuir mecanismos para sair de ótimos locais. Inclusive, as metaheurísticas diferenciam-se por esses mecanismos;

De acordo com [40] as metaheurísticas são divididas em duas categorias:

Busca Local - Novas soluções são obtidas por operadores de movimento que criam novas soluções vizinhas com base em mudanças simples na solução corrente. Exemplos: Tabu Search, Simulated Annealing, VND etc.

Busca Populacional - Tem o objetivo de manter conjunto de boas soluções e combiná-las para melhorar as soluções. Exemplos: Algoritmos Genéticos, Algoritmos Meméticos e Colônia de Formigas.

2.1 Otimização Multiobjetivo

Em geral, os problemas de busca possuem características a serem maximizadas ou minimizadas conforme a necessidade. Para isso se faz necessário definir funções objetivo que indiquem o valor de uma solução do problema em uma dessas características. Por exemplo, se uma transportadora pretende minimizar os custos com pedágio nas rodovias do país, uma solução para este problema é um dos possíveis trajetos entre a origem e o destino. O espaço de busca desse problema é formado por todos os trajetos possíveis nas rodovias do país. Para comparar trajetos, é necessário atribuir-lhes valor para que o algoritmo de busca possa descartar os trajetos de custo elevado. Para abordar este problema como multiobjetivo pode ser acrescentado o objetivo de minimizar a distância, por exemplo.

Para comparar duas soluções é necessário avaliá-las. Em um algoritmo multiobjetivo essa avaliação ocorre em dois aspectos ou mais. As funções de avaliação ou funções objetivo são responsáveis por tais medidas. Uma função objetivo determina se uma solução é melhor do que outra em um determinado objetivo. Além disso, os diversos problemas criam situações diversas em que é necessário minimizar todos os objetivos, maximizar todos os objetivos ou ainda minimizar uns e maximizar outros.

É necessário analisar todos os objetivos de uma solução simultaneamente, para atingir o grau de otimização multiobjetivo. Para isso é importante adotar um critério de seleção entre as melhores soluções durante a execução do algoritmo.

Descreve-se a seguir três formas de se tratar um problema multiobjetivo de acordo

com Coello et al [10]:

Agregação de Funções - Ocorre quando os objetivos são unidos em uma única função de avaliação por atribuição de pesos ou por soma dos valores de cada objetivo. No problema de redução de custos de transportadora a Agregação de Funções equivaleria a somar o valor dos pedágios com a distância. Esse seria o critério de comparação entre duas soluções diferentes. Tem a vantagem de ser simples para implementar. As desvantagens estão no fato de se misturar medidas diferentes e de se atribuir pesos aos objetivos empiricamente, podendo não refletir a realidade.

Ordenação Lexicográfica - Esse critério ordena os objetivos por prioridade e faz com que duas soluções sejam comparadas considerando cada um deles enquanto não for encontrada uma diferença relevante. Se uma solução for melhor em um objetivo de maior prioridade, os outros não precisam ser verificados. No problema da transportadora equivale a desconsiderar a solução de menor distância se já foi encontrada a de menor custo. Este critério evita a mistura de objetivos diferentes e também é simples. No entanto, a diferença entre duas soluções é medida de forma empírica.

Dominância de Pareto [33] - Critério que compara duas soluções em todos os objetivos simultaneamente sem priorizar ou ponderar objetivos. Esse modelo permite dizer qual solução domina e qual é dominada, ou seja, qual delas é mais interessante considerando mais de um objetivo.

2.1.1 Dominância de Pareto

Como mencionado, a Agregação de Funções e a Ordenação Lexicográfica apresentam alguns pontos negativos. Dentre as formas de seleção de soluções na otimização multiobjetivo, escolheu-se para este trabalho a Dominância de Pareto [33].

A Figura 2.1 exemplifica a Fronteira de Pareto com o problema de comprar um carro. Há diversos carros (soluções) possíveis, mas nesse caso, pretende-se facilitar ao usuário a escolha de um carro comparando-os segundo preço e conforto. As soluções mostradas minimizam preço e maximizam conforto e formam uma aproximação da Fronteira de

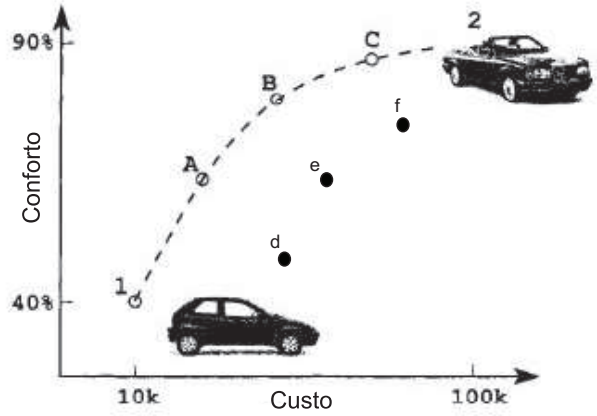

Figura 2.1: Fronteira de Alguns Modelos de Carro (adaptada de [8])

Pareto. A Fronteira de Pareto é o conjunto de soluções não dominadas do espaço de busca. Segundo Burke e Kendall [8] para uma solução x1 dominar x2:

- x1 não pode ser pior do que x2 em nenhum objetivo.
- x1 precisa ser estritamente melhor do que x2 em pelo menos um objetivo.

Não se deve utilizar o termo Fronteira de Pareto para designar os resultados obtidos por um algoritmo multiobjetivo. Dificilmente se poderá provar que o algoritmo tenha encontrado esse resultado máximo. Por isso, se utiliza o termo Aproximação da Fronteira de Pareto, pois conforme um algoritmo é executado ele avança em direção à Fronteira. Neste trabalho utiliza-se a sigla AFP para nomear a fronteira obtida por um algoritmo. Na figura 2.1, a AFP encontrada é composta pelo conjunto de soluções $\{1, A, B, C, 2\}$ no qual cada elemento do conjunto é um carro não dominado pelos que foram analisados. Esse conjunto só poderá ser chamado de Fronteira de Pareto depois de se analisar todos os carros existentes. Mas as soluções $\{d, e, f\}$ não fazem parte da AFP. Como exemplo, a solução e não faz parte da AFP porque:

Com relação à *A*: - A não é pior em nenhum objetivo. Inclusive é equivalente no objetivo conforto. Mas A é superior em pelo menos um objetivo (custo). Portanto, A domina e.

Com relação à *B*: - B não é pior em nenhum objetivo. B é melhor em pelo menos um objetivo, nesse caso nos dois. Portanto, B domina e.

Como já foi dito, um algoritmo multiobjetivo precisa de um critério para percorrer o espaço de busca que alterne entre os objetivos do problema a serem melhorados. Há casos em que se alternam as funções objetivo durante a busca utilizando uma porcentagem de interesse para cada objetivo. Algoritmos como Tabu Search (multiobjetivo) [6], NSGA-II e PACO percorrem a estrutura de vizinhança utilizando a Dominância de Pareto. Dessa forma não se prioriza uma função objetivo, mas todas têm a mesma importância no momento de comparar duas soluções.

2.1.2 Tabu Search (MTabu)

Com grande divulgação por atingir resultados ótimos em pouco tempo, esse algoritmo de busca de alta convergência foi proposto por Glover [19]. O objetivo inicial foi o de melhorar o algoritmo de busca básico chamado Hill Climbing [1], que pode ser aprofundado em Burke e Kendall [8]. O algoritmo de Busca Tabu adiciona ao Hill Climbing memória de curto prazo e critério de aspiração.

A Lista Tabu é uma memória de curto prazo do algoritmo que evita ciclos no espaço de busca. Isso é um avanço bastante significativo com relação ao Hill Climbing. Como pode ser observado na Figura 2.2, se durante a busca Tabu o algoritmo se encontra entre os pontos b e c, pode acontecer de o algoritmo voltar para o ponto b onde já havia passado. A Lista Tabu vai marcar o caminho recentemente percorrido e impedir que aquele trecho seja novamente explorado. Dessa forma o algoritmo é forçado a explorar as soluções da colina do ponto c, que são mais interessantes do que as soluções de b.

Na Lista Tabu pode ser colocada a solução inteira do problema, formando assim um tabu forte, ou as últimas transformações realizadas, formando assim um tabu mais fraco. O tempo de permanência na lista também determina se o tabu é forte. Quanto mais

[1] Em linhas gerais é um algoritmo de refinamento que procura soluções executando pequenas mudanças sobre elas visando a encontrar soluções melhores. Seu nome remete ao seu funcionamento: sobe uma colina sempre mudando para a melhor solução. É limitado porque nem sempre sobe a colina mais alta não encontrando a melhor solução global.

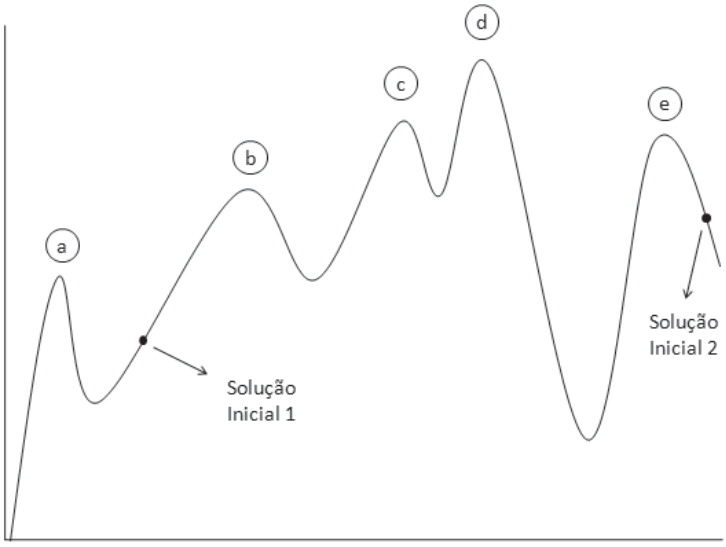

Figura 2.2: Representação dos Pontos Máximos de um Espaço de Busca (adaptada de [8])

tempo uma transformação ficar na lista, mais forte é o tabu. Durante a exploração do espaço de busca, o algoritmo checa se a solução encontrada ou se a transformação a ser feita está na Lista Tabu. Se estiver, a solução é ignorada naquele momento. Essa lista pode ser dinâmica não precisando manter os mesmos itens até o final da execução. A idéia é que uma transformação seja proibida em um determinado momento, mas possa ser executada novamente mais tarde. Para isso, essa lista tem um tamanho pequeno e a cada iteração uma solução ou transformação é retirada dela, para que as novas possam entrar. A Lista Tabu também é considerada dinâmica se tem o seu tamanho variado durante a execução.

O Critério de Aspiração é um ponto central do Tabu Search que permite aceitar soluções piores do espaço de busca com o objetivo de encontrar soluções melhores. O Critério de Aspiração mais conhecido é aquele em que se despreza uma restrição da Lista Tabu porque a solução que será gerada é melhor do que as encontradas até então. É essa inteligência que permite superar o Hill Climbing, passando por um vale e indo em outra direção do espaço de busca para "subir uma nova colina". Por isso, o Tabu Search possui grandes chances de encontrar o ótimo global.

Baykasoglu et al [6] propõem uma versão multiobjetivo do algoritmo Tabu Search chamada MOTS (*Multiobjective Optimization Tabu Search*) que explora uma vizinhança

utilizando a Dominância de Pareto. Essa implementação do Tabu Search tem um forte caráter multiobjetivo, pois a Dominância de Pareto é aplicada durante a busca e não somente ao final. Pode se dizer que a dominância guia a busca, fazendo com que todos os objetivos sejam levados em consideração simultaneamente. Esse algoritmo foi utilizado para resolver o problema de agendamento de *jobs*, que segundo o autor, dentre os problemas difíceis é um dos mais difíceis. Nesse trabalho foram obtidos resultados semelhantes aos conseguidos utilizando Algoritmos Genéticos para o mesmo problema.

Nessa versão do Tabu Search estão presentes as três listas abaixo:

Lista de Candidatos - Lista que recebe todos os vizinhos não dominados da solução corrente;

Lista Pareto - Lista que contém as soluções exploradas que não são dominadas;

Lista Tabu - Memória recente do algoritmo;

O algoritmo MOTS começa a partir de uma solução geradora. Essa solução é expandida e os vizinhos dela são adicionados à lista de candidatos da qual as soluções dominadas são imediatamente removidas. Então, a solução geradora é adicionada à Lista Pareto e as soluções dominadas são removidas.

Na Figura 2.3 visualiza-se o funcionamento do algoritmo MOTS para um problema hipotético extraído de [6]. Este é um problema de dupla maximização: pretende-se obter valores máximos para f1(x) e para f2(x). Uma barra vertical mantém separada a solução do problema de seus custos f1(x), f2(x) dessa mesma solução. Utilizando uma vizinhança pequena de tamanho três e Lista Tabu de tamanho quatro o algoritmo inicia com a solução (0,0) de custos (0,0). Essa é a primeira solução geradora. A vizinhança da solução geradora é adicionada à Lista de Candidatos de onde são removidas as soluções dominadas. No começo da iteração, uma nova solução geradora (semente) é escolhida aleatoriamente da Lista de Candidatos. Sempre ao final da iteração a solução escolhida como geradora é adicionada à Lista Pareto e à Lista Tabu. Quando o algoritmo terminar sua execução, a Lista Pareto terá a aproximação da fronteira obtida durante a busca, ou seja, as melhores soluções encontradas.

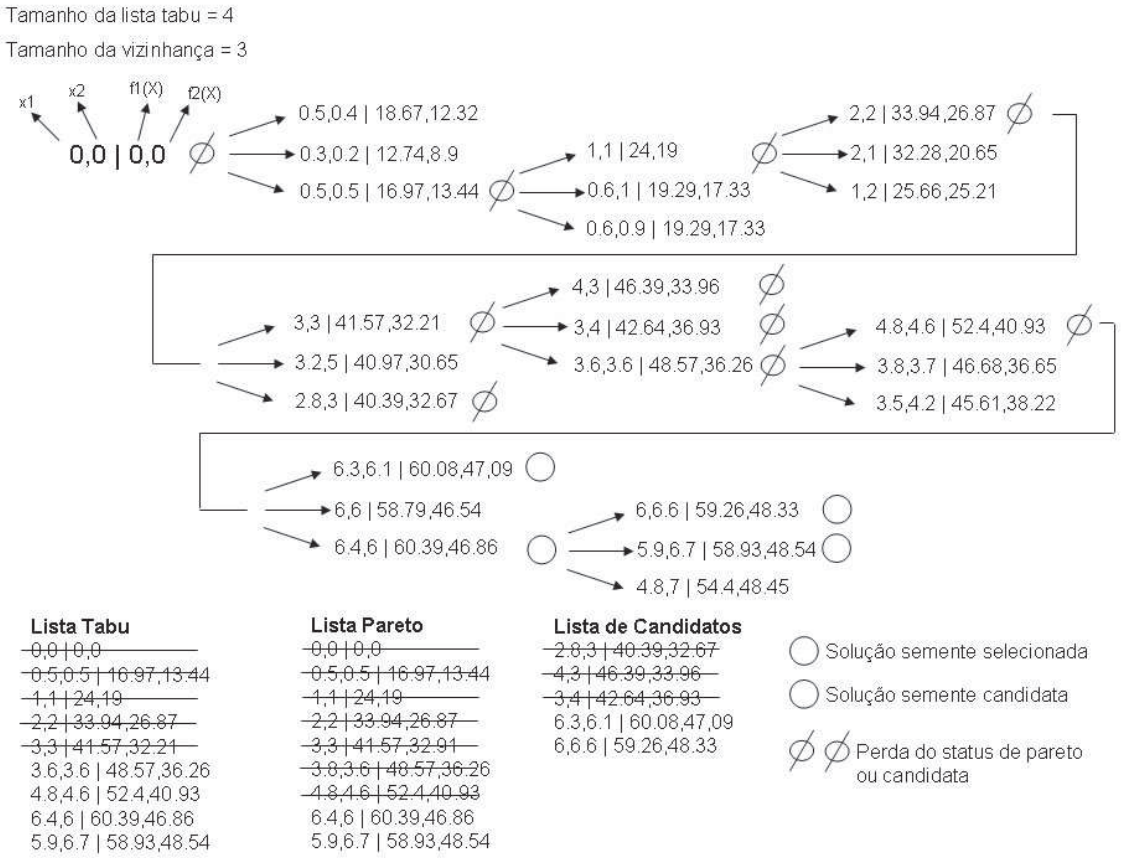

Figura 2.3: Exemplo de Funcionamento do Algoritmo MOTS (adaptada de [6])

O Algoritmo 1 mostra o pseudocógido do Tabu. O algoritmo opera com três laços principais. O primeiro ocorre para cada solução inicial cujo número total é definido em *maxseed*. No exemplo da Figura 2.3 *maxseed* equivale a 1, pois só há uma solução inicial: (0,0). O segundo é executado até ser atingido *max_iterations* (número total de iterações) e dentro dele ocorre a atualização das Listas de Candidatos e Pareto (*candidateList* e *paretoList*) removendo-se as soluções dominadas. Esse laço define o número de vezes que será explorada a vizinhança da solução geradora (semente). No exemplo da Figura 2.3 isso é feito 8 vezes. Nesse laço também ocorre o sorteio de uma nova solução geradora de vizinhança (*currentSolution*). No terceiro laço, o mais interno, é gerada a vizinhança e as soluções são avaliadas e adicionadas à Lista de Candidatos. Ao final da execução a Lista Pareto apresenta as melhores soluções encontradas formando uma AFP (Aproximação da Fronteira de Pareto).

Neste trabalho a implementação do algoritmo MOTS de Baykasoglu et al [6] receberá

Algoritmo 1: Pseudocódigo do MTabu

```
/* Sendo max_iterations o número de iterações e neighborhoodSize o tamanho da vizinhança */
Inicializar numberseed, candidateList, paretoList, tabuList;
while numberseed < maxseed do
   counter = 0;
   currentSolution = SeedSolution(numberseed);
   while counter < max_iterations do
      for i = 1 to neighborhoodSize do
         candidate = neighbor(currentSolution);
         if (not contains(tabuList, candidate)) then
            Evaluate(candidate);
            Update(candidateList, candidate);
         end if
      end for
      Update(tabuList, currentSolution);
      Update(paretoList, currentSolution);
      currentSolution = randomlySelectedCandidate(candidateList);
      counter =+ ;
   end while
   numberseed =+ ;
end while
```

o nome de MTabu por existir versões diferentes na literatura com o nome MOTS. MTabu remete à versão multiobjetivo da metaheurística Tabu Search.

2.1.3 Non-dominated Sorting GA (NSGA-II)

O Non-dominated Sorting GA (NSGA-II) é uma extensão do Algoritmo Genético (GA) para encontrar múltiplas soluções Pareto ótimas em um problema de otimização multiobjetivo [11]. Esse algoritmo possui as seguintes características:

1. Utiliza um princípio elitista;

2. Utiliza um mecanismo explícito de preservação de diversidade e

3. Enfatiza as soluções não dominadas.

O pseudocódigo pode ser visualizado no Algoritmo 2. Inicialmente, o NSGA-II gera uma população aleatória P_0, com $|P_0| = N$. Pela aplicação dos operadores genéticos habituais (seleção baseada em torneio (S), cruzamento (pc) e mutação (p_m)), a primeira população de filhos Q_0 ($|Q_0| = N$) é gerada. Ambos P_0 e Q_0 são unidos para formar R_0 ($|R_0| = 2N$).

Em todas as seguintes iterações n, tal que $n = 1, 2, 3, \ldots, max\_iterations$, o NSGA-II manipula uma população R_n. R_n é sorteada por não dominância. Isto significa que, inicialmente, todas as soluções não dominadas são colocadas na fronteira F_0. Em seguida, pontos de F_0 são descartados do conjunto global de soluções, e o novo conjunto de valores não dominados formam a fronteira F_1, e assim por diante.

Para formar a população, P_{n+1}, as soluções mais espalhadas são escolhidas utilizando o procedimento `CalculatesCrowdingDistances` para o ponto de F_j. Este procedimento retorna um valor *crowdingdistance* para cada solução que mede o quanto essa solução está perto de outras soluções.

Algoritmo 2: Pseudocódigo do Algoritmo NSGA-II

```
/* Sendo max_iterations o número de iterações */
P_0 = Q_0 = ∅ {Inicializar populações P_0 e Q_0}
n = 0 {Inicializar número de gerações}
Tournament Selection
Crossover
Mutation
Generate population Q_0
while n < max_iterations do
   R_n = P_n ∪ Q_n
   Sort R_n by non-dominance
   P_{n+1} = ∅
   while |P_{n+1}| ≤ N do
      CalculateCrowdingDistances for F_j
      P_{n+1} = P_{n+1} ∪ F_j
   end while
   Sort P_{n+1} using the ≥_n operator
   P_{n+1} = P_{n+1}[0 : (N − 1)] {Escolhe N primeiros elementos de P_{n+1}}
   Crossover
   Mutation
   Generate population Q_{n+1}
   n++
end while
```

Depois da seleção de N pontos de P_{n+1}, essa população é então ordenada utilizando-se a relação de dominância e o *crowdingdistance*. Então, somente as N melhores soluções de P_{n+1} permanecem na população. Por fim, os operadores genéticos são aplicados em P_{n+1} para formar a nova população de filhos Q_{n+1}. O *crowdingdistance* é calculado para cada solução e é considerado na escolha das soluções porque, segundo Deb [11], é importante manter um espalhamento nas soluções das fronteiras já encontradas para explorar melhor a população como um todo.

2.1.4 Pareto Ant Colony Optimization (PACO)

O algoritmo Ant Colony Optimization (ACO) foi introduzido em [18]. Foi inspirado pelo comportamento de colônias de formigas, principalmente por seu comportamento ao se espalharem em busca de alimento. Um dos pontos mais importantes é a comunicação indireta entre cada formiga de uma colônia. Isso ocorre quando elas liberam uma substância química chamada feromônio, formando uma trilha que é identificada por todos os indivíduos da colônia.

Na implementação do algoritmo, são criadas trilhas de feromônio artificiais formadas por formigas, também artificiais. Essas trilhas são uma distribuição numérica de informação, modificada pelas formigas para refletir a experiência acumulada enquanto tentam resolver um problema particular.

O algoritmo ACO parte do princípio de que as formigas possuem habilidades para encontrar o caminho mais curto do ninho até os locais onde há comida. Considerando um problema de otimização combinatorial, uma formiga constrói uma solução iterativamente. Esse procedimento construtivo é conduzido utilizando em cada passo uma distribuição de probabilidade, que corresponde aos caminhos de feromônios em formigas reais. Uma vez que uma solução é completada, as trilhas de feromônio são atualizadas de acordo com a qualidade da melhor solução construída. A cooperação entre formigas é realizada por uma estrutura comum que é a matriz compartilhada de feromônio.

O algoritmo discutido neste trabalho é multiobjetivo e é baseado no Pareto Ant Colony Optimization (PACO), que por sua vez, é baseado no Ant Colony System e foi proposto originalmente para resolver o problema Multiobjective Portfolio Selection [17].

O PACO trabalha com k matrizes de feromônio, onde k é o número de objetivos, e utiliza uma função heurística de agregação computada dos k objetivos. A regra de transição para escolher uma posição j é dada pela Equação 2.1.

$$j = \begin{cases} \text{argmax}_{j \in U} \left[\sum_{h=1}^{k} p_h \cdot \tau_{ij}^h \right]^{\alpha} \cdot \eta_{ij}^{\beta} & \text{if } q \leq q_0 \\ p\,(j) & \text{caso contrário} \end{cases} \tag{2.1}$$

O valor $p(j)$ é obtido pela probabilidade representada pela Equação 2.2. As matrizes de feromônio são representadas por τ^h e U é o conjunto de formigas disponíveis.

$$p(j) = \begin{cases} \dfrac{\left[\sum_{h=1}^{k} p_h \cdot \tau_{ij}^h\right]^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{l \in U}\left[\sum_{h=1}^{k} p_h \cdot \tau_{il}^h\right]^{\alpha}} & \text{if } j \in U \\ 0 & \text{caso contrário} \end{cases} \tag{2.2}$$

De acordo com Pasia et al [34] essa regra é uma extensão direta para múltiplas matrizes de feromônio da regra utilizada no Ant Colony System [18]. Os parâmetros α e β determinam a influência relativa do feromônio e informação heurística, respectivamente; η_{ij}^h é a informação heurística do objetivo h, $p^h \in [0, 1]$ são pesos, que são distribuídos uniformemente tais como $\sum_{h=1}^{k} = 1$. Em cada iteração, um vetor de peso é associado à cada formiga. O parâmetro q é um número aleatório distribuído uniformemente no intervalo [0, 1], e $q_0 \in [0, 1]$ é um parâmetro que define a intensificação e as propriedades de diversificação do algoritmo.

A atualização local de feromônio para todas as matrizes é executada sempre que uma formiga escolhe uma sequência de operadores de mutação (i, j) e usa as seguintes regras: $\tau_{ij}^h = (1 - \rho) \cdot \tau_{ij}^h + \rho \cdot \tau_0$, onde ρ é a taxa de evaporação e τ_0 é o valor inicial de feromônio.

A atualização global de feromônio é realizada depois que todas as formigas da população constroem uma solução. Para cada objetivo k, a melhor e a segunda melhor soluções são determinadas e então, a seguinte regra é aplicada: $\tau_{ij}^h = (1 - \rho) \cdot \tau_{ij}^h + \rho \cdot \Delta\tau_{ij}^h$, onde $\Delta\tau_{ij}^h$ recebe o valor: 15 se o componente (i, j) pertence à melhor ou à segunda melhor solução; 10 se (i, j) pertence somente ao melhor caminho; 5 se pertence ao segundo melhor caminho; e 0 em qualquer outro caso.

O Algoritmo 3 descreve os principais passos do algoritmo PACO. Primeiramente, um procedimento inicial é executado no qual as matrizes de feromônio são inicializadas com τ_0. O próximo passo é um laço. À cada iteração, todas as formigas constroem um conjunto s, de operadores de mutação ao executar os procedimentos `Build_Path` e `LocalPheronomeUpdate`.

Depois, buscas locais são realizadas para atingir uma exploração mais ampla do

Algoritmo 3: Pseudocódigo do PACO

```
/* Sendo max_iterations o número de iterações e Ants o número de formigas */
F1, F2 são as funções objetivo
Initialize pheromone (F1, F2, τ0)
while nriter ≤ max_iterations do
  for all ant in Ants do
    p1 = rand (0, 1)
    p2 = 1 − p1
    TakeInitialCandidates = ()
    s = GenerateInitialPath ()
    BuildPath s = (s, q, q0, p1, p2, F1, F2)
    LocalPheromoneUpdate (s, F1, F2)
    s1 = LocalSearch (s, F1)
    s2 = LocalSearch (s, F2)
  end for
  for all objective k do
    Determine best b and second-best b'
    GlobalPheromoneUpdate (b, b', Fk)
  end for
  ParetoSetUpdate (P, s1, s2)
  nriter += 1
end while
```

espaço de busca. Para cada objetivo, uma busca local é executada em cada formiga (`LocalSearch`). Essa busca local explora uma estrutura de vizinhança semelhante à do algoritmo MTabu.

Finalmente, todas as formigas são avaliadas e uma atualização global de feromônio é executada (`GlobalPheromoneUpdate`) baseando-se nas melhores soluções da iteração. O conjunto de melhores soluções encontradas até então, que é uma aproximação da Fronteira de Pareto, é também atualizado (`ParetoSetUpdate`). Ao final do laço, as soluções apresentadas no conjunto são as que o algoritmo obteve.

2.2 Considerações Finais

Neste capítulo foram descritos os principais conceitos de otimização multiobjetivo. Também foi mostrada resumidamente a evolução das heurísticas para as metaheurísticas para fugir de soluções que estão presas em um ponto ótimo local. Foram descritas três metaheurísticas que realizam a busca no espaço de solução de diferentes maneiras. Por esta razão, ou seja, por representarem diferentes tipos de algoritmos multiobjetivos eles foram

escolhidos para implementar a abordagem multiobjetivo descrita no presente trabalho. A implementação mais detalhada e relacionada ao problema em questão é apresentada no Capítulo 4.

CAPÍTULO 3

TESTE DE MUTAÇÃO

O teste de software é uma atividade fundamental dentre as atividades de garantia de qualidade de software cujo objetivo é executar o programa com objetivo de encontrar defeitos [28]. Inclui: planejamento de testes, projeto de casos de teste, execução e avaliação dos resultados de teste. Um caso de teste é composto pela entrada do programa (dado de teste) e a saída esperada [36]. Além disso, o teste pode ser utilizado em diferentes fases: teste de unidade, integração e de sistema [36].

O teste de unidade testa as menores partes do software isoladamente. Já o teste de integração testa as conexões entre as unidades determinadas pelo projeto. E por fim, o teste de sistema é posterior ao de integração e confronta o comportamento do software como um todo com a especificação.

Projetar casos de teste tem uma particularidade. Devido à impossibilidade de se testar o software com todas as suas possíveis entradas, é necessário criar métodos que direcionem a escolha dos casos de teste para aqueles que podem levar o sistema a falhar. Myers [28] diz que esse é o principal objetivo da atividade de teste. Para direcionar os casos de teste no cumprimento deste objetivo existem os critérios de teste. Esses critérios têm o papel de sinalizar se o software já foi suficientemente testado, pois marcam objetivos de teste a serem cumpridos. Esses objetivos de teste variam de acordo com a técnica de teste utilizada e podem ser, por exemplo, verificar se os casos de teste percorrem grande parte dos possíveis fluxos do código fonte. São três as técnicas de teste de software:

Técnica Funcional - É conhecida como teste de caixa preta, pois não considera o código fonte do software para gerar casos de teste. O software é testado de acordo com a especificação, conferindo-se somente as funcionalidades. O teste nesta técnica, geralmente, é composto de dois passos básicos: verificar quais são as funções e quais são os casos de teste para testá-las. A esta técnica estão associados os critérios

chamados Particionamento em Classes de Equivalência e Análise do Valor Limite que procuram explorar o domínio da entrada de dados do software, procurando combinações de dados ou os seus valores extremos que podem provocar uma falha. Os casos de teste são derivados a partir de fora do software, sem considerar sua estrutura interna [36].

Técnica Estrutural - Grande parte dos critérios desta técnica baseiam-se em um Grafo de Fluxo de Controle (GFC) que representa a estrutura interna do software. O grafo permite elaborar critérios baseados em fluxo de controle [38] e critérios baseados em fluxos de dados [26] [38]. Alguns dos critérios baseados em fluxo de controle são todos-nós, todos-ramos e todos-caminhos, os quais criam objetivos de teste relacionados ao fluxo do código. Assim é possível verificar a cobertura de teste, verificando quantos elementos requeridos por um critério foram satisfeitos com relação ao total. Os critérios baseados em fluxo de dados seguem a mesma linha, mas exigem a execução de caminhos para testar definições e consequentes usos das variáveis do programa.

Técnica Baseada em Defeitos - Esta técnica tem seus critérios de teste baseados nos enganos mais comuns cometidos por programadores no desenvolvimento do software e visa à geração de dados de teste para mostrar ausência ou não de certos tipos de defeitos no software. Dentre os critérios desta técnica destacam-se o critério Análise de Mutantes [1] e Semeadura de Erros [7].

Essas técnicas e critérios são aplicáveis para teste de unidade e teste de integração. As três técnicas são ditas complementares [36] porque revelam diferentes tipos de defeitos, e o mais interessante é explorar vantagens de cada uma.

A grande quantidade de critérios pode gerar uma certa dificuldade em determinar uma combinação satisfatória. Mas de qualquer forma, eles marcam um caminho a seguir para se obter uma boa cobertura no teste do software. Não é possível garantir que o programa está livre de defeitos, mas seguindo um método dado pelo critério é possível aumentar essa certeza. Assim os critérios marcam uma forma metódica de se selecionar dados de

teste, de se decidir se o programa foi testado o bastante [38].

A seguir, o critério Análise de Mutantes é descrito em detalhe, por ser o foco do presente trabalho.

3.1 Análise de Mutantes

O Critério Análise de Mutantes é um critério baseado em defeitos [1] [15] e apoia-se em dois pressupostos:

Hipótese do Programador Competente - Programadores são competentes e escrevem seus programas muito próximos ao que seria um programa correto. Portanto, considera-se que o programa em teste está muito próximo de estar correto;

Efeito do Acoplamento - Um pequeno defeito no software acarreta outros maiores [30]. Dessa forma, defeitos complexos são compostos de defeitos pequenos e revelando-se defeitos simples, defeitos complexos também podem ser revelados.

Partindo-se desses pressupostos, diversos programas mutantes são criados para um programa em teste P. Recebem esse nome porque são criados após uma mutação específica em P através dos operadores de mutação. Um operador é uma regra a partir da qual P é modificado e tem o objetivo de criar um mutante que descreve um possível defeito.

Um dado de teste (entrada para o programa) deve ser gerado para que os mutantes se comportem diferentemente de P e pode revelar um defeito quando ele e P forem executados com uma mesma entrada, mas produzirem uma saída diferente. Se a saída do programa é a saída esperada, foi encontrado um defeito. Se, pelo contrário, somente P emitir a saída esperada, o mutante é considerado morto. Por isso, há autores que usam o termo distinto para designar um mutante morto, porque houve distinção entre as saídas dos dois programas. Se os dois programas emitem a saída esperada, o testador precisa encontrar um dado de teste que diferencie a saída dos dois programas. Se for possível provar que para qualquer entrada do domínio de P não seja possível distinguir as saídas, então o mutante é considerado equivalente.

Determinar se um mutante é equivalente ao programa original é uma questão indecidível [7]. Ou seja, não é possível criar um programa para realizar essa tarefa de forma automatizada. Essa tarefa geralmente é realizada manualmente. Isso requer que cada mutante ainda vivo, ou seja, que não apresentou uma saída diferente do programa original, seja analisado particularmente.

Outro conceito da Análise de Mutantes é a cobertura ou escore de mutação. Indica a proporção dos mutantes mortos Mmt com relação ao total M de mutantes gerados depois da aplicação de um conjunto de casos de teste T, sem considerar os mutantes equivalentes Eq. A Equação 3.1 mostra a forma de se calcular o escore de mutação.

$$Escore(T, M) = \frac{Mmt}{M - Eq} \tag{3.1}$$

O escore de mutação pode ser utilizado para avaliar um dado conjunto de teste T e dizer o quão bom este conjunto é. Quando um conjunto de casos de teste T distingue todos os mutantes M ele é denominado adequado. Nesse caso, pode se designar T como M-Adequado.

3.2 Ferramentas para o Teste de Mutação de Programas

Na literatura encontram-se diferentes ferramentas para apoiar o teste de unidade utilizando Análise de Mutantes. Uma delas é a ferramenta Mothra [9] que possui 22 operadores e manipula programas escritos em Fortran. A outra é a Mujava [25] que trabalha com código orientado a objetos, mais concretamente, código fonte Java. Possui 15 operadores de mutação em nível de método e 28 em nível de classe, realizando testes unitários e de integração.

A Proteum [13] é uma ferramenta que permite a utilização da Análise de Mutantes para teste de unidade em programas escritos na linguagem C. Possui 71 operadores de mutação para teste de unidade classificados em quatro classes:

Mutação em Comandos - Realizam modificações nos comandos do programa omitindo ou trocando a ordem dos mesmos, etc;

Mutação em Operadores - Realizam transformações aplicadas em operadores relacionais, lógicos, aritméticos, etc;

Mutação em Variáveis - Realizar mudanças nas variáveis do programa, por exemplo, substituir variáveis por constantes, etc.

Mutação em Constantes - Realizar mudanças nas constantes do programa como, por exemplo, trocar um literal por outro, etc.

A ferramenta funciona através de uma sessão de teste na qual se fornece um programa a ser testado. A partir de então escolhem-se quais operadores serão utilizados para gerar mutantes, podendo ser escolhidos todos eles. Também pode-se optar por escolher parte dos operadores, ou somente os de uma determinada classe. A ferramenta possui um gerenciamento de casos de teste através do qual é possível adicionar, remover ou visualizar os já existentes. Depois de gerados os mutantes e criados os casos de testes, é possível executar os mutantes com os casos de teste selecionados e verificar o escore de mutação obtido. Os relatórios da ferramenta são bastante detalhados, permitindo verificar onde se deve concentrar os esforços de teste para aumentar o escore de mutação. Depois de criada a sessão e todas as vezes em que ela é reaberta, o software recomeça a ser testado incrementalmente, aumentando-se o escore de mutação, pois a cada caso de teste adicionado e executado com os mutantes, a ferramenta contabiliza o escore de mutação. Na versão gráfica é possível navegar pelos mutantes verificando seus estados, podendo-se marcá-los como equivalentes ou anômalos[1]. Nessa área a ferramenta disponibiliza o código fonte do programa original e o do mutante, marcando o local de mutação e qual operador a realizou. Além da interface gráfica a ferramenta também permite utilização através de scripts. Na Tabela 3.1 pode-se visualizar parte dos 71 operadores de unidade da ferramenta Proteum, com uma pequena descrição que remete ao nome do operador.

[1]Mutantes que podem apresentar erros em tempo de execução causados por defeitos como divisão por zero, laço infinito, etc.

Tabela 3.1: Exemplos de Operadores de Teste de Unidade da Ferramenta Proteum (extraída de Banzi et al [3])

Type	Description
CCCR	constant for constant replacement
CCSR	constant for scalar replacement
CRCR	required constant replacement
OASN	arithmetic operator by shift operator
OCGN	logical context negation
OEAA	plain assignment by arithmetic assignment
OEBA	plain assignment by bitwise assignment
OLBN	logical operator by bitwise operator
OLNG	logical negation
ORBN	relational operator by bitwise operator
ORAN	relational operator by arithmetic operator
ORLN	relational operator by logical operator
ORRN	relational operator mutation
ORSN	relational operator by shift operator
SMVB	move brace up and down
SMTC	n-trip continue
SRSR	return replacement
SSDL	statement deletion
SWDD	while (CT) replacement by do-while
STRI	trap on if (CT) condition
VDTR	domain traps
VSRR	mutate scalar references
VTWD	twiddle mutations

3.3 Mutação de Interface

O teste de integração baseado no critério Análise de Mutantes foi proposto como Mutação de Interface [12]. Ele considera fundamentalmente as conexões entre as unidades (métodos ou funções) do software. O operador de mutação é aplicado nas conexões entre duas unidades permitindo testar essas integrações.

Para propor os operadores no nível de interface, consideram-se três tipos de defeitos no teste de integração ilustrados na Figura 3.1

Dados um programa P e um caso de teste t; uma função f que chama outra função g, ambas pertencentes a P, e sendo S_i(g) os dados que entram em g e S_o(g) os valores retornados por g. Quando se executa P com t pode ocorrer um erro de integração na chamada de g a partir de f de três formas:

Erro 1 - f fornece S_i(g) incorreto causando uma saída incorreta em g;

Erro 2 - f fornece S_i(g) incorreto para g que, devido a isso, devolve S_o(g) incorreto para

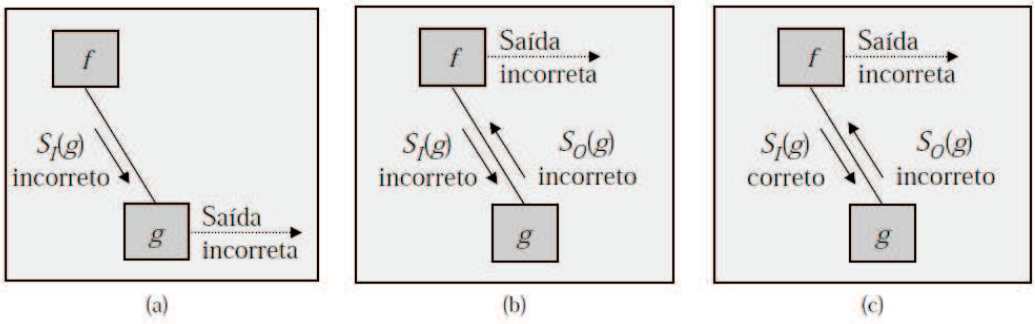

Figura 3.1: Tipos de Erro de Integração (extraída de [43])

f que emite uma saída incorreta;

Erro 3 - f fornece S_i(g) correto para g que devolve S_o(g) incorreto para f que emite uma saída incorreta.

3.3.1 Proteum IM

A Proteum *Interface Mutation* (IM) [14] é uma extensão da ferramenta Proteum que contempla o teste de integração. Ela possui 33 operadores de interface divididos em dois grupos que podem ser visualizados na Tabela 3.2. O número do grupo é o que inicia o nome do operador da Proteum IM. Os operadores do Grupo I aplicam mutações na função chamada. Os do Grupo II aplicam mutações na função que faz a chamada, no ponto de conexão entre as duas.

Foram utilizadas as seguintes abreviaturas na Tabela 3.2: P - parâmetros formais utilizados na chamada da função; L - variáveis declaradas na função chamada; G - variáveis globais utilizadas na função chamada; C - constantes utilizadas na função chamada; E - variáveis globais não utilizadas na função chamada; R - constantes requeridas. A tabela permite visualizar em que particularidades do código fonte cada operador realiza mutações.

Para exemplificar o funcionamento da ferramenta Proteum IM, abaixo são mostradas três funções de um programa P cujo código está completo.

Tabela 3.2: Operadores de Interface da Ferramenta Proteum IM [43]

NOME	DESCRIÇÃO
I-CovAllEdg	Garante cobertura de desvios
I-CovAllNod	Garante cobertura de nós
I-DirVarAriNeg	Acrescenta negação aritmética em variáveis de interface
I-DirVarBitNeg	Acrescenta negação de bit em variáveis de interface
I-DirVarIncDec	Acrescenta incremento (++) e decremento (–) em variável de interface
I-DirVarLogNeg	Acrescenta negação de lógica em variáveis de interface
I-DirVarRepCon	Troca variáveis de interface por elementos de C
I-DirVarRepExt	Troca variáveis de interface por elementos de E
I-DirVarRepGlo	Troca variáveis de interface por elementos de G
I-DirVarRepLoc	Troca variáveis de interface por elementos de L
I-DirVarRepPar	Troca variáveis de interface por elementos de P
I-DirVarRepReq	Troca variáveis de interface por elementos de R
I-IndVarAriNeg	Acrescenta negação aritmética em variáveis não de interface
I-IndVarBitNeg	Acrescenta negação de bit em variáveis não de interface
I-IndVarIncDec	Acrescenta incremento (++) e decremento (–) em variável não de interface
I-IndVarLogNeg	Acrescenta negação de lógica em variáveis não de interface
I-IndVarRepCon	Troca variáveis não de interface por elementos de C
I-IndVarRepExt	Troca variáveis não de interface por elementos de E
I-IndVarRepGlo	Troca variáveis não de interface por elementos de G
I-IndVarRepLoc	Troca variáveis não de interface por elementos de L
I-IndVarRepPar	Troca variáveis não de interface por elementos de P
I-IndVarRepReq	Troca variáveis não de interface por elementos de R
I-RetStaDel	Elimina comando return
I-RetStaRep	Troca comando return
II-ArgAriNeg	Acrescenta negação aritmética antes de argumento
II-ArgBitNeg	Acrescenta negação de bit antes de argumento
II-ArgDel	Elimina argumento
II-ArgIncDec	Incrementa e decrementa argumento
II-ArgLogNeg	Acrescenta negação lógica antes de argumento
II-ArgRepReq	Troca argumentos por elementos de R
II-ArgStcAli	Troca posição de argumentos de tipos compatíveis
II-ArgStcDif	Troca posição de argumentos de tipos diferentes
II-FunCalDel	Elimina chamada de função

```
// Programa P Original
int main() {
    int i;
    scanf("Forneça um valor: %d", &i);
    int valorRetornado = funcaoChamadora(i);
    printf("Valor retornado: %d", valorRetornado);
}
void funcaoChamadora(int param) {
    funcaoChamada(param);
}
int funcaoChamada(int param) {
    return param;
}
```

O mutante de P abaixo pode ser gerado pela ferramenta Proteum IM com o operador de mutação I-DirVarIncDec. A mutação foi aplicada na variável $param$ da $funcaoChamada$,

como era esperado por se tratar de um operador do Grupo I. Este mutante é morto com qualquer caso de teste que, ao ser passada a variável de interface *param* para a $funcaoChamada$, faça o programa mutante gerar uma saída diferente da saída de P. No exemplo, o dado de teste passado é o valor 10 e a saída esperada também é 10. No programa original a $funcaoChamada$ retorna 10, mostrando estar correta para este caso de teste. Quando o programa mutante é executado o valor retornado pela $funcaoChamada$ é 11. Assim, o mutante é considerado morto, pois sua saída foi diferenciada da de P.

```
// Programa Mutante de P
int main() {
    int i;
    scanf("Forneça um valor: %d", &i);
    int valorRetornado = funcaoChamadora(i);
    printf("Valor retornado: %d", valorRetornado);
}
void funcaoChamadora(int param) {
    funcaoChamada(param);
}
int funcaoChamada(int param) {
    return ++param;
}
```

O programa abaixo é considerado mutante equivalente de P, pois não há caso de teste que faça a mutação aplicada na variável *param* da $funcaoChamada$ gerar uma saída diferente da saída de P. Aplicando o mesmo caso de teste ao mutante equivalente, percebe-se que ele não é diferenciado de P, pois sua saída também é 10 que é a saída esperada.

```
// Mutante Equivalente de P
int main() {
    int i;
    scanf("Forneça um valor: %d", &i);
    int valorRetornado = funcaoChamadora(i);
    printf("Valor retornado: %d", valorRetornado);
}
void funcaoChamadora(int param) {
    funcaoChamada(param);
}
int funcaoChamada(int param) {
    return param++;
}
```

Nesse caso o mutante precisa ser avaliado e percebe-se que ele é equivalente porque *param* não é variável global e o incremento só é realizado na variável *param* depois que o valor já foi retornado. Assim, não é possível que ocorra propagação do resultado até o final da execução e não é possível diferenciar o mutante. Não é possível obter dado de teste que diferencie P de seu equivalente.

3.4 Reduzindo o Custo do Teste de Mutação

Apesar de o critério Análise de Mutantes ter grande potencial para revelar defeitos, ele apresenta um custo computacional elevado [41]. Quando uma ferramenta de teste procura pontos de mutação em um código fonte, encontra diversas possibilidades para cada operador que possui. O número de mutantes total que a ferramenta pode gerar varia de acordo com a estrutura de programação utilizada no software. Ou seja, não há uma proporção exata entre o número de linhas do software e o número de mutantes que serão gerados. No trabalho de Banzi et al [3], percebe-se que a cada 100 linhas de código geram-se em média 1290 mutantes. Durante, por exemplo, a avaliação de um conjunto de teste T todos esses programas mutantes precisam ser recompilados e executados pela ferramenta durante o teste para que as saídas sejam comparadas com o programa original. Além disso, esse critério exige um grande esforço para se determinar quais são os mutantes equivalentes. Por tudo isso, esse critério exige comparativamente mais casos de teste do que outros e possui alto custo computacional. Por isso, têm surgido iniciativas para reduzir esse impacto permitindo que a Análise de Mutantes seja aplicada na prática. Pretende-se diminuir o custo computacional, reduzindo-se o número de mutantes gerados e a quantidade de casos de teste, sem reduzir o escore de mutação.

Para reduzir o custo do teste de mutação algumas técnicas foram exploradas na literatura. A Mutação Aleatória [1] foi uma das primeiras propostas nesse sentido. Na Mutação Aleatória uma porcentagem ($X\%$) de mutantes de cada operador escolhida aleatoriamente é selecionada para gerar o conjunto M' não diminuindo o número de operadores utilizados.

A Mutação Restrita [27] escolhe um pequeno subconjunto de operadores de mutação para serem aplicados no programa a ser testado. O problema é então como estabelecer este conjunto.

Para resolver esta questão, a Mutação N Seletiva [32] seleciona um subconjunto de operadores excluindo os N operadores que geraram mais mutantes. Ou seja, com exceção dos N operadores que mais geraram mutantes, todos são utilizados para gerar o conjunto M'. Experimentos conduzidos com esta estratégia [32] motivaram a introdução da Mutação Suficiente [31].

Se cada operador descreve um tipo de defeito, o objetivo é identificar um conjunto pequeno de operadores de tal maneira que, revelando os defeitos descritos por esses operadores, também garantidamente sejam revelados os defeitos descritos pelo conjunto total de operadores. Considerando esta meta, Barbosa [4] definiu o problema de suficiência de operadores mais formalmente conforme a seguir:

Entrada: um conjunto M de mutantes gerados por uma conjunto O de operadores para um programa P e um conjunto T de dados de teste, M-Adequado.

Saída: um conjunto $M' \subseteq M$, produzido por um conjunto O' de operadores tal que $O' \subseteq O$ tal que o Escore(T',M) seja igual ou muito próximo ao Escore(T,M), sendo T' um conjunto M'-Adequado.

Offutt et al [31] introduziram um conjunto de cinco operadores de mutação para Fortran em experimentos realizados com a ferramenta Mothra [16]. Esse conjunto suficiente[2] obteve um escore superior a 98%, com 77% de mutantes a menos. Wong et al [44] introduziu um conjunto de operadores suficientes para linguagem C, e conduziu experimentos que mostraram forte relação entre os operadores de mutação suficiente para as linguagens C e Fortran [45].

Barbosa et al [5] propõem uma estratégia incremental para obtenção de operadores essenciais para gerar M'. Ao aplicarem esta estratégia à ferramenta Proteum no teste de unidade, obtiveram um conjunto com os seguintes operadores: SWDD, SMTC, SSDL, OLBN, ORRN, VTWD, VDTR, CCCR, CCSR, sendo $O' = 9$. O conjunto possui operadores de todas as classes (comandos, operadores, variáveis e constantes) e foi obtido ao se considerar operadores com maior escore de mutação. Também foram considerados os

[2]Conjunto resultado da aplicação de uma técnica de Mutação Suficiente.

operadores que mais incluem os demais. Essa verificação de inclusão foi realizada empiricamente. O procedimento essencial pode ser aplicado para diferentes conjuntos de programas e a cada aplicação apenas um conjunto de operadores essenciais pode ser obtido. Entretanto, diferentes conjuntos podem ser gerados para diferentes programas em diferentes domínios.

Já no teste de integração, mas no mesmo sentido de Barbosa et al [5] e também com passos incrementais, Vincenzi et al [43] introduziram o Conjunto Essencial de Interface que possui 8 operadores. Este trabalho é específico para o teste de integração e está voltado para os operadores da ferramenta Proteum IM. Os operadores são os seguintes: II-ArgAriNeg, I-CovAllNod, I-DirVarBitNeg, I-IndVarBitNeg, I-IndVarRepGlo, I-IndVarRepExt, I-IndVarRepLoc e I-IndVarRepReq, sendo $O' = 8$.

O trabalho de Namin et al [29] utiliza técnicas estatísticas e redução de variáveis para abordar o problema da suficiência de operadores aplicado ao teste de integração. Ele tem como objetivo reduzir o custo da utilização do teste de mutação como um critério de avaliação da adequação de um dado conjunto de teste T, reduzindo o número de mutantes gerados.

Para reduzir custos, algumas estratégias tentaram reduzir o tempo de execução elaborando alguns modelos que compilam e executam mais rápido [42]. Trabalhos mais recentes propuseram a combinação de dois defeitos descritos pelos mutantes para produzir um conjunto de mutantes de ordem mais alta [24]. Podem ser utilizadas técnicas baseadas em busca para selecionar as melhores combinações e reduzir o número de mutantes. Alguns resultados experimentais mostram que o uso desse tipo de mutantes não diminui a qualidade dos casos de teste gerados [35]. Outra estratégia, conhecida como *Mutation Clustering* [21] [23] utiliza técnicas de *clustering* (agrupamento) para agrupar mutantes similares e reduzir o número de casos de teste.

O principal problema das estratégias descritas é o fato de elas determinarem apenas um conjunto de operadores essenciais. Essas estratégias não permitem gerar conjuntos diferentes que sejam interessantes de acordo com objetivos diferentes. Além disso, há outros fatores que podem influenciar os custos de mutação, tais como o número de casos

de teste e tempo de execução associados a cada operador. Também não se trata apenas de obter um conjunto menor de operadores, pois isso não significa que será obtido um menor número de mutantes. Por tudo isso entende-se que o problema de determinar um conjunto de operadores essenciais é multiobjetivo. A próxima seção descreve um trabalho que propõe uma estratégia de redução de custos baseada em algoritmos de busca que permite selecionar conjuntos de operadores sem reduzir o escore de mutação.

3.5 Estratégia com Algoritmo de Busca

Como mencionado anteriormente, existem muitas possibilidades de combinação de operadores, correspondendo cada uma delas a um conjunto essencial. A seleção de um conjunto é um problema multiobjetivo influenciado por diversos fatores. Considerando este fato Banzi et al [3] introduziram uma abordagem de redução de custo do teste de mutação baseada em algoritmos de otimização multiobjetivo. A estratégia foi avaliada no nível do teste de unidade com os operadores da ferramenta Proteum e com 3 algoritmos NSGA-II, PACO e MTabu.

O experimento conduzido utilizou um conjunto de programas conhecido como *benchmark* da Siemens [22] e teve como objetivo minimizar o número de mutantes e maximizar o escore de mutação. Os resultados do algoritmo MTabu foram os melhores e utilizados em um procedimento para estabelecer um conjunto essencial de operadores. O procedimento permite aplicar diversas restrições segundo cada objetivo, podendo assim formar diversos conjuntos essenciais. Isso faz com que essa estratégia apresente grande versatilidade.

Para comparação com as estratégias tradicionais a restrição aplicada ao se utilizar o procedimento proposto foi sobre o objetivo escore de mutação, onde soluções das AFP's[3] do algoritmo MTabu que apresentavam escore igual ou superior a 0,9980 foram selecionadas. Esse valor de corte foi escolhido porque as estratégias tradicionais focam em alto escore de mutação. O conjunto selecionado foi obtido através das frequências dos operadores nas soluções escolhidas depois da aplicação da restrição, selecionando-se apenas

[3]Aproximação da Fronteira de Pareto. Conjunto de soluções não dominadas obtidas por uma execução do algoritmo, conforme explicado na Seção 2.1.1

60% dos operadores. O seguinte conjunto essencial contendo 12 operadores foi obtido: VTWD, VDTR, OEBA, ORBN, ORRN, VSRR, ORSN, ORAN, OEAA, SWDD, OASN e SSDL.

Esse conjunto quando comparado com estratégias tradicionais obteve resultados semelhantes ou superiores. O conjunto obtido permitiu maior redução de custo que o Conjunto Essencial de Barbosa e uma escore médio de 0,9984. O escore de mutação médio conseguido pelo Conjunto Essencial de Barbosa foi de 0,9979. A Estratégia Seletiva N=5 obteve escore de mutação médio de 0,9970. E a estratégia Aleatória 40% obteve o escore médio de 0,9965. Esses resultados mostram a eficácia do conjunto obtido por Banzi et al [3].

3.6 Considerações Finais

O presente capítulo forneceu um panorama sobre as estratégias de redução de custo do critério Análise de Mutantes e mostrou um trabalho pioneiro utilizando algoritmos de busca multiobjetivo. O principal problema das estratégias mencionadas é o fato de elas determinarem a cada aplicação apenas um conjunto de operadores essenciais. Não são apropriadas para gerar diferentes conjuntos de acordo com os diferentes objetivos. Determinar um conjunto de operadores é um problema multiobjetivo e pode ser melhor resolvido se encarado dessa forma. Como o trabalho de Banzi et al [3] obteve êxito na elaboração de um conjunto essencial para operadores de unidade da Proteum, torna-se importante realizar uma investigação sobre o uso dessa estratégia no teste de integração, e este é o objetivo do próximo capítulo no qual é descrito um experimento realizado com os operadores da Proteum IM.

CAPÍTULO 4

APLICANDO A ESTRATÉGIA MULTIOBJETIVO

Analisando os trabalhos apresentados no capítulo anterior para redução dos custos do teste de mutação verifica-se que a abordagem baseada em algoritmos de otimização multiobjetivo apresenta vantagens. Elas permitem a obtenção de diferentes conjuntos essenciais de operadores mantendo um alto escore de mutação e que podem ser utilizados de diferentes maneiras, diferentemente das estratégias tradicionais que estabelecem um conjunto fixo de operadores.

Entretanto esta estratégia ainda não foi explorada no teste de integração, considerando mutação de interface e descrever o estudo realizando esta investigação é o objetivo do presente capítulo. O estudo descrito adotou estratégias para representar a população e ajustar os parâmetros dos algoritmos multiobjetivos similares às adotadas em Banzi et al [3]. Detalhes do estudo são fornecidos a seguir.

4.1 Representação da População

Como mencionado o problema de redução de custos do teste de mutação e determinação de conjuntos essenciais de operadores pode envolver um número B de diferentes objetivos a serem otimizados: escore de mutação, número de casos de teste, número de mutantes, número de mutantes equivalentes, número de defeitos relacionados, tempo de execução, etc. Dado um programa P, um conjunto de operadores de mutação O, e um vetor de B funções objetivo, $f_i, i = 1...B$. O problema é encontrar um subconjunto O', tal que O' é o conjunto Pareto ótimo com relação às funções objetivo, $f_i, i = 1...B$.

Uma importante característica na implementação de um algoritmo metaheurístico é a representação escolhida para descrever as soluções do problema. Essa escolha irá influenciar na implementação de todos os estágios do algoritmo. Neste caso, a representação escolhida para o problema é simples, com a solução sendo representada por um vetor cujas

posições assumem um número inteiro no intervalo $[1, N]$, sendo N o número de operadores de mutação. Apesar de não importar a ordem, um operador não pode aparecer mais que uma vez em uma solução. Um exemplo de uma possível solução pode ser visualizado na Tabela 4.1.

Tabela 4.1: Exemplo de Uma Solução

15	1	8	0	6

A solução é formada por quatro operadores numerados: 15, 1, 8, 0 e 6, correspondendo, por exemplo, aos operadores: I-IndVarLogNeg, I-CovAllNod, I-DirVarRepGlo, I-CovAllEdg e I-DirVarRepCon da Proteum IM [14]. A descrição desses operadores já foi mostrada na Tabela 3.2.

Outra característica relacionada aos algoritmos multiobjetivo é a escolha das funções objetivo (f_i). Neste trabalho, foram implementadas duas funções baseadas no escore de mutação e no número de mutantes. O objetivo de número de mutantes será medido somando o número de mutantes de cada operador. O escore de mutação é medido:

- Obtendo o conjunto T' de casos de teste que é M'-Adequado, sendo M' o conjunto de mutantes formado pela junção dos mutantes dos operadores da solução.
- Aplicando o conjunto T' sobre M (conjunto de mutantes de todos os operadores utilizados para o programa) obtendo assim o escore correspondente ao conjunto total de mutantes.

Baseado nas duas medidas apresentadas acima, o problema é a busca por conjuntos de operadores de mutação que maximizam o escore e minimizam o número de mutantes.

4.2 Implementação dos Algoritmos

No Capítulo 2 os algoritmos foram abordados de forma mais abstrata, sem estar relacionados a algum problema concreto. Mas como é sabido, ao se aplicar o algoritmo ao problema, é preciso configurar seus parâmetros experimentalmente para explorar bem o

espaço de busca. Algumas vezes, também é necessário fazer adaptações devido à estrutura da solução.

Os três algoritmos mencionados anteriormente foram escolhidos pelo fato de explorarem o espaço de busca de maneiras diferentes, obtendo assim uma confiabilidade maior com relação a exploração realizada.

A presente seção visa a mostrar como os algoritmos foram implementados e configurados.

4.2.1 Detalhes de Implementação do MTabu

O algoritmo foi implementado com a intenção de ser bastante aleatório. Os movimentos realizados nas soluções para gerar a vizinhança são os seguintes: trocar, remover e adicionar operadores com 50%, 30% e 20% de probabilidade de serem escolhidos, respectivamente. Um dos três movimentos acima é escolhido por sorteio cada vez que se procura obter um vizinho da solução corrente. O operador de adição não é utilizado se a solução já possuir tamanho 7. Foi dada maior prioridade ao operador de troca, pois com um tamanho de soluções pequeno é mais interessante combinar as soluções em vez de permitir que elas aumentem de tamanho. Considerando que o tamanho da solução inicial influencia na exploração do espaço de busca, foram utilizadas 4 soluções iniciais ($maxseed$) com as seguintes quantidades de operadores de mutação: 0, 2, 3 e 4.

4.2.2 Detalhes de Implementação do NSGA-II

Foi escolhido o operador de crossover de um ponto. Já que não se pode permitir operadores repetidos na solução, também entende-se que os indivíduos não podem ter valores repetidos. Devido à essa restrição, uma pequena mudança foi realizada no operador crossover de um ponto. O algoritmo que faz o crossover entre dois indivíduos i1 e i2 funciona da seguinte forma:

1. Uma posição aleatória p do vetor é escolhida.

2. Trocam-se todas i posições iniciais de i_1 e i_2.

3. No caso de existirem operadores de mutação repetidos tanto em i_1 quanto em i_2, o operador de menor posição é eliminado.

Além do crossover, o operador de mutação também foi adaptado para este trabalho. A mutação ocorre como se segue. Uma das três ações disponíveis é escolhida aleatoriamente: remover, adicionar ou substituir um operador de mutação. Há a probabilidade de 50% de um operador ser removido, 25% de probabilidade de ser substituído e 25% de ser adicionado.

4.2.3 Detalhes de Implementação do PACO

Alguns detalhes de implementação do algoritmo para resolver o presente problema já foram apresentados na Seção 2.1.4. O ponto de partida do algoritmo são algumas formigas (ants) que começam a percorrer um caminho a partir de alguns operadores de mutação da Proteum IM selecionados aleatoriamente a cada iteração. Pode-se pensar nos operadores da Proteum IM utilizados em um programa espalhados geometricamente, onde cada formiga pode ir para qualquer operador com apenas um passo. Uma solução para o problema é equivalente a um caminho entre esses operadores. Enquanto as formigas caminham, marcam o trajeto com feromônio que, a partir de então, começa a evaporar. Ao caminhar as formigas verificam o trajeto à frente que possua mais feromônio e dependendo do resultado encontrado com a nova solução, podem reforçar o feromônio daquele trajeto. É importante que o feromônio evapore, para que as formigas tentem outros caminhos, que podem conduzir a soluções ainda melhores.

4.3 Programas Utilizados

Foi utilizado um conjunto de programas da Siemens do repositório disponível na Universidade de Nebraska-Lincoln chamado Subject Infraestructure Repository (SIR[1]) [22]. Diversos trabalhos que envolvem teste de software vêm utilizando esse repositório [2] [3] [29] [24]. Nesse repositório estão disponíveis, além de outras informações, o código C e

[1] http://sir.unl.edu/php/index.php

um conjunto de casos de teste que foi utilizado no experimento. Estes programas contêm em média 270 NLOC e incluem a maioria das estruturas C usadas em grandes sistemas, tais como: struct, apontador, alocação de memória, comandos *switch*, expressões condicionais complexas, etc. Maiores informações sobre os programas utilizados encontram-se na Tabela 4.2, tais como o número de linhas de código (LOC), número de mutantes gerados pela Proteum IM (MG), número de mutantes assumidos como equivalentes (MEq), conjunto de mutantes considerados para este trabalho após a eliminação dos equivalentes (M), quantidade de operadores da Proteum IM que geraram mutantes (O) e o número de casos de teste disponíveis no repositório (NCR). Como se pode observar, foram utilizados seis programas no experimento.

Tabela 4.2: Programas Utilizados

Programa	NLOC	MG	MEq	M	O	NCR
Printtokens	343	6762	1756	5006	28	4070
Printtokens2	355	5045	789	4256	29	4115
Schedule	296	1745	161	1584	31	2650
Schedule2	263	3434	1111	2323	31	2710
Tcas	137	1883	379	1504	22	1608
Totinfo	281	2057	92	1965	31	1052

Os programas possuem as seguintes funcionalidades: Printtokens e Printtokens2 são analizadores léxicos; Schedule e Schedule2 são associados a escalonadores de tarefas; Totinfo calcula estatísticas a partir dos dados de entrada; e o Tcas é relacionado à prevenção de colisão de aeronaves. Esses programas foram submetidos à versão script da Proteum IM [14] utilizando-se todos os 33 operadores da ferramenta, mas como se vê na coluna O da Tabela 4.2, nem todos os operadores da ferramenta geraram mutantes, pois não encontraram pontos de mutação nos programas. De qualquer forma, grande parte dos operadores foi utilizada. Também foram utilizados todos os casos de teste NCR disponíveis para cada programa.

Primeiramente os programas foram submetidos ao teste de mutação com a ferramenta Proteum IM com um caso de teste por vez. Sendo assim, cada versão do código foi executada n vezes, onde n é o número de casos de teste disponíveis no repositório SIR [22] multiplicado pelo número de mutantes gerados pela ferramenta Proteum IM para

o programa. A informação da execução de cada mutante com cada caso de teste foi armazenada em uma tabela. Essa tabela relata quais são os mutantes mortos e vivos. Por razões de simplificação, os mutantes que não foram mortos por nenhum caso de teste, foram definidos como equivalentes e removidos dessa tabela.

4.4 Geração de Conjuntos Adequados

Dentre os casos de teste disponíveis no repositório (conjunto NCR) nem todos são necessários para matar todos os mutantes M. Existem diferentes conjuntos T, tal que $T \subseteq NCR$ que são M-Adequados. Diferentes ordens de execução dos dados de teste implicam em diferentes conjuntos. Para este trabalho foram aleatoriamente selecionadas 10 ordens de execução, correspondendo a 10 diferentes conjuntos T, M-Adequados. Durante a execução de uma ordem, o conjunto T foi construído apenas com dados de teste que realmente contribuíam para o aumento do escore de mutação, ou seja, realmente matavam algum mutante ainda vivo não morto por nenhum dado anterior de T. Na Tabela 4.3 encontram-se os números de casos de teste de cada conjunto T que é M-Adequado.

Tabela 4.3: Tamanhos dos Conjuntos de Casos de teste M-Adequados

Programa	$T1$	$T2$	$T3$	$T4$	$T5$	$T6$	$T7$	$T8$	$T9$	$T10$
Printtokens	58	75	59	59	66	64	66	57	52	56
Printtokens2	31	33	29	31	23	33	32	38	33	32
Schedule	13	12	10	20	15	9	14	19	11	13
Schedule2	27	33	29	32	29	28	31	31	30	34
Tcas	71	77	77	84	78	70	74	73	71	68
Totinfo	15	13	15	21	12	13	16	14	19	19

4.5 Aplicação dos Algoritmos ao Problema

O espaço de busca do problema consiste em todas as combinações dos 33 operadores de mutação da Proteum IM. Como não faria sentido repetir operadores em uma solução, é necessário criar essa restrição: um operador não pode aparecer duas vezes na mesma solução. Inicialmente, o tamanho da solução pode variar de um operador até o número de operadores que a Proteum IM utilizou sobre um programa. Esse número pode ser

encontrado na Tabela 4.2 e varia de 22 a 31. Depois de se implementar os algoritmos, percebeu-se que as soluções Pareto tendem a ter 6 operadores ou menos. Por isso foi limitado o tamanho da solução nos algoritmos, apesar de continuar sendo possível combinar todos os operadores de determinado programa. Nos três algoritmos, o tamanho da solução foi limitado a 7.

Os parâmetros de cada algoritmo podem ser visualizados na Tabela 4.4. Esses valores são característicos de cada algoritmo e neste trabalho foram configurados experimentalmente, observando-se quais valores atingiam melhores resultados com menor custo computacional. Para o MTabu percebeu-se que utilizar mais 200 iterações não implicava em resultados melhores. As soluções iniciais ($maxseed$) ajudaram o algoritmo a convergir mais rapidamente, pois possuem de zero a três operadores. Isso influencia toda a busca, pois muda a região da qual ela é iniciada. Com o tamanho de vizinhança por volta de 25 o algoritmo se comportou bem para todos os programas. A Lista Tabu ficou com tamanho próximo a 20.

O NSGA-II também teve seus parâmetros configurados experimentalmente visando a não aumentar o custo computacional sem haver incremento nos resultados da busca. Os seus parâmetros foram fixados para limitar as execuções dos outros dois algoritmos. Considerou-se que o tamanho da população multiplicado pelo número de iterações ($max\_iterations$) deveria ser o número exato de chamadas à função objetivo dos três algoritmos. Isso foi feito com o objetivo de possibilitar a comparação entre os diferentes algoritmos.

Da mesma forma, o PACO teve o número de formigas ($ants$) aumentados em busca de um equilíbrio entre o custo computacional e os resultados alcançados.

Os três algoritmos foram executados dez vezes tendo como entrada os dez conjuntos Ti de cada programa. Além disso, cada conjunto Ti foi espaço de busca de cada algoritmo dez vezes, ou seja, cada algoritmo foi executado dez vezes com cada conjunto Ti de cada programa.

Cada algoritmo realizou dez execuções de cada conjunto de T de cada programa. Sendo assim, ao final das execuções havia dez conjuntos de Pareto para cada conjunto Ti, para

Tabela 4.4: Parâmetros dos Algoritmos

Programa	`Printtokens`	`Printtokens2`	`Schedule`	`Schedule2`	`Tcas`	`Totinfo`
MTabu						
maxseed	4	4	4	4	4	4
max_iterations	200	200	200	200	200	200
neighborhoodSize	28	29	30	34	32	12
tabuListSize	15	18	15	20	25	15
NSGA-II						
population size	150	150	150	150	150	150
max_iterations	115	113	121	147	147	46
S	4	4	6	6	6	4
p_c	40%	30%	60%	90%	70%	70%
p_m	40%	30%	60%	100%	30%	70%
PACO						
ants	8	8	10	10	13	5
max_iterations	20	20	20	20	20	20
q_0	0,3	0,3	0,3	0,3	0,3	0,3
ρ	0,2	0,2	0,2	0,2	0,2	0,2
α	1	1	1	1	1	1
β	1	1	1	1	1	1
t_0	0,1	0,1	0,1	0,1	0,1	0,1

cada algoritmo.

4.6 Comparação dos Algoritmos de Busca

Para que os algoritmos fossem comparáveis, foi estabelecido um número exato de chamadas à função objetivo. Esse número foi obtido através dos parâmetros do algoritmo NSGA-II multiplicando-se o número de iterações pelo tamanho da população. Cada execução dos três algoritmos teve o mesmo número de chamadas à f que pode ser observado na Tabela 4.5.

Tabela 4.5: Número Fixo de Chamadas à Função Objetivo

Programa	Chamadas à f
Printtokens	17250
Printtokens2	16950
Schedule	18150
Schedule2	22050
Tcas	22050
Totinfo	6900

Os algoritmos foram comparados de acordo com o número de soluções encontradas para cada execução. Para cada um dos dez conjuntos T de cada programa foi elaborada uma fronteira única a partir das fronteiras das dez execuções dos três algoritmos, removendo-se

as soluções dominadas. Para cada execução de cada algoritmo foi verificado o número de soluções obtidas pertencentes a esta fronteira. Na Tabela 4.6 encontram-se as médias desses valores para cada algoritmo. A coluna #S indica quantas soluções não dominadas os três algoritmos juntos encontraram executando sobre o conjunto Ti. Dessa forma, o algoritmo que, para cada execução, atinge os valores mais próximos de #S é o que obtém melhores resultados.

Tabela 4.6: Média de Soluções por Execução na Fronteira do Conjunto T

Printtokens				
	MTabu	NSGA-II	PACO	#S
$T1$	15,30	10,60	11,00	16
$T2$	25,00	16,60	13,70	26
$T3$	21,90	13,30	11,90	22
$T4$	20,40	14,80	11,00	21
$T5$	25,80	14,60	13,40	26
$T6$	25,50	16,60	15,40	26
$T7$	25,00	17,10	18,30	25
$T8$	19,00	14,10	11,60	19
$T9$	18,00	11,60	10,70	18
$T10$	19,40	12,60	11,40	20

Printtokens2				
	MTabu	NSGA-II	PACO	#S
$T1$	18,30	16,50	18,70	20
$T2$	22,20	17,40	19,00	23
$T3$	15,50	13,10	15,00	17
$T4$	23,30	18,20	19,30	24
$T5$	16,70	14,40	15,80	18
$T6$	14,50	12,20	13,20	16
$T7$	11,80	10,90	11,80	22
$T8$	18,20	14,50	15,50	19
$T9$	19,90	17,50	18,10	22
$T10$	16,80	14,10	15,40	18

Schedule				
	MTabu	NSGA-II	PACO	#S
$T1$	9,90	8,30	9,20	11
$T2$	10,90	9,80	11,00	11
$T3$	8,50	6,90	8,90	9
$T4$	13,20	9,10	9,10	14
$T5$	11,00	9,40	10,30	11
$T6$	7,70	6,10	7,60	11
$T7$	14,00	11,80	10,10	14
$T8$	13,30	11,20	9,30	18
$T9$	9,00	8,00	8,70	9
$T10$	11,00	8,90	8,40	11

Schedule2				
	MTabu	NSGA-II	PACO	#S
$T1$	5,00	5,30	4,10	11
$T2$	22,20	19,00	11,90	23
$T3$	20,00	17,30	11,10	20
$T4$	19,00	15,80	9,80	19
$T5$	24,00	20,90	12,20	24
$T6$	27,10	22,60	13,00	28
$T7$	20,30	13,90	10,40	21
$T8$	21,80	18,90	12,10	22
$T9$	18,00	14,40	10,10	18
$T10$	32,10	26,00	14,20	33

Tcas				
	MTabu	NSGA-II	PACO	#S
$T1$	32,00	14,80	20,70	32
$T2$	30,80	10,30	18,30	32
$T3$	35,00	10,80	15,10	35
$T4$	28,00	9,70	14,00	28
$T5$	35,30	9,00	18,10	36
$T6$	25,00	7,50	15,30	25
$T7$	28,00	10,70	15,70	28
$T8$	29,10	12,80	14,90	30
$T9$	41,00	11,00	20,60	41
$T10$	36,00	9,50	17,10	36

Totinfo				
	MTabu	NSGA-II	PACO	#S
$T1$	11,10	9,20	8,80	13
$T2$	6,50	5,50	4,90	8
$T3$	6,10	5,80	5,00	7
$T4$	9,00	7,10	6,20	9
$T5$	4,00	4,00	3,50	4
$T6$	7,40	4,60	3,70	8
$T7$	11,00	7,50	6,80	13
$T8$	7,50	5,30	5,50	8
$T9$	10,60	8,20	8,00	15
$T10$	5,70	4,50	5,00	8

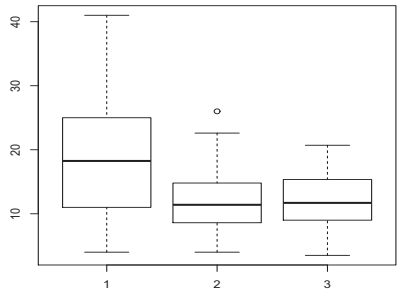

Figura 4.1: Comparação geral dos algoritmos MTabu (1), NSGA-II (2) e PACO (3)

Os resultados da Tabela 4.5 foram analisados através do Teste de Friedman a partir do Software R [37]. Esse teste considera o valor p-value de 0,05 para haver diferença estatística. Primeiramente os dados foram analisados separadamente por programa. Essa análise revelou que:

- Para os programas Printtokens e Totinfo, o algoritmo MTabu é estatisticamente

superior ao NSGA-II e ao PACO, sendo que esses dois últimos são estatisticamente equivalentes;

- Para o programa Printtokens2, o algoritmo MTabu é estatisticamente equivalente ao PACO e os dois são superiores ao NSGA-II;
- Para os programas Schedule e Tcas, o algoritmo MTabu é estatisticamente superior ao NSGA-II. NSGA-II e Paco são equivalentes;
- Para o programa Schedule2, o algoritmo MTabu é equivalente ao NSGA-II e os dois são superiores ao PACO.

Os dados também foram analisados considerando-se todos os programas juntos. Neste caso foi constatado através do mesmo teste do Software R que o algoritmo MTabu é estatisticamente superior aos outros dois. O gráfico gerado pelo Software R a partir desses dados, pode ser visualizado na Figura 4.1.

4.7 Considerações Finais

Considerando que o algoritmo MTabu obteve melhores resultados estatísticos que os demais algoritmos, suas soluções serão analisadas no próximo capítulo para a elaboração de um dos possíveis conjuntos essenciais que a abordagem possibilita, e permitir comparação com as estratégias tradicionais.

CAPÍTULO 5

CONJUNTO ESSENCIAL

Conforme dito anteriormente, o problema não apresenta somente uma solução. No processo de otimização, é encontrado um conjunto de boas soluções que formam uma Aproximação da Fronteira de Pareto (AFP). Assim, nessa aproximação são encontradas diferentes soluções não dominadas (conjuntos de operadores).

Entretanto, a fronteira é dependente do programa, mas é computacionalmente custoso executar os algoritmos de otimização todas as vezes em que o teste ocorre. Por isso, os conjuntos essenciais de operadores podem ser previamente estabelecidos da análise de diferentes fronteiras obtidas para programas semelhantes ou no mesmo domínio de aplicação.

Na utilização da abordagem multiobjetivo para estabelecer conjuntos essenciais de operadores no teste de integração apresentada no capítulo anterior, o algoritmo MTabu apresentou os melhores resultados. Ao final da execução de cada algoritmo um conjunto de boas soluções é obtido. Este conjunto de boas soluções pode ser usado de diferentes maneiras mas neste capítulo um exemplo de uso é apresentado mostrando como o testador pode acrescentar critérios na seleção dos conjuntos obtidos. É apresentado um procedimento de seleção das soluções, adaptado de Banzi et al [3]. Este procedimento é avaliado em comparação com as estratégias tradicionais.

5.1 Procedimento de Uso das Soluções Obtidas

Dado um conjunto F de soluções obtidas a partir de uma AFP fornecida por um algoritmo multiobjetivo considerando programas similares, e restrições $r_i, i = 1..B$ para cada objetivo, executam-se os seguintes passos para se determinar um conjunto essencial de operadores O'.

1. Determinar um subconjunto de soluções F' de F, que satisfaça as restrições r_i;

2. Obter uma lista de operadores OL que aparece em F';

3. Ordenar os operadores em OL de acordo com sua frequência nas soluções de F', que é o número de vezes que eles aparecem nas soluções e o número de programas em que esses operadores estão presentes;

4. Obter O' de OL de acordo com os requisitos para teste, ou recursos alocados para esta atividade.

5.2 Aplicação do Procedimento

O procedimento detalhado anteriormente foi aplicado aos programas e resultados do algoritmo MTabu descritos no capítulo anterior, para o qual estão disponíveis 10 conjuntos Fi de soluções não dominadas correspondentes aos 10 conjuntos de teste Ti, para $1 \leqslant i \leqslant 10$. F foi composto pela união dos Fi, sem entretanto remover as soluções não dominadas.

Passo 1 - Para a aplicação do procedimento, o conjunto F' foi composto selecionando-se para cada programa de cada Fi a solução que gerasse o menor número de mutantes mas que também satisfizesse a restrição de escore de mutação maior que 0,998.

Passo 2 - Estabelecimento de uma lista de operadores que aparecem nas soluções de F', considerando todos os programas.

Passo 3 - Ordenação dos operadores de acordo com a frequência, ou seja, número de vezes que cada operador aparece nas soluções de F', e com o número de diferentes programas correspondentes a estas soluções.

A Tabela 5.1 apresenta a lista de operadores OL obtida após o Passo 3, já ordenados. A coluna #Programas indica em quantos programas o operador apareceu. Considerando que o algoritmo de busca encontrou soluções que se aproximam cada vez mais da Fronteira de Pareto, os operadores que possuem maiores frequências são os que mais contribuem para um alto escore e baixo número de mutantes. O operador torna-se ainda mais relevante se aparece em vários programas. Pela mesma tabela nota-se que um operador de alta

Tabela 5.1: Frequência dos Operadores nas Fronteiras do Algoritmo MTabu

Operador	Frequência	#Programas
I-IndVarIncDec	19	4
I-IndVarAriNeg	19	2
I-IndVarRepPar	18	4
I-RetStaDel	16	5
I-DirVarRepExt	12	3
I-DirVarIncDec	12	2
II-FunCalDel	10	1
I-DirVarRepReq	8	1
I-RetStaRep	7	2
I-IndVarBitNeg	7	1
I-DirVarRepPar	7	1
I-DirVarRepLoc	6	4
I-IndVarRepLoc	5	2
I-DirVarAriNeg	5	1
I-CovAllEdg	3	3
II-ArgIncDec	3	2
I-IndVarRepExt	3	2
I-IndVarLogNeg	3	2
I-IndVarRepGlo	3	1
I-DirVarRepCon	2	2
I-DirVarLogNeg	2	2
I-DirVarBitNeg	2	1
I-IndVarRepCon	2	1
II-ArgLogNeg	1	1

frequência tende a aparecer em mais programas, fugindo do caso particular. Portanto, para a obtenção de um conjunto essencial, foi considerada a frequência do operador em primeiro lugar. Em caso de desempate, foi considerado o número de programas nos quais ele aparece.

O critério adotado para definir o Conjunto Essencial baseando-se na Tabela 5.1 foi selecionar os operadores que apresentam frequência igual ou superior a 10. Pode-se visualizar na Tabela 5.2 os 7 operadores do conjunto essencial aqui proposto que será denominado Tabu Set IM.

Tabela 5.2: Tabu Set IM

I-IndVarIncDec	Acrescenta incremento (++) e decremento (-) em variável não de interface
I-IndVarAriNeg	Acrescenta negação aritmética em variáveis não de interface
I-IndVarRepPar	Troca variáveis não de interface por elementos de P
I-RetStaDel	Elimina comando return
I-DirVarRepExt	Troca variáveis de interface por elementos de E
I-DirVarIncDec	Acrescenta incremento (++) e decremento (-) em variável de interface
II-FunCalDel	Elimina chamada de função

A Tabela 5.3 apresenta as frequências associadas aos operadores essenciais de interface de Barbosa et al [43] nas soluções encontradas pelo MTabu. Verifica-se uma semelhança

Tabela 5.3: Operadores Essenciais de Interface e Soluções do Algoritmo MTabu

Operador EI	Frequência
I-IndVarBitNeg	7
I-IndVarRepLoc	5
I-IndVarRepExt	3
I-IndVarRepGlo	3
I-DirVarBitNeg	2
I-IndVarRepReq	0
I-CovAllNod	0
II-ArgAriNeg	0

entre a relevância dos operadores nos dois casos, pois cinco dentre oito dos operadores essenciais de interface aparecem nas soluções do MTabu. Se um operador não aparece na Tabela 5.1 de frequência dos operadores do MTabu, não significa que não haja relevância para o MTabu. Essa ordem de frequência foi construída segundo restrições que escolhem uma solução com operadores com escore superior a 0,998 com o menor número de mutantes. Essa solução é retirada das não dominadas de cada conjunto de cada programa. Sendo assim, os três últimos operadores que apresentam frequência 0, ainda podem ser relevantes e aparecerem nas AFP's do MTabu.

Uma breve análise foi feita com relação aos operadores de frequência 0. O operador I-IndVarRepReq aparece no conjunto $T4$ do Printtokens com escore superior a 0,999 e com muitos mutantes. Também aparece em todos os conjuntos do Schedule2, não estando na solução com escore 0,998 que apresenta menor número de mutantes ou com escore 0,999 ou 1. O operador I-CovAllNod é encontrado em uma execução do conjunto $T9$ e na maioria das execuções do conjunto $T1$ do Schedule2. Por fim, o II-ArgAriNeg aparece no Printtokens e Printtokens2 em todos os conjuntos com poucos mutantes e baixo escore. Conclui-se que apesar da baixa frequência, ainda há semelhança entre os operadores dos conjuntos Essencial de Interface e o obtido neste trabalho.

5.2.1 Aplicação das Estratégias Tradicionais

Todas as estratégias foram aplicadas aos 10 conjuntos Ti ($1 \leqslant i \leqslant 10$) descritos na Tabela 4.3. De acordo com cada estratégia, um subconjunto de operadores $O' \subseteq O$ foi selecionado, assim como os respectivos mutantes M' gerados por O'. A partir desses

mutantes foram determinados conjuntos de dados de teste adequados $Ti' \subseteq Ti$, ou seja, conjuntos formados por dados de teste de Ti que realmente contribuíram para aumentar o escore com relação a M' na ordem de execução correspondente a T. O escore de T' para M foi então calculado. A seguir é fornecida uma breve descrição de como foram obtidos os conjuntos O' e M' para cada estratégia.

Mutação Aleatória (A10), (A20) e (A40): foram utilizadas três porcentagens (X%): 10%, 20% e 40%. Ou seja foram selecionados aleatoriamente 10%, 20% e 40% dos mutantes gerados por cada operador de mutação. Portanto $O' = O$. Neste trabalho, esta estratégia teve uma peculiaridade com relação às demais. Foi executada 10 vezes para se obter a média, retirando um pouco o peso do fator aleatório.

Mutação Seletiva (S5), (S10) e (S20): foram considerados três valores para N: 5, 10 e 20. Então, os mutantes gerados pelos $N = 5$ operadores que mais geraram mutantes foram excluídos do conjunto total M e o conjunto de operadores O' utilizado ficou com $(O - 5)$ operadores. Analogamente com $(O - 10)$ e $(O - 20)$ para $N = 10$ e $N = 20$.

Conjunto Essencial de Interface (EI): foi utilizado o conjunto determinado por Vincenzi et al [43] formado pelos seguintes operadores: II-ArgAriNeg, I-CovAllNod, I-DirVarBitNeg, I-IndVarBitNeg, I-IndVarRepGlo, I-IndVarRepExt, I-IndVarRepLoc, I-IndVarRepReq, com $O' = 8$.

Na Tabela 5.4 podem ser verificados os conjuntos M' e T' e o escore de mutação obtido pelas estratégias aplicadas aos dez conjuntos Ti do programa Printtokens. As tabelas correspondentes aos demais programas encontram-se no Apêndice A. Por exemplo, nesta tabela a estratégia Aleatória A10 gerou um conjunto M' de 487 mutantes, precisando de 28 casos de teste do conjunto $T1$ que alcançou um escore de mutação de 0,9928 considerando o conjunto total de mutantes gerados por todos os operadores.

Tabela 5.4: Resultados das Estratégias Para o Programa Printtokens

Printtokens								
		A10	A20	A40	S5	S10	S20	EI
T1	M'	487	990	1991	2473	1439	329	1248
	T'	28	35	43	53	52	18	40
	E	0,9928	0,9965	0,9981	0,9994	0,9994	0,9579	0,9972
T2	M'	487	990	1991	2473	1439	329	1248
	T'	30	41	52	67	63	24	53
	E	0,9902	0,9951	0,9981	0,9994	0,9994	0,9626	0,9978
T3	M'	487	990	1991	2473	1439	329	1248
	T'	27	35	43	53	51	20	41
	E	0,9903	0,9957	0,9981	0,9996	0,9994	0,9606	0,9974
T4	M'	487	990	1991	2473	1439	329	1248
	T'	32	38	47	55	53	23	45
	E	0,9906	0,9962	0,9984	0,9996	0,9994	0,9626	0,9992
T5	M'	487	990	1991	2473	1439	329	1248
	T'	26	36	48	56	52	17	48
	E	0,9896	0,9954	0,9979	0,9986	0,9982	0,9539	0,9972
T6	M'	487	990	1991	2473	1439	329	1248
	T'	35	42	50	58	54	23	49
	E	0,9926	0,9947	0,9984	0,9994	0,9992	0,9684	0,9984
T7	M'	487	990	1991	2473	1439	329	1248
	T'	34	40	50	57	54	23	45
	E	0,9924	0,9955	0,9982	0,9992	0,999	0,9694	0,9976
T8	M'	487	990	1991	2473	1439	329	1248
	T'	23	31	41	50	49	15	40
	E	0,9927	0,9957	0,9986	0,9996	0,9996	0,9577	0,9972
T9	M'	487	990	1991	2473	1439	329	1248
	T'	22	27	36	47	46	15	35
	E	0,993	0,9949	0,9982	0,9994	0,9994	0,96	0,9982
T10	M'	487	990	1991	2473	1439	329	1248
	T'	25	33	41	52	48	17	39
	E	0,9894	0,9955	0,9981	0,9998	0,9998	0,9553	0,9972

5.3 Comparação com as Estratégias Tradicionais

Os resultados obtidos pelas estratégias tradicionais foram avaliados de acordo com as AFP's obtidas pelo MTabu. Em se tratando de escore, a maioria das estratégias foi dominada pelos resultados obtidos pelo algoritmo MTabu cujas melhores soluções de cada conjunto de casos de teste obtiveram escore de mutação de 100%. A única estratégia que conseguiu 100% de escore de mutação foi a Seletiva 5, somente para os programas Schedule e Totinfo. O algoritmo MTabu obteve esse valor em todas as execuções, apesar de apresentar um alto número de mutantes nesses casos.

Foi comparado cada resultado das estratégias tradicionais sobre um conjunto Ti com todas as dez execuções do algoritmo MTabu sobre o mesmo conjunto Ti. Em nenhuma das execuções uma solução obtida por uma estratégia tradicional dominou uma solução

do algoritmo MTabu. Essa comparação foi realizada sem misturar soluções de execuções diferentes do algoritmo MTabu para formar um conjunto de Pareto.

5.3.1 Avaliação do Procedimento Proposto

Para avaliar o procedimento proposto, o conjunto essencial de operadores Tabu Set IM descrito na Seção 5.2 foi também aplicado aos 10 conjuntos Ti, neste caso $O' = 7$. Os resultados obtidos para cada programa são apresentados na última coluna das tabelas do Apêndice A.

A Tabela 5.5, apresenta uma média dos valores dos 10 conjuntos Ti, de todas as estratégias. Verifica-se que a estratégia $S5$ apresenta alto escore para todos os programas, com exceção do Tcas, mas o número de mutantes também é elevado. As estratégias $S20$ e $A10$ apresentam os menores números de mutantes sendo as que mais reduzem o custo computacional, mas também são as que apresentam menor escore. O Tabu Set IM apresenta um escore superior a 0,997, com o maior escore para Printtokens2 e Tcas e apresenta um número de mutantes intermediário que reduz satisfatoriamente o custo computacional.

Tabela 5.5: Média de Resultados de Estratégias e do Tabu Set IM

	Printtokens			Printtokens2			Schedule		
	M'	Escore	T'	M'	Escore	T'	M'	Escore	T'
A10	487	0,9914	28,2	412	0,9941	21,2	145	0,9921	6,9
A20	990	0,9955	35,8	840	0,9976	23,3	304	0,9973	8,8
A40	1991	0,9982	45,1	1690	0,9988	26,4	621	0,9989	10,3
S5	2473	0,9994	54,8	1755	0,9948	23,2	681	0,9997	12,8
S10	1439	0,9993	52,2	885	0,9927	20,3	378	0,9984	8,4
S20	329	0,9608	19,5	117	0,9513	9,7	106	0,9959	6,8
EI	1248	0,9977	43,5	1093	0,9931	20,8	398	0,9987	10,3
Tabu Set IM	1168	0,9972	43,8	1001	0,9994	28,3	662	0,9989	10,1

	Schedule2			Tcas			Totinfo		
	M'	Escore	T'	M'	Escore	T'	M'	Escore	T'
A10	219	0,9838	15,2	141	0,9480	30,6	185	0,9970	8,8
A20	454	0,9937	20,1	293	0,9749	41,1	377	0,9991	10,7
A40	916	0,9982	24,4	594	0,9886	53,1	772	0,9995	12,5
S5	1370	0,9991	28,3	603	0,9873	47,4	924	0,9999	14,5
S10	783	0,9928	21,9	298	0,9703	35,9	470	0,9862	11,2
S20	218	0,9898	18,2	22	0,6237	8,4	83	0,9838	7,6
EI	553	0,9983	25,2	515	0,9466	34,6	634	0,9992	12,1
Tabu Set IM	648	0,9978	20,7	534	0,9929	60,8	391	0,9994	11,7

Na Tabela 5.6 verifica-se a média geral das estratégias considerando todos os progra-

Tabela 5.6: Média dos Resultados de Estratégias Para Todos os Programas

	M'	Escore	T'	O'	Redução
A10	264,8	0,9844	18,5	O	90,4%
A20	543,0	0,9930	23,3	O	80,4%
A40	1097,3	0,9971	28,6	O	60,4%
S5	1301,0	0,9967	30,2	O - 5	53,1%
S10	708,8	0,9899	25,0	O - 10	74,4%
S20	145,8	0,9176	11,7	O - 20	94,7%
EI	740,2	0,9889	24,4	8	73,3%
Tabu Set IM	734,0	0,9976	29,2	7	73,5%

mas. As que mais reduzem custo são $S20$ e $A10$.

A estratégia $A40$, obteve um escore bastante alto de 0,9971, que ainda é inferior ao valor do Tabu Set IM, mas com o prejuízo de ter gerado um número de mutantes bastante superior de 1097,3, que é o segundo maior das estratégias avaliadas. A estratégia $A40$ obteve o terceiro valor mais alto de casos de teste, 28,6.

Pode-se notar pela tabela que os valores são bastante proporcionais. Isso quer dizer que, se o escore aumenta o número de mutantes e ou o número de casos de teste também cresce. A abordagem envolvendo busca multiobjetivo deu certa flexibilidade ao Tabu Set IM, pois ele não se comporta completamente de acordo com essa proporcionalidade. Isso pode ser visto analisando os valores obtidos: com um número intermediário de mutantes o Tabu Set IM consegue alto escore, apesar de ter o segundo valor mais alto para o conjunto T' de 29,2. Percebe-se que o conjunto permite contornar a proporção, pois para o mais alto escore era esperado um valor alto de mutantes.

A estratégia $A10$ é bastante influenciada pela proporção. Apresenta a segunda mais alta redução de 90,4% e o segundo menor número de casos de testes, 18,5, contudo obtém o segundo valor mais baixo para o escore, 0,9844.

A menor redução, de 94,7%, e o menor número de casos de teste, 11,7, foram apresentados pela estratégia $S20$. Entretanto o escore foi o mais baixo, 0,9176. A estratégia $A10$ que obteve os resultados mais próximos da estratégia $S20$ obteve escore bastante superior com um número não tão maior de mutantes.

A estratégia $S5$ apresenta a menor redução, 53,1%, o terceiro maior escore 0,9967 e o número mais alto de casos de teste, 30,2. Já a estratégia $S10$ apresentou valores

intermediários em todos os sentidos, não se destacando significativamente.

O Conjunto Essencial de Interface (EI) apresentou resultados intermediários. Com relação ao Tabu Set IM, necessitou de menos casos de teste para um número parecido de mutantes, mas obteve menor escore.

Com 543 mutantes, a estratégia $A20$ obteve redução de 80,4%, a terceira maior. Também obteve o terceiro menor número de casos de teste, 23,3. Obteve o quarto maior escore, de 0,9930. Essa estratégia foi a que mais se equiparou ao conjunto Tabu Set IM sendo inferior em escore de mutação, apesar da redução não apresentar diferenças tão grandes.

O Tabu Set IM supera a estratégia EI nos dois objetivos, apresentando escore geral 0,9976, maior do que todas as estratégias, utilizando um número aceitável de mutantes e apresentando um valor viável de custo benefício.

5.4 Comparação com Teste de Unidade

A atividade de teste requer grande parte do tempo total do projeto de software [36]. Uma de suas fases é o projeto de casos de teste que é relativamente demorado, precisa de uma dedicação de esforço e de tempo grandes, principalmente quando se trata de técnicas que utilizam a estrutura interna do software. Nesses casos, elaborar casos de teste se torna uma atividade mais complexa, devido ao nível de detalhe para aumentar a cobertura dos elementos requeridos pelo critério de teste.

Por isso, se fosse possível utilizar os mesmos casos de teste utilizados no teste de unidade para o teste de integração, seria possível obter uma contribuição significativa para diminuição do esforço de teste. Nesse sentido, no contexto de Análise de Mutantes, Vincenzi et al [43] realizaram um experimento e concluíram que começar os testes de integração utilizando os casos de teste de unidade permite começar o teste com um escore significativo, reduzindo o esforço para elaboração de novos casos de teste específicos para os operadores de mutação do teste de integração.

Nesta seção é descrito um experimento elaborado com o mesmo objetivo do trabalho de Vincenzi et al [43], mas utilizando os conjuntos Tabu Set de Banzi et al [3] composto de

operadores de mutação do teste de unidade e o Tabu Set IM. Na Tabela 5.7 são descritos os valores dessa avaliação para cada programa, separadamente. Na Tabela 5.8 são descritas as médias dos resultados dos dez conjuntos Tu_i e Tin_i.

Mu é o conjunto de todos os mutantes gerados pelo conjunto Tabu Set de operadores de mutação de unidade da Proteum. Tu é o conjunto de casos de testes Mu-Adequado. Min é o conjunto de todos os mutantes gerados pelo conjunto Tabu Set IM de operadores de mutação de integração da Proteum IM. Tin é o conjunto de casos de testes Min-Adequado. As colunas $\#Tu$ e $\#Tin$ mostram quantos casos de testes possui o conjunto correspondente.

O experimento foi repetido para os 10 conjuntos de casos de teste de unidade Tu_i $Mu-Adequados$ e para 10 conjuntos de casos de teste de integração Tin_i Min-Adequados, sendo $1 \leq i \leq 10$.

Para cada programa, foram aplicados os casos de teste Tu_i $(1 \leq i \leq 10)$ no conjunto de mutantes Min. O resultado está descrito para cada programa na coluna "Unidade na Integração"da Tabela 5.7. Também, para cada programa, foram aplicados os casos de teste Tin_i $(1 \leq i \leq 10)$ no conjunto de mutantes Mu. O resultado está descrito para cada programa na coluna "Integração na Unidade".

O conjunto Tabu Set de unidade possui um maior número operadores e também maior número de mutantes que o conjunto Tabu Set IM. Tabu Set possui $O' = 12$ e Tabu Set IM possui $O' = 7$.

Pode-se verificar que os resultados mostrados nesta seção são semelhantes aos de Vincenzi et al [43] et al. Pela Tabela 5.8 verifica-se que o escore mais baixo com que se pode começar o teste de integração tendo utilizado os conjuntos de casos de teste Tu é, em média, para este experimento, de 0,9558. Para três programas esse valor foi superior a 0,99. Esses valores são superiores ao obtido por Vincenzi et al [43] que obtiveram um valor inicial para o teste de integração de 0,91956. Observando a aplicação dos casos de teste de integração no teste de unidade, verifica-se que o mínimo escore obtido foi, em média, de 0,9243. Para dois programas esse valor foi superior a 0,99. Dessa forma conclui-se que a estratégia proposta por [43] é viável e apresenta resultados para a redução de custo do critério Análise de Mutantes, também no contexto da estratégia multiobjetivo.

Tabela 5.7: Comparação de Escore entre os Casos de Teste de Unidade e Integração

Printtokens		Escore	Min	$\#Tu$
Unidade na Integração	Tu_1	0,9820	1168	34
	Tu_2	0,9794	1168	27
	Tu_3	0,9820	1168	29
	Tu_4	0,9863	1168	38
	Tu_5	0,9828	1168	37
	Tu_6	0,9828	1168	46
	Tu_7	0,9785	1168	38
	Tu_8	0,9811	1168	40
	Tu_9	0,9820	1168	30
	Tu_{10}	0,9820	1168	38
Printtokens		Escore	Mu	$\#Tin$
Integração na Unidade	Tin_1	0,9944	1432	51
	Tin_2	0,9944	1432	41
	Tin_3	0,9965	1432	51
	Tin_4	0,9944	1432	45
	Tin_5	0,9944	1432	46
	Tin_6	0,9944	1432	44
	Tin_7	0,9944	1432	38
	Tin_8	0,9937	1432	46
	Tin_9	0,9944	1432	43
	Tin_{10}	0,9965	1432	42

Printtokens2		Escore	Min	$\#Tu$
Unidade na Integração	Tu_1	0,9790	1001	26
	Tu_2	0,9750	1001	30
	Tu_3	0,9700	1001	22
	Tu_4	0,9800	1001	27
	Tu_5	0,9790	1001	29
	Tu_6	0,9740	1001	25
	Tu_7	0,9850	1001	30
	Tu_8	0,9760	1001	19
	Tu_9	0,9790	1001	32
	Tu_{10}	0,9760	1001	28
Printtokens2		Escore	Mu	$\#Tin$
Integração na Unidade	Tin_1	0,9969	1325	32
	Tin_2	0,9947	1325	32
	Tin_3	0,9962	1325	21
	Tin_4	0,9924	1325	24
	Tin_5	0,9909	1325	27
	Tin_6	0,9947	1325	32
	Tin_7	0,9924	1325	30
	Tin_8	0,9939	1325	26
	Tin_9	0,9886	1325	27
	Tin_{10}	0,9932	1325	29

Schedule		Escore	Min	$\#Tu$
Unidade na Integração	Tu_1	0,9969	662	30
	Tu_2	0,9969	662	29
	Tu_3	0,9939	662	32
	Tu_4	0,9954	662	32
	Tu_5	0,9969	662	31
	Tu_6	0,9697	662	31
	Tu_7	0,9969	662	28
	Tu_8	0,9939	662	28
	Tu_9	0,9969	662	30
	Tu_{10}	0,9954	662	31
Schedule		Escore	Mu	$\#Tin$
Integração na Unidade	Tin_1	0,9293	962	13
	Tin_2	0,9189	962	9
	Tin_3	0,9209	962	10
	Tin_4	0,9251	962	9
	Tin_5	0,9241	962	6
	Tin_6	0,9386	962	7
	Tin_7	0,9251	962	10
	Tin_8	0,9282	962	15
	Tin_9	0,9178	962	7
	Tin_{10}	0,9147	962	7

Schedule2		Escore	Min	$\#Tu$
Unidade na Integração	Tu_1	0,9922	648	34
	Tu_2	0,9907	648	28
	Tu_3	0,9922	648	23
	Tu_4	0,9969	648	29
	Tu_5	0,9922	648	32
	Tu_6	0,9969	648	28
	Tu_7	0,9922	648	34
	Tu_8	0,9953	648	26
	Tu_9	0,9922	648	27
	Tu_{10}	0,9922	648	30
Schedule2		Escore	Mu	$\#Tin$
Integração na Unidade	Tin_1	0,9583	1057	14
	Tin_2	0,9640	1057	18
	Tin_3	0,9602	1057	22
	Tin_4	0,9678	1057	20
	Tin_5	0,9716	1057	21
	Tin_6	0,9668	1057	19
	Tin_7	0,9659	1057	19
	Tin_8	0,9678	1057	18
	Tin_9	0,9631	1057	17
	Tin_{10}	0,9649	1057	17

Tcas		Escore	Min	$\#Tu$
Unidade na Integração	Tu_1	0,9382	534	65
	Tu_2	0,9438	534	63
	Tu_3	0,9644	534	70
	Tu_4	0,9606	534	64
	Tu_5	0,9569	534	72
	Tu_6	0,9625	534	73
	Tu_7	0,9569	534	70
	Tu_8	0,9662	534	70
	Tu_9	0,9588	534	73
	Tu_{10}	0,9494	534	63
Tcas		Escore	Mu	$\#Tin$
Integração na Unidade	Tin_1	0,9710	1416	64
	Tin_2	0,9759	1416	55
	Tin_3	0,9738	1416	61
	Tin_4	0,9759	1416	57
	Tin_5	0,9653	1416	63
	Tin_6	0,9766	1416	61
	Tin_7	0,9809	1416	65
	Tin_8	0,9766	1416	62
	Tin_9	0,9682	1416	59
	Tin_{10}	0,9668	1416	57

Totinfo		Escore	Min	$\#Tu$
Unidade na Integração	Tu_1	0,9974	391	29
	Tu_2	0,9974	391	34
	Tu_3	0,9974	391	31
	Tu_4	0,9974	391	31
	Tu_5	0,9974	391	28
	Tu_6	0,9974	391	36
	Tu_7	0,9974	391	31
	Tu_8	0,9974	391	31
	Tu_9	1,0	391	33
	Tu_{10}	0,9974	391	31
Totinfo		Escore	Mu	$\#Tin$
Integração na Unidade	Tin_1	0,9684	2729	13
	Tin_2	0,9728	2729	11
	Tin_3	0,9739	2729	10
	Tin_4	0,9747	2729	12
	Tin_5	0,9783	2729	13
	Tin_6	0,9846	2729	13
	Tin_7	0,9827	2729	8
	Tin_8	0,9769	2729	12
	Tin_9	0,9611	2729	12
	Tin_{10}	0,9406	2729	10

Tabela 5.8: Médias entre os Casos de Teste de Unidade e Integração

Unidade na Integração	**Escore**	**Min**	**#Tu**
Printtokens	0,9819	1168,0	35,7
Printtokens2	0,9773	1001,0	26,8
Schedule	0,9933	662,0	30,2
Schedule2	0,9933	648,0	29,1
Tcas	0,9558	534,0	68,3
Totinfo	0,9976	391,0	31,5

Integração na Unidade	**Escore**	**Mu**	**#Tin**
Printtokens	0,9947	1432,0	44,7
Printtokens2	0,9934	1325,0	28,0
Schedule	0,9243	962,0	9,3
Schedule2	0,9650	1057,0	18,5
Tcas	0,9731	1416,0	60,4
Totinfo	0,9714	2729,0	11,4

5.5 Considerações Finais

Neste capítulo foi aplicado o procedimento para obtenção do conjunto Tabu Set IM. Também foram comparadas as estratégias tradicionais com a abordagem proposta. O conjunto Tabu Set IM obteve os seguintes valores de média para mutantes, escore de mutação e casos de teste, respectivamente: 734, 0,9976 e 29,2. O valor de mutantes é intermediário, mas o escore obtido é o mais alto. Os valores superam as outras estratégias tendo semelhança com a $A20$ que tem os seguintes valores: 543, 0,9930 e 23,3 para mutantes, escore de mutação e casos de teste, respectivamente.

Conclui-se com o experimento de avaliação descrito neste capítulo que a abordagem de otimização multiobjetivo é uma alternativa bastante interessante para reduzir o custo do critério Análise de Mutantes, porque mantém ótimos resultados e ainda permite considerar outros fatores (objetivos) que influem no custo do critério.

CAPÍTULO 6

CONCLUSÕES

Sendo o critério de teste Análise de Mutantes eficaz em revelar defeitos é importante reduzir seu custo computacional. Como uma alternativa às estratégias tradicionais de redução desse custo é possível aplicar metaheurísticas para selecionar operadores de mutação que mantém alto escore de mutação e baixo número de mutantes gerados. O presente trabalho teve como objetivo explorar o uso da abordagem baseada em metaheurísticas multiobjetivos para reduzir o custo da aplicação do critério Análise de Mutantes no teste de integração.

A estratégia foi implementada com três algoritmos diferentes: NSGA-II, PACO e MTabu. Para avaliar e comparar os algoritmos foi conduzido um estudo com os operadores de mutação de interface da Proteum IM.

O estudo utilizou seis programas reais e dez conjuntos de casos de teste para cada um deles correspondentes a dez ordens de execuções de dados de teste geradas aleatoriamente. Cada conjunto de caso de testes de cada programa foi espaço de busca de cada algoritmo. Os resultados dos três algoritmos foram comparados e o MTabu obteve resultados melhores. Isso foi verificado utilizando-se uma comparação estatística através do teste de Friedman disponibilizado pelo Software R[37]. De forma geral, o MTabu foi superior aos algoritmos NSGA-II e PACO. Por isso, seus resultados foram selecionados para as análises de operadores essenciais. As soluções do MTabu foram submetidas ao procedimento para obtenção de conjuntos essenciais utilizado em Banzi et al [3]. Como mais de um conjunto pode ser obtido com a abordagem multiobjetivo, optou-se pela geração de somente um conjunto com as soluções de menor número de mutantes e escore de mutação superior a 0,998 para permitir a comparação com estratégias tradicionais. A esse conjunto deu-se o nome de Tabu Set IM.

As dificuldades encontradas para realização do trabalho foram gerar os dados de en-

trada para as metaheurísticas, balancear os parâmetros dos algoritmos e encontrar uma forma justa de comparar os três algoritmos diferentes.

Os resultados das estratégias tradicionais foram comparados com as soluções das AFP's do MTabu e com o conjunto Tabu Set IM. Nos dois casos o MTabu apresentou resultados superiores. Na comparação com as estratégias, o Tabu Set IM obteve em média 734, 0,9976 e 29,2 respectivamente para número de mutantes, escore de mutação e casos de teste. O valor de mutantes obtido é intermediário, o escore é o mais alto obtido entre todas as estratégias. Os valores superam todas as outras estratégias apresentando o melhor custo benefício.

Em resumo, o trabalho contribuiu com uma abordagem multiobjetivo para os testes de integração, definindo um conjunto essencial e uma lista ordenada com os operadores mais relevantes encontrados. Mostra os benefícios da abordagem multiobjetivo que apresenta uma exploração global do problema, considerando diversas combinações de soluções. Devido à flexibilidade da abordagem baseada em algoritmos multiobjetivos, outros conjuntos podem ser obtidos, alterando-se restrições aplicadas. Pode-se aplicar restrições que utilizem soluções das AFP's que tenham baixo número de mutantes. Dessa forma, o conjunto obtido apresentará alta redução e baixo número de casos de teste, mas consequentemente não apresentará alto escore. Também é possível escolher as soluções das AFP's que apresentem alto escore. Todas as AFP's possuem uma solução com escore de 100%, mas usar restrições que escolham essas soluções para obter um conjunto essencial deixará o número de mutantes e de casos de teste extremamente elevados. Entre os dois extremos há muitos conjuntos que podem ser estabelecidos.

Foi realizada uma comparação com a Estratégia Essencial de Teste, realizada por Vincenzi et al [43] na qual se reforçou o fato de que os casos de teste de unidade podem ser utilizados para iniciar os testes de integração cumprindo os requisitos mínimos de teste ou chegando a um escore bastante satisfatório próximo a 0,99.

Sendo assim conclui-se que a abordagem aqui aplicada de fato possibilita uma grande variedade de opções para a redução do custo do critério Análise de Mutantes.

6.1 Trabalhos Futuros

Existem muitas possibilidades de continuidade para este trabalho. Por exemplo pode-se explorar outros possíveis conjuntos essenciais obtidos analisando novas possibilidades de aumento de escore de mutação e redução de número de mutantes e outras restrições associadas a atividade de teste.

Novos experimentos podem ser conduzidos com outros programas. O tempo necessário para aplicação da abordagem não é alto. Para isso seria necessário obter a entrada para as metaheurísticas e configurar seus parâmetros para o caso concreto. Há ainda a possibilidade da aplicação da abordagem em outros contextos como teste de software orientado a objetos utilizando a ferramenta Mujava ou teste de software orientado a aspectos. Além disso, podem ser realizados outros experimentos utilizando outros objetivos como tempo de execução dos programas e número de casos de teste.

Neste trabalho foram explorados três algoritmos de busca diferentes, mas outros algoritmos existentes na literatura podem gerar resultados ainda melhores. É possível implementar novas metaheurísticas e conduzir novos experimentos de avaliação.

Um importante ponto a ser também analisado em trabalhos futuros é a eficácia dos dados de teste associados aos operadores obtidos. Comparações entre as estratégias deverão ser realizadas. A eficácia poderá ser adicionada ao algoritmo em termos do número de defeitos revelados como um terceiro objetivo.

BIBLIOGRAFIA

[1] A. T. Acree, T. A. Budd, R. A. DeMillo, R. J. Lipton, e F. G. Sayward. *Mutation analysis*. School of Information and Computer Science, Georgia Institute of Technology, 1979.

[2] A. S. Banzi, D. Antunes, G. Pinheiro, J. C. G. Árias, R. Hornun, R. Cabral, R. Friedemann, S. R. Vergilio, e T. Nobre. Avaliando Diferentes Estratégias de Redução de Custo do Teste de Mutação. 2009. Workshop de Teste e Tolerância a Falhas.

[3] A. S. Banzi, T. Nobre, G. B. Pinheiro, J. C. G. Árias, A. T. Pozo, e S. R. Vergilio. Selecting Mutation Operators with a Multiobjective Approach. 2010. Submetido para Journal of System and Software.

[4] E. F. Barbosa. *Uma contribuição para determinação de um conjunto essencial de operadores de mutação no teste de programas C.* Biblioteca Digital de Teses e Dissertações da USP, 1998.

[5] E. F. Barbosa, J. C. Maldonado, e A. M. R. Vincenzi. Toward the determination of sufficient mutant operators for C. *Software Testing, Verification and Reliability*, 11(2):113–136, 2001.

[6] A. Baykasoglu, S. Owen, e N. Gindy. A Taboo Search Based Approach to Find the Pareto Optimal Set in Multiple Objective Optimisation. *Journal of Engineering Optimization*, páginas 731–748, 1999.

[7] T. A. Budd. *Mutation Analysis: Ideas, Example, Problems and Prospects, chapter Computer Program Testing.* North-Holand Publishing Company, 1981.

[8] E. K. Burke e G. Kendall, editors. *Search Methodologies Introductory Tutorials in Optimization and Decision Support Techniques.* Springer, 2005.

[9] B. J. Choi, R. A. DeMillo, E. W. Krauser, R. J. Martin, A. P. Mathur, A. J. Offutt, H. Pan, e E. H. Spafford. The Mothra Tool Set. *Proceedings of the 22nd Annual Hawaii International Conference on System Sciences (HICSS'22)*, páginas 275–284, 3-6 Janeiro de 1989.

[10] C. A. Coello, G. B. Lamont, e D. A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Springer, second edition, 2007.

[11] K. Deb, S. Agrawal, A. Pratap, e T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. *Lecture Notes in Computer Science*, páginas 849–858, 2000.

[12] M. E. Delamaro. *Mutação de Interface: Um Critério de Adequação Interprocedimental para o Teste de Integração*. Tese de doutorado, IFSC/USP, São Carlos, SP, Junho de 1997.

[13] M. E. Delamaro e J. C. Maldonado. Proteum–a tool for the assessment of test adequacy for C programs. *Proceedings of the Conference on Performability in Computing Systems (PCS 96)*, páginas 79–95, 1996.

[14] M. E. Delamaro e A. M. Rizzo. Proteum/IM 2.0: An integrated mutation testing environment. *Mutation Testing for the New Century*, páginas 91, 2001.

[15] R. DeMillo, R. J. Lipton, e F. G. Sayward. Hints on test data selection: Help for the practicing programmer. *IEEE Software*, 11:34–41, 2001.

[16] R. A. DeMillo, D. S. Guindi, W. M. McCracken, A. J. Offutt, e K. N. King. An extended overview of the Mothra software testing environment. *Software Testing, Verification, and Analysis, 1988., Proceedings of the Second Workshop on*, páginas 142–151, 1988.

[17] K. Doerner, W. J. Gutjahr, R. F. Hartl, C. Strauss, e C. Stummer. Pareto Ant Colony Optimization: A Metaheuristic Approach to Multiobjective Portfolio Selection. *Annals of Operation Research*, (131):79–99, 2004.

[18] M. Dorigo e K. Socha. *An Introduction to Ant Colony Optimization*. Number TR/IRIDIA/2006-010. Technical Report - IRIDIA, Abril de 2006.

[19] F. Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers & Operations Research*, páginas 13:533–549, 1986.

[20] F. Glover e M. Laguna. Tabu Search. Kluwer, Dordrecht, 1997.

[21] S. Hussain. *Mutation Clustering*. Tese de Doutorado, Master Thesis, Kings's College, London, UK, 2008.

[22] M. Hutchins, H. Foster, T. Goradia, e T. Ostrand. Experiments of the effectiveness of data data flow and control flow-based w-test adequacy criteria. *16th International Conference on Software Engineering (ICSE 1994)*, páginas 191–200, 1994.

[23] C. Ji, Z. Chen, B. Xu, e Z. Zhao. A Novel Method of Mutation Clustering Based on Domain Analysis. *the 21st International Conference on Software Engineering Knowledge Engineering (SEKE'2009)*, 2009.

[24] Y. Jia e M. Harman. Higher Order Mutation Testing. *Information and Software Technology*, 51(10):1379–1393, Outubro de 2009.

[25] Y. Ma, A. J. Offutt, e Y. R. Kwon. MuJava: an automated class mutation system: Research Articles. *Softw. Test. Verif. Reliab.*, 15:97–133, Junho de 2005.

[26] J. C. Maldonado. *Critérios Potenciais Usos: Uma Contribuição ao Teste Estrutural de Software*. Tese de Doutorado, DCA/FEE/Unicamp, Julho de 1991.

[27] A. P. Mathur. Performance, effectiveness, and reliability issues in software testing. *the Fifteenth Annual International Computer Software and Applications Conference, COMPSAC'91*, páginas 604–605, 1991.

[28] G. J. Myers. *Art of Software Testing*. John Wiley & Sons, Inc., New York, NY, USA, 1979.

[29] A. S. Namin, J. H. Andrews, e D. Murdoch. Sufficient Mutation Operators for Measuring Test Effectiveness. *ICSE'2008*, páginas 351–360, 2008.

[30] A. J. Offutt. The coupling effect: fact or fiction. *SIGSOFT Softw. Eng. Notes*, 14:131–140, Novembro de 1989.

[31] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, e C. Zapf. An experimental determination of sufficient mutant operators. *ACM Transactions on Software Engineering and Methodology*, 2(5):99–118, 1996.

[32] A. J. Offutt, G. Rothermel, e C. Zapf. An Experimental Evaluation of Selective Mutation. *ICSE*, páginas 100–107, 1993.

[33] V. Pareto. *Manuel D'Économie Politique*. Ams Pr, Paris., 1927.

[34] J. M. Pasia, R.F. Hart, e K. F. Doerner. Solving a Bi-objective Flowshop Scheduling Problem by Pareto-Ant Colony Optimization. *Lecture Notes in Computer Science*, páginas 294–305, Agosto de 2006.

[35] M. Polo, M. Piattini, e I. García-Rodríguez. Decreasing the cost of mutation testing with second-order mutants. *Software, Testing, Verification and Reliability*, 19(2):111–131, Junho de 2009.

[36] R. S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Higher Education, 5th edition, 2001.

[37] R Project. Software R. http://www.r-project.org.

[38] S. Rapps e E. J. Weyuker. Selecting Software Test Data Using Data Flow Information. *IEEE Trans. Softw. Eng.*, 11:367–375, Abril de 1985.

[39] C. C. Ribeiro. Metaheuristics and Applications. In Advanced School on Artificial Intelligence, Estoril, Portugal, 1996.

[40] M. J. F. Souza. Inteligência Computacional para Otimização. Departamento de Computação, Instituto de Ciências Exatas e Biológicas, UFOP, 2009.

[41] S. R. S. Souza. *Avaliação do custo e eficácia do critério análise de mutantes na atividade de teste de programas.* Tese de Doutorado, ICMC-USP - São Carlos - SP, Junho de 1996.

[42] R. Untch, J. Offutt, e M. J. Harraold. Mutation analysis using program schemata. *International Symposium on Software Testing, and Analysis*, páginas 139–148, 1993.

[43] A. M. R. Vincenzi, J. C. Maldonado, E. F. Barbosa, e M. E. Delamaro. Operadores Essenciais de Interface: Um Estudo de Caso. *XIII Simpósio Brasileiro de Engenharia de Software (SBES 99)*, páginas 373–391, Florianópolis, SC, Outubro de 1999.

[44] W. E. Wong, M. E. Delamaro, J. C. Maldonado, e A. P. Mathur. Constrained mutation in C programs. *Proceedings of the 8th Brazilian Symposium on Software Engineering*, páginas 439–452, 1994.

[45] W. E. Wong, J. C. Maldonado, M. E. Delamaro, e S. R. S. Souza. A comparison of selective mutation in C and Fortran. *Workshop of the Project: Validation and Test of Operation Systems*, páginas 71–84, Brazil, 1997.

APÊNDICE A

RESULTADOS DAS ESTRATÉGIAS TRADICIONAIS E TABU SET IM

A seguir são apresentadas as tabelas de cada programa que comparam as estratégias tradicionais e o resultado obtido pelo conjunto Tabu Set IM para cada conjunto de casos de teste Ti ($1 \leq i \leq 10$). A média da Tabela 5.5 foi realizada a partir dos valores aqui apresentados.

Tabela A.1: Resultados das Estratégias Para o Programa Printtokens

Printtokens									
		A10	A20	A40	S5	S10	S20	EI	Tabu Set IM
T1	M'	487	990	1991	2473	1439	329	1248	1168
	T'	28	35	43	53	52	18	40	43
	E	0,9928	0,9965	0,9981	0,9994	0,9994	0,9579	0,9972	0,9986
T2	M'	487	990	1991	2473	1439	329	1248	1168
	T'	30	41	52	67	63	24	53	50
	E	0,9902	0,9951	0,9981	0,9994	0,9994	0,9626	0,9978	0,9946
T3	M'	487	990	1991	2473	1439	329	1248	1168
	T'	27	35	43	53	51	20	41	40
	E	0,9903	0,9957	0,9981	0,9996	0,9994	0,9606	0,9974	0,9978
T4	M'	487	990	1991	2473	1439	329	1248	1168
	T'	32	38	47	55	53	23	45	43
	E	0,9906	0,9962	0,9984	0,9996	0,9994	0,9626	0,9992	0,9972
T5	M'	487	990	1991	2473	1439	329	1248	1168
	T'	26	36	48	56	52	17	48	42
	E	0,9896	0,9954	0,9979	0,9986	0,9982	0,9539	0,9972	0,9970
T6	M'	487	990	1991	2473	1439	329	1248	1168
	T'	35	42	50	58	54	23	49	51
	E	0,9926	0,9947	0,9984	0,9994	0,9992	0,9684	0,9984	0,9994
T7	M'	487	990	1991	2473	1439	329	1248	1168
	T'	34	40	50	57	54	23	45	53
	E	0,9924	0,9955	0,9982	0,9992	0,999	0,9694	0,9976	0,9994
T8	M'	487	990	1991	2473	1439	329	1248	1168
	T'	23	31	41	50	49	15	40	39
	E	0,9927	0,9957	0,9986	0,9996	0,9996	0,9577	0,9972	0,9954
T9	M'	487	990	1991	2473	1439	329	1248	1168
	T'	22	27	36	47	46	15	35	38
	E	0,993	0,9949	0,9982	0,9994	0,9994	0,96	0,9982	0,9982
T10	M'	487	990	1991	2473	1439	329	1248	1168
	T'	25	33	41	52	48	17	39	39
	E	0,9894	0,9955	0,9981	0,9998	0,9998	0,9553	0,9972	0,9942

Tabela A.2: Resultados das Estratégias Para o Programa Printtokens2

Printtokens2									
		A10	A20	A40	S5	S10	S20	EI	Tabu Set IM
T1	M'	412	840	1690	1755	885	117	1093	1001
	T'	23	26	27	22	20	9	21	28
	E	0,9942	0,9984	0,9990	0,9885	0,9871	0,9434	0,9906	0,9993
T2	M'	412	840	1690	1755	885	117	1093	1001
	T'	21	24	28	26	23	9	22	30
	E	0,9928	0,9972	0,9990	0,9986	0,9951	0,9490	0,9953	0,9991
T3	M'	412	840	1690	1755	885	117	1093	1001
	T'	18	19	23	21	16	7	18	25
	E	0,9941	0,9969	0,9992	0,9979	0,9880	0,9293	0,9892	0,9993
T4	M'	412	840	1690	1755	885	117	1093	1001
	T'	21	22	26	22	20	11	19	27
	E	0,9924	0,9974	0,9987	0,9901	0,9892	0,9572	0,9828	0,9993
T5	M'	412	840	1690	1755	885	117	1093	1001
	T'	13	15	19	15	14	7	15	18
	E	0,9963	0,9979	0,9990	0,9977	0,9972	0,9709	0,9974	0,9995
T6	M'	412	840	1690	1755	885	117	1093	1001
	T'	22	25	27	24	19	11	20	30
	E	0,9957	0,9986	0,9989	0,9984	0,9969	0,9659	0,9969	0,9998
T7	M'	412	840	1690	1755	885	117	1093	1001
	T'	22	24	27	24	22	10	21	30
	E	0,9927	0,9965	0,9987	0,9906	0,9897	0,9401	0,9901	0,9993
T8	M'	412	840	1690	1755	885	117	1093	1001
	T'	27	30	32	28	26	11	26	36
	E	0,9941	0,9977	0,9985	0,9927	0,9922	0,9523	0,9951	0,9998
T9	M'	412	840	1690	1755	885	117	1093	1001
	T'	23	24	28	27	23	12	23	30
	E	0,9955	0,9982	0,9988	0,9986	0,9977	0,9532	0,9979	0,9995
T10	M'	412	840	1690	1755	885	117	1093	1001
	T'	22	24	27	23	20	10	23	29
	E	0,9927	0,9971	0,9986	0,9953	0,9944	0,9516	0,9953	0,9993

Tabela A.3: Resultados das Estratégias Para o Programa Schedule

Schedule									
		A10	A20	A40	S5	S10	S20	EI	Tabu Set IM
T1	M'	145	304	621	681	378	106	398	662
	T'	8	9	11	13	8	8	11	10
	E	0,9949	0,9991	0,9997	1	0,9975	0,9975	0,9994	0,9994
T2	M'	145	304	621	681	378	106	398	662
	T'	6	8	9	12	8	7	10	8
	E	0,9938	0,9986	0,9993	1	0,9994	0,9987	0,9994	0,9994
T3	M'	145	304	621	681	378	106	398	662
	T'	4	6	7	9	5	4	6	6
	E	0,9946	0,9970	0,9986	0,9994	0,9975	0,9962	0,9962	0,9981
T4	M'	145	304	621	681	378	106	398	662
	T'	9	12	15	18	12	8	14	14
	E	0,9920	0,9944	0,9987	1	0,9981	0,9968	0,9994	0,9981
T5	M'	145	304	621	681	378	106	398	662
	T'	8	10	12	15	11	9	13	11
	E	0,9908	0,9974	0,9990	1	0,9987	0,9968	0,9994	0,9987
T6	M'	145	304	621	681	378	106	398	662
	T'	6	7	7	9	7	6	7	8
	E	0,9943	0,9985	0,9992	1	0,9994	0,9981	0,9994	0,9994
T7	M'	145	304	621	681	378	106	398	662
	T'	6	9	10	12	8	7	10	10
	E	0,9892	0,9978	0,9987	1	0,9981	0,9981	0,9994	0,9981
T8	M'	145	304	621	681	378	106	398	662
	T'	10	12	15	16	8	8	13	16
	E	0,9899	0,9960	0,9988	0,9981	0,9962	0,9962	0,9975	0,9994
T9	M'	145	304	621	681	378	106	398	662
	T'	6	7	8	11	8	5	9	9
	E	0,9914	0,9977	0,9991	1	0,9994	0,9855	0,9994	0,9994
T10	M'	145	304	621	681	378	106	398	662
	T'	6	8	9	13	9	6	10	9
	E	0,9896	0,9967	0,9983	1	0,9994	0,9949	0,9981	0,9994

Tabela A.4: Resultados das Estratégias Para o Programa Schedule2

Schedule2									
		A10	A20	A40	S5	S10	S20	EI	Tabu Set IM
T1	M'	219	454	916	1370	783	218	553	648
	T'	14	18	22	25	19	15	24	19
	E	0,9829	0,9931	0,9985	0,9991	0,9931	0,9927	0,9991	0,9983
T2	M'	219	454	916	1370	783	218	553	648
	T'	16	23	26	31	28	23	28	23
	E	0,9813	0,9942	0,9977	0,9991	0,9940	0,9931	0,9987	0,9974
T3	M'	219	454	916	1370	783	218	553	648
	T'	14	18	24	27	21	17	22	21
	E	0,9854	0,9904	0,9982	0,9991	0,9910	0,9901	0,9953	0,9978
T4	M'	219	454	916	1370	783	218	553	648
	T'	14	20	25	30	20	17	26	19
	E	0,9861	0,9923	0,9981	0,9991	0,9914	0,9905	0,9974	0,9974
T5	M'	219	454	916	1370	783	218	553	648
	T'	16	22	25	26	19	16	25	22
	E	0,9811	0,9941	0,9985	0,9991	0,9918	0,9858	0,9978	0,9983
T6	M'	219	454	916	1370	783	218	553	648
	T'	17	21	24	26	23	22	24	22
	E	0,9890	0,9950	0,9982	0,9991	0,9940	0,9931	0,9987	0,9978
T7	M'	219	454	916	1370	783	218	553	648
	T'	14	18	24	29	22	17	24	20
	E	0,9850	0,9933	0,9986	0,9991	0,9935	0,9854	0,9996	0,9978
T8	M'	219	454	916	1370	783	218	553	648
	T'	15	20	25	29	21	17	26	18
	E	0,9797	0,9960	0,9985	0,9991	0,9914	0,9832	0,9978	0,9983
T9	M'	219	454	916	1370	783	218	553	648
	T'	14	18	23	28	20	17	24	20
	E	0,9828	0,9930	0,9983	0,9991	0,9935	0,9931	0,9987	0,9978
T10	M'	219	454	916	1370	783	218	553	648
	T'	18	23	26	32	26	21	29	23
	E	0,9850	0,9951	0,9973	0,9991	0,9940	0,9910	0,9996	0,9974

Tabela A.5: Resultados das Estratégias Para o Programa Tcas

Tcas									
		A10	A20	A40	S5	S10	S20	EI	Tabu Set IM
T1	M'	141	293	594	603	298	22	515	534
	T'	29	39	49	46	34	9	35	57
	E	0,9540	0,9795	0,9870	0,9860	0,9681	0,6410	0,9568	0,9953
T2	M'	141	293	594	603	298	22	515	534
	T'	31	43	54	47	38	8	35	62
	E	0,9529	0,9721	0,9871	0,9854	0,9648	0,5931	0,9468	0,9847
T3	M'	141	293	594	603	298	22	515	534
	T'	32	43	55	53	40	10	39	65
	E	0,9449	0,9777	0,9893	0,9914	0,9820	0,6090	0,9535	0,9953
T4	M'	141	293	594	603	298	22	515	534
	T'	31	42	56	45	32	7	33	72
	E	0,9461	0,9734	0,9884	0,9874	0,9648	0,6469	0,9402	0,9874
T5	M'	141	293	594	603	298	22	515	534
	T'	32	43	57	48	34	8	31	65
	E	0,9392	0,9749	0,9874	0,9854	0,9668	0,5652	0,9315	0,9940
T6	M'	141	293	594	603	298	22	515	534
	T'	31	40	52	48	37	8	36	58
	E	0,9469	0,9758	0,9898	0,9880	0,9721	0,6762	0,9535	0,9947
T7	M'	141	293	594	603	298	22	515	534
	T'	30	40	54	47	36	8	36	57
	E	0,9450	0,9712	0,9893	0,9880	0,9721	0,6848	0,9328	0,9953
T8	M'	141	293	594	603	298	22	515	534
	T'	31	42	51	47	38	9	35	59
	E	0,9497	0,9779	0,9890	0,9887	0,9714	0,6184	0,9501	0,9947
T9	M'	141	293	594	603	298	22	515	534
	T'	30	39	52	48	37	9	32	60
	E	0,9521	0,9708	0,9892	0,9880	0,9754	0,6370	0,9528	0,9953
T10	M'	141	293	594	603	298	22	515	534
	T'	29	40	51	45	33	8	34	53
	E	0,9489	0,9761	0,9899	0,9847	0,9661	0,5658	0,9481	0,9920

Tabela A.6: Resultados das Estratégias Para o Programa Totinfo

Totinfo									
		A10	A20	A40	S5	S10	S20	EI	Tabu Set IM
T1	M'	185	377	772	924	470	83	634	391
	T'	8	11	12	15	11	7	10	12
	E	0,9970	0,9991	0,9993	1	0,9852	0,9807	0,9990	1
T2	M'	185	377	772	924	470	83	634	391
	T'	8	10	11	13	9	6	11	12
	E	0,9965	0,9996	0,9997	1	0,9858	0,9812	0,9995	1
T3	M'	185	377	772	924	470	83	634	391
	T'	10	11	12	15	12	9	11	12
	E	0,9978	0,9995	0,9998	1	0,9858	0,9858	0,9995	1
T4	M'	185	377	772	924	470	83	634	391
	T'	10	13	15	16	12	9	17	14
	E	0,9966	0,9992	0,9994	1	0,9822	0,9812	0,9995	0,9990
T5	M'	185	377	772	924	470	83	634	391
	T'	6	8	9	12	10	6	8	8
	E	0,9981	0,9996	0,9998	1	0,9858	0,9858	0,9995	1
T6	M'	185	377	772	924	470	83	634	391
	T'	6	8	10	11	10	5	7	9
	E	0,9982	0,9992	0,9993	1	0,9995	0,9980	0,9980	1
T7	M'	185	377	772	924	470	83	634	391
	T'	11	12	13	14	11	9	14	13
	E	0,9986	0,9991	0,9995	1	0,9832	0,9822	0,9995	0,9990
T8	M'	185	377	772	924	470	83	634	391
	T'	8	9	12	13	10	7	12	11
	E	0,9951	0,9992	0,9997	1	0,9847	0,9827	0,9995	0,9990
T9	M'	185	377	772	924	470	83	634	391
	T'	10	13	16	18	14	9	17	12
	E	0,9946	0,9973	0,9987	0,9990	0,9847	0,9761	0,9985	0,9969
T10	M'	185	377	772	924	470	83	634	391
	T'	11	12	15	18	13	9	14	14
	E	0,9975	0,9994	0,9996	1	0,9847	0,9847	0,9995	1