UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

MARIANA SOLLER RAMADA

Geração de sequências de teste a partir de Máquinas de Estados Finitos baseada em busca

Goiânia
2020

UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO (TECA) PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TESES

E DISSERTAÇÕES NA BIBLIOTECA DIGITAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFG), regulamentada pela Resolução CEPEC nº 832/2007, sem ressarcimento dos direitos autorais, de acordo com a Lei 9.610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo das Teses e Dissertações disponibilizado na BDTD/UFG é de responsabilidade exclusiva do autor. Ao encaminhar o produto final, o autor(a) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

1. Identificação do material bibliográfico

[ ] Dissertação [X] Tese

2. Nome completo do autor

Mariana Soller Ramada

3. Título do trabalho

Geração de sequências de teste a partir de Máquinas de Estados Finitos baseada em busca

4. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador)

Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante:
a) consulta ao(à) autor(a) e ao(à) orientador(a);
b) novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo da tese ou dissertação.
O documento não será disponibilizado durante o período de embargo.
Casos de embargo:
- Solicitação de registro de patente;
- Submissão de artigo em revista científica;
- Publicação como capítulo de livro;
- Publicação da dissertação/tese em livro.

Obs. Este termo deverá ser assinado no SEI pelo orientador e pelo autor.

Documento assinado eletronicamente por **MARIANA SOLLER RAMADA**, **Discente**, em 10/08/2020, às 15:28, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Telma Woerle De Lima Soares**, **Professora do Magistério Superior**, em 11/08/2020, às 10:33, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1479211** e o código CRC **C8462344**.

Referência: Processo nº 23070.034804/2020-36 SEI nº 1479211

MARIANA SOLLER RAMADA

Geração de sequências de teste a partir de Máquinas de Estados Finitos baseada em busca

Tese apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás em associação com a Universidade Federal de Mato Grosso do Sul, como requisito parcial para obtenção do título de Doutor em Ciência da Computação.

Área de concentração: Ciência da Computação.

Orientadora: Profa. Telma Woerle de Lima Soares

Co-Orientador: Prof. Adenilso da Silva Simão

Goiânia
2020

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Soller Ramada, Mariana
Geração de sequências de teste a partir de Máquinas de Estados Finitos baseada em busca [manuscrito] / Mariana Soller Ramada. - 2020.
CXXXIV, 134 f.: il.

Orientador: Profa. Dra. Telma Woerle de Lima Soares; co orientador Dr. Adenilso da Silva Simão.
Tese (Doutorado) - Universidade Federal de Goiás, Instituto de Informática (INF), Programa de Pós-Graduação em Ciência da Computação em rede (UFG/UFMS), Goiânia, 2020.
Bibliografia.
Inclui siglas, abreviaturas, símbolos, gráfico, tabelas, algoritmos, lista de figuras, lista de tabelas.

1. Teste baseado em modelo. 2. Máquina de estados finitos. 3. Geração de casos de teste. 4. Problema de otimização. 5. Metaheurística. I. Woerle de Lima Soares, Telma, orient. II. Título.

CDU 004

UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

ATA DE DEFESA DE TESE

Ata Nº **04/2020** da sessão de Defesa de Tese de **Mariana Soller Ramada,** que confere o título de Doutora em Ciência da Computação, na área de concentração em Ciência da Computação**.**

Aos quatorze dias do mês de julho de dois mil e vinte, a partir das oito horas e trinta minutos, via sistema de webconferência da RNP, realizou-se a sessão pública de Defesa de Tese intitulada **"Geração de sequências de teste a partir de Máquinas de Estados Finitos baseada em busca**". Os trabalhos foram instalados pela Orientadora, Professora Doutora Telma Woerle de Lima Soares (INF/UFG) com a participação dos demais membros da Banca Examinadora: Professor Doutor Adenilso da Silva Simão (ICMC-USP), coorientador; Professor Doutor Alexandre Cláudio Botazzo Delbem (ICMC-USP), membro titular externo; Professor Doutor André Takeshi Endo (UTFPR), membro titular externo; Professor Doutor Celso Gonçalves Camilo Junior (INF/UFG), membro titular interno; Professor Doutor Fernando Marques Federson (INF-UFG), membro titular externo. A realização da banca ocorreu per meio de videoconferência, em atendimento à recomendação de suspensão das atividades presenciais na UFG emitida pelo Comitê UFG para o Gerenciamento da Crise COVID-19, bem como à recomendação de isolamento social da Organização Mundial de Saúde e do Ministério da Saúde para enfrentamento da emergência de saúde pública decorrente do novo coronavírus. Durante a argüição os membros da banca não fizeram sugestão de alteração do título do trabalho. A Banca Examinadora reuniu-se em sessão secreta a fim de concluir o julgamento da Tese tendo sido a candidata **aprovada** pelos seus membros. Proclamados os resultados pela Professora Doutora Telma Woerle de Lima Soares, Presidente da Banca Examinadora, foram encerrados os trabalhos e, para constar, lavrou-se a presente ata que é assinada pelos Membros da Banca Examinadora, aos catorze dias do mês de julho de dois mil e vinte.

TÍTULO SUGERIDO PELA BANCA

Documento assinado eletronicamente por **André Takeshi Endo**, **Usuário Externo**, em 14/07/2020, às 11:39, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Fernando Marques Federson**, **Professora do Magistério Superior**, em 14/07/2020, às 11:40, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Telma Woerle De Lima Soares**, **Professora do Magistério Superior**, em 14/07/2020, às 11:40, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Adenilso da Silva Simão**, **Usuário Externo**, em 14/07/2020, às 11:40, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Celso Gonçalves Camilo Junior**, **Usuário Externo**, em 14/07/2020, às 11:41, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Alexandre Cláudio Botazzo Delbem**, **Usuário Externo**, em 14/07/2020, às 11:43, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **MARIANA SOLLER RAMADA**, **Discente**, em 14/07/2020, às 12:04, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1416139** e o código CRC **7B1A06C7**.

Referência: Processo nº 23070.028901/2020-90 SEI nº 1416139

Mariana Soller Ramada

Graduou-se em Ciência da Computação na UFG - Universidade Federal de Goiás. Durante sua graduação, foi estagiária do Centro de Recurso Computacionais (Cercomp) da UFG. Durante o Mestrado, realizado também na UFG, foi bolsista REUNI e monitora no Instituto de Informática na disciplina Banco de Dados do curso de Ciência da Computação. Após o Mestrado foi professora substituta no Instituto de Informática da UFG. Atualmente é Analista de Tecnologia da Informação do Instituto de Informática da Universidade Federal de Goiás.

Dedico este trabalho ao meu amado pai (o melhor!).

Agradecimentos

Primeiramente, gostaria de agradecer e dedicar esse trabalho ao meu pai, o qual é a pessoa que mais se orgulha e se empolga com minhas conquistas. Isso me faz querer ir sempre mais além, pelo simples fato de ter a oportunidade de vê-lo escutar e vibrar com cada passo meu. É um momento só nosso e é muito especial para mim.

Gostaria de agradecer ao meu marido, Murilo, que por sorte minha também é formado em Ciência da Computação e praticamente está fazendo o doutorado junto comigo. Gostaria de agradecê-lo pelo companheirismo, por me motivar, por dizer que está tudo bem quando passo por semanas que não rendem, por corrigir códigos que eu passei dias tentando resolver sem sucesso e por se esforçar em entender minha pesquisa, permitindo que tenhamos discussões sobre o tema que agregam muito na solução do projeto.

Agradeço à minha mãe, que é sempre uma pessoa leve e tranquila, e que graças à oportunidade de poder fazer o doutorado afastada do trabalho, pude ir desfrutar da sua comida e companhia todos os dias da semana. Obrigada mãe, por sempre me receber de braços abertos, mesmo nos dias que precisei almoçar correndo para voltar logo à faculdade, e por não ficar me perguntando do doutorado. Às vezes é bom ter a cabeça um pouco desligada disso tudo e ter uma pessoa que não nos remete a ele.

Um dos grandes motivos de fazer o doutorado é meu irmão mais velho e já doutor, Marcelo. Desde pequeno, sempre foi uma pessoa com grandes realizações e eu não poderia ficar pra trás. Agradeço por ele sempre estar à minha frente me instigando (implicitamente) a percorrer seus caminhos, pois admiro suas conquistas e espero poder chegar próximo do que ele é.

Agradeço à minha orientadora, Telma, por me dar a oportunidade de ser orientada por ela. Agradeço à ela pela escolha do tema de pesquisa, que une a computação evolutiva (área de pesquisa dela) com engenharia de sofwtare (uma área próxima da área de pesquisa que realizei no mestrado), permitindo que eu me sentisse mais à vontade com a temática. Agradeço também pela sua disponibilidade em me receber todas as vezes que necessitei e por me guiar durante algumas encruzilhadas da pesquisa, tornando o processo do doutorado menos árduo.

Agradeço ao meu coorientador, Adenilso, que independente da distância, contri-

buiu muito com o trabalho e sem ele eu não teria o nível de conhecimento do problema que hoje possuo. Gostaria de agradecê-lo, também, por me receber na USP de São Carlos e ter disponibilizado sua agenda para me atender todos os dias da semana que estive por lá, tornando minha ida à São Carlos muito produtiva.

Agradeço ao meus colegas do grupo de pesquisa de Metaheurísticas, por estarmos juntos todas às quartas-feiras durante esses quatro anos de caminhada, seja conversando, seja contribuindo com a pesquisa um do outro, seja surtando juntos.

Agradeço à UFG, em especial ao INF, por apoiar e permitir desenvolver este trabalho de maneira exclusiva, concedendo a oportunidade de estar afastada das minhas atividades laborais durante um período.

Resumo

Ramada, Mariana Soller. **Geração de sequências de teste a partir de Máquinas de Estados Finitos baseada em busca**. Goiânia, 2020. 134p. Tese de Doutorado. Instituto de Informática, Universidade Federal de Goiás.

O desenvolvimento de *software* é um processo complexo e, portanto, suscetível a erros. Visando aumentar a confiança no produto gerado, a atividade de teste tem como objetivo identificar erros ainda desconhecidos. Uma das principais etapas desta atividade é a geração de casos de teste, a qual exige grande esforço. Neste contexto, o Teste Baseado em Modelo (TBM) surgiu como uma estratégia para a geração de testes automatizados. Para tanto, o modelo de Máquinas de Estados Finitos (MEFs) tem sido amplamente utilizado. Idealmente, um conjunto de teste deve ter o menor custo possível e ser capaz de revelar o maior número possível de defeitos. Contudo, em geral, este problema é NP-Difícil. Vários métodos de geração de sequências de teste foram propostos, porém, apesar de muitos deles gerarem conjuntos com capacidade de detecção total de defeitos, esses métodos costumam obter conjuntos muito longos e com muitas sequências. Assim, nesta tese, um método de geração baseado em busca foi proposto com o objetivo de gerar conjuntos de teste que representem uma boa relação entre custo e cobertura de defeitos. Uma representação, denominada *Node-depth-outdegree Encoding* (NDOE), e seus operadores de mutação e inicialização foram propostos. Uma função de *fitness* foi apresentada. A função é composta por três objetivos: minimizar o comprimento do conjunto de teste, minimizar o número de sequências de teste e maximizar a cobertura de defeitos, cuja análise é baseada em *score* de mutação e condições de suficiência. Os resultados experimentais mostraram que o método proposto é capaz de gerar conjuntos de teste com alta cobertura de defeitos e com custo menor do que os métodos de geração do estado da arte.

Palavras–chave

Teste baseado em modelo, máquina de estados finitos, geração de casos de teste, problema de otimização, metaheurística

Abstract

Ramada, Mariana Soller. **A search-based testing from Finite State Machines**. Goiânia, 2020. 134p. PhD. Thesis. Instituto de Informática, Universidade Federal de Goiás.

Since the software development process is a complex task, it is susceptible to errors. In order to increase confidence in the correctness of the implementation under test, the testing activity aims to identify errors still unknown. One of the main steps of this activity is the generation of test cases, which requires great effort. In this context, the Model Based Testing (MBT) emerged as a promising strategy to support automated test case generation. For this purpose, Finite State Machines (FSMs) have been widely used. Ideally, a test suite should have lowest possible cost and be able to reveal as many faulty implementations as possible. Unfortunately, in general, it is an NP-hard problem. Several test generation methods have been proposed, however, although many of them generate a test suite with full fault detection capability, they usually generate large test suites with many sequences. In this sense, we propose a search-based software testing method aiming to generate test suites that represent a good trade-off between its cost and its fault coverage. A encoding, called Node-depth-outdegree Encoding (NDOE), and its mutation and initialization operators were proposed. A fitness function was presented. The proposed fitness is composed of three objectives: minimize the test suite length, minimize the number of test sequences, and maximize the fault coverage, which analysis is based on mutation score and sufficient conditions. The experimental results show that the proposed method is able to generate lower cost test suites, with high fault coverage, than literature test generation methods.

Keywords

Model-based testing, finite state machine, test case generation, problem optimization, metaheuristic

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Abreviaturas e Siglas

AE ..... Algoritmo Evolutivo

EATS ..... *Evolutionary Algorithm for Test Suite* (Algoritmo evolutivo para conjunto de teste)

EATS-MS . *Evolutionary Algorithm for Test Suite - Mutation Score* (Algoritmo evolutivo para conjunto de teste - Escore de mutação)

EATS-SC . *Evolutionary Algorithm for Test Suite - Sufficient Conditions* (Algoritmo evolutivo para conjunto de teste - Condições de suficiência)

HSI ..... *Harmonized State Identification* (Identificação de estado harmonizada)

INRES ... *INitiator-REsponder*

MEF ..... Máquinas de Estados Finitos

NDDE .... *Node-depth-degree Encoding* (Representação nó-profundidade-grau)

NDE ..... *Node-depth Encoding* (Representação nó-profundidade)

NDOE .... *Node-depth-outdegree Encoding* (Representação nó-profundidade-grau de saída)

NFL ..... *No Free Lunch* (Sem almoço grátis)

PPRs ..... Problemas de Projeto de Redes

SBSE ..... *Search-Based Software Engineering* (Engenharia de software baseada em busca)

SBST ..... *Search-Based Software Testing* (Teste de software baseado em busca)

SUT ..... *System Under Test* (Sistema sob teste)

TBM ..... Teste Baseado em Modelo

VV&T .... Verificação, Validação e Teste

Lista de Símbolos

a - vértice onde o novo vértice criado pelo operado AUN será adicionado; ou a subárvore podada pelos operadores RCS e CCS será realigada

A - programa sob teste

c - constante positiva

D - número de restrições de desigualdade

DS - sequência de distinção

D_M - domínio de especificação da MEF M

E - conjunto finito de arestas do grafo

E_T - conjunto finito de arestas da árvore T

f - vértice pai da raiz da subárvore a ser podada pelo operador RUS

$FC(m, M, TS)$ - *score* de mutação

G ou $G = (V, E)$ - grafo formado pelos conjuntos V e E

H_i - conjunto de identificação do estado i que representa a união das sequências de separação de i com cada estado da MEF

i_a - índice do vértice a

i_L - índice do último vértice pertencente a subàrvore podada

i_p - índice do vértice p

I - MEF de implementação

J - número de restrições de igualdade

k - número de partições do grafo

K - conjunto confirmado

$K(v)$ - *core number* do vértice v

$K(G)$ - *core number* do grafo G

l - número de variáveis de decisão

m - número de estados da MEF de implementação

M - MEF de especificação

n - número de estados da MEF de especificação

n_i - número de símbolos de entrada da MEF de especificação

n_{nc} - número de vértices confirmados

n_t - número de transições da MEF

n_{tcTS} - número de transições cobertas por *TS*

n_{tcNC} - número de transições cobertas pelos vértices confirmados

nv - número de vértices do grafo

N - conjunto de vizinhos de um vértice, incluindo o próprio

$N_c(m, M)$ - número de mutantes que estão em conformidade com a MEF M

$N_p(m, M, TS)$ - número de mutantes que passam no conjunto de teste TS

$N_t(m, M)$ - total de mutantes da MEF M

O - número de funções objetivo

p - raíz da subárvore a ser podada pelos operadores RCS e CCS

p_n - tamanho da população

P - conjunto *transition cover*

POP - população de um algoritmo

$P(v)$ - grau de partição do vértice v

$P(G)$ - grau de partição do grafo G

Q - conjunto *state cover*

r - raíz da subárvore a ser podada pelo operador RUS

R - conjunto diferença entre os conjuntos *transition cover* e *state cover*

s_0 - estado inicial da MEF de especificação

S - conjunto finito de estados da MEF de especificação

T ou $T = (V_T, E_T)$ - árvore composta pelo conjuntos V_T e E_T

TS - conjunto de teste

T_x - representação NDOE de uma árvore

u - vértice a ser modificado pelo operador MUN

u_0 - estado inicial da MEF de implementação

U - conjunto finito de estados da MEF de implementação

V - conjunto finito de vértices do grafo

V_T - conjunto finito de vértices da árvore T

w_c - peso do objetivo de cobertura

w_l - peso do objetivo de comprimento

w_r - peso do objetivo de número de *resets*

W - conjunto de caracterização

W_s - conjunto de identificação do estado s, o qual contém ao menos uma sequência que distingue s de qualquer outro estado da MEF

X - conjunto finito de entradas da MEF de especificação

$\hat{X}$ - espaço de variáveis de decisão

Y - conjunto finito de saídas da MEF de especificação

Y' - conjunto finito de saídas da MEF de implementação

z_i - valor a ser normalizado

$(z_i)_N$ - valor normalizado

z_{max} - maior valor da amostra

z_{min} - menor valor da amostra

Z - espaço objetivo

δ - função de transição da MEF de especificação

Δ - função de transição da MEF de implementação

ϵ - sequência vazia

$\Im(M)$ - domínio de defeitos da MEF M

λ - função de saída da MEF de especificação

Λ - função de saída da MEF de implementação

Ω_M - conjunto de todas as sequências de entradas definidas para a MEF M

CAPÍTULO 1

Introdução

1.1 Contextualização

O teste de software é uma atividade importante do ciclo de vida do software, pois consiste em encontrar defeitos inseridos no software durante o seu desenvolvimento, visando garantir a qualidade do produto final. Quando um sistema é implementado, é necessário testar se a implementação está de acordo com a especificação do sistema. Isso geralmente é realizado por meio de testes de conformidade, que tentam encontrar diferenças entre o comportamento de uma implementação sob teste e sua especificação (SALAM; HIERONS; SWIFT, 2008).

Uma vez que o teste manual é caro, propenso a erros e de difícil reprodução, a automação de teste tornou-se uma tendência devido à sua capacidade de reduzir custos e melhorar a eficácia. Assim, de acordo com Shu et al. (2016), a automatização da geração de casos de teste a partir de teste baseado em modelo (TBM) tornou-se um assunto importante tanto para a indústria, quanto para a pesquisa acadêmica. Segundo Endo e Simao (2013), o TBM é uma abordagem que deriva automaticamente casos de teste a partir de um modelo formal e tem apresentado um conjunto de benefícios, como alta taxa de detecção de defeitos, custo e tempo reduzidos, rastreabilidade e facilidade de lidar com a evolução dos requisitos.

Para gerar casos de teste a partir de uma especificação, é necessário um modelo que represente a mesma. Máquinas de Estados Finitos (MEFs) são comumente usadas para este fim. De acordo com Jourdan, Ural e Yenigün (2016) e Cutigi, Simao e Souza (2016), MEFs têm sido amplamente utilizadas devido à sua simplicidade e capacidade de especificar o comportamento de sistemas reativos. Usualmente, sistemas reativos são empregados em atividades que envolvem questões críticas, nas quais falhas podem implicar em perda de vidas humanas e prejuízos financeiros. Por essa razão, o processo de desenvolvimento desses sistemas costumam ter um alto custo devido à necessidade de realizar as atividades de teste com o máximo de rigor, para que o sistema atinja um grau aceitável de qualidade e confiabilidade. Assim, o teste baseado em MEF é um candidato promissor a ser aplicado no contexto de sistemas reativos, pois além da geração

automática de casos de teste, que possibilita a redução do custo da atividade de teste, outra característica relevante para os sistemas reativos é alta taxa de detecção de defeitos. Como exemplos de sistemas reativos, pode-se citar sistemas embarcados críticos, controle de tráfego e o monitoramento de ambientes industriais e hospitalares.

A geração de testes a partir de MEFs é um problema de longa data, sendo que os primeiros trabalhos datam das décadas de 50 (MOORE, 1956) e 60 (HENNINE, 1964). Muitos métodos de geração de sequências de teste a partir de MEFs foram propostos, dentre os mais conhecidos pode-se destacar os métodos W (CHOW, 1978), DS (GONENC, 1970), UIO (SABNANI; DAHBURA, 1988), W_p (FUJIWARA et al., 1991), HSI (PETRENKO; BOCHMANN, 1995), H (DOROFEEVA; EL-FAKIH; YEVTUSHENKO, 2005), SPY (SIMÃO; PETRENKO; YEVTUSHENKO, 2009) e P (SIMÃO; PETRENKO, 2010b).

1.2 Motivação

O custo de um conjunto de teste é determinado pelo seu tempo de execução e normalmente é o fator dominante quando se avalia o custo da aplicação de um método de geração de conjunto de teste. Quanto maior for o comprimento de um conjunto de teste, mais tempo será gasto na sua execução. Além disso, o número de sequências geradas também influencia no seu tempo de execução, uma vez que é necessária a execução de uma sequência (operação *reset*) que leva tanto a MEF quanto sua implementação ao seu estado inicial antes de executar uma nova sequência de teste. A operação *reset* é inserida no início de cada sequência do conjunto de teste e, portanto, o número de operações *resets* é igual ao número de sequências do mesmo.

Além de ter baixo custo, um conjunto de teste deve ser capaz de identificar o máximo de defeitos que qualquer implementação possa conter. Um conjunto de sequências de teste é chamado de completo se ele é capaz de detectar todos os defeitos em um determinado domínio de defeitos. Geralmente, o domínio de defeitos é definido em função do número máximo de estados que uma máquina de estados deve ter para ser equivalente à especificação.

A completude de um conjunto de teste pode ser verificada, por exemplo, por meio de *score* de mutação ou por meio de condições de suficiência. O *score* de mutação determina a porcentagem de implementações defeituosas que são reveladas pelo conjunto de teste. As condições de suficiência determinam quais são as condições que tornam um conjunto de teste completo. A título de exemplo, os conjuntos gerados pelos métodos W, W_p e HSI satisfazem as condições de Petrenko, Bochmann e Yao (1996) e o método H as condições de Dorofeeva, El-Fakih e Yevtushenko (2005), portanto são métodos capazes de gerarem conjuntos completos. Apesar de muitos dos métodos do estado da arte gerarem

conjuntos completos, ou seja, garantirem que todos os defeitos em um dado domínio de defeitos são revelados, esses métodos costumam obter conjuntos de teste muito grandes em comprimento e com muitas sequências.

De fato, o problema de encontrar um conjunto de teste capaz de revelar o máximo de defeitos em um dado domínio com custo mínimo é caracterizado como um problema NP-Difícil (LEE; YANNAKAKIS, 1996). Dada a complexidade do problema é fundamental o estudo e desenvolvimento de algoritmos eficientes para solução do mesmo.

A reformulação dos problemas de engenharia de *software* como problemas de otimização permite que metaheurísticas, como algoritmos genéticos (HOLLAND, 1975), *simulated annealing* (KIRKPATRICK; GELATT; VECCHI, 1983) e busca tabu (GLOVER, 1986), sejam aplicadas. Esse campo da engenharia de *software* é conhecido como SBSE (*Search-Based Software Engineering*) e, de acordo com Harman e Jones (2001), surgiu uma vez que a natureza dos problemas de engenharia de software são ideais para a aplicação de técnicas de busca dado que tais técnicas podem prover formas de encontrar soluções aceitáveis em situações em que soluções ótimas são teoricamente impossíveis ou praticamente inviáveis. SBST (*Search-Based Software Testing*) é um subcampo da SBSE que trata do uso de metaheurísticas para solucionar problemas de engenharia de software relacionados com atividades de teste.

No contexto de teste baseado em MEFs, a maior gama de métodos de geração existentes é determinística. Diversos estudos experimentais demonstram que os métodos mais recentes se mostram capazes de gerar conjuntos completos com menor custo dos que os métodos propostos ao longo dos anos. Contudo, os estudos experimentais de Cutigi, Simao e Souza (2016) evidenciaram que é possível satisfazer a completude com um conjunto de teste ainda menor, em comprimento e/ou número de sequências, que os gerados pelos métodos recentes. Cutigi, Simao e Souza (2016) propuseram abordagens de redução que, a partir de um conjunto de teste completo gerado por um método de geração clássico, combinam as sequências do conjunto e verificam se o novo conjunto obtido mantém a completude. A verificação da completude é baseada nas condições de suficiência definidas por Simão e Petrenko (2010a). Por serem menos exigentes, ou seja, mais fáceis de serem satisfeitas, as condições propostas por Simão e Petrenko (2010a) permitem que as abordagens de redução resultem em conjuntos de testes menores.

No cenário de SBST, há poucos métodos propostos (LEHRE; YAO, 2014; DERDERIAN et al., 2006; GUO et al., 2005; GUO et al., 2004) e eles têm a característica em comum de gerar sequências únicas de entrada e saida (UIO), que são sequências especiais utilizadas para verificar se a MEF está em um estado particular. Contudo, tais sequências podem não existir para todas as MEFs, e portanto, a aplicação do método fica limitada à existência da referida sequência.

Dadas a evidência que é possível encontrar soluções mais eficientes que as

geradas pelos métodos determinísticos e a limitação da aplicação dos métodos geradores baseado em busca, torna-se relevante explorar o cenário de SBST. Dito isso, o presente trabalho apresenta o estudo e investigação de um método de geração baseado em busca capaz de gerar conjuntos de teste que apresentem um bom *trade-off* entre seu custo e cobertura de defeitos.

1.3 Objetivos do trabalho

O objetivo do trabalho é desenvolver um algoritmo evolutivo (AE) para solucionar de maneira eficiente o problema de geração de sequências de teste a partir de MEFs. Este problema pode ser modelado como um problema de otimização com os seguintes objetivos: maximizar a cobertura de defeitos, minimizar o comprimento do conjunto gerado e minimizar o número de sequências geradas. No entanto, a utilização de um AE padrão pode não ser suficiente para garantir a eficiência. Conforme Rothlauf (2011), para obter uma metaheurística de alto desempenho é preciso considerar o conhecimento específico do problema para o projeto da representação e função de *fitness*, bem como dos operadores de pesquisa e inicialização. Com base nisso, os seguintes objetivos específicos nortearam o desenvolvimento da pesquisa:

- **Desenvolver uma representação para árvores:** considerando que um conjunto de teste, solução candidata, pode ser representado como uma árvore, uma representação para o referido problema de busca deve incorporar as restrições necessárias para um grafo corresponder a uma árvore. O uso de representações convencionais, em geral, pode produzir soluções infactíveis, como grafos com ciclos ou grafos desconexos, os quais não representam soluções válidas. A produção de tais soluções exige a necessidade de correção das mesmas, até que seja obtida uma solução factível, consumindo parte do tempo de computação e reduzindo drasticamente a eficiência do AE. Portanto, uma nova representação (NDOE - *Node-depth-outdegree Encoding*) foi proposta levando em consideração as restrições necessárias para construção de apenas soluções factíveis;
- **Desenvolver operadores de mutação:** similarmente, o uso de operadores padrões poderia resultar em soluções infactíveis degradando o desempenho do AE. Nesse sentido, novos operadores de mutação foram propostos considerando os conhecimentos específicos do problema;
- **Desenvolver uma função de *fitness*:** dado que a modelagem do problema como um problema de otimização se trata de um problema multiobjetivo, será explorada uma abordagem baseada em preferência, transformando os três objetivos em um único objetivo. Além disso, com respeito ao objetivo de cobertura, para a verificação

de completude serão exploradas duas técnicas: *score* de mutação e condições de suficiência. Serão consideradas as condições de suficiência de Simão e Petrenko (2010a). Tais condições generalizam condições propostas anteriormente, sendo possível encontrar conjuntos com custo menor e com a mesma eficácia. Até o momento, nenhum método de geração foi proposto considerando tais condições;

- **Desenvolver uma heurística para encontrar clique máxima em grafos k-partidos:** para avaliar a cobertura de defeitos, com base nas condições de suficiência de Simão e Petrenko (2010a), é necessário o conhecimento da clique máxima de um determinado grafo que possui a propriedade de ser k-partido. Dado que o problema de encontrar clique máxima em grafos k-partidos é NP-Completo (ASAHIRO; MIYANO; SAMIZO, 2010; MALOD-DOGNIN; ANDONOV; YANEV, 2010), foi proposta uma heurística que resolve o problema de maneira eficiente;
- **Desenvolver um método de inicialização:** preocupando-se em gerar a população inicial com diversidade suficiente, será explorado uma estratégia de inicialização aleatória; e
- **Avaliação do método proposto:** realização de estudos experimentais a partir de MEFs geradas aleatoriamente e um estudo de caso que mostram a viabilidade e aplicabilidade do método proposto.

As principais contribuições deste trabalho são a formulação do problema de geração de sequências de teste a partir de MEFs como um problema de busca e o projeto de uma metaheurística para a solução deste problema. Além disso, o presente trabalho apresenta resultados de estudos experimentais conduzidos para tornarem claras as vantagens e desvantagens do método proposto comparando-o com outros métodos do estado arte.

1.4 Organização da tese

Além desta seção introdutória, o restante desta tese está organizada da seguinte forma. Os Capítulos 2 e 3 apresentam os principais conceitos inseridos no contexto de teste de *software* e na técnica de teste baseado em MEFs, além dos principais métodos conhecidos na literatura para geração de sequências de teste a partir de MEFs. No Capítulo 4 são apresentados os principais conceitos e definições sobre metaheurísticas, bem como os principais elementos de projeto para a construção de uma metaheurística de alto desempenho. O Capítulo 5 descreve o método de geração de sequências de teste baseado em busca proposto neste trabalho. Nesse capítulo são explanados as características e o funcionamento do método proposto. No Capítulo 6 realiza-se uma avaliação experimental

do método proposto em relação a alguns métodos de geração existentes e consolidados na literatura. Por fim, o Capítulo 7 apresenta as conclusões, contribuições e trabalhos futuros deste trabalho.

CAPÍTULO 2

Teste de *Software*

2.1 Considerações iniciais

Como a maioria das atividades de engenharia, a construção de software depende principalmente da habilidade, da interpretação e da execução das pessoas que o constroem; por isso; erros acabam surgindo. Para que tais erros não perdurem durante todo o processo de desenvolvimento, existe uma série de atividades fundamentais para o desenvolvimento de um produto final com um nível aceitável de qualidade, dentre elas o teste de *software*.

A atividade de teste visa a identificação de erros ainda desconhecidos em um sistema. Ela consiste em executar o programa sob teste utilizando algumas entradas em particular e verificar se seu comportamento está de acordo com o esperado. A qualidade obtida no teste de um *software*, geralmente, está associada à geração de casos de teste que permitem realizar uma série de cenários de execução de uma implementação do sistema (PETRENKO; BOCHMANN; YAO, 1996). Testes exaustivos, que exigem que todos os possíveis cenários de execução da implementação sejam realizados, é capaz de comprovar a corretude do sistema sob teste. Contudo, o teste exaustivo é impraticável, pois pode envolver um número enorme ou infinito de cenários de execução a serem avaliados. Dado o alto custo associado à atividade de teste, a automatização de geração de casos de teste tornou-se uma tendência devido à sua capacidade de reduzir custos e melhorar a eficácia (YANG et al., 2015).

No teste baseado em modelo, o comportamento do sistema em teste é representado e descrito por um modelo formal, permitindo a automatização da geração de testes. Além disso, a modelagem formal de um sistema permite que as atividades de teste possam ser iniciadas antes da implementação do *software*, uma vez que o processo de modelagem, geralmente, é realizado no início do ciclo de desenvolvimento.

Neste capítulo são apresentados os conceitos gerais de teste de *software*, bem como os conceitos relacionados ao teste baseado em modelo.

2.2 Fundamentos de Teste de *Software*

Segundo Pfleeger (2004), os estágios do ciclo de desenvolvimento de *software* envolvem não somente as habilidades relacionadas à computação, mas também a capacidade de comunicação e habilidade interpessoais. Sendo assim, apesar de dispor de diversas metodologias, técnicas e ferramentas capazes de facilitar e aprimorar o desenvolvimento do *software*, este é um processo propenso a enganos podendo acarretar em falhas no produto gerado.

Com o uso crescente de *software* nos mais diversos setores da sociedade, a demanda e a preocupação com a produção de *software* de alta qualidade motivou a introdução de atividades agregadas, sob o nome de garantia de qualidade de *software*, ao longo de todo o processo de desenvolvimento (ROCHA; MALDONADO; WEBER, 2001). Dentre as principais atividades de garantia de qualidade de *software* estão as atividades de Verificação, Validação e Teste (VV&T), que possuem a finalidade de garantir que tanto o modo pelo o qual o *software* está sendo construído quanto o produto em si estejam em conformidade com o especificado (ROCHA; MALDONADO; WEBER, 2001; DELAMARO; MALDONADO; JINO, 2016).

Segundo Delamaro, Maldonado e Jino (2016), as atividades de VV&T não se restringem ao produto final. Ao contrário, podem e devem ser conduzidas durante todo o processo de desenvolvimento do *software*. A aplicação desses tipos de atividades durante o processo de desenvolvimento possibilitam a identificação precoce de erros inseridos no *software*, minimizando assim falhas e riscos associados. As atividades de teste, em específico, constituem um elemento crítico na garantia da qualidade de um sistema, visto que essas atividades representam a revisão final da especificação, projeto e codificação do sistema (MYERS; SANDLER; BADGETT, 2011).

As atividades de teste de *software* consistem em executar um sistema ou componente sob condições específicas com o objetivo de encontrar defeitos no produto e aumentar a confiança nesse. Um teste bem sucedido é aquele que revela a presença de um ou mais defeitos até então não encontrados (MYERS; SANDLER; BADGETT, 2011).

Para fins de entendimento deste trabalho, as definições utilizadas para os termos defeito, engano, erro e falha seguem o padrão IEEE 610.12-1990 (IEEE, 1990), tais que:

- **defeito** (do inglês, *fault*) – passo, processo ou definição de dados incorretos, como por exemplo uma instrução ou comando incorreto;
- **engano** (do inglês, *mistake*) – ação humana que produz um resultado incorreto, como uma ação incorreta tomada pelo programador;
- **erro** (do inglês, *error*) – diferença entre o valor obtido e o valor esperado, ou seja, qualquer estado intermediário incorreto ou resultado inesperado na execução do programa constitui um erro; e

- **falha** (do inglês, *failure*) – produção de uma saída incorreta com relação à especificação.

O padrão define também o conceito de **caso de teste**, como sendo um conjunto de entradas de teste, condições de execução e saídas esperadas desenvolvido para uma finalidade particular, como por exemplo exercitar um determinado caminho do programa ou verificar a conformidade com um determinado requisito. Um **conjunto de teste** é composto por casos de teste.

Quando executado de forma sistemática e criteriosa, o teste contribui para aumentar a confiança no produto. Contudo, as atividades de teste não podem garantir a ausência completa de erro, ou se o *software* se comportará conforme especificado em qualquer situação. Sommerville (2011) afirma que é possível que um determinado teste seja esquecido e justamente ele seria o responsável por descobrir mais problemas no sistema. Conforme Dahl, Dijkstra e Hoare (1972), a atividade de teste pode ser usada para mostrar a presença de erros, mas nunca para mostrar sua ausência. Dessa forma, essas atividades devem ser planejadas de modo a projetar testes que descubram sistematicamente diferentes classes de erros e façam-no com uma quantidade de tempo e esforço mínimo (PRESSMAN, 2006).

Segundo Pressman (2009), uma atividade de teste deve seguir uma estratégia que possibilite um acompanhamento gerencial à medida que o projeto avança, além de proporcionar um roteiro que descreva os passos a serem seguidos de maneira controlada. Para isso, a execução de uma atividade de teste envolve quatro etapas (BEIZER, 1990; PRESSMAN, 2009):

- **Planejamento:** é responsável por formular a maneira em que a atividade de teste será conduzida e os recursos necessários para a condução do processo de teste, como: cronograma, recursos humanos, escolha das técnicas e critérios a serem utilizados, dentre outros elementos que são arbitrados pelas necessidades do projeto;
- **Projeto dos casos de teste:** consiste na elaboração dos casos de teste.
- **Execução do teste:** conduz a aplicação dos casos de teste criados anteriormente. Ainda segundo Pfleeger (2004), a execução de um teste começa com a revisão dos casos de teste, a fim de verificar se estão corretos, se são viáveis, se fornecem o grau desejado de cobertura e se demonstram a funcionalidade pretendida; e
- **Coleta e avaliação dos resultados:** após a execução do teste, os resultados produzidos são coletados, registrados, organizados e apresentados na forma de relatórios. Tais resultados são avaliados em relação aos resultados esperados.

2.2.1 Fases de teste

Erros podem ser ocasionados por tipos distintos de defeitos os quais podem estar em diferentes níveis de abstração. Portanto, é desejável que o teste de software seja dividido em fases que permitam testar, além do sistema por completo, as pequenas partes que compõem o sistema. Conforme Delamaro, Maldonado e Jino (2016), de forma geral, pode-se estabelecer como fases o teste de unidade, o teste de integração e o teste de sistema. Tais fases são descritas a seguir (ROCHA; MALDONADO; WEBER, 2001; PRESSMAN, 2009):

- **Teste de unidade:** tem por objetivo explorar a menor unidade do projeto, procurando identificar erros de especificação e/ou implementação em cada módulo, separadamente. Segundo o padrão (IEEE, 1990), uma unidade é um componente de *software* que não pode ser subdividido. Assim, esse tipo de teste se concentra na verificação de falhas nos parâmetros de entrada/saída, nas estruturas de dados que influenciam na integridade dos dados armazenados e nas condições que determinam os limites de operação da unidade;
- **Teste de integração:** tem por objetivo descobrir erros associados às interfaces entre os módulos quando esses são integrados para construir a estrutura do *software* que foi estabelecida na fase de projeto. Esse teste propicia a identificação de diversos tipos de falhas, tais como: interface incorreta, falta ou conflito de funcionalidades, sequência incorreta de unidades; e
- **Teste de sistema:** tem por objetivo testar o sistema como um todo, levando em consideração os requisitos funcionais e não funcionais especificados. Além disso, nesse tipo de teste é avaliado o funcionamento do sistema em relação ao ambiente para o qual foi projetado, incluindo a plataforma de execução e os outros sistemas externos que estão direta ou indiretamente relacionados ao sistema em teste.

Visto que cada fase abrange aspectos distintos do processo de desenvolvimento de *software*, a atividade de teste pode ser estruturada de maneira que, em cada fase, diferentes tipos de erros sejam abordados, culminando no estabelecimento de estratégias adequadas de seleção de dados de teste. De fato, é necessária a seleção de um subconjunto de casos de teste finito capaz de garantir a qualidade desejada, pois um conjunto de casos de teste ideal seria aquele composto por todos os elementos do domínio de entrada do programa em teste. Contudo, isso geralmente é inviável, uma vez que o domínio de entrada pode ser extremamente grande ou até mesmo infinito. Como modo de solucionar essa questão, são definidos os **critérios de teste**.

Um critério de teste permite selecionar e avaliar casos de teste de forma a aumentar as possibilidades de revelar a presença de defeitos ou, quando isso não ocorre,

estabelecer um nível elevado de confiança na correção do produto (ROCHA; MALDONADO; WEBER, 2001). Em outras palavras, os critérios de teste podem ser utilizados como critério de geração de casos de teste e critério de adequação de casos de teste. No primeiro caso, o critério é utilizado para gerar um conjunto de casos de teste adequado por construção ao critério. No outro caso, o critério é utilizado para verificar se um conjunto de casos de teste satisfaz os requisitos de teste estabelecidos.

2.2.2 Técnicas de teste

Diferentes técnicas de teste foram estabelecidas para classificar os critérios de acordo com a origem da informação que é utilizada para estabelecer os requisitos de teste. As principais técnicas são: funcional, estrutural, baseada em defeitos e baseada em modelos.

Na técnica **funcional**, conhecida também como caixa preta, os testes são projetados para validar os requisitos funcionais, sem se preocupar com o funcionamento interno de um programa (PRESSMAN, 2006). Os requisitos de teste são estabelecidos com base na especificação, não utilizando conhecimento sobre uma determinada implementação. No teste funcional, os principais critérios de teste empregados são: particionamento em classes de equivalência, análise do valor limite e grafo de causa-efeito (PRESSMAN, 2006). As principais classes de erros passíveis de serem detectados por esse tipo de teste são: funcionalidades incorretas ou omitidas, erros de interface, erros de estrutura de dados ou de acesso a dados externos, erros de comportamento ou desempenho e, por fim, erros de iniciação e término.

O teste **estrutural** (caixa branca), ao contrário do teste funcional, necessita do conhecimento do código fonte do programa em testes. Os testes são projetados para garantir que todas as instruções do programa tenham sido exercitadas pelo menos uma vez durante os testes e que todas as condições lógicas tenham sido exercitadas (PRESSMAN, 2006). Os critérios pertencentes a essa técnica são classificados com base no fluxo de controle, no fluxo de dados e na complexidade (ROCHA; MALDONADO; WEBER, 2001). Segundo Pressman (2009), nesse tipo de teste é possível identificar, principalmente, as seguintes classes de defeitos: defeitos lógicos, pressuposições incorretas e defeitos de projeto.

Na técnica de teste **baseada em defeitos**, os requisitos de teste são estabelecidos explorando os erros típicos e comuns cometidos no processo de desenvolvimento de *software*. Dentre os principais critérios inseridos nessa categoria estão os critérios de análise de mutantes (DEMILLO et al., 1987) e semeadura de erros (RAMAMOORTHY; BASTANI, 1982).

O teste **baseado em modelo** está diretamente relacionado ao foco deste trabalho e, portanto, será abordado em detalhes na seguinte seção.

2.3 Teste Baseado em Modelo

Um modelo é uma representação que é mais simples do que o artefato representado, mas preserva (ou, ao menos, aproxima) alguns atributos importantes do artefato real (PEZZÈ; YOUNG, 2008). No contexto de teste baseado em modelo (TBM), o modelo formal, extraído de alguma especificação de requisitos ou fonte de conhecimento sobre o sistema, descreve as sequências de ações possíveis e as saídas esperadas. A partir dessa descrição, podem ser aplicados critérios de seleção de casos de teste.

O processo de modelagem de um sistema permite iniciar as atividades de teste mais cedo, sendo possível revelar defeitos na especificação de requisitos ou na fase de projeto de *software*. Também permite a aplicação de métodos analíticos que cobrem uma gama de cenários muito maior do que a que se pode testar explicitamente. Não menos importante, muitas dessas análises podem ser automatizadas (PEZZÈ; YOUNG, 2008). A abordagem do TBM, quando automatizada, pode reduzir os custos para reaplicação do processo de teste, visto que o *software* e o modelo, geralmente, são alterados com frequência. Dessa forma, os profissionais de teste podem reaplicar rapidamente o processo de geração de casos de teste a cada mudança realizada no modelo.

O TBM pode ser aplicado em qualquer fase de teste (unidade, integração e sistema) e comumente é considerado como teste funcional, pois existe uma predominância de técnicas de modelagem que consideram o sistema sob teste (do inglês, *System Under Test* - SUT) como uma caixa preta. Contudo, o oráculo que define e separa o comportamento adequado do comportamento errôneo é constituído pelo modelo formal da especificação do sistema.

Segundo Utting e Legeard (2006), o processo do TBM pode ser dividido em 5 etapas:

- **Modelagem do comportamento do sistema:** tem como intuito obter um modelo formal suficientemente correto e condizente com a especificação do *software* a ser testado. O modelo criado deve ter um nível de abstração maior que o sistema em si, ou seja, deve ser menor e mais simples, focando apenas nos detalhes para descrever com precisão as partes do SUT que serão testadas. Terminada a construção do modelo, uma boa prática é utilizar técnicas e ferramentas para verificar se o modelo está consistente e realmente apresenta o comportamento esperado;
- **Geração dos casos de teste abstratos:** tem como objetivo a geração dos casos de teste a partir do modelo criado. Geralmente, são utilizadas ferramentas que

implementam técnicas para geração de casos de teste. Além disso, são necessários critérios de seleção de teste para limitar o número de casos de teste derivados do modelo de teste. Como resultado, nessa etapa é gerado um conjunto de sequências de operações executáveis no modelo;

- **Concretização dos casos de teste abstratos:** tem como objetivo transformar os casos de testes abstratos em casos teste concretos e executáveis no sistema em teste;
- **Execução dos casos de teste:** etapa em que se executam, no sistema em teste, os casos de testes concretos gerados na etapa anterior. O resultado dessa etapa é um relatório de execução dos testes; e
- **Avaliação do resultado do teste:** tem como intuito analisar os resultados obtidos no relatório de execução do teste. O resultado dessa etapa é a definição de ações corretivas no sistema ou no modelo.

A geração de casos de teste está diretamente relacionada tanto ao modelo quanto à técnica de modelagem utilizada. Existe uma grande variedade de técnicas de modelagem distintas que podem ser usadas. O que diferencia essas técnicas são o grau de formalidade e mecanismos de modelagem oferecidos por elas. Dentre as principais técnicas de modelagem, pode-se citar: Máquinas de Estados Finitos (MEFs) (GILL, 1962), Máquinas de Estados Finitos Estendidas (MEFEs) (PETRENKO; BORODAY; GROZ, 2004), Statecharts (HAREL, 1987), Redes de Petri (PETERSON, 1977) e Estelle (TURNER, 1993).

A modelagem de *software* por meio de MEFs vem sendo frequentemente utilizada devido à sua simplicidade e expressividade. Este trabalho é embasado nessa técnica de modelagem, a qual é melhor descrita no próximo capítulo.

2.4 Considerações finais

A atividade de teste dispõe de diversas técnicas distintas direcionadas ao mesmo objetivo: revelar a presença de defeitos. A técnica de teste baseado em modelo, visa a auxiliar a tarefa de geração de casos de teste por meio da especificação formal do sistema em teste. A partir do modelo formal é possível a aplicação de métodos sistemáticos capazes de gerar conjuntos de casos de teste de maneira automática.

Dentre os principais modelos, encontram-se as Máquinas de Estados Finitos (MEFs) utilizadas para a modelagem de diversas classes de sistema, principalmente a classe de sistemas que possuem como característica um comportamento reativo, ou seja, arbitrado por estímulos de entrada. Os conceitos específicos ao teste baseado em MEFs são apresentados no próximo capítulo. Além disso, também são apresentados alguns dos principais métodos de geração de casos de teste do estado da arte.

CAPÍTULO 3

Teste baseado em Máquinas de Estados Finitos

3.1 Considerações iniciais

Neste capítulo são apresentados os principais conceitos e definições sobre MEFs e teste baseado em MEFs. Além disso, são apresentados alguns métodos de geração da literatura, bem como métricas para avaliar a qualidade dos conjuntos gerados.

Os conceitos introduzidos neste capítulo fundamentam a pesquisa realizada nesta tese no que diz respeito ao tema de geração de teste baseada em MEFs.

3.2 Máquina de Estados Finitos

Em sua forma mais simples, uma máquina de estados finitos é um conjunto finito de estados e um conjunto de transições entre os estados (PEZZÈ; YOUNG, 2008). Geralmente, MEFs são representadas na forma de diagrama de transição de estado, que se trata de um grafo dirigido, no qual os vértices representam os estados do programa e as arestas representam operações que transformam um estado do programa em outro. Em cada transição são rotulados os eventos (entrada), responsáveis pela mudança de estado.

Há dois tipos, comumente, utilizados de MEFs: máquina de *Mealy* (MEALY, 1955) e máquina de *Moore* (MOORE, 1956). Na máquina de *Mealy*, além da entrada, a saída também é definida na transição, enquanto na máquina de *Moore* a saída é definida no estado. Neste trabalho será considerada a máquina de *Mealy*.

Formalmente, de acordo com Petrenko e Yevtushenko (2005), uma MEF M (determinística e de *Mealy*) é uma 7-tupla $<S, s_0, X, Y, D_M, \delta, \lambda>$, onde:

- S é um conjunto finito de estados, incluindo o estado inicial s_0
- X é um conjunto finito de entradas
- Y é um conjunto finito de saídas
- D_M é o domínio da especificação, $D_M \subseteq S \times X$
- δ é uma função de transição, $\delta : D_M \rightarrow S$
- λ é uma função de saída, $\lambda : D_M \rightarrow Y$

A Figura 3.1 apresenta um exemplo de MEF com três estados $S = \{S_0, S_1 \text{ e } S_2\}$, duas entradas $X = (a \text{ e } b)$ e duas saídas $Y = (0 \text{ e } 1)$. O estado inicial é o S_0, indicado pela seta de entrada no canto superior esquerdo. Cada transição é representada por uma aresta direcionada rotulada com a entrada que ocasiona a transição e a saída que será produzida quando a transição for executada. Por exemplo, a aresta do estado S_1 para o estado S_2 com rótulo 'b/0' representa a transição (S_1,b), onde $\delta(S_1,b) = S_2$ e $\lambda(S_1,b) = 0$.

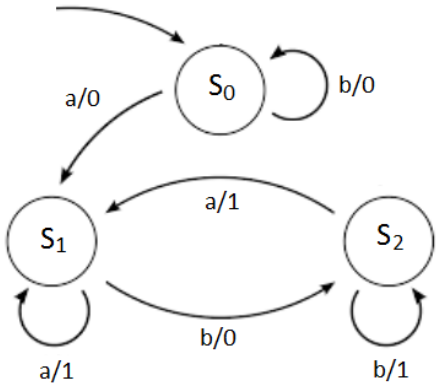


Figura 3.1: Exemplo de uma máquina de estados finitos

Uma tupla $(s, x) \in D_M$ determina unicamente uma transição (*definida*) de M no estado s. Uma sequência $\alpha = x_1 ... x_k$, $\alpha \in X*$, é dita ser uma sequência de entrada definida no estado $s \in S$ se existe $s_1, ..., s_{k+1}$, onde $s_1 = s$, tal que $(s_i, x_i) \in D_M$ e $\delta(s_i, x_i) = s_{i+1}$, para todo $1 \leq i \leq k$ (SIMÃO; PETRENKO, 2010a). A notação $\Omega_M(s)$ representa o conjunto de todas as sequências de entradas definidas no estado s e a notação Ω_M representa o conjunto de todas as sequências de entrada definidas no estado s_0, isto é, para MEF M (PETRENKO; YEVTUSHENKO, 2005).

A função de transição δ e a função de saída λ podem ser estendidas para as sequências de entradas definidas compostas por um ou mais símbolos de entrada, incluindo a sequência vazia denotada por ϵ. Tem-se que $\delta(s, \epsilon) = s$ e $\lambda(s, \epsilon) = \epsilon$ para todo $s \in S$. Seja β uma sequência de entradas e $\delta(s, \beta) = s'$, então, para todo $x \in X$ define-se $\delta(s, \beta x) = \delta(s', x)$ e $\lambda(s, \beta x) = \lambda(s, \beta)\lambda(s', x)$ (CUTIGI; SIMAO; SOUZA, 2016).

Dois estados s_i, $s_j \in S$ são *equivalentes* se $\lambda(s_i, \gamma) = \lambda(s_j, \gamma)$ para todo $\gamma \in \Omega(s_i) \cap \Omega(s_j)$. MEFs são equivalentes se seus estados iniciais são equivalentes (CUTIGI; SIMAO; SOUZA, 2016). Dois estados s_i, $s_j \in S$ são *distinguíveis* se existe um $\gamma \in \Omega(s_i) \cap \Omega(s_j)$ tal que $\lambda(s_i, \gamma) \neq \lambda(s_j, \gamma)$. MEFs são distinguíveis se seus estados iniciais são distinguíveis.

Duas sequências α e β são distinguíveis se existe uma sequência de entrada γ, tal que $\alpha\gamma$, $\beta\gamma \in \Omega_M$ e $\lambda(\delta(s_0, \alpha), \gamma) \neq \lambda(\delta(s_0, \beta), \gamma)$. Para um conjunto de teste TS, duas sequências α e β são *TS-distinguíveis* se existe $\alpha\gamma$, $\beta\gamma \in TS$, tal que $\lambda(\delta(s_0, \alpha), \gamma)$

$\neq \lambda(\delta(s_0, \beta), \gamma)$. De forma análoga, duas sequências α e β são TS-equivalentes se $\lambda(\delta(s_0, \alpha), \gamma) = \lambda(\delta(s_0, \beta), \gamma)$ para todo $\alpha\gamma$, $\beta\gamma \in TS$ (CUTIGI; SIMAO; SOUZA, 2016).

Uma sequência de entrada α é um *prefixo* da sequência β, denotado por $\alpha \leq \beta$, se existe uma sequência de entrada γ, tal que $\alpha\gamma = \beta$. Se γ não é a sequência vazia, então α é um *prefixo próprio*. A sequência γ é um *sufixo* de β. Para um conjunto de entradas A, denota-se por $pref(A)$ o conjunto de todos os prefixos de qualquer sequência de entrada de A, isto é, $pref(A) = \{\alpha | \exists \beta \in A \wedge \alpha \leq \beta\}$ (CUTIGI; SIMAO; SOUZA, 2016).

Com relação as propriedades das MEFs, diz-se que (DELAMARO; MALDONADO; JINO, 2016; SIMÃO; PETRENKO, 2010a):

- Uma MEF é *completa* ou *completamente especificada* se $D = S \times X$; caso contrário a MEF é *parcial* ou *parcialmente especificada*.
- Uma MEF é *fortemente conectada* se para cada par de estados (s_i, s_j) existe um caminho por transições que vai de s_i a s_j. Ela é inicialmente conectada se a partir do estado inicial é possível atingir todos os demais estados da MEF.
- Uma MEF é *minimal* (ou *reduzida*) se na MEF não existem quaisquer dois estados equivalentes.
- Uma MEF é *determinística* quando, para qualquer estado e para uma dada entrada, a MEF permite uma única transição para um próximo estado. Caso contrário, ela é *não-determinística.*

3.3 Teste baseado em MEFs

Quando um sistema é implementado, é necessário testar se a implementação está de acordo com a especificação do sistema. Isso geralmente é realizado por meio de testes de conformidade que tentam encontrar quaisquer diferenças entre o comportamento do SUT e sua especificação (SALAM; HIERONS; SWIFT, 2008).

Considere uma MEF M que representa a especificação de um sistema, sobre a qual tem-se todas as informações a respeito de suas funções de transição e saída, e uma MEF I que representa uma implementação, a qual é uma caixa-preta e apenas pode-se observar seu comportamento de entrada e saída. O teste baseado em MEF é geralmente formalizado como o problema de determinar se a máquina de implementação I confirma (ou seja, está correta em relação) a máquina de especificação M, testando I como uma caixa preta e observando suas saídas. Este problema é definido em Lee e Yannakakis (1996) como teste de conformidade ou problema de detecção de defeito.

O teste baseado em MEFs tem como objetivo gerar um conjunto de casos de teste a partir da MEF que representa a especificação do *software* a ser testado. Esse conjunto, ao ser aplicado em uma implementação, deve ser capaz de identificar qualquer divergência entre a implementação e a especificação, ou seja, evidenciar as falhas da implementação.

Quando a MEF possui determinadas características, como possuir sequência de distinção ou capacidade de *reset* confiável, é possível resolver o problema de teste de conformidade em tempo polinomial. Contudo, as sequências de teste geradas podem ser muito longas para aplicações reais. Segundo Lee e Yannakakis (1996), o problema de encontrar sequências mais curtas pode ser modelado como o problema de *Rural Postman Tour* (GAREY; JOHNSON, 1990 apud LEE; YANNAKAKIS, 1996), caracterizado como um problema NP-Difícil.

Uma *sequência de teste* α (ou *caso de teste*) é uma sequência de entrada definida na MEF M, tal que $\alpha \in \Omega_M$ (CUTIGI; SIMAO; SOUZA, 2016). Um *conjunto de teste* de M é um conjunto finito de casos de teste de M, tal que não existam dois casos de teste α e β em que α é prefixo de β. De fato, se um caso de teste α é um prefixo próprio do caso de teste β, a execução de β sempre irá implicar na execução de α, então α pode ser removido do conjunto de teste sem alterar o resultado do teste (ENDO; SIMAO, 2013). Um conjunto de teste pode ser representado como uma árvore, em que os nós são rotulados com os estados da MEF e as arestas são rotuladas com símbolos de entrada (SIMÃO; PETRENKO; YEVTUSHENKO, 2009). Cada sequência de teste corresponde à sequência composta pelos símbolos de entrada ao longo de um caminho da raiz até um nó. A raíz da árvore é o estado inicial. Considerando a MEF da Figura 3.1, a árvore da Figura 3.2 representa o conjunto de teste $TS = \{\text{aaab, abba, baa, bb}\}$.

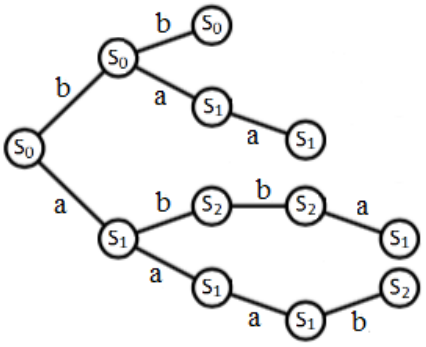


Figura 3.2: Representação em árvore do conjunto de teste $TS = \{\text{aaab, abba, baa, bb}\}$

A operação *reset* é uma sequência de entrada r que leva a MEF M de qualquer estado para o estado inicial s_0, isto é, $\delta_M(s_i, r) = s_0$ para todo $s_i \in$ S. O *reset* é dito confiável se seu funcionamento correto é garantido na máquina de implementação I, isto é, $\delta_I(s_i, r) = s_0$ para todo s_i de I; caso contrário, não é confiável (LEE; YANNAKAKIS, 1996). No presente trabalho serão consideradas máquinas com capacidade de *reset* confiável.

A operação *reset* aparece como o primeiro símbolo em uma sequência de teste. Por exemplo, a sequência de teste aab pode ser representada por $raab$, onde o primeiro símbolo indica que a operação *reset* deve ser aplicada antes de cada teste. Portanto, o número de *resets* de um conjunto de teste é o número de sequências que ele possui.

O *comprimento* de um conjunto de teste é obtido pela soma do número de símbolos de entrada contido no conjunto e o número de operações *resets*, ou seja, 1 é assumido como o comprimento da operação *reset* (CUTIGI; SIMAO; SOUZA, 2016).

Seja uma especificação M, um *domínio de defeitos* é o conjunto de todas as MEFs determinísticas com o mesmo alfabeto de entrada X de M, denotado por $\Im(M)$. Seja $m \geq 1$, um inteiro, $\Im_m(M)$ denota todas as MEFs de $\Im(M)$ com no máximo m estados (ENDO; SIMAO, 2013). Ainda segundo Endo e Simao (2013), dada uma MEF M com n estados, um conjunto de teste $TS \subseteq \Omega_M$ é *m-completo* ($m \geq n$), ou simplesmente *completo*, se para cada MEF $I \in \Im_m(M)$, tal que I e M são distinguíveis, existe uma sequência pertencente a TS que distingue I de M. Em outras palavras, um conjunto de testes m-completo é capaz de relevar todos os defeitos em qualquer implementação do domínio definido que tenha no máximo m estados. Nesse trabalho, será considerado $m = n$.

Conforme Delamaro, Maldonado e Jino (2016), os métodos de geração de casos de teste são fortemente baseados em algumas sequências ou conjunto de sequências básicas. A seguir, são abordadas algumas dessas sequências básicas.

Dados os estados $s, t \in S$, uma sequência $\alpha \in \Omega_M(s)$, tal que $\delta(s, \alpha) = t$, é uma *sequência de transferência* de s para t (PETRENKO; YEVTUSHENKO, 2005). Por exemplo, dados os estados s_0 e s_2 (Figura 3.1), α = ab é uma sequência de transferência de s_0 para s_2.

Um conjunto *state cover* Q de uma MEF M com n estados é definido como o conjunto de n sequências de transferência, incluindo a sequência vazia, que leva a máquina M a partir do estado inicial para cada um dos estados (PETRENKO; YEVTUSHENKO, 2005). A MEF da Figura 3.1 possui o conjunto *state cover* $Q = \{\epsilon, a, ab\}$.

Um conjunto *transition cover* P de uma MEF M se trata do conjunto de sequências de entrada que exercita todas as transições de M, incluindo a sequência vazia, isto é, para cada estado $s \in S$ e para cada entrada definida $x \in X$, existe uma sequência de entrada $\alpha \in P$ tal que $\delta(s_0, \alpha) = s$ e $\alpha, x \in P$ (CUTIGI; SIMAO; SOUZA, 2016). Um conjunto *transition cover* da MEF da Figura 3.1 é $P = \{\epsilon, a, b, aa, ab, aba, abb\}$.

Uma *sequência de distinção* DS é uma sequência de símbolos de entrada que, quando aplicada aos estados da MEF, produz saídas distintas para cada um dos estados. Ou seja, observando-se a saída produzida pela MEF como resposta à DS, pode-se determinar em qual estado a MEF estava originalmente (DELAMARO; MALDONADO; JINO, 2016). Em uma determinada MEF pode ou não existir uma sequência de distinção e portanto a aplicação de métodos baseados nessa sequência, como o método DS (GONENC, 1970), ficam limitados à existência da sequência de distinção. A sequência de distinção da MEF da Figura 3.1 é DS = {ba}.

Um *conjunto de caracterização* W é um conjunto de sequências de entrada que podem, juntas, identificar qual era o estado original (DELAMARO; MALDONADO;

JINO, 2016). Diferentemente de uma DS, um conjunto W pode possuir mais de uma sequência. Assim, quando se aplica uma sequência contida em W, é necessário utilizar alguma forma de restaurar a MEF para o estado original (que se deseja determinar), para que as demais sequências possam ser aplicadas. Uma sequência DS pode ser considerada um conjunto W unitário, portanto a MEF da Figura 3.1 possui o conjunto $W = \{\text{ba}\}$. Essa MEF também possui o conjunto $W = \{\text{a}, \text{b}\}$, que não é uma DS, pois é composto por mais de uma sequência.

Tanto a sequência DS quanto o conjunto W têm como objetivo básico identificar o estado em que a MEF estava antes da execução destas sequências. Contudo, nem toda MEF possui uma sequência DS. Em contrapartida, toda MEF minimal possui um conjunto W. Caso a MEF não seja minimal, ela pode ser reduzida para uma MEF minimal equivalente.

Um *conjunto de identificação* W_s para o estado s é um subconjunto do conjunto W, o qual contém ao menos uma sequência que distingue s de qualquer outro estado. Ou seja, para cada estado $s_i \in S$, $s_i \neq s$, existe $\alpha \in (W_{s_i} \cap W)$ tal que $\lambda(s, \alpha) \neq \lambda(s_i, \alpha)$ (CUTIGI; SIMAO; SOUZA, 2016). A união de todos os conjuntos de identificação W_i é um conjunto W. Para a MEF da Figura 3.1, o conjunto de identificação W_s para cada estado pode ser $W_0 = \{\text{a}, \text{b}\}$, $W_1 = \{\text{a}, \text{b}\}$ e $W_2 = \{\text{b}\}$.

Uma *família de separação* é um conjunto de sequências de identificadores de estado H_i para cada estado $s_i \in S$ que satisfaz a seguinte condição: para quaisquer estados s_i, s_j distintos, existem as sequências $\beta \in H_i$ e $\gamma \in H_j$ que possuem um prefixo comum α, ou seja, $\alpha \leq \beta$ e $\alpha \leq \gamma$, tal que $\alpha \in \Omega_M(s_i) \cap \Omega_M(s_j)$ e $\lambda(s_i, \alpha) \neq \lambda(s_j, \alpha)$ (ENDO; SIMAO, 2013). Para a MEF da Figura 3.1, o conjunto de identificação H_i para cada estado pode ser $H_0 = \{\text{a}\}$, $H_1 = \{\text{a}, \text{b}\}$ e $H_2 = \{\text{a}, \text{b}\}$.

3.3.1 Métodos de geração de sequências de teste

Na literatura, é possível encontrar diversos métodos propostos para geração de sequências de teste (CHOW, 1978; DOROFEEVA; EL-FAKIH; YEVTUSHENKO, 2005; SIMÃO; PETRENKO; YEVTUSHENKO, 2009; SIMÃO; PETRENKO, 2010b; SOUCHA; BOGDANOV, 2018). Embora os métodos possuam um objetivo em comum, eles diferem em relação:

- aos tipos de sequências básicas utilizadas,
- às restrições exigidas para sua aplicação,
- ao comprimento das sequências de teste geradas,
- à eficácia quanto a cobertura de defeitos, e
- à quantidade de operações *reset* contidas no conjunto de casos de teste.

De acordo com Fujiwara et al. (1991), um conjunto de sequências de teste deve apresentar as seguintes características:

- possuir a menor quantidade de sequências e o menor comprimento de sequências possível, agilizando sua execução e
- ser capaz de identificar, o máximo possível, os defeitos que qualquer implementação possa conter.

Geralmente, os métodos de geração de sequências de teste possuem duas fases: *state identification* e *transition testing*. A primeira fase verifica que cada estado especificado por M (especificação) também existe em I (implementação). A segunda fase assegura que para cada transição de M existe uma transição correspondente em I.

A seguir serão apresentados os métodos **W**, W_p, HSI, H, SPY e P que serão utilizados nos experimentos. Todos esses métodos são aplicáveis à MEFs inicialmente conectadas, minimais, completas e determinísticas. Os métodos HSI, H e P também são aplicáveis à máquinas parciais. Todos os métodos são determinísticos e geram conjuntos de teste completos.

3.3.1.1 Método W

Proposto por Chow (1978), é um dos métodos mais clássicos e conhecidos entre os métodos de geração de sequências de teste baseado em MEF. Esse método usa o conjunto *transition cover* P para exercitar todas as transições e, então, aplica o conjunto W para identificar cada estado alcançado. Considerando $m = n$, o conjunto de teste é gerado concatenando os conjuntos P e W e removendo os prefixos próprios.

Por exemplo, considere os conjuntos $P = \{\epsilon, a, b, aa, ab, abb, aba\}$ e $W = \{a, b\}$ para a MEF da Figura 3.1. Após realizar $P \cdot W$ e remover as sequências que são prefixos de outros testes, o conjunto de sequências de teste final é $TS_W = \{rba, rbb, raaa, raab, rabaa, rabab, rabba, rabbb\}$, de comprimento 34 e com 8 operações *reset*.

3.3.1.2 Método W_p

Fujiwara et al. (1991) apresentaram uma melhoria ao método **W**, que foi chamado de método W_p (*partial* W), que consiste em gerar conjuntos menores que os gerados pelo método **W** e com a mesma garantia de cobertura dos defeitos. Com o método W_p, é possível reduzir o conjunto de teste, pois ele não utiliza o conjunto W para verificar cada estado s_i da MEF, e sim um subconjunto deste (conjunto de identificação W_i).

O método W_p consiste em duas fases:

- **Fase 1:** gerar o conjunto $C_1 = Q \cdot W$

- **Fase 2:** gerar o conjunto $C_2 = \bigcup_{p \in R}\{p\} \cdot W_i$, tal que i é definido pelo estado final da execução da sequência p e $R = P - Q$.

Como conjunto final, tem-se a união dos conjuntos gerados pelas duas fases, ou seja, $TS_{Wp} = C_1 \cup C_2$. Considerando a MEF da Figura 3.1 e os conjuntos $Q = \{\epsilon, a, ab\}$, $P = \{\epsilon, a, b, aa, ab, abb, aba\}$, $R = \{b, aa, abb, aba\}$ $W = \{a, b\}$, $W_0 = \{a, b\}$, $W_1 = \{a, b\}$, $W_2 = \{b\}$, a aplicação do método W_p seria:

Na Fase 1 tem-se $C_1 = \{a, b, aa, ab, aba, abb\}$

Na Fase 2, $C_2 = \{ba, bb, aaa, aab, abbb, abaa, abab\}$

O conjunto final, eliminando os prefixos, é $TS_{Wp} = C_1 \cup C_2 = \{rba, rbb, raaa, raab, rabbb, rabaa, rabab\}$ de comprimento 29 e com 7 operações *reset*.

3.3.1.3 Método *HSI*

Proposto por Petrenko e Bochmann (1995), o método *HSI* (*Harmonized State Identification*) é o primeiro a gerar sequências tanto para MEFs parciais quanto para MEFs completas. A construção das sequências de teste é baseada nas sequências de identificação de estado (H_i), ou seja, família de separação. O conjunto de teste é obtido concatenando todas as sequências $\alpha \in P$ com H_i, tal que $\delta(s_0, \alpha) = s_i$.

Considerando a MEF da Figura 3.1 e os conjuntos $Q = \{\epsilon, a, ab\}$, $P = \{\epsilon, a, b, aa, ab, abb, aba\}$, $H_0 = \{a\}$, $H_1 = \{a, b\}$, $H_2 = \{a, b\}$, o conjunto final, eliminando os prefixos, é $TS_{HSI} = \{rba, raaa, raab, rabaa, rabab, rabba, rabbb\}$ de comprimento 31 e com 7 operações *reset*.

3.3.1.4 Método H

Similar ao método *HSI*, o método *H* (DOROFEEVA; EL-FAKIH; YEVTUSHENKO, 2005) também utiliza o conjunto de família de separação e pode ser visto como uma melhoria do método *HSI*. A diferença é que o método *H* seleciona as sequências de identificação de estado durante a construção do conjunto de teste. No método *HSI*, os identificadores de estado são pré-definidos antes do início da construção do conjunto de teste, mas pode haver mais de um identificador para cada estado. Considerando isso, o método *H* analisa o melhor identificador a ser utilizado durante a construção do conjunto, visando reduzir o comprimento final do conjunto de testes. Dessa forma, pode-se obter conjuntos menores de identificação, reaproveitando sequências que já estão no conjunto de teste.

O método *H* utiliza as condições de suficiência definidas por Dorofeeva, El-Fakih e Yevtushenko (2005) para avaliar a qualidade dos conjuntos gerados. Segundo Cutigi, Simao e Souza (2016), as condições de suficiência determinam propriedades que garantem que um conjunto de teste é capaz de revelar todos os defeitos de um dado

domínio. Em outras palavras, se determinado conjunto de teste satisfaz as condições de suficiência, garante-se que esse conjunto é completo, caso contrário não se pode afirmar nada a respeito da completude do conjunto.

Considerando a MEF da Figura 3.1 e os conjuntos $Q = \{\epsilon, a, ab\}$, $H_0 = \{a\}$, $H_1 = \{a, b\}$, $H_2 = \{a, b\}$, a primeira parte do método é equivalente à $Q \cdot H_i = \{aa, ab, aba, abb\}$, após remover os prefixos. Na segunda fase, o método H verifica as transições utilizando as condições de suficiência. O comprimento do conjunto gerado depende da ordem na qual as transições são verificadas. Se as transições não forem verificadas, identificadores de estado apropriados são selecionados em tempo de execução para diminuir o comprimento do conjunto de teste gerado. O conjunto de teste final é $TS_H = \{rba, raaa, raab, rabaa, rabab, rabba, rabbb\}$ de comprimento 31 e com 7 operações *reset*.

Para a MEF exemplo considerada, os métodos H e HSI produziram o mesmo conjunto de teste devido ao fato de que as sequências de separação geradas pelo método H foram as mesmas geradas pelo HSI, ou seja, não foi encontrada nenhuma condição durante a execução do método que pudesse reaproveitar alguma sequência do conjunto para fazer a separação entre dois estados. Contudo, nos experimentos realizados em Dorofeeva, El-Fakih e Yevtushenko (2005) mostrou-se que os conjuntos gerados pelo método H são, em média, 66% do tamanho dos conjuntos gerados pelo método HSI.

3.3.1.5 Método SPY

Definido por Simão, Petrenko e Yevtushenko (2009), o método SPY é um dos métodos mais recentes. Foi proposto com o objetivo de gerar um conjunto de teste com um número menor de sequências, usando uma estratégia que evita a criação de novas ramificações no conjunto referido. Assim como o método H, o método SPY também escolhe identificadores de estado durante a construção do conjunto de teste, de modo a estender casos de teste já existentes ao invés de criar novas ramificações. Usando o SPY, o conjunto de teste final para a MEF da Figura 3.1 é $TS_{SPY} = \{raaa, rabaa, rabba, rbaababbb\}$ de comprimento 23 e com 4 operações *reset*.

3.3.1.6 Método P

Outro método recente é o método P (SIMÃO; PETRENKO, 2010b), o qual inicialmente foi proposto para obter um conjunto n-completo incrementando um conjunto de teste já existente. Contudo, se for considerado um conjunto de teste que contém apenas a sequência vazia, o método P pode ser visto como um método de geração de sequências de teste.

Basicamente, o método trabalha iterativamente verificando as condições de suficiência definidas em Simão e Petrenko (2010b) e aplicando regras definidas para

derivar o conjunto de testes. Após remover os prefixos, o conjunto de teste final para a MEF da Figura 3.1 é TS_P = {rba, raaa, raab, rabaa, rabab, rabbb} de comprimento 26 e com 6 operações *reset*.

As características dos conjuntos de teste e o comprimento dos conjuntos de teste gerados por cada método citado acima foram alvos de comparação nos estudos experimentais realizados nos trabalhos de Dorofeeva, El-Fakih e Yevtushenko (2005), Dorofeeva et al. (2010) e Endo e Simao (2013). Os métodos foram comparados com a utilização de MEFs geradas aleatoriamente, mostrando que os métodos mais recentes produzem conjuntos de teste com menor custo.

3.3.2 Custo e cobertura de defeitos

Segundo Cutigi, Simao e Souza (2016), o custo de um conjunto de teste é determinado pelo seu tempo de execução; quanto maior for o comprimento de um conjunto de teste, mais tempo será gasto na sua execução. Além disso, o número de sequências (operações *resets*) também influencia no seu tempo de execução, uma vez que, a operação *reset* representa a execução de uma sequência que leva a MEF para o estado inicial. Sendo assim, o custo de um conjunto de teste é, geralmente, um fator dominante quando se avalia um método de geração de sequências de teste.

No que diz respeito à cobertura de defeitos, ou, eficácia, é avaliada a capacidade do conjunto de teste em revelar defeitos. As técnicas de análise de mutantes e condições de suficiência podem ser utilizadas para esse tipo de avaliação e serão detalhadas a seguir.

3.3.2.1 Análise de mutantes

A análise de mutantes visa avaliar a eficácia de um conjunto de teste *TS* com respeito a um programa A. Um conjunto de programas, denominados mutantes de A, é gerado incluindo modificações em A por meio de operadores de mutação, representando possíveis erros cometidos no processo de desenvolvimento.

Essa análise busca verificar se *TS* é capaz de revelar as diferenças entre o programa A e seus mutantes. Se o resultado da execução de um mutante for diferente do resultado do programa A, para qualquer caso de teste pertencente à *TS*, o defeito do mutante é relevado e é dito que *TS* "mata" esse mutante.

De acordo com Petrenko, Bochmann e Yao (1996), todas as MEFs pertencentes à $\Im_m(M)$ podem ser tratadas como mutantes de M. Um mutante é uma MEF resultante da aplicação de operadores de mutação em M, a qual pode ser uma máquina parcial. Os seguintes operadores de mutação são frequentemente usados e podem ser aplicados em

qualquer ordem, um determinado número de vezes (incluindo zero vezes) (PETRENKO; BOCHMANN; YAO, 1996):

- Tipo 1: alterar o estado final de uma transição (defeito de transferência);
- Tipo 2: alterar a saída de uma transição (defeito de saída);
- Tipo 3: adicionar uma transição (usada no contexto de máquinas parciais e não-determinísticas); e
- Tipo 4: adicionar um estado extra (quando $m > n$).

A capacidade de um conjunto de teste detectar implementações defeituosas, pode ser caracterizada por um número real entre 0 (o conjunto não revela nenhum defeito) e 1 (o conjunto é completo), calculada com o *score* de mutação. Dado um conjunto de teste TS e uma máquina de especificação M, o *score* de mutação (ou *fault coverage*) é definido como (PETRENKO; BOCHMANN; YAO, 1996):

$$FC(m, M, TS) = \frac{N_t(m, M) - N_p(m, M, TS)}{N_t(m, M) - N_c(m, M)} \tag{3-1}$$

onde $N_t(m, M)$ é o número de máquinas em $\Im_m(M)$ (total de mutantes), $N_p(m, M, TS)$ é o número de máquinas em $\Im_m(M)$ que passam no conjunto TS e $N_c(m, M)$ o número de máquinas que estão em conformidade com M. Assim, o *score* de mutação consiste na razão entre o número de mutantes mortos por TS e o número de mutantes que não estão em conformidade com a especificação.

3.3.2.2 Condições de suficiência

Outra maneira de avaliar a eficácia é o uso de condições de suficiência, que tem como objetivo garantir a completude dos testes, indicando sob quais condições um conjunto de teste é capaz de revelar todos os defeitos de um dado domínio. A completude de um conjunto de teste é comprovada quando o conjunto satisfaz as condições de suficiência. Caso as condições não sejam satisfeitas, não se pode fazer nenhuma afirmação a respeito da completude do conjunto.

Segundo Simão e Petrenko (2010a), existem outras técnicas para determinar a completude de um conjunto de teste, como abordagens exaustivas que enumeram explicitamente todas as possíveis MEFs defeituosas (POAGE; MCCLUSKEY, 1964 apud SIMÃO; PETRENKO, 2010a), por exemplo, ou todas as formas mínimas de uma MEF parcialmente especificada representando um conjunto de testes como uma árvore (YAO; PETRENKO; BOCHMANN, 1994). Contudo tais técnicas não escalam. Como alternativa, as condições de suficiência se apresentam como uma maneira menos complexa de determinar a completude de um conjunto de casos de teste. Além disso, condições que são mais fáceis de serem satisfeitas podem ser usadas para provar a comple-

tude de uma classe maior de testes e permitem aprimorar os métodos de geração de testes, reduzindo o tamanho do conjunto sem perder a eficácia na detecção de defeitos. Diversos trabalhos definiram condições de suficiência para determinar a completude de conjuntos de testes a partir de MEFs (PETRENKO; BOCHMANN; YAO, 1996; URAL; WU; ZHANG, 1997; DOROFEEVA; EL-FAKIH; YEVTUSHENKO, 2005; SIMÃO; PETRENKO; YEVTUSHENKO, 2009; SIMÃO; PETRENKO, 2010a; SIMÃO; PETRENKO, 2010b).

Os métodos **W**, W_p e HSI apesar de não fazerem uso explicitamente de condições de suficiência para garantia da completude do conjunto gerado, tiveram sua completude comprovada, por meio das condições de suficiência definidas por Petrenko, Bochmann e Yao (1996). Ou seja, tais métodos já satisfaziam tais condições de suficiência de maneira implícita.

De modo diferente, o método H constrói o conjunto de teste baseado nas condições de suficiência definidas por Dorofeeva, El-Fakih e Yevtushenko (2005), satisfazendo as condições explicitamente. Tais condições generalizam as condições de Petrenko, Bochmann e Yao (1996). Da mesma maneira, os métodos P e SPY também fazem uso explícito de condições de suficiência definidas em Simão e Petrenko (2010b) e Simão, Petrenko e Yevtushenko (2009), respectivamente.

Existem alguns métodos como Hierons e Ural (2002), Hierons e Ural (2006), Ural e Zhang (2006), Simão e Petrenko (2008), que tratam a geração de conjuntos de teste unitário, ou seja, que possuem apenas um caso de teste, dispensando assim a necessidade da operação *reset*. Para tais métodos, as condições de suficiência de Ural, Wu e Zhang (1997) podem ser aplicadas, visto que as condições de Dorofeeva, El-Fakih e Yevtushenko (2005) são aplicáveis apenas para conjuntos não unitários.

Dado que as condições de Dorofeeva, El-Fakih e Yevtushenko (2005) e Ural, Wu e Zhang (1997) são ortogonais, ou seja, não podem ser reduzidas umas às outras, Simão e Petrenko (2010a) definiram um conjunto de condições de suficiência que generalizam ambos os trabalhos. Sendo possível assim, provar a completude tanto de conjuntos unitários quanto de conjuntos que fazem uso da operação *reset*. Além disso, são menos restritivas podendo ser usadas para provar a completude de uma classe maior de testes.

Até o momento nenhum método de geração foi proposto considerando as condições de Simão e Petrenko (2010a), apenas abordagens de redução (CUTIGI; SIMAO; SOUZA, 2016). A partir de um conjunto de teste completo gerado por um método de geração clássico, as abordagens de redução combinam as sequências do conjunto e verificam se o novo conjunto obtido mantém a completude. A verificação da completude é baseada nas condições de suficiência definidas por Simão e Petrenko (2010a). Os resultados dos experimentos evidenciaram que é possível encontrar conjuntos de teste com menor custo, com a mesma eficácia, do que os até então encontrados pelos métodos de geração

do estado da arte. Por serem menos exigentes, ou seja, mais fáceis de serem satisfeitas, as condições propostas por Simão e Petrenko (2010a) permitiram que as abordagens de redução de Cutigi, Simao e Souza (2016) gerassem conjuntos de testes menores. Considerando isso, o método de geração proposto neste trabalho faz uso dessas condições as quais são detalhadas a seguir.

As condições de Simão e Petrenko (2010a) são baseadas nas propriedades de um conjunto confirmado, definido da seguinte forma:

Definição 3.1 (Conjunto confirmado) *Seja TS um conjunto de teste de uma MEF* $M = (S, s_0, X, Y, D_M, \delta, \lambda)$ *e* $K \subseteq TS$. *O conjunto K é confirmado se* $\delta(s_0, K) = S$ *e, para cada* $I = (U, u_0, X, Y', D_I, \Delta, \Lambda) \in \Im(M)$, *tal que I passa em TS, para todo* $\alpha, \beta \in K$, $\Delta(q_0, \alpha) = \Delta(q_0, \beta)$ *se e somente se* $\delta(s_0, \alpha) = \delta(s_0, \beta)$. *Uma sequência de entradas é confirmada se existe um conjunto confirmado que a contém.*

Em outras palavras, um conjunto confirmado contém sequências de transferência para todos os estados de M e se duas sequências atingem um mesmo estado (convergem) em M, então por definição, elas atingem um mesmo estado em qualquer MEF pertencente a $\Im(M)$ que passa no teste *TS*.

Os Lemas 3.1-3.3, propostos por Simão e Petrenko (2010a), indicam algumas maneiras de se construir um conjunto confirmado. O Lema 3.1 apresenta uma condição de suficiência para um conjunto *state cover* minimal ser um conjunto confirmado.

Lema 3.1 *Seja TS um conjunto de teste de uma MEF e K um conjunto state cover minimal. Se cada duas sequências de K são TS-distinguíveis entre si, então o conjunto K é confirmado.*

Os demais lemas definem condições de suficiência para adicionar uma sequência em um conjunto confirmado, de modo que o conjunto resultante continua mantendo a propriedade de ser confirmado e dessa forma pode-se construir conjuntos confirmados de maneira incremental.

Lema 3.2 *Seja K um conjunto de sequências confirmadas e* α *uma sequência de transferência para o estado s. Se para cada* $s' \in S \setminus \{s\}$, *existe* $\beta \in K$, $\delta(s_0, \beta) = s'$, *tal que* α *e* β *são TS-distinguíveis, então o conjunto* $K \cup \{\alpha\}$ *é confirmado.*

Lema 3.3 *Seja K um conjunto confirmado e* $\alpha \in TS$ *. Se existir* $\beta, \chi \in K$ *tal que* $\delta(s_0, \beta) = \delta(s_0, \chi)$, *e uma sequência* φ, *tal que* $\beta\varphi \in K$ *e* $\chi\varphi = \alpha$, *então o conjunto* $K \cup \{\alpha\}$ *também é confirmado.*

Por fim, o Teorema 3.1 define as condições de suficiência para um conjunto de teste ser completo.

Teorema 3.1 (Condições de suficiência para n-completude de um conjunto de testes) *Seja TS um conjunto de teste de uma MEF* $M = (S, s_0, X, Y, D_M, \delta, \lambda)$ *com n estados, inicialmente conectada e minimal. TS é n-completo para M, se existe um conjunto confirmado* $K \subseteq TS$ *com as seguintes propriedades:*

1. $\epsilon \in K$;
2. *Para cada* $(s, x) \in D_M$, *existem* $\alpha, \alpha x \in K$, *tal que* $\delta(s_0, \alpha) = s$.

Em outras palavras, um determinado conjunto de testes é completo para determinada MEF caso exista um conjunto confirmado que contenha a sequência vazia e cubra cada transição da MEF.

Um algoritmo para verificar a completude de um conjunto de teste *TS* baseado no Teorema 3.1 é proposto por Simão e Petrenko (2010a). Inicialmente, o algoritmo constrói um grafo de distinção de *TS*. Um grafo de distinção *G* de *TS* é um grafo cujos vértices são todos os prefixos das sequências em *TS*, de modo que dois vértices são adjacentes em *G* se e somente se as sequências correspondentes são *TS*-distinguíveis. Em seguida, conjuntos confirmados minimais são construídos aplicando o Lema 3.1. O problema de construir conjuntos confirmados minimais aplicando o Lema 3.1 é reduzido ao problema de se encontrar uma clique de tamanho *n* no grafo de distinção *G*. Após a construção dos conjuntos confirmados minimais, novas sequências são adicionadas aos mesmos utilizando os Lemas 3.2 e 3.3. E por fim, verifica-se se os conjuntos confirmados obtidos na etapa anterior satisfazem o Teorema 3.1. O Algoritmo 3.1, adaptado de Cutigi (2010), apresenta tais etapas em detalhes. Para exemplificar a aplicação do algoritmo será considerada a MEF da Figura 3.1 e o conjunto de teste *TS* = {rba,raaba,rabbb,rababa} de tamanho 19 e com 4 operações *resets*. Para facilitar a leitura, a MEF da Figura 3.1 é apresentada novamente na Figura 3.3.

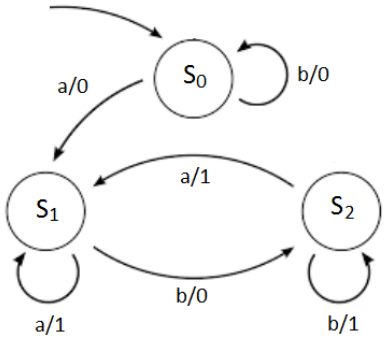


Figura 3.3: Exemplo de uma máquina de estados finitos

Algoritmo 3.1: Verificar a n-completude de um conjunto de teste (Fonte: adaptado de Cutigi (2010))

Entrada: MEF M e conjunto de teste TS
Saída: **Verdadeiro**, se TS é n-completo de acordo com o Teorema 3.1

```
1  Construir grafo de distinção G de TS
2  Determinar uma clique de tamanho n em G e inserir os nós da clique no
     conjunto K
3  se encontrou clique então
4      enquanto encontrar uma sequência α ∈ TS \K, em que os Lemas 3.2
         ou 3.3 podem ser aplicados faça
5          Inserir α em K
6      fim
7      se K satisfaz o Teorema 3.1 então
8          retorne verdadeiro
9      fim
10 senão
11     retorne falso
12 fim
```

Primeiramente, na linha 1 do Algoritmo 3.1 cria-se o grafo de distinção G. A Figura 3.4 ilustra o grafo de distinção para o exemplo proposto. Em um grafo de distinção G de TS, cada nó do grafo representa uma sequência de TS que leva a um estado da MEF. Por exemplo os nós ϵ e a levam aos estados S_0 e S_1, respectivamente. As arestas de G representam pares de sequências TS-distinguíveis, ou seja, sequências $\alpha\gamma$, $\beta\gamma \in TS$, tal que $\lambda(\delta(S_0,\alpha),\gamma) \neq \lambda(\delta(S_0,\beta),\gamma)$. Considerando $\alpha = \epsilon$, $\beta = a$ e $\gamma = a$, temos que $\lambda(\delta(S_0,\epsilon),a) = \lambda(S_0,a) = 0$ e $\lambda(\delta(S_0,a),a) = \lambda(S_1,a) = 1$. Portanto, como $\lambda(\delta(S_0,\epsilon),a) \neq \lambda(\delta(S_0,a),a)$, ϵ e a são TS-distinguíveis e estão ligados no grafo de distinção da Figura 3.4.

Todos os nós ligados no grafo representam estados diferentes na MEF, uma vez que produziram saídas diferentes para determinadas entradas. Contudo, os nós que não estão ligados não necessariamente representam um mesmo estado, apenas um subconjunto destes nós representam sequências que levam a um mesmo estado.

Um grafo k-partido se trata de uma categoria de grafos na qual os nós podem ser divididos em k conjuntos distintos, sendo que os nós de um mesmo conjunto não possuem arestas entre si. Dessa forma, o grafo de distinção é um grafo n-partido, onde n é o número de estados da máquina, uma vez que o conjunto dos nós que representam sequências que levam a um mesmo estado não estarão ligados no grafo. Observa-se na Figura 3.4 os três conjuntos de nós que levam a cada estado da MEF, caracterizando um grafo 3-partido. As

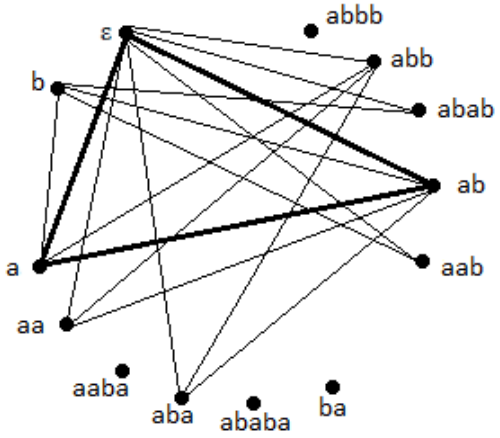


Figura 3.4: Grafo de distinção de TS = {rba, raaba, rabbb, rababa}

sequências do conjunto {ϵ,b} levam ao estado S_0, do conjunto {a, aa, aaba, aba, ababa, ba} levam ao estado S_1 e do conjunto {aab, ab, abab, abb, abbb} levam ao estado S_2.

Na linha 2 do Algoritmo 3.1 é determinado uma clique de tamanho n. Uma clique de um grafo é um subconjunto de seus vértices tais que cada dois vértices do subconjunto são conectados por uma aresta. O conjunto de sequências que aparecem em uma clique de tamanho n é um conjunto confirmado, uma vez que, como todos os nós estão ligados, tem-se a certeza que são estados distintos entre si, satisfazendo as condições do Lema 3.1. As arestas destacadas na Figura 3.4 indicam uma clique, a qual gera o conjunto confirmado inicial K. O conjunto de sequências extraído da clique é um conjunto *state cover* da MEF que, nesse caso, é K = {ϵ, a, ab}.

Segundo Simão e Petrenko (2010a), apesar do problema de encontrar n-cliques em um grafo arbitrário ser NP-completo, várias propriedades do grafo de distinção podem ser usadas, como o fato de ser n-partido, para formular heurísticas que permitem lidar com grafos maiores, como por exemplo, as heurísticas propostas em (GRUNERT et al., 2002 apud SIMÃO; PETRENKO, 2010a).

Uma vez gerado um conjunto confirmado inicial, pode-se expandir esse conjunto inserindo (confirmando) novas sequências por meio dos lemas 3.2 e 3.3 (linhas 4 e 5). Considerando uma clique de tamanho n, o Lema 3.2 pode ser interpretado da seguinte maneira: se uma sequência α ainda não confirmada está ligada, no grafo de distinção, a um conjunto de $n-1$ sequências que fazem parte da clique, então α é confirmado. Dado o grafo de distinção da Figura 3.4, a sequência aa pode ser confirmada, uma vez que ela está ligada com as sequências ϵ e ab, as quais fazem parte da clique K. Da mesma forma, as sequências b, aba, abb podem ser confirmadas aplicando o Lema 3.2 e o conjunto confirmado resultante é K = {ϵ, a, b, aa, ab, aba, abb}.

Quando não é mais possível a aplicação do Lema 3.2, o Lema 3.3 pode ser aplicado como uma maneira alternativa para confirmar novas sequências. Essa alternativa

confirma uma sequência α, por meio de sequências β e χ já confirmadas. Se χ é prefixo de α, ou seja $\alpha = \chi\varphi$, e leva ao mesmo estado de β, e $\beta\varphi$ gera uma sequência confirmada, então se confirma α. Por exemplo, considere a sequência α = abbb ainda não confirmada. Dado χ = abb e β = ab, sequências confirmadas e que levam ao mesmo estado, como χ é um prefixo de α, e portanto φ = b, e $\beta\varphi$ = abb é uma sequência confirmada, então a sequência α = abbb também pode ser confirmada, ou seja, ser inserida no conjunto K. Da mesma maneira, mais cinco sequências podem ser confirmadas com o Lema 3.3, conforme a Tabela 3.1.

Tabela 3.1: Confirmação de sequências aplicando o Lema 3.3

			Conjunto resultante
α = ba	χ = b φ = a	$\beta = \epsilon$ $\beta\varphi$ = a	K = {ϵ, a, b, aa, ab, ba, aba, abb, abbb}
α = aab	χ = aa φ = b	β = a $\beta\varphi$ = ab	K = {ϵ, a, b, aa, ab, ba, aba, aab, abb, abbb}
α = aaba	χ = aa φ = ba	β = a $\beta\varphi$ = aba	K = {ϵ, a, b, aa, ab, ba, aba, aab, abb, aaba, abbb}
α = abab	χ = aba φ = b	β = a $\beta\varphi$ = ab	K = {ϵ, a, b, aa, ab, ba, aba, aab, abb, aaba, abab, abbb}
α = ababa	χ = aba φ = ba	β = aa $\beta\varphi$ = aaba	K = {ϵ, a, b, aa, ab, ba, aba, aab, abb, aaba, abab, abbb, ababa}

Assim, após a aplicação do Lema 3.3 tem-se o conjunto confirmado K = {ϵ, a, b, aa, ab, ba, aba, aab, abb, aaba, abab, abbb, ababa} que contém todas sequências do grafo de distinção. Caso restassem sequências ainda não confirmadas, o Lema 3.2 poderia ser utilizado novamente, uma vez que as sequências confirmadas pelo Lema 3.3 podem abrir mais possibilidades de confirmação.

Por fim, na linha 7 é verificado se o conjunto K satisfaz o Teorema 3.1. K contém ϵ, resta verificar se existem sequências α e αx para todo par $(s, x) \in D_M$. A Tabela 3.2 apresenta todas as sequências α e αx que cobrem todas as transições da MEF. Portanto, de acordo com as condições de suficiência de Simão e Petrenko (2010a), prova-se a completude do conjunto exemplo TS = {rba, raaba, rabbb, rababa}.

Com os dados apresentados na Tabela 3.3, é possível observar que esse conjunto do exemplo é menor, tanto em comprimento quanto no número de *resets*, que todos os conjuntos gerados pelos métodos apresentados na Seção 3.3.1.

Tabela 3.2: Verificação da cobertura de transições do conjunto de sequências confirmadas

Par estado/entrada	**Sequência** α	**Sequência** αx
(S_0, a)	ϵ	a
(S_0, b)	ϵ	b
(S_1, a)	a	aa
(S_1, b)	a	ab
(S_2, a)	ab	aba
(S_2, b)	ab	abb

Tabela 3.3: Comparação do custo dos conjuntos de teste

Método	**Comprimento**	**Número de *resets***
W	34	8
W_p	29	7
HSI	31	7
H	31	7
SPY	23	4
P	26	6
Exemplo	19	4

3.4 Considerações finais

Neste capítulo foram apresentados os principais conceitos e definições da técnica de teste baseada em máquinas de estados finitos, na qual esse trabalho está inserido. O teste baseado em MEFs auxilia a tarefa de geração de casos de teste eficientes por meio da especificação formal do *software* sob teste.

Foram introduzidos e exemplificados alguns dos principais métodos para geração de casos de teste utilizando MEFs, dentre eles, o **W**, W_p, HSI, H, SPY e P. Tais métodos possuem a característica importante de gerar conjuntos de teste com cobertura de defeitos garantida.

É possível observar, de um método para outro, uma tentativa de melhoria da qualidade do conjunto gerado, reduzindo o custo e mantendo a completude. Por esse motivo, os métodos mais recentes produzem conjuntos de teste menores. Contudo, Cutigi, Simao e Souza (2016) mostrou que é possível satisfazer a completude com um conjunto de teste ainda menor, em comprimento e/ou número de sequências, que os gerados por tais métodos. Portanto, o estado da arte ainda não soluciona o problema da maneira mais eficiente. Existem soluções melhores a serem encontradas o que justifica o desenvolvimento de um método de geração que seja capaz de alcançá-las.

Dito isso, o intuito do presente trabalho é propor um método de geração de sequências de teste que: 1) revele o máximo de defeitos possível, 2) possua o menor com-

primento possível e 3) contenha o menor número de sequências possível. Considerando que esse problema é NP-Difícil, será utilizada uma metaheurística para tal propósito.

No próximo capítulo serão apresentados conceitos e princípios necessários para o projeto de uma metaheurística eficiente.

CAPÍTULO 4

Projeto de Metaheurísticas

4.1 Considerações iniciais

Segundo Rothlauf (2011), as metaheurísticas são métodos poderosos para resolver um problema específico de otimização, cujo objetivo é encontrar uma solução ótima ou quase ótima com um esforço computacional baixo. Contudo, para obter um alto desempenho, deve-se projetar a metaheurística de uma maneira mais específica para o problema, de modo a explorar o conhecimento específico do problema no projeto de seus elementos básicos.

Neste capítulo são apresentados os principais conceitos e definições sobre metaheurísticas, bem como os principais elementos de projeto para a construção de uma metaheurística de alto desempenho. Os conceitos introduzidos neste capítulo fundamentam a pesquisa realizada nesta tese no que diz respeito ao tema de metaheurísticas e seu projeto.

4.2 Fundamentos de Metaheurísticas

Luke (2013) define as metaheurísticas como métodos utilizados em problemas sobre os quais há poucas informações: não se sabe com o que uma solução ótima se parece, há pouca informação heurística disponível e força-bruta é desconsiderada devido ao espaço de solução ser muito grande. Porém, dada uma solução candidata ao problema, ela pode ser testada e sua otimalidade averiguada. De acordo com Rothlauf (2011), metaheurísticas, também conhecidas como heurísticas modernas, podem ser facilmente aplicadas a problemas que são próximos do mundo real, encontrando boas soluções, mas sem a garantia de que as soluções encontradas sejam ótimas.

Assim como as heurísticas tradicionais, as metaheurísticas executam etapas de melhoria, ou seja, caminham em direção a melhores soluções para o problema, porém as metaheurísticas também permitem que soluções inferiores sejam geradas durante a busca. Considerando isso, Rothlauf (2011) define as metaheurísticas, como heurísticas de

propósito geral que usam durante a busca fases de intensificação (do inglês, *exploitation*) e diversificação (do inglês, *exploration*). Na fase de intensificação, modificações são realizadas em soluções existentes de alta qualidade com o intuito de concentrar a busca em áreas promissoras do espaço de soluções. Na diversificação, são realizadas modificações nas soluções existentes, aceitando soluções inferiores, de forma que novas áreas do espaço de busca sejam exploradas.

Uma metaheurística normalmente inicia com uma ou mais soluções de um problema geradas aleatoriamente. Em seguida, em etapas iterativas, são realizadas modificações na(s) solução(ões) existente(s) gerando uma ou mais soluções novas, por meio dos operadores de busca. Essas etapas se repetem até que um determinado nível de qualidade é atingido ou vários passos de busca são executados.

Existe uma grande variedade de metaheurísticas, dentre elas, algoritmos evolutivos, *simulated annealing*, busca tabu, colônia de formigas e entre outras. No entanto, conforme Rothlauf (2011), muitas vezes as diferenças entre esses métodos são pequenas e conceitos semelhantes são apresentados usando rótulos diferentes. Neste trabalho, serão utilizados os conceitos de algoritmos evolutivos. A seguir são apresentados esses conceitos, bem como definições comuns entre os métodos.

4.2.1 Conceitos e definições

Os algoritmos evolutivos (AEs) caracterizam-se como metaheurísticas inspiradas nos princípios da teoria de evolução e seleção natural da espécie. Em geral, um AE consiste em uma *população* de soluções codificadas (indivíduos) *selecionadas* e manipuladas por um conjunto de *operadores* e avaliadas por alguma *função fitness* (COELLO; LAMONT; VELDHUIZEN, 2007). Em outras palavras, uma população de indivíduos passa por algumas transformações, e durante esse processo de evolução, os indivíduos lutam pela sua sobrevivência.

A *população* é um conjunto de soluções do problema. Uma estrutura ou *indivíduo* é uma solução codificada. Tipicamente, um indivíduo é representado como um *array* binário e sua forma codificada é chamada de *cromossomo*. O cromossomo é dito ser o *genótipo* da solução e este genótipo quando expresso (decodificado) no domínio do problema é chamado de *fenótipo*. Cada cromossomo é composto de *genes*, isto é, características da solução do problema, que assumem certos valores (*alelos*) de algum alfabeto genético. Um *locus* identifica a posição de um gene dentro do cromossomo (COELLO; LAMONT; VELDHUIZEN, 2007). Esses conceitos são retratados na Figura 4.1.

O ciclo de evolução de um AE ocorre da seguinte maneira: dada uma população inicial de indivíduos, a mudança de características dos indivíduos, ao longo de iterações, produz uma nova população. Essas mudanças são ocasionadas por meio de operadores de

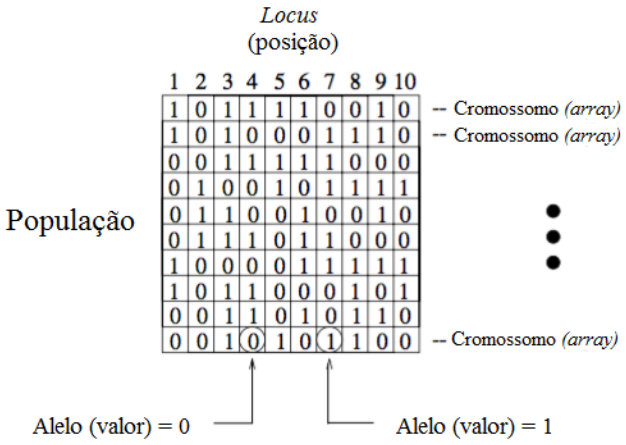


Figura 4.1: Estrutura de dados generalizada e terminologia de AEs (Fonte: adaptado de Coello, Lamont e Veldhuizen (2007))

reprodução aplicados aos cromossomos. A seleção dos cromossomos para reprodução é baseada em uma função, denominada de função de aptidão, avaliação ou *fitness*, a qual tem a capacidade de medir a aptidão (qualidade) de cada indivíduo. Os indivíduos com melhor aptidão possuem maior probabilidade de serem selecionados como pais, ou seja, participarem do processo de reprodução, e transferirem suas características para novos indivíduos (filhos), aumentando a qualidade da população a cada geração. O valor de aptidão também é utilizado para determinar quais indivíduos irão permanecer (sobreviver) na população na próxima geração (BACK; FOGEL; MICHALEWICZ, 1999).

De maneira mais simplista, Jong (2006) descreve a dinâmica dos AEs como uma população de tamanho p que é mantida durante todo o processo evolutivo, a população corrente é, então, usada como uma fonte de pais para produzir f filhos e, em seguida, a população expandida é reduzida de $p+f$ para p indivíduos.

Como os EAs trabalham com um conjunto de soluções, existe um paralelismo implícito caracterizado pela avaliação simultânea de vários pontos do espaço de soluções potenciais para um problema. Uma solução para o problema é representada como um vetor $x = (x_1, \ldots, x_i)$ de i variáveis de decisão, com uma atribuição de valores específicos para cada uma das variáveis de decisão. O conjunto de soluções de um problema é denominado *espaço de busca*.

Um espaço de busca pode definir relações (por exemplo, distâncias) entre soluções (ROTHLAUF, 2011). Dessa forma, é possível definir semelhanças entre soluções baseadas na distância entre elas. Uma *vizinhança* determina quais soluções são semelhantes entre si.

Rothlauf (2011) define vizinhança como um mapeamento $N(x) : \hat{X} \rightarrow 2^{\hat{X}}$ em que $\hat{X}$ é o espaço de busca e $2^{\hat{X}}$ significa o conjunto de todos os possíveis subconjuntos de $\hat{X}$. Em outras palavras, a vizinhança $N(x)$ é um mapeamento que atribui a cada solução $x \in$

$\hat{X}$ um conjunto de soluções $y \in \hat{X}$ que são vizinhas de x, ou seja, são em algum sentido similares a x.

A *localidade* descreve quão bem a distância entre duas soluções reflete a diferença de *fitness* de ambas. A localidade de um problema é alta se soluções vizinhas possuem valores de *fitness* semelhantes. Caso pequenas distâncias não correspondam a diferenças pequenas dos valores *fitness*, então o problema possui baixa localidade.

Conforme Rothlauf (2011), problemas com alta localidade permitem que métodos de busca guiada obtenham um alto desempenho, uma vez que tais métodos fazem uso dos valores de *fitness* de soluções encontradas anteriormente para guiar o processo de busca. Assim, é possível encontrar soluções de alta qualidade na vizinhança de boas soluções já encontradas. Enquanto que em problemas de baixa localidade, os métodos de busca guiada possuem desempenho similiar aos métodos de busca aleatória, pois não é possível tirar proveito de informações extraídas de soluções anteriores.

Assim como a localidade, outra propriedade inerente de um problema é a *decomposição*, a qual descreve quão bem um problema pode ser decomposto em vários subproblemas menores que são independentes uns dos outros. A decomposição de um problema é dita alta se a estrutura da função *fitness* é tal que nem todas as variáveis de decisão devem ser simultaneamente consideradas para o cálculo da função. Um problema tem baixa decomposição se não for possível decompor um problema em subproblemas com poucas interdependências entre os grupos de variáveis (ROTHLAUF, 2011).

4.2.2 Teorema *No Free Lunch*

Em 1995, Wolpert e Macready (1995) definiram o teorema *No Free Lunch* (NFL) baseado na expressão popular "*There is no free lunch*" que expressa a ideia de que é impossível conseguir algo sem dar nada em troca. O teorema NFL afirma que um algoritmo de propósito geral, na qual o algoritmo não faz uso de informações aprendidas sobre um problema em particular, não é eficiente. Em outras palavras, quanto mais problemas puderem ser resolvidos com um método de otimização específico, menor será seu desempenho médio.

Em 1997, Wolpert e Macready (1997) estabeleceram que qualquer algoritmo que possui desempenho elevado em algum tipo de problema irá pagar em desempenho com relação a outro tipo de problema. Ou seja, isolando uma classe de problemas em que o algoritmo A é mais eficiente que o algoritmo B, o algoritmo B seria mais eficiente que A em outra classe de problemas.

Em resumo, o teorema NFL afirma que é impossível criar uma estratégia de otimização de propósito geral com desempenho superior em todos os problemas em

relação à métodos específicos. A única forma de uma estratégia superar outra é explorar propriedades específicas de um problema.

Por ser de propósito geral, é característico de qualquer metaheurística que sua funcionalidade seja independente de problemas, facilitando a aplicação de tal método a diferentes domínios de aplicação. Contudo, pelo teorema NFL há um *trade-off* entre a eficácia e a capacidade de generalização dos métodos de otimização. Portanto, de acordo com Rothlauf (2011), as heurísticas modernas tradicionais, que não são específicas para um problema, geralmente funcionam apenas para problemas pequenos e à medida que o problema se torna maior e mais realista, o desempenho diminui.

Para superar essa limitação, a metaheurística deve ser projetada de uma maneira que a torne capaz de explorar propriedades específicas do problema. Assim, Rothlauf (2011) afirma que para desenvolver heurísticas modernas de alta qualidade é necessário a compreensão dos conceitos gerais por trás dos principais elementos de projeto, dentre eles: representação, operador de busca, inicialização e função *fitness*. Tais elementos serão detalhados na próxima seção.

4.3 Elementos de projeto

Para aumentar o desempenho de uma heurística moderna, deve-se compreender os elementos básicos de projeto e como o conhecimento específico do problema pode ser usado no projeto dos mesmos. De acordo com Rothlauf (2011), não é a escolha de um tipo particular de heurística moderna que é relevante para o alto desempenho, mas sim, uma consideração apropriada do conhecimento específico do problema, usando-o para o projeto de representações, operadores de pesquisa, soluções iniciais e *fitness*.

4.3.1 Representação

A representação é caracterizada como um mapeamento entre o genótipo e o fenótipo. O genótipo define a forma codificada de um indivíduo e permite descrever o indivíduo no nível de genes. O fenótipo define o indivíduo no domínio do problema, descreve a aparência que o conjunto de informações armazenados no cromossomo representa. O mapeamento genótipo-fenótipo (representação) permite construir o fenótipo a partir da informação contida no genótipo e vice-versa (ROTHLAUF, 2011).

Segundo Rothlauf (2011), esse mapeamento é necessário pois há uma distinção entre os espaços de busca nos quais os operadores de busca e a função objetivo podem ser aplicados. Quando se trata da aplicação de operadores de busca, as soluções devem estar no nível genotípico. Enquanto que para avaliar a qualidade de uma solução o genótipo

precisa ser mapeado para o fenótipo, pois o valor objetivo de uma solução é determinado pela sua aparência fenotípica.

A aplicação bem sucedida e eficiente da heurística moderna depende da escolha adequada do mapeamento genótipo-fenótipo, ou seja codificação e decodificação do indivíduo (ROTHLAUF, 2011). Nesse tipo de heurística os mapeamentos são necessários para a aplicação dos operadores de busca e da função objetivo, respectivamente, durante o processo de busca.

Alguns benefícios do uso de representações são o reuso de genótipos/operadores de busca e redução do espaço de busca. No primeiro caso, um mesmo genótipo pode ser aplicado para diferentes tipos de problemas somente interpretando-os diferentemente usando um mapeamento genótipo-fenótipo de cada problema. Assim, não é necessário desenvolver nenhum novo operador, pois pode-se utilizar operadores de busca tradicionais, com propriedades e comportamentos conhecidos, correspondentes ao genótipo utilizado. No segundo caso, em problemas de árvore, por exemplo, em que a solução ótima é uma árvore estrela, pode-se utilizar genótipos que codificam apenas árvores estrela, resultando em um espaço de busca muito menor.

4.3.2 Operadores de busca

De forma similar ao processo reprodutivo que ocorre na natureza, novos indivíduos podem ser obtidos por meio de operadores de reprodução (busca) aplicados em soluções existentes. De acordo com Rothlauf (2011), os operadores de busca podem ser distinguidos entre *operadores de busca local* e *operadores de recombinação*, e simulam o processo de reprodução assexuada e sexuada, respectivamente. No processo assexuado, um único pai (solução) é necessário enquanto que no sexuado são necessários pelo menos dois pais. Os operadores de busca local geram soluções semelhantes a uma solução existente e os operadores de recombinação criam um nova solução recombinando propriedades de soluções existentes.

O objetivo do uso de operadores de busca local é realizar pequenas alterações iterativamente, criando novas soluções com distância mínima para seus pais, considerando a hipótese de que soluções de alta qualidade estão agrupadas no espaço de busca. Dessa forma, soluções melhores podem ser encontradas pesquisando na vizinhança de boas soluções já encontradas. Os passos de busca (alterações) não podem ser muito grandes, pois impossibilitariam a busca guiada em torno de boas soluções e resultariam em uma busca aleatória (ROTHLAUF, 2011).

Os operadores baseados em recombinação assumem que os problemas podem ser decompostos em subproblemas menores. Tem como objetivo formar novas soluções recombinando soluções de subproblemas menores que existem em diferentes pai. Em

outras palavras, a busca baseada em recombinação resolve problemas decomponíveis decompondo-os em subproblemas menores, resolvendo os subproblemas menores separadamente e combinando as soluções resultantes dos subproblemas para formar soluções globais. A aplicação de operadores de recombinação resulta em descendentes em que a distância entre os pais é maior ou igual que as distâncias entre os filhos e seus pais.

Assim como existem genótipos tradicionais, como *array* binário, inteiro e real, também existem operadores de busca tradicionais que podem ser aplicados considerando a escolha do genótipo. Quando um genótipo tradicional for utilizado para codificar o indivíduo, os operadores de busca tradicionais podem ser usados. Caso contrário, operadores especializados são necessários para permitir aos filhos herdarem propriedades importantes de seus pais. Em geral, esses operadores são específicos para o problema e devem ser desenvolvidos separadamente para cada problema de otimização.

Para o projeto de operadores de busca apropriados, Radcliffe introduziu o conceito de *Formae* e propôs um conjunto de diretrizes de projeto, os quais foram desenvolvidos em uma série de artigos (RADCLIFFE, 1991a; RADCLIFFE, 1991b; RADCLIFFE, 1992a; RADCLIFFE, 1992b; RADCLIFFE, 1993; RADCLIFFE, 1994). *Formae* são subconjuntos do espaço de busca e são definidos como classes de equivalência induzidas por um conjunto de relações de equivalência. Por exemplo, em um espaço de busca de cadernos, relações básicas de equivalência podem ser “mesma cor de capa” ou “mesma quantidade de matérias”, o que induziria o *formae* “capa azul”, “capa rosa”, “dez matérias”, etc.

Segundo Rothlauf (2011), a diretriz mais importante é que o *formae* gerado deve agrupar soluções com *fitness* similares, a fim de criar uma estrutura do espaço de busca que possa ser explorada pelos operadores de busca. Assumindo que os operadores de busca criam novas soluções a partir de um conjunto de soluções pai, Radcliffe (1991a), Radcliffe e Surry (1994) formularam os quatro princípios a seguir:

- Respeito (do inglês, *respect*): descendentes gerados por recombinação devem ser membros dos *formae* comuns aos seus pais. Por exemplo, no espaço de busca de cadernos, um descendente deve ter capa rosa e dez matérias se ambos os pais tiverem capa rosa e dez matérias.
- Transmissão (do inglês, *transmission*): um descendente deve ser equivalente a pelo menos um dos seus pais sob cada relação de equivalência. Exemplificando, se um dos pais tiver capa rosa e o outro capa azul, o descendente terá capa rosa ou azul.
- Escolha (do inglês, *assortment*): um descendente deve ser formado com características compatíveis provenientes dos pais. Caso haja características incompatíveis, o descendente herdará uma característica compatível de um pai aleatório. Por exemplo, os descendentes herdam a característica capa azul do primeiro pai e dez matérias do segundo pai apenas se capa azul e dez matérias forem compatíveis.

- Ergocidade (do inglês, *ergodicity*): deve ser possível, através de uma sequência finita de aplicações dos operadores de busca, acessar qualquer ponto no espaço de busca a partir de qualquer população.

4.3.3 Inicialização

Um método de inicialização provê um ou mais pontos de partida para a busca. Se forem utilizadas abordagens de busca local uma única solução inicial é suficiente. Caso seja utilizado abordagens baseadas em recombinação um conjunto maior de soluções iniciais é necessário, visto que tais abordagens têm como objetivo recombinar propriedades de diferentes soluções.

De acordo com Rothlauf (2011), as soluções iniciais podem ser geradas de forma aleatória ou com alguma tendência (do inglês, *bias*) para certos tipos de solução. A escolha da estratégia de inicialização depende do conhecimento que se tem a respeito do problema. Caso não se tenha nenhum conhecimento específico do problema ou sobre as propriedades das soluções de alta e baixa qualidade, é recomendado o método aleatório que escolhe as soluções com igual probabilidade e distribui os pontos de partida uniformemente no espaço de busca. Nesse caso, a inicialização é sem tendência e não favorece a escolha de alguns tipos de soluções. Caso se tenha conhecimento específico do problema, as soluções iniciais podem ser criadas considerando esse conhecimento de modo a iniciar a busca na região de soluções de alta qualidade.

A escolha da estratégia de inicialização determina a diversidade da população inicial e pode impactar no desempenho da metaheurística. A diversidade de uma população é alta se a distância média entre as soluções é alta; e é baixa caso a distância média é baixa. A baixa diversidade pode ser resultado de uma população muito pequena ou soluções iniciais geradas com tendência, pois a tendência geralmente concentra a busca em uma parte específica do espaço de busca (ROTHLAUF, 2011).

Como as populações iniciais com tendência costumam ter baixa diversidade, em relação à inicialização aleatória, nem todas as partes relevantes do espaço de busca podem ser exploradas. Assim, reduz-se a probabilidade das abordagens de busca local e baseada em recombinação encontrarem soluções de alta qualidade que tenham uma grande distância para as soluções iniciais. Ademais, a baixa diversidade é um problema para abordagens baseadas em recombinação, pois as propriedades distintas que podem ser recombinadas para formar novas soluções é limitada e talvez não seja possível formar soluções ótimas por recombinação (ROTHLAUF, 2011).

Dito isso, ao definir uma estratégia de inicialização, inicialmente é preciso identificar possíveis conhecimentos que podem ser incorporados de modo a inserir uma tendência para soluções de alta qualidade. Caso haja esse conhecimento, deve-se preocupar

em gerar a população com diversidade suficiente para garantir que os operadores de busca funcionem adequadamente e sejam capazes de encontrar soluções diferentes da população inicial. Do contrário, caso o conhecimento sobre a estrutura de soluções de alta qualidade ou baixa qualidade seja confuso ou insuficiente, deve-se evitar a inserção de tendência uma vez que pode prejudicar o desempenho do processo de busca.

4.3.4 Função *fitness*

A função *fitness* permite avaliar e estimar a capacidade relativa de um indivíduo solucionar o problema. Heurísticas modernas utilizam a função *fitness* para comparar a qualidade das soluções. A avaliação da qualidade permite selecionar os indivíduos para reprodução e/ou sobrevivência e deve ser capaz de guiar a busca.

Frequentemente a função *fitness* é equivalente à função objetivo, também conhecida como função de avaliação. De acordo com Rothlauf (2011), em geral, a função objetivo baseia-se no problema e na formulação do modelo, enquanto que a função *fitness* mede a qualidade das soluções na perspectiva da heurística moderna.

Ao projetar uma função *fitness*, deve-se preocupar com dois aspectos: a atribuição dos valores objetivo e o custo do cálculo da função. Com respeito à atribuição dos valores objetivo, deve-se assegurar que as soluções ótimas (soluções que atendem completamente ao objetivo) têm a melhor avaliação e soluções que têm qualidade inferior à solução ótima também têm um valor objetivo menor. Além disso, soluções de qualidade similar devem ter valores objetivo semelhantes. Com respeito ao custo, como as metaheurísticas geralmente realizam um grande número de avaliações de soluções, o cálculo da funcão não deve consumir muito tempo.

Um problema de otimização com um único objetivo envolve uma única função objetivo e geralmente resulta em uma única solução, chamada solução ótima (BRANKE et al., 2008). Por outro lado, um problema de otimização multiobjetivo considera simultaneamente vários objetivos, que podem ser conflitantes entre si, e portanto, pode não haver uma única solução que seja ótima para todos os objetivos. Nesse caso, espera-se como resultado do problema um conjunto de soluções com diferentes compensações (melhor em um objetivo e pior em outro). A seção a seguir aborda alguns conceitos dos problemas de otimização multiobjetivo.

4.4 Otimização Multiobjetivo

Um problema de otimização multiobjetivo lida com mais de uma função objetivo, que devem ser maximizadas ou minimizadas, bem como podem haver restrições que

devem ser satisfeitas para que uma determinada solução seja factível. De forma geral, Deb e Kalyanmoy (2001) definem o problema de otimização multiobjetivo como:

$$\begin{aligned} \text{Maximizar/Minimizar} \quad & f_o(x), && o = 1,2,\ldots,O; \\ \text{sujeito a} \quad & g_d(x) \geq 0, && d = 1,2,\ldots,D; \\ & h_j(x) = 0, && j = 1,2,\ldots,J; \\ & x_i^{(L)} \leq x_i \leq x_i^{(B)}, && i = 1,2,\ldots,I. \end{aligned} \tag{4-1}$$

Existem O funções objetivo consideradas na formulação acima. Uma solução x é um vetor de I variáveis de decisão $x = (x_1, x_2, \ldots, x_I)^T$ tal que $x \in \hat{X}$ (espaço de variável de decisão). As funções $g_d(x)$ e $h_j(x)$ compõem o conjunto de restrições associadas ao problema, com D e J restrições de desigualdade e igualdade, respectivamente. O último conjunto de restrições define os limites inferiores ($x_i^{(L)}$) e superiores ($x_i^{(B)}$) para cada variável de decisão x_i. Uma solução x que satisfaz o conjunto de restrições associado ao problema é denominada de solução factível, caso contrário é uma solução infactível.

Em problemas de otimização multiobjetivo, além do espaço de variáveis de decisão $\hat{X}$, as funções objetivo constituem um espaço multi-dimensional adicional, chamado de espaço objetivo Z. Para cada solução $x \in \hat{X}$, existe um ponto no espaço objetivo Z (imagem de $\hat{X}$), denotado por $f(x) = z = (z_1, z_2, \ldots, z_O)^T$ (DEB; KALYANMOY, 2001). A Figura 4.2 ilustra os dois espaços e o mapeamento do espaço de variável de decisão I-dimensional $\hat{X}$ para o espaço objetivo o-dimensional Z. Ainda na Figura 4.2, as regiões em cinza representam o espaço de decisão factível X^*, formado apenas por soluções factíveis, e o espaço objetivo factível (imagem de X^*), denotado por Z^*.

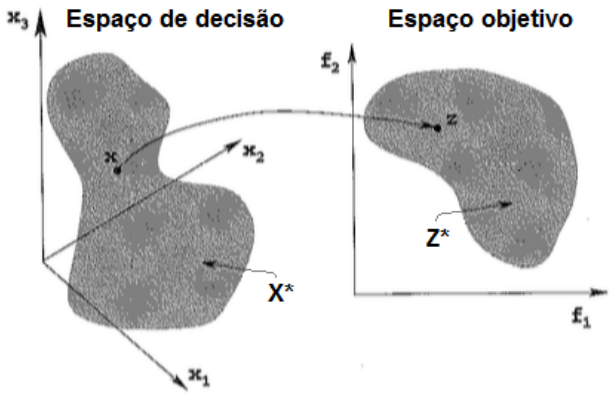


Figura 4.2: Representação do espaço de variável de decisão e o espaço objetivo correspondente. (Fonte: modificado de Deb e Kalyanmoy (2001))

Os algoritmos propostos para o problema de otimização multiobjetivo podem ser classificados em dois tipos: métodos baseados em preferência e métodos de geração (COHON, 1985 apud DEB; KALYANMOY, 2001). Nos métodos baseados em preferência, caso exista alguma informação de preferência *a priori*, como o nível de importância

de cada objetivo, esta informação é, então, usada no processo de otimização. Nos métodos de geração, nenhum conhecimento *a priori* da importância de cada objetivo é utilizado. Neste tipo de método, um conjunto de soluções não-dominadas (soluções ótimas de Pareto) é gerado para o tomador de decisor, que escolhe, então, uma solução a partir deste conjunto (DEB; KALYANMOY, 2001).

Geralmente, as abordagens baseadas em preferências transformam, de diferentes maneiras, os problemas multiobjetivo em problemas de um único objetivo (YU; GEN, 2012). O procedimento de transformação de múltiplos objetivos em um único pode ser realizado, por exemplo, pelo método de soma ponderada. Este método é a abordagem mais simples e é provavelmente a abordagem clássica mais amplamente utilizada (DEB; KALYANMOY, 2001).

Nas seções a seguir serão detalhadas a abordagem de soma ponderada (abordagem baseada em preferência) e os conceitos de soluções ótimas de Pareto, os quais são a base para os métodos de geração.

4.4.1 Método de soma ponderada

Em muitos problemas da vida real, os objetivos em consideração são conflitantes. Portanto, otimizar uma solução com respeito a um único objetivo muitas vezes resulta em resultados inaceitáveis em relação aos outros objetivos (KONAK; COIT; SMITH, 2006). Uma forma de lidar com essa questão é transformar e resolver um problema com múltiplos objetivos como um problema de um único objetivo.

O método de soma ponderada transforma os múltiplos objetivos em um único objetivo, de modo que o único objetivo representa a soma de todos objetivos, em que cada objetivo é multiplicado por um peso fornecido pelo usuário. De maneira mais formal, um problema de otimização multiobjetivo (definido na Equação 4-1) é convertido em um problema de otimização de um único objetivo do seguinte modo (DEB; KALYANMOY, 2001):

$$\begin{aligned} \text{Maximizar/Minimizar} \quad & F(x) = \textstyle\sum_{m=1}^{O} w_m f_m(x), & \\ \text{sujeito a} \quad & g_d(x) \geq 0, & d = 1,2,\ldots,D; \\ & h_j(x) = 0, & j = 1,2,\ldots,J; \\ & x_i^{(L)} \leq x_i \leq x_i^{(B)}, & i = 1,2,\ldots,I. \end{aligned} \tag{4-2}$$

onde w_m é o peso da m-ésima função objetivo.

O peso de um objetivo é geralmente escolhido considerando sua importância no contexto do problema. Portanto, é necessário um conhecimento a priori do problema para

uma boa escolha desses valores, pois a qualidade das soluções dependem totalmente dos pesos, tornando essa estratégia muito sensível à mudanças nestes parâmetros.

Além disso, é preciso considerar que diferentes objetivos podem ter ordens de grandeza diferentes e, portanto, deve ser realizada a normalização desses valores. Por fim, o problema de otimização multiobjetivo pode conter objetivos de maximização e objetivos de minimização, sendo necessário convertê-los todos para um único tipo.

4.4.2 Soluções ótimas de Pareto

Na otimização de um único objetivo, uma solução ótima é aquela que fornece o valor mínimo (ou máximo) da funcão objetivo. Em um problema de otimização multiobjetivo, quando são considerados objetivos conflitantes, em geral, não existe uma única solução que seja ótima com respeito a todos os objetivos.

De acordo com Yu e Gen (2012), o conceito de dominância proposto por Pareto permite lidar com a dificuldade de comparar dois objetivos simultaneamente. A partir deste conceito, o relacionamento entre duas soluções pode ser dividido em três categorias: uma solução é pior que a outra (é dominada), uma solução é melhor que a outra (domina), ou não é comparável à outra (não-dominada). Formalmente, assumindo um problema de maximização multiobjetivo, sem perda de generalidade, uma solução factível x é melhor que outra solução factível y se (ISHIBUCHI; TSUKAMOTO; NOJIMA, 2008):

$$\forall i = 1, \ldots, O : f_i(y) \leq f_i(x) \quad e \quad \exists j = 1, \ldots, O : f_j(y) < f_j(x)$$

Quando x é melhor que y é dito que x domina y (ou y é dominado por x). Quando y não é dominado por nenhuma outra solução factível, y é referido como uma solução ótima de Pareto. Por exemplo, considere o problema de maximização dos objetivos f_1 e f_2 e o conjunto de soluções factíveis, ilustrados na Figura 4.3. O retângulo cinza claro determina a região no espaço objetivo que é dominada pela solução B, enquanto que as soluções contidas no retângulo cinza escuro correspondem às soluções que dominam a solução B. Todas as soluções que se encontram fora desses dois retângulos são soluções não-dominadas por B. As soluções A e F são soluções ótimas de Pareto, pois não são dominadas por nenhuma outra solução factível.

O conjunto de soluções factíveis não-dominadas em $\hat{X}$ é chamado de conjunto ótimo de Pareto, e a imagem de um determinado conjunto ótimo de Pareto, no espaço objetivo Z, é chamada de fronteira de Pareto (KONAK; COIT; SMITH, 2006). Cada solução do conjunto de soluções não-dominadas é ótima no sentido de que nenhuma melhoria pode ser obtida com respeito a qualquer objetivo sem que exista piora em pelo menos algum outro objetivo (FONSECA; FLEMING, 1993).

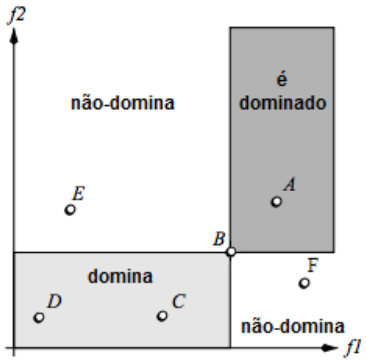


Figura 4.3: Possíveis relações entre as soluções. (Fonte: modificado de (ZITZLER, 1999))

Considerando a dificuldade de existir uma solução factível para um problema multiobjetivo que otimiza simultaneamente todas funções objetivo, uma opção razoável é investigar um conjunto de soluções, cada uma satisfazendo os objetivos em um nível aceitável sem ser dominada por qualquer outra solução. Em outras palavras, o objetivo final de um algoritmo de otimização multiobjetivo é identificar soluções no conjunto ótimo de Pareto (KONAK; COIT; SMITH, 2006). Contudo, devido ao tamanho deste conjunto, torna-se impraticável encontrar todas as soluções contidas nele.

Dessa forma, é necessário encontrar um subconjunto do conjunto ótimo de Pareto de modo que sejam alcançados os seguintes objetivos:

- Encontrar um conjunto de soluções o mais próximo possível da fronteira ótima de Pareto (convergência); e
- Encontrar um conjunto de soluções que seja diversificado o suficiente para garantir uma fronteira bem distribuída de soluções com diferentes compensações entre os objetivos (diversidade).

4.5 Considerações finais

Neste capítulo foram apresentados os principais conceitos e definições de metaheurísticas, bem como os principais elementos de projeto necessários para a construção de uma metaheurística eficiente. O próximo capítulo apresenta um novo método de geração de teste a partir de MEFs baseado em busca, projetado considerando os conhecimentos descritos neste capítulo.

CAPÍTULO 5

Método EATS

5.1 Considerações iniciais

O problema de geração de sequências de teste a partir de uma MEF de especificação pode ser caracterizado como um problema multiobjetivo, no qual espera-se gerar um conjunto de teste de modo a: maximizar a quantidade de defeitos revelados, minimizar o comprimento do conjunto de teste e minimizar a quantidade de sequências (*resets*).

A formulação do problema de teste baseado em MEFs como um problema de busca foi pouco explorada na literatura (LEHRE; YAO, 2014; DERDERIAN et al., 2006; GUO et al., 2005; GUO et al., 2004). Os trabalhos propostos investigam a geração de sequências especiais que podem não existir para todas as MEFs, e portanto, a aplicação de tais métodos fica limitada à existência da referida sequência.

Em contrapartida, diversos métodos de SBST foram propostos para o problema de teste baseado em Máquinas de Estados Finitos Estendidas (MEFEs) (DERDERIAN et al., 2005; LEFTICARU; IPATE, 2007; KALAJI; HIERONS; SWIFT, 2009; YANO; MARTINS; SOUSA, 2011; ZHOU; ZHAO; YOU, 2014; LU; MIAO, 2014; ASOUDEH; LABICHE, 2014). O modelo de MEFE inclui um conjunto de variáveis de modo que uma transição pode ter pré-condições definidas sobre as entradas e variáveis da máquina; e também pode ter operações (atribuições) para essas variáveis. No teste baseado em MEFEs, cada caso de teste define um *transition path* (TP) que representa o caminho percorrido quando a MEFE é estimulada com a sequência de entrada do caso de teste. No entanto, como as transições de uma MEFE podem ter pré-condições e operações, um determinado TP pode ser infactível. Por exemplo, a operação de uma transição pode atribuir o valor 0 a uma variável x, enquanto a pré-condição de uma transição posterior requerer $x > 0$. Esse TP é infactível, portanto, é impossível encontrar uma sequência de entrada (caso de teste) para exercitá-lo. No entanto, o problema de determinar se um determinado TP é factível é indecidível. Assim, no cenário de teste baseado em MEFEs, os métodos de SBST são construídos de modo a incorporar nos elementos de projeto métricas que permitem mensurar a factibilidade de um TP.

Como no teste baseado em MEF não existe o problema de factibilidade de TPs, não convém reutilizar representações e funções de aptidões propostas no cenário de teste baseado em MEFEs. Portanto, neste capítulo é apresentado o método EATS (do inglês, *Evolutionary Algorithm for Test Suite*) que se trata de uma metaheurística desenvolvida considerando os conhecimentos específicos do problema de geração de conjunto de teste a partir de MEF.

5.2 Processo evolutivo

O método EATS se trata de um algoritmo evolutivo desenvolvido com o objetivo de solucionar o seguinte problema:

Definição 5.1 (Problema) *Gerar um conjunto de sequências de teste, a partir de uma MEF, que represente um bom trade-off entre cobertura de defeitos e custo, onde custo representa o comprimento do conjunto de teste e o número de sequências do conjunto.*

O Problema 5.1 pode ser modelado como um problema de busca multiobjetivo, o qual contém um objetivo de maximização da cobertura de defeitos e outros dois objetivos de minimização, um do comprimento e o outro do número de *resets* do conjunto de teste. O método EATS considera apenas MEFs de Mealy, determinísticas, completas, minimais e inicialmente conectadas.

O processo de busca definido no método EATS segue o modelo incremental, ou *steady state*, no qual os descendentes são produzidos um de cada vez e competem imediatamente pela sua sobrevivência (JONG, 2006). Como operador de busca é utilizado a mutação, que consiste em provocar perturbações nos indivíduos da população.

Depois de gerar (Seção 5.5) e avaliar (Seção 5.6) os indivíduos da população inicial, o método EATS inicia uma geração selecionando um membro da população atual aleatoriamente como pai para produzir um filho idêntico. O método então introduz alguma variação no filho por meio de um operador de mutação (Seção 5.4). O filho gerado é forçado a competir imediatamente pela sua sobrevivência contra o pior indivíduo da população. Se a aptidão do novo indivíduo for maior, o novo indivíduo sobrevive e passa a fazer parte da população e o pior indivíduo da população atual morre. Caso contrário, o novo indivíduo morre. Uma nova geração é iniciada caso o critério de parada não seja atingido. O processo evolutivo é interrompido após 10.000 gerações sem melhora da melhor solução da população corrente. O Algoritmo 5.1 apresenta o processo evolutivo do método EATS.

Algoritmo 5.1: Processo evolutivo do Método EATS

```
1 Gerar e avaliar população inicial
2 enquanto não atingir 10000 gerações sem melhoria do melhor indivíduo da
    população corrente faça
3     Selecionar aleatoriamente e clonar indivíduo pai
4     Mutar indivíduo clone
5     se fitness(individuoMutado) > fitness(piorIndividuo) então
6         Inserir individuoMutado na população
7     fim
8 fim
```

O projeto da representação utilizada para codificar os indivíduos, dos operadores de mutação, do método de inicialização e da função de avaliação será detalhado nas seções abaixo.

5.3 Representação

Considerando o problema definido em 5.1, uma solução (indivíduo) para o problema é um conjunto de sequências de teste. Como visto na seção 3.3, um conjunto de sequências de teste pode ser visualizado como uma árvore. Dessa forma, considerando a codificação do indivíduo como uma árvore, uma representação para esse problema deve incorporar as restrições necessárias para um grafo corresponder a uma árvore.

De forma similar, problemas de projeto de redes (PPRs) são codificados, em geral, por árvores e foi observado que representações convencionais em AEs para PPRs não produzem resultados relevantes. Essas abordagens evolutivas geram muitas soluções infactíveis (grafos com ciclos ou grafos desconexos), sendo necessário dispensar tempo de computação para correção da solução até que seja obtida uma solução válida, reduzindo drasticamente a eficiência do AE (LIMA, 2009).

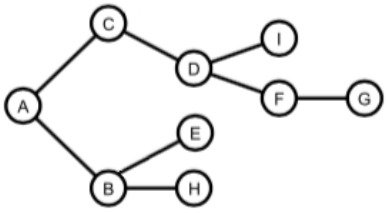
Com o objetivo de propor uma representação mais adequada para os AEs aplicados a PPRs, Delbem, Lima e Telles (2012), Lima (2009) propuseram a representação nó-profundidade-grau (NDDE, do inglês *Node-depth-degree Encoding*), que consiste no aperfeiçoamento da representação nó-profundidade (NDE - do inglês *Node-Depth Encoding*) (DELBEM; CARVALHO, 2003; DELBEM et al., 2004). Para essa classe de problemas, a NDDE teve sua eficiência comprovada com respeito às outras representações de árvore, como Vetor de Características (DAVIS et al., 1993), Número de Prüfer (ABUALI; SCHOENEFELD; WAINWRIGHT, 1994; ZHOU; GEN, 1997), *NetKeys* (ROTHLAUF; GOLDBERG; HEINZL, 2002) e Conjunto de Arestas (RAIDL; JULSTROM, 2003), e tem apresentado resultados satisfatórios para problemas modelados por grafo.

Considerando esse cenário, a representação NDDE foi escolhida como base para a construção da representação utilizada no método EATS e será detalhada a seguir.

5.3.1 Representação Nó-profundidade-grau

A NDDE de uma árvore $T = (V_T, E_T)$ consiste em um vetor de triplas contendo os nós (vértices) da árvore, suas profundidades e seus graus (v_i, prof$_i$, grau$_i$), tal que $i \in \{1,2,\ldots,n_v\}$ e n_v é a quantidade de vértices da árvore. A ordem das triplas é tal que os vértices em cada subárvore são consecutivos e a raiz de cada subárvore precede seus vértices.

Dada uma árvore T, sua NDDE pode ser obtida por uma busca em profundidade em T, adicionando um vértice, sua profundidade e grau no final da NDDE quando a busca encontrar um vértice não visitado (DELBEM; LIMA; TELLES, 2012). Como um exemplo, a Figura 5.1 mostra uma árvore e a NDDE correspondente produzida por uma busca em profundidade iniciada no vértice A.



(a) *Árvore com 9 vértices*

vértice	A	B	H	E	C	D	F	G	I
profundidade	0	1	2	2	1	2	3	4	3
grau	2	3	1	1	2	3	2	1	1
índice	1	2	3	4	5	6	7	8	9

(b) *NDDE*

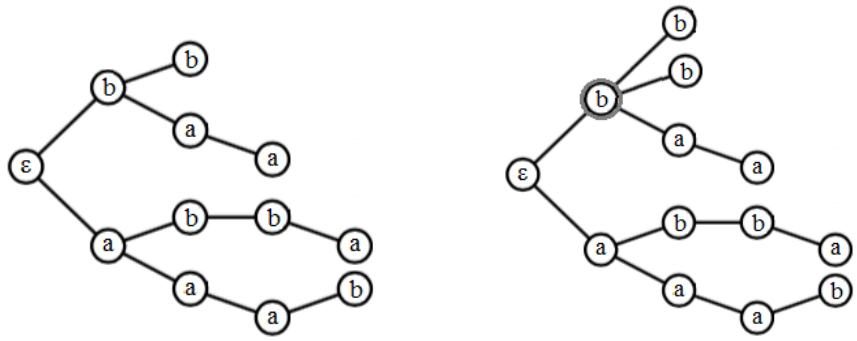
Figura 5.1: Exemplo de uma árvore e sua NDDE correspondente.

5.3.2 Representação Nó-profundidade-grau de saída

A representação em árvore de um conjunto de teste possui informações nos vértices (estados da MEF) e arestas (símbolos de entrada), como é possível observar na Figura 3.2. Como a NDDE não armazena informações de aresta e um conjunto de teste é definido com base nos símbolos de entrada, o conjunto de teste foi representado como uma árvore em que os vértices são rotulados com os símbolos de entrada e as arestas não possuem rótulos. A raiz da árvore é rotulado com ϵ. A Subfigura 5.2(a) mostra o conjunto de teste da Figura 3.2 considerando tal representação.

Contudo, dada as características da classe de problemas de PPRs, a NDDE não permite a repetição de rótulos dos vértices. Porém, conforme pode ser observado na Subfigura 5.2(a), os vértices representam os símbolos de entradas da MEF e, portanto, será necessária a repetição dos rótulos.

Dada a possibilidade de repetição dos rótulos, pode-se observar na Figura 5.2(b) a representação de um conjunto de teste similar ao conjunto da Figura 5.2(a), com uma



(a) *Conjunto de teste* TS = *{aaab, abba,baa,bb}*

(b) *Conjunto de teste* TS = *{aaab, abba,baa,bb,bb}*

Figura 5.2: Representação do conjunto de teste em árvore, na qual os vértices são símbolos de entrada.

sequência a mais: *bb*. Contudo, essa sequência é uma repetição da sequência *bb* já existente. Isso ocorre pois o vértice evidenciado na Figura 5.2(b) possui um número de filhos (três) maior que o número de entradas da MEF (dois), sendo necessário repetir um símbolo de entrada e ocasionando a geração de uma sequência de teste repetida. Para evitar gerar esse tipo de solução, foi considerada a seguinte restrição: o número de filhos de um vértice deve ser menor ou igual ao número de entradas da MEF. Além disso, mesmo o número de filhos sendo menor, é necessário verificar se as entradas dos filhos de um mesmo vértice pai não se repetem.

Considerando o fato que o problema de teste baseado em MEFs possui características distintas daquelas de PPRs, foi proposta a representação NDOE (do inglês, Node-depth-outdegree Enconding). Diferentemente da NDDE, a NDOE permite a repetição dos rótulos dos vértices, exceto entre filhos de um mesmo vértice pai. Dada que a restrição de repetição dos rótulos se baseia no número de filhos de um vértice, ao invés de armazenar a informação do grau de cada vértice, a NDOE armazena o grau de saída, que representa o número de filhos de um determinado vértice.

Assim, A NDOE de uma árvore $T = (V_T, E_T)$ consiste em um vetor de triplas contendo os nós (vértices) da árvore, suas profundidades e seus graus de saída (v_i, prof_i, grau_saida_i), tal que $i \in \{1,2,\ldots,n_v\}$ e n_v é a quantidade de vértices da árvore. A ordem das triplas é tal que os vértices em cada subárvore são consecutivos e a raiz de cada subárvore precede seus vértices. A Figura 5.3 apresenta a NDOE da Figura 5.2(a).

5.4 Operadores de mutação

A representação NDDE possui os operadores de mutação PAO e CAO (LIMA, 2009). Contudo tais operadores foram propostos para PPRs, nos quais os indivíduos

vértice	ε	a	a	a	b	b	b	a	b	a	a	b
profundidade	0	1	2	3	4	2	3	4	1	2	3	2
grau de saída	2	2	1	1	0	1	1	0	2	1	0	0
índice	1	2	3	4	5	6	7	8	9	10	11	12

Figura 5.3: NDOE correspondente à árvore da Figura 5.2(a)

possuem tamanho fixo e não há repetição dos rótulos dos vértices.

No contexto deste trabalho, esses operadores não se aplicam, porque os indivíduos possuem tamanho variável e pode haver a repetição dos rótulos de vértices, conforme pode ser observado na Figura 5.3. Além disso, há a restrição de repetição de entradas entre filhos de um mesmo vértice pai. Como os operadores da NDDE não foram projetados considerando tal restrição, sua aplicação resultaria em indivíduos infactíveis sendo necessária a correção dos mesmos, impactando assim no desempenho final do algoritmo. Por este motivo, foram propostos novos operadores de busca.

Um indivíduo é mutado no Algoritmo 5.1 por meio de um dos cincos operadores de mutação propostos: 1) Recorta e Cola Subárvore (RCS), 2) Copia e Cola Subárvore (CCS), 3) Adiciona Um Nó (AUN), 4) Remove Uma Subárvore (RUS) e 5) Modifica Um Nó (MUN). Os três primeiros operadores permitem aumentar o tamanho de um indivíduo e o quarto e quinto permitem diminuir e manter o mesmo tamanho, respectivamente. A escolha de qual operador aplicar é feita de forma aleatória, onde aleatoriamente escolhe-se entre aumentar, diminuir ou manter o tamanho do indivíduo e caso seja feita a escolha de aumentar, escolhe-se aleatoriamente entre os três primeiros operadores.

Os operadores RCS, CCS e AUN geram indivíduos filhos com mais vértices do que o indivíduo pai, ou seja, os indivíduos gerados representam conjuntos de teste com custo maior, mas que em contrapartida podem beneficiar o objetivo de maximizar a cobertura dos defeitos. De modo contrário, o operador RUS reduz o número de vértices, que consequentemente representam conjuntos de teste com menor custo, porém esta redução pode degradar a cobertura de defeitos do conjunto. Por fim, o operador MUN gera conjuntos com o mesmo custo, podendo beneficiar/degradar apenas o objetivo de cobertura de defeitos.

Apesar de alguns desses operadores causarem grandes mudanças no indivíduo gerado, esses operadores são chamados de mutação, pois geram uma nova solução a partir de uma única outra solução. Os operadores foram propostos de modo a garantir o princípio de ergocidade definido na Seção 4.3.2. Os demais princípios dizem respeito a operadores de recombinação. A seguir será apresentado em detalhes cada um dos cinco operadores propostos neste trabalho.

5.4.1 Recorta e Cola Subárvore (RCS)

O operador RCS baseia-se no operador PAO, no qual é transferida uma subárvore de uma parte da árvore para outra. Contudo, no operador RCS é feita uma cópia da raiz da subárvore a ser transferida. Assim, RCS mantém a raiz da subárvore transferida no local original. Essa modificação é feita para aumentar o número de vértices no novo indivíduo gerado, passando a ter um vértice a mais que o indivíduo pai. Caso fosse feita apenas a transferência da subárvore de uma parte da árvore para outra, o indivíduo gerado teria o mesmo tamanho do pai.

Assim como o PAO, o RCS utiliza dois vértices em especial: o vértice p, que indica a raiz da subárvore a ser transferida e o vértice a, onde a subárvore podada será conectada. O Algoritmo 5.2 descreve os passos do operador RCS.

Algoritmo 5.2: Operador RCS

Entrada: MEF M e NDOE T_x
Saída: NDOE T'_x (indivíduo gerado)

1 Escolha aleatoriamente um vértice diferente da raiz e chame-o de p
2 Escolha aleatoriamente um vértice diferente da raiz e verifique se ele não está na subárvore enraizada em p e se seu número de filhos é menor que o número de símbolos de entrada de M. Caso afirmativo, chame esse vértice de a. Caso contrário, escolha aleatoriamente outro vértice
3 Determine o intervalo $(i_p, \ldots, i_L)$ em T_x, correspondente aos vértices da subárvore enraizada em p
4 **se** $i_p < i_a$ **então**
5 Copie em T'_x as posições de T_x de $i = 1$ até i_p e atualize o número de filhos de i_p para 0
6 Copie as posições de $(i_{L+1}, \ldots, i_a)$ de T_x em T'_x e incremente o número de filhos de i_a
7 Copie as posições de $(i_p, \ldots, i_L)$ de T_x em T'_x e atualize os valores de profundidade: $prof_i = prof_i - prof_{i_p} + prof_{i_a} + 1$. Caso o símbolo de entrada de i_p já seja utilizado em algum outro filho de i_a, escolha um símbolo não utilizado e atualize o símbolo de entrada de i_p em T'_x
8 Copie em T'_x as posições de T_x de $i = i_{a+1}$ até $|T_x|$
9 **senão**
10 Copie em T'_x as posições de T_x de $i = 1$ até i_a e incremente o número de filhos de i_a
11 Copie as posições de $(i_p, \ldots, i_L)$ de T_x em T'_x e atualize os valores de profundidade: $prof_i = prof_i - prof_{i_p} + prof_{i_a} + 1$. Caso o símbolo de entrada de i_p já seja utilizado em algum outro filho de i_a, escolha um símbolo não utilizado e atualize o símbolo de entrada de i_p em T'_x
12 Copie em T'_x as posições de T_x de $i = i_{a+1}$ até i_p e atualize o número de filhos de i_p para 0
13 Copie em T'_x as posições de T_x de $i = i_{L+1}$ até $|T_x|$
14 **fim**
15 **retorne** T'_x

Para exemplificar a aplicação do Algoritmo 5.2, considere a NDOE da Figura 5.4 e a MEF da Figura 3.1, a qual possui dois símbolos de entrada. Considere, também, a escolha de $i_p = 6$ e $i_a = 2$, nas linhas 1 e 2 do algoritmo.

vértice	ε	a	a	a	b	b	a
profundidade	0	1	2	3	3	1	2
grau de saída	2	1	2	0	0	1	0
índice	1	2	3	4	5	6	7

Figura 5.4: NDOE T_x

Na linha 3 do algoritmo, o intervalo (6, 7) é determinado como a subárvore enraizada em p, indicada em vermelho na Figura 5.4. Em azul representa o vértice a.

Como $i_p = 6$ é maior que $i_a = 2$, na linha 10 copia-se em T'_x as posições de T_x de $i = 1$ até $i_a = 2$ e o número de filhos de i_a é incrementado deixando de ser 1 e passando a ser 2. Na linha 11, é copiado para T'_x o intervalo (6, 7) de T_x e os valores de profundidade são atualizados. A profundidade do vértice indexado em 6 de T_x em T'_x será $\text{prof}_6 = 1 - 1 + 1 + 1 = 2$ e do vértice indexado em 7 será $\text{prof}_7 = 2 - 1 + 1 + 1 = 3$.

Na linha 12, é copiado para T'_x o intervalo (3, 6) de T_x atualizando o número de filhos do vértice 6 para 0. Por fim, na linha 13 seriam copiadas as posições de T_x a partir de $L+1 = 8$, mas nesse caso não há tais posições e portanto T'_x está completo. A Figura 5.5 ilustra os passos realizados pelo algoritmo na construção do novo indivíduo.

vértice	ε	a						
profundidade	0	1						
grau de saída	2	2						
índice	1	2	3	4	5	6	7	8

(a) *NDOE T'_x após linha 10*

vértice	ε	a	b	a				
profundidade	0	1	2	3				
grau de saída	2	2	1	0				
índice	1	2	3	4	5	6	7	8

(b) *NDOE T'_x após linha 11*

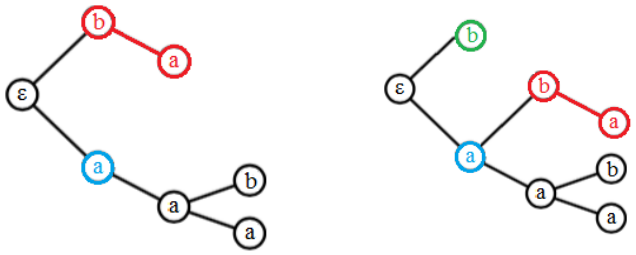
vértice	ε	a	b	a	a	a	b	b
profundidade	0	1	2	3	2	3	3	1
grau de saída	2	2	1	0	2	0	0	0
índice	1	2	3	4	5	6	7	8

(c) *NDOE T'_x após linha 12*

Figura 5.5: NDOE T'_x gerada pelo Algoritmo 5.2

A Figura 5.6 ilustra as árvores pai e filha, correspondente às NDOEs das Figuras

5.4 e 5.5(c). Em vermelho é ressaltada a subárvore transferida, em azul o local no qual a subárvore é inserida e em verde a cópia da raiz da subárvore transferida.



(a) *Árvore pai* (b) *Árvore filha*

Figura 5.6: Exemplo de indivíduo gerado pelo operador RCS

5.4.2 Copia e Cola Subárvore (CCS)

O operador CCS é similar ao RCS, contudo no RCS é mantida a cópia da raiz da subárvore transferida e no CCS é mantida a cópia da subárvore inteira. Assim, uma cópia da subárvore a ser transferida (enraizada em p) continua no local que ela estava e a subárvore é conectada em outra parte (vértice a). Com base nisso, o novo indivíduo gerado pelo operador RCS possui um vértice a mais em relação ao seu pai, enquanto que com a aplicação do CCS o novo indivíduo possui $|sub(p)|$ vértices a mais, onde $|sub(p)|$ representa o número de vértices da subárvore enraizada em p. O Algoritmo 5.3 descreve os passos do operador CCS.

Algoritmo 5.3: Operador CCS

Entrada: MEF M e NDOE T_x
Saída: NDOE T'_x (indivíduo gerado)

1 Escolha aleatoriamente um vértice diferente da raiz e chame-o de p
2 Escolha aleatoriamente um vértice diferente da raiz e verifique se ele não está na subárvore enraizada em p e se seu número de filhos é menor que o número de símbolos de entrada de M. Caso afirmativo, chame esse vértice de a. Caso contrário, escolha aleatoriamente outro vértice
3 Determine o intervalo $(i_p, \ldots, i_L)$ em T_x, correspondente aos vértices da subárvore enraizada em p
4 Copie em T'_x as posições de T_x de $i = 1$ até i_a e incremente o número de filhos de i_a
5 Copie as posições de $(i_p, \ldots, i_L)$ de T_x em T'_x e atualize os valores de profundidade: $prof_i = prof_i - prof_{i_p} + prof_{i_a} + 1$. Caso o símbolo de entrada de i_p já seja utilizado em algum outro filho de i_a, escolha um símbolo não utilizado e atualize o símbolo de entrada de i_p em T'_x
6 Copie em T'_x as posições de T_x de $i = i_{a+1}$ até $|T_x|$
7 **retorne** T'_x

Considerando o mesmo exemplo da seção anterior, a Figura 5.7 ilustra a construção do novo indivíduo T'_x utilizando o Algoritmo 5.3. A Figura 5.8 ilustra as árvores

pai e filha, correspondente às NDOEs das Figuras 5.4 e 5.7(c). Em vermelho é ressaltada a subárvore transferida, em azul o local no qual a subárvore é inserida e em verde a cópia da subárvore transferida.

vértice	ε	a							
profundidade	0	1							
grau de saída	2	2							
índice	1	2	3	4	5	6	7	8	9

(a) *NDOE T'_x após linha 4*

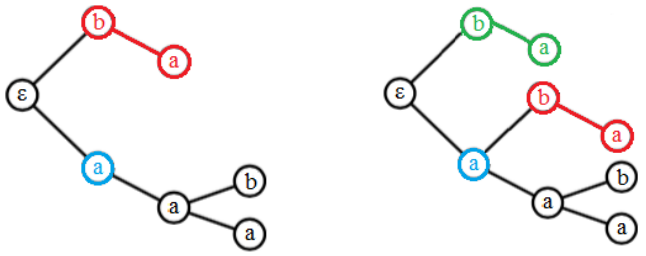
vértice	ε	a	b	a					
profundidade	0	1	2	3					
grau de saída	2	2	1	0					
índice	1	2	3	4	5	6	7	8	9

(b) *NDOE T'_x após linha 5*

vértice	ε	a	b	a	a	a	b	b	a
profundidade	0	1	2	3	2	3	3	1	2
grau de saída	2	2	1	0	2	0	0	1	0
índice	1	2	3	4	5	6	7	8	9

(c) *NDOE T'_x após linha 6*

Figura 5.7: NDOE T'_x gerada pelo Algoritmo 5.3



(a) *Árvore pai* (b) *Árvore filha*

Figura 5.8: Exemplo de indivíduo gerado pelo operador CCS

5.4.3 Adiciona Um Nó (AUN)

Como os operadores RCS e CCS representam passos grandes na busca, foi proposto o operador AUN para gerar o novo indivíduo com apenas um vértice a mais, porém sem deslocar partes do mesmo. O operador AUN utiliza apenas o vértice *a*, que nesse caso representa o local onde será inserido o novo vértice criado. O operador AUN é descrito no Algoritmo 5.4.

Algoritmo 5.4: Operador AUN

Entrada: MEF M e NDOE T_x

Saída: NDOE T'_x (indivíduo gerado)

1 Escolha aleatoriamente um vértice diferente da raiz e verifique se seu número de filhos é menor que o número de símbolos de entrada de M. Caso afirmativo, chame esse vértice de a. Caso contrário, escolha aleatoriamente outro vértice

2 Copie em T'_x as posições de T_x de $i = 1$ até i_a e incremente o número de filhos de i_a

3 Insira em T'_x, na posição $i_a + 1$, um vértice com profundidade igual a $\text{prof}_{i_a} + 1$, um símbolo de entrada, escolhido aleatoriamente, ainda não utilizado por nenhum outro filho de a e número de filhos 0

4 Copie em T'_x as posições de T_x de $i = i_{a+1}$ até $|T'_x|$

5 **retorne** T'_x

Considerando a NDOE da Figura 5.4, a MEF da Figura 3.1 e a escolha de $i_a = 2$, a Figura 5.9 ilustra a construção do novo indivíduo T'_x utilizando o Algoritmo 5.4.

vértice	ε	a						
profundidade	0	1						
grau de saída	2	2						
índice	1	2	3	4	5	6	7	8

(a) *NDOE T'_x após linha 2*

vértice	ε	a	b					
profundidade	0	1	2					
grau de saída	2	2	0					
índice	1	2	3	4	5	6	7	8

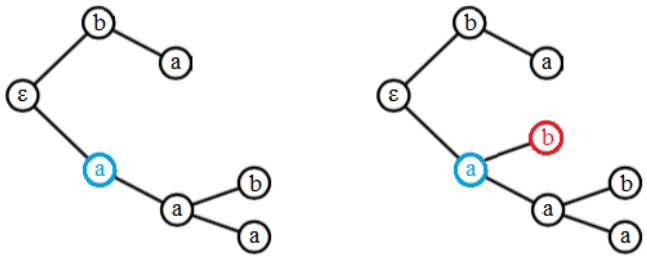
(b) *NDOE T'_x após linha 3*

vértice	ε	a	b	a	a	b	b	a
profundidade	0	1	2	2	3	3	1	2
grau de saída	2	2	0	2	0	0	1	0
índice	1	2	3	4	5	6	7	8

(c) *NDOE T'_x após linha 4*

Figura 5.9: NDOE T'_x gerada pelo Algoritmo 5.4

A Figura 5.10 ilustra as árvores pai e filha, correspondente às NDOEs das Figuras 5.4 e 5.9(c). Em vermelho é ressaltado o novo vértice inserido e em azul o local no qual o vértice foi inserido.



(a) *Árvore pai* (b) *Árvore filha*

Figura 5.10: Exemplo de indivíduo gerado pelo operador AUN

5.4.4 Remove Uma Subárvore (RUS)

Para diminuir o tamanho de um indivíduo, o operador RUS escolhe aleatoriamente um vértice r, diferente da raiz, e remove a subárvore enraizada em r, incluindo r. Dessa forma, o operador RUS gera um novo indivíduo com $|sub(r)|$ vértices a menos, onde $|sub(r)|$ representa o número de vértices da subárvore enraizada em r. O Algoritmo 5.5 apresenta em detalhes o operador RUS.

Algoritmo 5.5: Operador RUS

Entrada: MEF M e NDOE T_x
Saída: NDOE T'_x (indivíduo gerado)

1 Escolha aleatoriamente um vértice diferente da raiz e chame-o de r
2 Determine o intervalo $(i_r, \ldots, i_L)$ em T_x, referente aos vértices da subárvore enraizada em r
3 Determine o vértice pai f de r
4 Copie em T'_x as posições de T_x de $i = 1$ até $i_r - 1$ e decremente o número de filhos de f
5 Copie em T'_x as posições de T_x de $i = i_{L+1}$ até $|T_x|$
6 **retorne** T'_x

Considere a MEF da Figura 3.1, a escolha de $i_r = 3$ e a NDOE T_x, utilizada nos exemplos anteriores, e ilustrada na Figura 5.11. A subárvore enraizada em r (linha 2 do Algoritmo 5.5) está representada em vermelho e em azul está representado o vértice f, pai de r, determinado na linha 3. A Figura 5.12 ilustra a construção do novo indivíduo T'_x utilizando o Algoritmo 5.5.

A Figura 5.13 ilustra as árvores pai e filha, correspondente às NDOEs das Figuras 5.11 e 5.12(b). Em vermelho é ressaltada a subárvore removida e em azul o pai do vértice r.

vértice	ε	a	a	a	b	b	a
profundidade	0	1	2	3	3	1	2
grau de saída	2	1	2	0	0	1	0
índice	1	2	3	4	5	6	7

Figura 5.11: NDOE T_x

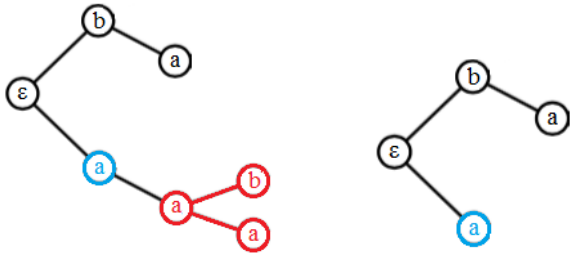
vértice	ε	a		
profundidade	0	1		
grau de saída	2	0		
índice	1	2	3	4

(a) *NDOE T'_x após linha 4*

vértice	ε	a	b	a
profundidade	0	1	1	2
grau de saída	2	0	1	0
índice	1	2	3	4

(b) *NDOE T'_x após linha 5*

Figura 5.12: NDOE T'_x gerada pelo Algoritmo 5.5



(a) *Árvore pai* (b) *Árvore filha*

Figura 5.13: Exemplo de indivíduo gerado pelo operador RUS

5.4.5 Modifica Um Nó (MUN)

O operador MUN é responsável por modificar o símbolo de entrada de um vértice, escolhido aleatoriamente. Como é feita apenas uma alteração do valor do vértice, o tamanho do indivíduo gerado não sofre alteração, sendo igual ao tamanho do indivíduo pai que o gera. O Algoritmo 5.6 descreve em detalhes o operador MUN.

Considerando a NDOE da Figura 5.4, a MEF da Figura 3.1 e a escolha de $i_u = 3$, a Figura 5.14 ilustra a construção do novo indivíduo T'_x utilizando o Algoritmo 5.6. A Figura 5.15 ilustra as árvores pai e filha, correspondente às NDOEs das Figuras 5.4 e 5.14(c). Em azul é ressaltado o vértice modificado.

Algoritmo 5.6: Operador MUN

Entrada: MEF M e NDOE T_x

Saída: NDOE T'_x (indivíduo gerado)

1 Escolha aleatoriamente um vértice diferente da raiz e verifique se o número de filhos de seu pai é menor que o número de símbolos de entrada de M. Caso afirmativo, chame esse vértice de u. Caso contrário, retorne e sorteia novamente um operador de mutação para aplicar

2 Copie em T'_x as posições de T_x de $i = 1$ até i_u

3 Escolha aleatoriamente um símbolo de entrada ainda não utilizado por nenhum outro vértice irmão de u e modifique o símbolo de entrada de u

4 Copie em T'_x as posições de T_x de $i = i_{u+1}$ até $|T_x|$

5 **retorne** T'_x

vértice	ε	a	a				
profundidade	0	1	2				
grau de saída	2	1	2				
índice	1	2	3	4	5	6	7

(a) *NDOE T'_x após linha 2*

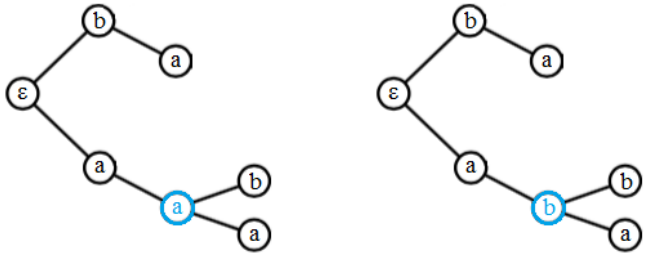
vértice	ε	a	b				
profundidade	0	1	2				
grau de saída	2	1	2				
índice	1	2	3	4	5	6	7

(b) *NDOE T'_x após linha 3*

vértice	ε	a	b	a	b	b	a
profundidade	0	1	2	3	3	1	2
grau de saída	2	1	2	0	0	1	0
índice	1	2	3	4	5	6	7

(c) *NDOE T'_x após linha 4*

Figura 5.14: NDOE T'_x gerada pelo Algoritmo 5.6



(a) *Árvore pai* (b) *Árvore filha*

Figura 5.15: Exemplo de indivíduo gerado pelo operador MUN

5.5 Inicialização

Como mencionado na Seção 4.3.3, as soluções iniciais podem ser geradas de forma aleatória ou com alguma tendência para certos tipos de solução dependendo do conhecimento que se tem a respeito do problema. Como proposta deste trabalho será explorada a inicialização aleatória.

Na inicialização do método EATS, os indivíduos são construídos fazendo o uso do operador AUN, conforme pode ser observado no Algoritmo 5.7.

Algoritmo 5.7: Inicialização da população de indivíduos

Entrada: MEF M e tamanho da população p_n
Saída: População inicial POP com p_n indivíduos

```
1  //Seja n o número de estados da MEF M
2  //Seja n_i o número de símbolos de entradas da MEF M
3  //Seja c uma constante
4  numeroMaximoDeNos ← c×n×n_i
5  enquanto o tamanho de POP for menor que p_n faça
6      Crie uma NDOE T_x com um vértice raiz e n_i vértices de profundidade 1 (sem repetição dos
         símbolos de entrada)
7      Escolha um número aleatório entre n_i + 1 e numeroMaximoDeNos e chame-o de
         quantidadeDeNos
8      enquanto o número de vértices de T_x for menor que quantidadeDeNos faça
9          AUN(M, T_x)
10     fim
11 fim
12 retorne P
```

Inicialmente, na linha 6 do Algoritmo 5.7, cada indivíduo é formado pela "estrutura básica" que representa o vértice raiz e seus n_i filhos, cada um com um símbolo de entrada distinto. Para a MEF da Figura 3.1 a "estrutura básica" dos indivíduos é apresentada na Figura 5.16.

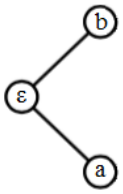


Figura 5.16: "Estrutura básica" das soluções iniciais com base na MEF da Figura 3.1

O tamanho de cada indivíduo é escolhido aleatoriamente com base no intervalo $[n_i+1, numeroMaximosDeNos]$ (linha 7). O limite inferior do intervalo diz respeito ao fato do indivíduo já possuir n_i+1 vértices ("estrutura básica") e o limite superior representa o fato de que uma MEF completa possui $n \times n_i$ transições e a constante c permite a ocorrência de *loops*, ou seja, passar mais de uma vez em uma mesma transição. Uma vez escolhido o tamanho do indivíduo (*quantidadeDeNos*), é necessário aplicar o operador AUN x vezes tal que x = *quantidadeDeNos* - (n_i+1).

5.6 Função *fitness*

Considerando a modelagem do Problema 5.1 como um problema de otimização é necessário considerar três objetivos: maximizar a cobertura de defeitos, minimizar o comprimento do conjunto de teste e minimizar o número de operações *resets*.

5.6.1 Cálculo dos valores objetivos

O *comprimento* é obtido pela soma das profundidades dos vértices folhas acrescido do número de folhas. O *número de operações resets* é o número de vértices folhas. Para o cálculo da *cobertura de defeitos* serão investigadas (i) a técnica de análise de mutantes e (ii) as condições de suficiência de Simão e Petrenko (2010a).

Na técnica de análise de mutantes (Seção 3.3.2.1) o *score* de mutação representa um intervalo entre 0 e 1, no qual 1 representa que o conjunto de teste é completo. O cálculo do *score* de mutação se dá pela Equação 3-1, em que, dado $m = n$, o fator $N_t(m, M)$ é dado por $(n|Y|)^{n|X|}$, $N_c(m, M)$ como $(n-1)!$ e $N_p(m, M, TS)$ representa a quantidade de mutantes executados com o conjunto de teste TS e que não foram mortos. Contudo para encontrar o valor exato de $N_p(m, M, TS)$ é necessário executar todos os mutantes ($N_t(m, M)$) representando um alto custo computacional. Por este motivo para o cálculo de $N_p(m, M, TS)$ foi utilizada uma estimativa baseada em análise estrutural (PETRENKO; BOCHMANN; YAO, 1996). Dessa forma, o uso da estimativa dispensa a necessidade de execução de todos os mutantes e segundo Petrenko, Bochmann e Yao (1996) possui uma complexidade computacional baixa, $O(L^2)$, onde L é o tamanho do conjunto de teste em termos do número total de símbolos de entrada. Além disso, como o método EATS

considera MEFs completas e $m = n$, os mutantes representam MEFs com alterações no estado final e/ou saída de uma transição.

Com relação ao uso das condições de suficiência para o cálculo do objetivo de cobertura, o Algoritmo 3.1, o qual verifica se determinado conjunto de teste satisfaz as condições, possui duas particularidades que dificultam sua aplicação direta como cálculo de uma função de *fitness*.

O primeiro quesito é a ausência de uma escala para diferenciar as soluções, pois utilizando o algoritmo é possível apenas diferenciar se uma solução é ótima (quando retornar verdadeiro) ou não é ótima (quando retornar falso). Contudo, considerando duas soluções não completas, ambas serão avaliadas como não ótimas, porém se uma das soluções possui um conjunto *transition cover*, por exemplo, e a outra não, a solução que possui *transition cover* é capaz de revelar mais defeitos que a solução que não possui e portanto deveria possuir um valor de *fitness* maior com respeito ao objetivo de cobertura.

O segundo ponto diz respeito à eficiência. Como é realizado um grande número de avaliações de soluções, o cálculo da função de *fitness* não deve ter um alto custo. Apesar do Algoritmo 3.1 fazer uso de uma heurística para encontrar uma n-clique no grafo de distinção é interessante encontrar meios de reduzir a necessidade de execução dessa tarefa com o intuito de reduzir o custo das avaliações. Levando em consideração tais questões foi proposto o Algoritmo 5.8 que faz uso das condições de suficiência de Simão e Petrenko (2010a) para avaliar uma solução com respeito ao objetivo de cobertura de defeitos.

Um conjunto de teste para ser completo precisa cobrir todas as transições da MEF, pois caso contrário uma transição não coberta poderá conter um defeito o qual não será revelado pelo conjunto de teste em questão. Assim, espera-se que um conjunto de teste completo contenha um conjunto *transition cover*.

Um conjunto de teste que não contém um *transition cover* não é completo e portanto não há a necessidade de verificar as condições de suficiência. Dessa forma sua qualidade diz respeito ao número de transições da MEF que ele é capaz de cobrir (linha 21 do Algoritmo 5.8). Quanto maior o número de transições cobertas maior será seu valor objetivo.

Caso o indivíduo possua um *transition cover*, o grafo de distinção é construído e as condições de suficiências são verificadas. Caso não haja uma n-clique, é preciso encontrar uma clique máxima no grafo de distinção. A qualidade do indivíduo será mensurada considerando o tamanho da clique máxima encontrada (linha 18 do Algoritmo 5.8). Quanto maior a clique encontrada maior a qualidade da solução.

Caso o indivíduo possua um conjunto *transition cover* e é encontrada uma n-clique então os Lemas 3.2 e 3.3 serão aplicados. Se for possível confirmar todos os vértices do grafo de distinção então o conjunto será completo e portanto terá valor objetivo

igual à 1. Caso contrário, serão contabilizados quantos vértices foram confirmados e quantas transições foram cobertas por esses vértices. Quanto maior for o número de vértices confirmados e transições cobertas por eles, mais próximo o indivíduo é do ótimo.

Algoritmo 5.8: Avaliação do objetivo de cobertura de defeitos baseado nas condições de Simão e Petrenko (2010a)

Entrada: MEF M e conjunto de teste TS (indivíduo)
Saída: Valor do objetivo de cobertura do indivíduo TS

```
1  //Seja n o número de estados da MEF M
2  //Seja n_t o número de transições da MEF M
3  //Seja n_v o número de vértices de G
4  //Seja n_vc o número de vértices confirmados
5  //Seja n_tcTS o número de transições cobertas por TS
6  //Seja n_tcVC o número de transições cobertas pelos vértices confirmados
7  se TS possui conjunto transition cover então
8      Construir grafo de distinção G de TS
9      Determinar uma clique de tamanho n em G e inserir os vértices da clique no
         conjunto K
10     se encontrou clique então
11         enquanto encontrar uma sequência α ∈ TS \K, em que os Lemas 3.2 ou 3.3
             podem ser aplicados faça
12             Inserir α em K
13         fim
14         valorObjetivo ← |clique|/n + n_vc/n_v + n_tcVC/n_t
15     senão
16         Determinar uma clique máxima em G
17         valorObjetivo ← |cliqueMaxima|/n
18     fim
19 senão
20     valorObjetivo ← n_tcTS/n_t
21 fim
22 valorObjetivo ← valorObjetivo/3
23 retorne valorObjetivo
```

O objetivo de cobertura de defeitos passa a ser calculado considerando a média da quantidade de transições cobertas, tamanho da clique encontrada e quantidade de vértices confirmados pelos Lemas 3.2 e 3.3. Dessa forma, o valor objetivo é definido no intervalo entre 0 e 1, assim como o *score* de mutação.

Com respeito à eficiência, a hierarquia do cálculo do objetivo de cobertura de defeitos minimiza a necessidade de execução da tarefa de encontrar cliques. Essa redução deve-se ao fato de que somente quando houver *transition cover* as condições de suficiência serão verificadas.

O Algoritmo 5.8 envolve a solução de dois problemas de cliques: encontrar k-cliques em grafos k-partidos (linha 10) e encontrar clique máxima em grafos k-partidos (linha 17). Segundo Simão e Petrenko (2010a), o problema de encontrar k-cliques em grafos k-partidos já foi investigado em (GRUNERT et al., 2002) e foi proposto um algoritmo que faz uso de heurísticas para se ter uma boa eficiência na solução do problema. Quanto ao problema de encontrar clique máxima em grafos k-partidos, encontrou-se na literatura apenas trabalhos que solucionam o problema para grafos gerais. Dessa forma, foi proposto no presente trabalho uma heurística para fins de solucionar o problema de encontrar clique máxima em grafos k-partidos. A seção a seguir detalha o algoritmo proposto.

5.7 Heurística para encontrar clique máxima em grafos k-partidos

Inicialmente, foi feito um levantamento bibliográfico para encontrar possíveis soluções para o problema de encontrar clique máxima em grafos k-partidos. Contudo, foram encontrados apenas trabalhos que solucionam o problema de encontrar clique máxima em grafos gerais.

Dentre estes trabalhos, Rossi et al. (2014) propuseram um algoritmo exato baseado em duas etapas, sendo que uma delas é uma heurística de tempo quase linear. De acordo com os experimentos, a heurística se mostrou ser capaz de encontrar a clique máxima em mais da metade dos grafos considerados e portanto pode ser usada como um procedimento *stand-alone*.

Como no Algoritmo 5.8 o tamanho da clique máxima representa um critério para avaliar e diferenciar soluções, então é suficiente ter um algoritmo que seja capaz de encontrar cliques grandes, mesmo que não haja a garantia de encontrar a maior. Portanto, a heurística de Rossi et al. (2014) servirá de base para a construção da heurística proposta e detalhada a seguir.

A heurística de Rossi et al. (2014) cria uma clique pesquisando em torno de cada vértice v no grafo e adicionando gulosamente vértices da vizinhança, a qual inclue o próprio vértice v, desde que eles formem uma clique. A pesquisa é feita considerando a ordenação dos vértices baseada em *core number*.

Definição 5.2 (Core number) *O core number de um vértice v é o maior inteiro c, de modo que v tem grau maior que 0 quando todos os vértices de grau menor que c são removidos. De maneira equivalente, o core number de um vértice v é o maior número inteiro c, de modo que v existe em um grafo em que todos os graus são pelo menos c.*

Denotamos o core number de um vértice v por $K(v)$ e denotamos por $K(G)$ o maior core number em todo o grafo G.

A Figura 5.17 exibe um grafo $G = (V, E)$ com os *core numbers* de cada um dos seus vértices. Se considerarmos $c = 1$, nenhum vértice é removido. Se considerarmos $c = 2$, o vértice a é removido. Portanto, $K(a) = 1$, pois $c = 1$ é o maior valor que garante que a exista no grafo. Da mesma forma, quando $c = 3$ os vértices d, e são removidos e portanto $K(d) = K(e) = 2$. Por fim, quando $c = 4$ todos os vértices são removidos e portanto $K(b) = K(c) = K(f) = K(g) = K(h) = 3$. Temos então que $K(G) = 3$.

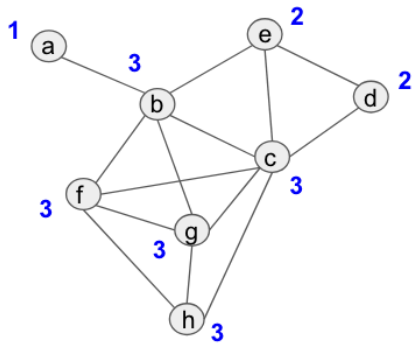


Figura 5.17: Exemplo de um grafo com os *core numbers* de cada um de seus vértices

Suponha um grafo G' que contenha uma clique de tamanho q. Cada vértice da clique tem grau $q-1$ e portanto o grafo deve ter um $(q-1)$-*core*. Assim, $K(G')+1$ é um limite superior para o tamanho da maior clique de G'. Portanto, o maior tamanho de clique possível no grafo da Figura 5.17 é 4.

A heurística de Rossi et al. (2014) utiliza esse limite para ser capaz de encerrar o algoritmo depois de encontrar uma clique que atinja o limite superior ou interrompa uma pesquisa local mais cedo, porque não existe clique maior. O Algoritmo 5.9, traduzido de Rossi et al. (2014), descreve em detalhes a heurística.

Para exemplificar a aplicação do Algoritmo 5.9 será considerada o grafo da Figura 5.17. Na linha 2 temos a seguinte ordenação dos vértices $\{f, g, h, c, b, e, d, a\}$. Como $K(f) = 3$ é maior que $max = 0$, temos na linha 4 que $N = \{f, g, h, c, b\}$. Na primeira iteração do laço de repetição das linhas 6-10, $C = \{f\}$. Na segunda iteração, $C = \{f, g\}$, na terceira $C = \{f, g, h\}$ e na quarta $C = \{f, g, h, c\}$. Na quinta iteração, $C \cup b$ não é uma clique e portanto permanece $C = \{f, g, h, c\}$. Na linha 12, $H = \{f, g, h, c\}$ e $max = 4$. Ao retornar ao laço da linha 2, o próximo vértice é g, porém $K(g) = 3$ é menor que $max = 4$ (linha 3), portanto, retorna-se ao laço da linha 2. Para todos os próximos vértices $\{h, c, b, e, d, a\}$ a verificação da linha 3 irá falhar e então será retornado $H = \{f, g, h, c\}$ na linha 16.

Algoritmo 5.9: Heurística gulosa para encontrar clique máxima em grafos gerais. O *array* $\hat{K}$ armazena os *core numbers* dos vértices.

Entrada: G = (V, E), $\hat{K}$

```
1  H = { }, max = 0
2  para cada v ∈ V em ordem decrescente de core number faça
3      se K(v) ≥ max então
4          Seja N os vizinhos de v com core number ≥ max
5          C = { }
6          para cada vértice u ∈ N em ordem decrescente de core number faça
7              se C ∪ {u} é uma clique então
8                  Adicione u à C
9              fim
10         fim
11         se |C| > max então
12             H = C, max = |H|
13         fim
14     fim
15 fim
16 retorne H, uma clique em G
```

Objetivando propor um algoritmo de clique máxima que considere as especificidades de um grafo k-partido, foram feitas algumas modificações, destacadas em azul, no Algoritmo Rossi et al. (2014) que resultaram no Algoritmo 5.10.

Além do conceito de *core number*, o algoritmo proposto se baseia no conceito de grau de partição, definido da seguinte forma:

Definição 5.3 (Grau de partição) *O grau de partição de um vértice v é o número de partições distintas adjacentes ao vértice v. Denotamos o grau de partição de um vértice v por $P(v)$ e denotamos por $P(G)$ o maior grau de partição em todo o grafo G.*

A Figura 5.18 exibe os graus de partição de cada um dos vértices do grafo 5-partido, dada as partições $V_1 = \{a\}$, $V_2 = \{b, c, d\}$, $V_3 = \{e\}$, $V_4 = \{f\}$ e $V_5 = \{g\}$.

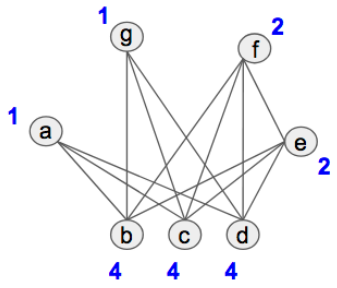


Figura 5.18: Exemplo de um grafo com os graus de partição de cada um de seus vértices

Algoritmo 5.10: Heurística gulosa para encontrar clique máxima em grafos k-partidos. O *array* $\hat{K}$ armazena os *core numbers* dos vértices. O *array* $\hat{P}$ armazena os graus de partição dos vértices.

Entrada: G = (V, E), com conjunto de partições $V_1, V_2, \ldots, V_{v_n}$, $\hat{K}$, $\hat{P}$

```
1  //Seja V_c o conjunto de partições cobertas por C
2  //Seja i o número da partição à qual o vértice u pertence
3  para cada v ∈ V em ordem decrescente de min[grau de partição, core number] faça
4      se min[K(v), P(v)] ≥ max então
5          Seja N os vizinhos de v com min[grau de partição, core number] ≥ max
6          se |N| ≥ max então
7              C = { }, V_c = { }
8              para cada vértice u ∈ N em ordem decrescente de min[grau de partição,
                 core number] faça
9                  se i ∉ V_c então
10                     se C ∪ {u} é uma clique então
11                         Adicione u à C, Adicione i à V_c
12                     fim
13                 fim
14             fim
15             se |C| > max então
16                 H = C, max = |H|
17             fim
18         fim
19     senão
20         interrompa
21     fim
22 fim
23 retorne H, uma clique em G
```

O grafo da Figura 5.18 possui três cliques máximas $\{b, e, f\}$, $\{c, e, f\}$ e $\{d, e, f\}$. Os vértices a e g não participam de nenhuma clique máxima e por este motivo uma boa estratégia seria não iniciar a pesquisa em torno desses vértices. Contudo, todos os vértices do grafo da Figura 5.18 possuem *core number* igual à 3. Portanto, qualquer ordenação dos vértices seria válida considerando a heurística de Rossi et al. (2014).

Assim como o *core number*, o grau de partição também pode ser considerado como critério para ordenação uma vez que também representa um limite superior. Por exemplo, o vértice a possui grau de partição igual à 1, o que representa que seus vizinhos estão dentro de uma única partição, no caso a partição V_2. Portanto, uma clique em torno do vértice a é limitada superiormente à $P(a) + 1 = 2$. Como, no grafo da Figura 5.18, o *core number* dos vértices não é capaz de diferenciá-los, o grau de partição dos vértices pode ser usado para prover uma ordenação mais apropriada.

Dito isso, a primeira modificação que se pode notar no Algoritmo 5.10 é o uso do grau de partição, além do *core number*, para determinar o limite superior. Observa-se que é considerado o menor valor, dentre os dois, de modo a definir um limite mais restrito.

A modificação referente à linha 7, insere uma verificação para interromper uma pesquisa local mais cedo, quando não for possível construir uma clique maior. Suponha que a vizinhança N de v é uma clique, portanto temos uma clique de tamanho $|N|$. Se $|N| < max$, não será possível encontrar uma clique maior, em torno do vértice v, do que a que já foi encontrada. Portanto, a pesquisa local em torno do vértice v pode ser interrompida.

No Algoritmo 5.9, a pesquisa em torno de um vértice v, considera apenas os vizinhos de v que são capazes de formar uma clique com tamanho maior que max (linha 4). Em um grafo k-partido, vértices de uma mesma partição não possuem arestas entre si e portanto não formam uma clique. Assim, se um vizinho u, de uma determinada partição V_i, pertence à clique C, também não há a necessidade de considerar os vizinhos de v que pertencem à mesma partição V_i (essa modificação engloba as linhas 8, 10 e 12 do Algoritmo 5.10). Por exemplo, o vértice f possui os vizinhos $\{f, b, c, d, e\}$. Considerando $C = \{f, b\}$ não é necessário verificar se os vizinhos c e d são capazes de aumentar a clique, pois eles pertencem à mesma partição de b e portanto não há arestas entre eles.

A modificação correspondente à linha 21 interrompe o laço de repetição da linha 4 quando não é possível encontrar uma clique maior que o tamanho max já encontrado. No exemplo da aplicação do Algoritmo 5.9, após verificar que o *core number* do vértice g era menor que max, o algoritmo continuou verificando os vértices restantes $\{h, c, b, e, d, a\}$. Porém, como os vértices estão ordenados, se a verificação falhou para o vértice g então também irá falhar para os demais vértices. Portanto, o algoritmo poderia ser interrompido, uma vez que não será possível encontrar uma clique maior do que a que já foi encontrada.

Considerando essas modificações, ao aplicar o Algoritmo 5.10 ao grafo da Figura 5.18 temos na linha 4 a seguinte ordenação dos vértices {b,c,d,e,f,a,g}. Como $min[K(b) = 3, P(b) = 4] = 3$ é maior que $max = 0$, temos na linha 6 que $N = \{b, e, f, g, a\}$. Na primeira iteração do laço de repetição das linhas 9-15, $C = \{b\}$ e $V_c = \{2\}$. Na segunda iteração, $C = \{b, e\}$ e $V_c = \{2, 3\}$, na terceira $C = \{b, e, f\}$ e $V_c = \{2, 3, 4\}$, e na quarta $C = \{b, e, f, g\}$ e $V_c = \{2, 3, 4, 5\}$. Na quinta iteração, $C \cup a$ não é uma clique e portanto permanece $C = \{b, e, f, g\}$ e $V_c = \{2, 3, 4, 5\}$. Na linha 17, $H = \{b, e, f, g\}$ e $max = 4$. Ao retornar ao laço da linha 4, o próximo vértice a ser considerado é o vértice c, porém $min[K(c) = 3, P(c) = 4] = 3$ é menor que $max = 4$ (linha 5), portanto, o algoritmo é interrompido na linha 21 e $H = \{b, e, f, g\}$ é retornado na linha 24.

5.8 Abordagem mono-objetivo

Dado que o Problema 5.1 se trata de um problema multiobjetivo, no presente trabalho será explorada uma abordagem baseada em preferência, transformando os múltiplos objetivos em um único objetivo. O procedimento de transformação será realizado pelo método de soma ponderada.

Como o cálculo do objetivo de cobertura representa o intervalo [0,1], foi necessário normalizar a escala de valores do cálculo do comprimento e número de *resets*. Dado que ambos os objetivos são de minimização, foi utilizada a seguinte normalização de modo que o valor mínimo represente o valor 1 o valor máximo represente 0, sendo possível assim converter todos os objetivos para um único tipo (maximização):

$$(z_i)_N = \frac{z_{max} - z_i}{z_{max} - z_{min}}$$

Dessa forma, o problema passa a ser a *maximização* da seguinte função:

$$w_c \times cobertura + w_l \times comprimentoNormalizado + w_r \times resetNormalizado \quad (5\text{-}1)$$

onde w_c, w_l e w_r representam os pesos correspondentes aos objetivos de cobertura, comprimento e número de *resets*, respectivamente.

Como não se tem a priori um conhecimento dos valores mínimos e máximos de comprimento e número de *resets*, esse valores são definidos com base nos indivíduos da população inicial. À medida que o algoritmo evolui, esses valores são ajustados com base nos indivíduos da população corrente. Em outras palavras, dada uma geração, se um indivíduo é incluído na população e os valores de seu comprimento e/ou número de *resets* são maiores que os máximos e/ou menores que os mínimos, esses valores são atualizados e é necessário atualizar o valor da *fitness* de todos os indivíduos da população corrente. O critério que define se um indivíduo novo será incluído na população se baseia na comparação da *fitness* do novo indivíduo e do pior indivíduo da população corrente (linha 5 do Algoritmo 5.1). Caso o novo indivíduo possua valores que extrapolem os valores máximos e mínimos, o valor da *fitness* do pior indivíduo deve ser normalizada considerando os valores do novo indivíduo para que a comparação seja realizada.

5.9 Considerações finais

O problema de geração de sequências de teste a partir de MEFs pode ser modelado como um problema de otimização com três objetivos: maximizar a cobertura de defeitos, minimizar o comprimento do conjunto de teste e minimizar o número de sequências de teste. Neste capítulo foi proposto o método EATS, um algoritmo evolutivo

projetado considerando as especificidades do referido problema de geração de sequências de teste. Foi apresentado o projeto da representação, operadores de mutação, inicialização e função *fitness*. No próximo capítulo é apresentada a avaliação experimental do método aqui proposto.

CAPÍTULO 6

Avaliação do método EATS

6.1 Considerações iniciais

Com o objetivo de avaliar o método proposto sob uma perspectiva da área de aplicação, estudos experimentais foram realizados com o intuito de responder sobre quão eficientes e eficazes são os conjuntos de teste gerados pelo método proposto. Para tanto, o método EATS foi comparado com os métodos de geração da Seção 3.3.1. Os experimentos consistem na comparação dos conjuntos de sequências de teste gerados com respeito ao comprimento do conjunto e o número de operações *resets*. Ademais, foi conduzido um experimento para verificar a capacidade de cobertura de defeitos dos conjuntos gerados pelo método EATS. As MEFs utilizadas nos experimentos foram geradas aleatoriamente por meio do método utilizado nos trabalhos Endo e Simao (2012), Endo e Simao (2013) e Cutigi, Simao e Souza (2016). De fato, foram utilizadas as mesmas MEFs de Cutigi, Simao e Souza (2016). Apesar de ter sido considerada uma ampla gama de MEFs aleatórias, pode ocorrer das mesmas não representarem corretamente MEFs do mundo real. Portanto, foi conduzido um estudo de caso considerando um protocolo de comunicação real.

O método EATS também será avaliado sob uma perspectiva evolutiva. Nessa análise são considerados o comportamento da curva de convergência e do método de inicialização. Por fim, será feita uma análise da eficiência e eficácia da heurística para encontrar clique máxima em grafos $k-$partidos proposta no Capítulo 5.

6.2 EATS-MS e EATS-SC

Serão avaliadas duas variações do método EATS. Ambas fazem uso da representação NDOE, dos cinco operadores de mutação, inicialização aleatória e da função de *fitness* modelada como um problema mono-objetivo (técnica de soma ponderada), todos esses apresentados no capítulo anterior. A diferença entre as duas variações é no cálculo do objetivo de cobertura de defeitos. No método EATS-MS (do inglês, *Evolutionary*

Algorithm for Test Suite - Mutation Score) o cálculo é baseado na análise de mutantes enquanto que no método EATS-SC (do inglês, *Evolutionary Algorithm for Test Suite - Sufficient Conditions*) a cobertura é medida com base nas condições de suficiência de Simão e Petrenko (2010a) considerando o Algoritmo 5.8.

A Tabela 6.1 resume os parâmetros considerados na execução dos algoritmos.

Tabela 6.1: Parametrização dos algoritmos EATS-MS e EATS-SC

Parâmetros		**Valores**				
Critério de parada		10.000 gerações sem melhora da melhor solução				
Tamanho da população		10 50 100 200				
Pesos dos objetivos	w_c	0,33	0,4	0,5	0,7	0,9
	w_l	0,33	0,3	0,25	0,15	0,05
	w_r	0,33	0,3	0,25	0,15	0,05
Constante c (inicialização)		1 5 10 20				
Taxa de aplicação dos operadores de mutação		Escolhe-se aleatoriamente entre aumentar, diminuir ou manter o tamanho do indivíduo. Caso seja feita a escolha de aumentar, escolhe-se aleatoriamente entre CCS, RCS e AUN				

Devido à estocasticidade dos algoritmos, ambos foram executados 30 vezes para cada MEF e considerada a melhor solução de cada execução. Para cada MEF é calculada a mediana das 30 execuções. Foram realizadas comparações pareadas entre o EATS e os métodos do estado da arte usando o teste *Wilcoxon rank sum* para testar a significância estatística ($p < 0,05$). Além disso, usamos a medida Vargha-Delaney $\hat{A}_{12}$ para mensurar o tamanho de efeito e avaliar o número de vezes em que o custo dos conjuntos de testes gerados pelo EATS foi menor do que o gerado pelos métodos da literatura. Consideramos a magnitude da superioridade de EATS (melhor em gerar conjuntos reduzidos) como "pequena" quando $0,56 < \hat{A}_{12} \leq 0,64$, "média" quando $0,64 < \hat{A}_{12} \leq 0,71$ e "grande" quando $0,71 < \hat{A}_{12} \leq 1$. O teste *Wilcoxon rank sum* e Vargha-Delaney $\hat{A}_{12}$ foram realizados através da ferramenta estatística R (R Core Team, 2013).

6.3 Quão eficientes são os conjuntos de teste gerados por EATS-MS e EATS-SC?

O presente experimento tem como objetivo avaliar o custo dos conjuntos gerados pelas duas variantes, EATS-MS e EATS-SC. Para tanto, será feito um comparativo com o custo dos conjuntos de teste gerados pelos métodos W, W_p, HSI, H, SPY e P. Os conjuntos serão gerados a partir de 30 MEFs completas com 10 estados, 3 entradas e 3 saídas criadas de forma aleatória.

Como resultado da parametrização dos algoritmos EATS-MS e EATS-SC, observou-se que ambas as variantes obtiveram melhor desempenho com os pesos (0,7;0,15;0,15), sendo possível gerar soluções que apresentavam custos menores que o estado da arte e valor 1 ou próximo à 1 no objetivo de cobertura, indicando que os conjuntos são completos ou com alta capacidade de detecção de defeitos. Os conjuntos de pesos (0,33;0,33;0,33), (0,4;0,3;0,3) e (0,5;0,25;0,25) resultaram em soluções que priorizam o custo, porém se distanciam do objetivo de cobertura de defeitos. De forma contrária, os pesos (0,9;0,05;0,05) resultaram em soluções que são boas no objetivo de cobertura, porém piores nos objetivos de comprimento e número de *resets*.

Com respeito à constante c do método de inicialização, observou-se que valores maiores da constante permitiram que os algoritmos apresentassem ao final da evolução conjuntos mais eficientes e eficazes, exceto com a configuração $c = 20$, a qual não apresentou diferença significativa em relação à configuração $c = 10$ e por este motivo serão apresentados os resultados referentes à $c = 10$.

Quanto à parametrização do tamanho da população, a Tabela 6.2 apresenta os custos observados variando o tamanho da população e considerando os pesos (0,7;0,15;0,15) e $c = 10$. São apresentadas as medianas e o desvio padrão do comprimento e número de *resets*.

Tabela 6.2: Custo das soluções geradas por EATS-MS e EATS-SC variando o tamanho da população

	Tamanho da população	**EATS-MS** Mediana	DV	**EATS-SC** Mediana	DV
Comprimento	10	98,75	9,56	174,75	10,65
	50	80,25	7,91	152,5	12,77
	100	73,75	7,80	147,5	11,51
	200	69,25	7,15	142	12,34
Número de *resets*	10	13	1,57	25	1,58
	50	10	1,30	22	2,11
	100	8,75	1,31	21,25	1,98
	200	7,75	1,05	20,25	2,05

A variação do método proposto baseada em análise de mutantes (EATS-MS) mostrou-se mais promissora em gerar conjuntos de teste mais eficientes do que a variação baseada em condições de suficiência (EATS-SC). Ademais, é possível observar na Tabela 6.2 que ambas as variantes apresentam um comportamento similar: populações maiores resultam em uma maior capacidade de reduzir o custo dos conjuntos gerados. Possivelmente isso ocorre devido ao fato que populações maiores podem resultar em um aumento da diversidade, a qual permite que a busca escape de soluções ótimas locais.

A Tabela 6.3 apresenta o custo dos conjuntos gerados pelos métodos do estado da arte. Como esperado, é possível observar que o método mais recente, P, produz conjuntos menores tanto no comprimento quanto no número de sequências e o método W produz o maior conjunto no que diz respeito aos dois critérios. A variante EATS-MS, mesmo com uma população de tamanho 10, é capaz de gerar um conjunto com custo ainda menor que o conjunto gerado pelo método P. Enquanto que a variante EATS-SC, com uma população de tamanho 10, é capaz de gerar conjuntos com um menor número de sequências que P, porém com um comprimento maior. Aumentando-se o tamanho da população, os conjuntos gerados por EATS-SC passam a ter um comprimento menor que os conjuntos resultantes do método P.

Tabela 6.3: Custo dos conjuntos de teste gerados pelos métodos do estado da arte

Método	**Comprimento**		**Número de *resets***	
	Mediana	DV	Mediana	DV
W	346,5	55,96	63	9,11
W_p	268,5	24,04	52	4,65
HSI	263,5	22,59	50,5	4,63
H	191,5	14,66	30	3,34
SPY	201,5	18,29	32,5	3,54
P	161	17,40	26,5	3,03

Contudo, o aumento do tamanho da população resulta em um maior tempo de execução, conforme pode ser observado na Figura 6.1, a qual apresenta o tempo de execução, em escala logarítmica, do método P e das variantes EATS-MS e EATS-SC com populações de tamanho 10, 50 e 200. Ademais, é possível notar que a variante EATS-SC requer um tempo de execução notoriamente superior em relação aos métodos P e EATS-MS, mesmo com populações menores, possivelmente devido ao fato do cálculo do objetivo de cobertura possuir a complexidade de encontrar cliques máximas. Enquanto que a variante EATS-MS apresentou tempos similares ao método P. Assim, considerando o custo dos conjuntos gerados e o tempo de execução necessário, a variante EATS-MS é a mais recomendada para gerar conjuntos eficientes.

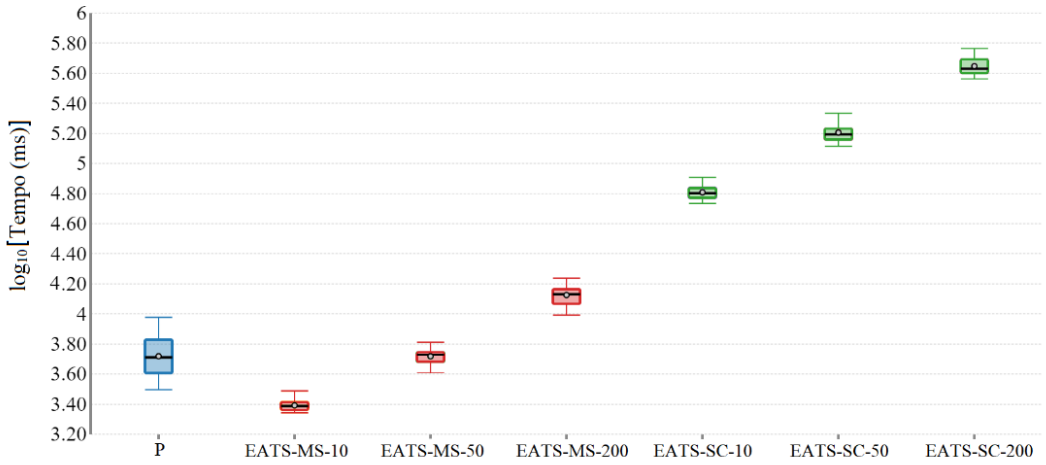


Figura 6.1: Tempo de execução do método P e das variantes EATS-MS e EATS-SC com populações de tamanho 10, 50 e 200

6.3.1 Variações nos parâmetros das MEFs

Com o objetivo de avaliar o impacto da variação dos parâmetros das MEFs no custo dos conjuntos gerados, foram utilizadas MEFs com diferentes números de estados, entradas e saídas. Com respeito à variação do número de estados, foram geradas 30 MEFs aleatórias com o número de estados variando entre 5 e 15 e o número de entradas e saídas fixados em 3. Em relação à variação do número de entradas/saídas, foram utilizadas 30 MEFs com 10 estados variando o número de entradas/saídas entre 2 e 5 e fixando o outro parâmetro em 3.

Assim como nos experimentos da Seção 6.3, foram analisados os conjuntos de teste gerados pelos métodos W, W_p, HSI, H, SPY e P e as variantes EATS-MS e EATS-SC. Considerando os resultados da Seção 6.3 relativos a parametrizações, a configuração utilizada, em ambas variantes, foi pesos (0.7,0.15,0.15) para a função objetivo, população de tamanho 50 e $c = 10$, visto que $c = 20$ não apresentou melhora significativa.

Os resultados serão apresentados em dois gráficos de linhas, um para o comparativo com respeito ao comprimento do conjunto de teste e outro para o comparativo com respeito ao número de *resets*.

6.3.1.1 Variação do número de estados

A Figura 6.2 apresenta como o custo dos conjuntos produzidos pelos métodos da literatura e os algoritmos EATS-MS e EATS-SC variam em relação à variação do número de estados. Para cada MEF, os métodos EATS-MS e EATS-SC são executados 30 vezes e considerada a melhor solução de cada execução. Para cada MEF é calculada a mediana das 30 execuções. Como são consideradas 30 MEFs para cada configuração de número de

estados, é calculada a mediana para cada configuração. As tabelas 6.4 e 6.5 apresentam o *p-value* do comparativo da variante EATS-MS e EATS-SC com os métodos do estado da arte, respectivamente. A Tabela 6.6 apresenta os valores de $\hat{A}_{12}$ do comparativo com o método EATS-SC. Para variante EATS-MS os valores de $\hat{A}_{12}$ foram todos iguais à 1.

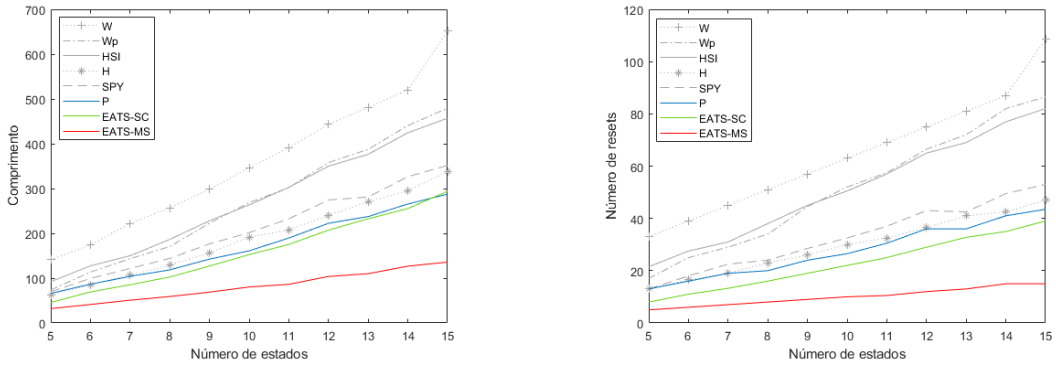


Figura 6.2: Custo variando o número de estados

Todos os métodos apresentam um crescimento aproximadamente linear, com W tendo o maior custo e EATS-MS o menor custo. Existe uma forte correlação positiva entre o comprimento/número de *resets* e o número de estados (em média acima de 0.99/0.99); portanto, os métodos tendem a gerar conjuntos de teste com maior custo ao adicionar mais estados à máquina. Dentre os métodos do estado da arte, apenas o método P está ressaltado na figura para evidenciar que ele representa o método da literatura que apresenta uma maior capacidade de gerar conjuntos com menor custo e, portanto, atrair a atenção do comparativo dos métodos EATS-MS e EATS-SC com o referido método.

A Figura 6.3 exibe em detalhes apenas o comparativo entre os métodos H, P, EATS-MS e EATS-SC, evidenciando que dentre os métodos do estado da arte, o método P tem uma maior capacidade de gerar conjuntos com menor custo. Apenas nas configurações com 5 e 6 estados o método H apresenta conjuntos menores que o método P com respeito ao comprimento, porém não com respeito ao número de *resets*. O método EATS-SC é capaz de gerar conjuntos com custo ainda menor que o método H e P, exceto na configuração com 15 estados, na qual o método EATS-SC gera conjuntos com menor número de *resets* que o método P, porém com comprimento maior.

Pode-se observar nas tabelas 6.5 e 6.6, que na maioria dos casos houve diferença estatística ($p < 0,05$) e a magnitude da superioridade do método EATS-SC foi grande (próximo à 1) em relação ao custo dos conjuntos gerados pelos métodos do estado da arte. Em relação ao método P, nas configurações com 10, 12, 13 e 14 estados não houve diferença estatística com respeito ao comprimento, porém o método EATS-SC foi superior apresentando magnitude média (para as configurações com 10, 12 e 14 estados) e pequena (para configuração com 13 estados). Na configuração com 15 estados, além de

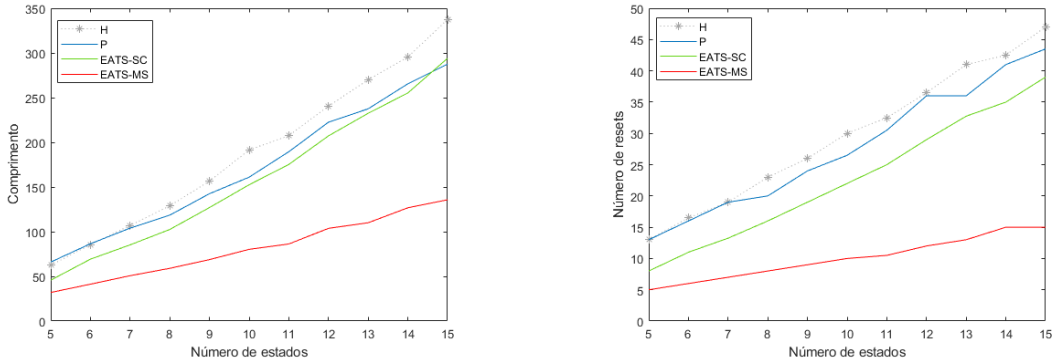


Figura 6.3: Detalhe da Figura 6.2 exibindo apenas os conjuntos gerados por H, P, EATS-MS e EATS-SC

não haver diferença estatística com respeito ao comprimento, o método P foi superior ao método EATS-SC, sendo capaz de gerar conjuntos com menor comprimento em 55% das vezes que foi aplicado. Visto que o aumento do número de estados representa uma maior complexidade do problema, seria recomendável um ajuste no tamanho da população da variante EATS-SC, de modo que o aumento do número de estados fosse acompanhado do aumento do tamanho da população. Dessa forma, a variante se beneficiaria, dado que populações maiores resultam em uma maior capacidade de reduzir o custo dos conjuntos gerados.

Por fim, de uma maneira estatisticamente significativa (conforme observado na Tabela 6.4), o método EATS-MS foi capaz de gerar conjuntos com o menor custo dentre todos os métodos. Além disso, apresentou $\hat{A}_{12} = 1$ em relação à todos os métodos do estado da arte, ou seja foi capaz de gerar conjuntos com menor custo em 100% das vezes que foi aplicado, reforçando o que foi evidenciado nos resultados anteriores de que a variante EATS-MS é a mais recomendada para gerar conjuntos eficientes.

6.3.1.2 Variação do número de entradas

A Figura 6.4 apresenta como o custo dos conjuntos variam em relação ao número de entradas. O comportamento de cada método é semelhante ao apresentado na Figura 6.3, com W tendo o maior custo e EATS-MS o menor custo. Uma forte correlação positiva também é observada entre o comprimento/número de *resets* e entradas (em média acima de 0.99/0.99); portanto, os métodos tendem a gerar conjuntos de testes com maior custo ao adicionar mais entradas à máquina.

As tabelas 6.7 e 6.8 apresentam o *p-value* do comparativo da variante EATS-MS e EATS-SC com os métodos do estado da arte, respectivamente. A Tabela 6.9 apresenta os valores de $\hat{A}_{12}$ do comparativo com o método EATS-SC. Para variante EATS-MS os valores de $\hat{A}_{12}$ foram todos iguais à 1.

Tabela 6.4: EATS-MS *versus* métodos da literatura - valores de *p-value* do teste de *Wilcoxon rank sum* variando o número de estados

Método	**5 estados**		**6 estados**		**7 estados**		**8 estados**		**9 estados**		**10 estados**		**11 estados**		**12 estados**		**13 estados**		**14 estados**		**15 estados**	
	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets
W	5,60E-10	2,00E-10	5,30E-10	1,30E-10	5,90E-10	2,20E-11	6,00E-10	5,10E-11	6,00E-10	8,80E-11	6,20E-10	1,20E-10	6,00E-10	2,60E-10	6,30E-10	2,60E-10	6,30E-10	3,10E-10	6,10E-10	2,90E-10	6,30E-10	3,30E-10
W_p	6,20E-10	4,00E-10	6,20E-10	5,20E-10	6,20E-10	3,30E-10	6,30E-10	4,90E-10	6,30E-10	5,20E-10	6,30E-10	5,60E-10	6,20E-10	5,80E-10	6,30E-10	5,30E-10	6,30E-10	6,00E-10	6,30E-10	5,70E-10	6,30E-10	5,70E-10
HSI	6,20E-10	4,00E-10	6,20E-10	5,30E-10	6,20E-10	3,20E-10	6,30E-10	5,00E-10	6,30E-10	5,10E-10	6,30E-10	5,60E-10	6,20E-10	5,80E-10	6,30E-10	5,30E-10	6,30E-10	5,90E-10	6,30E-10	5,60E-10	6,30E-10	5,50E-10
H	6,20E-10	3,90E-10	6,20E-10	5,20E-10	6,20E-10	3,00E-10	6,20E-10	4,80E-10	6,30E-10	5,10E-10	6,30E-10	5,50E-10	6,20E-10	5,70E-10	6,30E-10	5,40E-10	6,30E-10	5,90E-10	6,30E-10	5,60E-10	6,30E-10	5,60E-10
SPY	6,20E-10	3,90E-10	6,20E-10	5,20E-10	6,20E-10	3,20E-10	6,20E-10	4,60E-10	6,30E-10	5,10E-10	6,30E-10	5,40E-10	6,20E-10	5,70E-10	6,30E-10	5,40E-10	6,30E-10	5,80E-10	6,30E-10	5,70E-10	6,30E-10	5,50E-10
P	6,10E-10	3,50E-10	6,20E-10	4,90E-10	6,20E-10	3,00E-10	6,30E-10	4,80E-10	6,30E-10	5,10E-10	6,30E-10	5,50E-10	6,20E-10	5,70E-10	6,30E-10	5,10E-10	6,30E-10	5,80E-10	6,30E-10	5,60E-10	6,30E-10	5,50E-10

Tabela 6.5: EATS-SC *versus* métodos da literatura - valores de *p-value* do teste de *Wilcoxon rank sum* variando o número de estados. Os valores em **negrito** representam que não houve diferença estatística

Método	**5 estados**		**6 estados**		**7 estados**		**8 estados**		**9 estados**		**10 estados**		**11 estados**		**12 estados**		**13 estados**		**14 estados**		**15 estados**	
	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets
W	5,60E-10	2,70E-10	5,30E-10	1,40E-10	5,90E-10	8,60E-10	6,00E-10	5,50E-11	6,00E-10	9,60E-11	6,20E-10	1,30E-10	6,10E-10	2,70E-10	6,20E-10	2,80E-10	6,30E-10	3,20E-10	6,20E-10	3,10E-10	6,20E-10	3,40E-10
W_p	5,90E-09	5,40E-10	2,50E-09	5,40E-10	7,30E-10	5,30E-10	6,30E-10	5,30E-10	8,50E-10	5,50E-10	6,30E-10	5,90E-10	6,30E-10	6,00E-10	6,30E-10	5,80E-10	6,30E-10	6,10E-10	6,30E-10	6,00E-10	6,90E-10	5,90E-10
HSI	6,20E-10	5,40E-10	6,20E-10	5,50E-10	6,30E-10	6,80E-10	6,30E-10	5,30E-10	6,30E-10	3,50E-08	6,30E-10	5,90E-10	6,30E-10	6,00E-10	6,30E-10	5,70E-10	6,30E-10	6,00E-10	6,30E-10	5,90E-10	6,30E-10	5,70E-10
H	2,50E-07	7,30E-10	6,00E-05	9,40E-09	4,60E-07	1,40E-09	6,10E-07	2,30E-09	4,10E-06	5,50E-10	2,10E-08	1,20E-09	1,20E-06	6,10E-08	1,40E-04	5,50E-07	1,70E-06	8,90E-07	1,50E-08	2,90E-08	1,60E-07	2,60E-06
SPY	2,50E-09	1,00E-09	1,30E-08	1,60E-09	8,50E-10	4,20E-11	9,30E-10	1,50E-09	8,30E-09	9,10E-10	5,40E-09	1,50E-09	8,50E-10	5,90E-10	6,30E-10	6,70E-10	9,60E-09	1,70E-09	1,70E-09	6,00E-10	4,90E-09	8,80E-10
P	1,60E-07	7,90E-10	2,20E-05	1,10E-08	5,30E-08	5,40E-10	2,00E-04	5,30E-09	4,70E-04	7,10E-08	**8,40E-01**	1,60E-04	1,94E-02	3,50E-07	**5,24E-02**	4,00E-08	**1,00E+00**	1,30E-05	**2,31E-01**	1,20E-06	**1,00E+00**	1,96E-03

Tabela 6.6: EATS-SC *versus* métodos da literatura - valores de tamanho de efeito (Vargha-Delaney $\hat{A}_{12}$) variando o número de estados. O valor em **negrito** representa que o método EATS-SC não foi superior.

Método	5 estados		6 estados		7 estados		8 estados		9 estados		10 estados		11 estados		12 estados		13 estados		14 estados		15 estados	
	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets
W	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
W_p	0,97	1	0,98	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
HSI	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
H	0,93	0,99	0,85	0,97	0,92	0,99	0,92	0,98	0,89	0,95	0,96	0,99	0,91	0,95	0,84	0,92	0,90	0,91	0,96	0,96	0,93	0,90
SPY	0,98	0,99	0,97	0,99	1	0,99	1	0,99	0,97	0,99	0,98	0,99	1	1	1	1	0,97	0,99	0,99	1	0,98	0,99
P	0,93	0,99	0,87	0,96	0,95	1	0,83	0,97	0,82	0,94	0,66	0,84	0,75	0,92	0,73	0,95	0,60	0,87	0,69	0,91	**0,45**	0,79

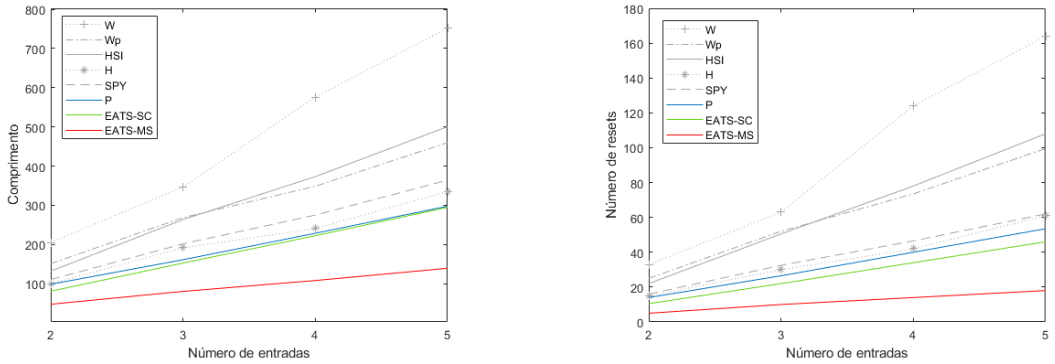


Figura 6.4: Custo variando o número de entradas

Nesse experimento, o método P apresentou conjuntos com o menor custo dentre os métodos do estado da arte, como pode ser melhor observado na Figura 6.5. Novamente, o comprimento dos conjuntos gerados por EATS-SC se aproxima dos valores apresentados pelo método P à medida que aumenta-se a complexidade da máquina, sendo possível observar na Tabela 6.8 que não houve diferença estatística para MEFs com um maior número de entradas e a superioridade do método EATS-SC em relação ao comprimento diminui à medida que aumenta-se o número de entradas, conforme pode ser observado na Tabela 6.9. Contudo, de uma maneira estatisticamente significativa (conforme observado na Tabela 6.8), o número de *resets* dos conjuntos gerados por EATS-SC se mantém menor mesmo adicionando mais entradas à maquina. É importante salientar que o comprimento da operação *reset* está sendo considerado como 1, porém na prática essa operação possui valores maiores e que portanto irão impactar no comprimento total do conjunto.

O método EATS-MS, assim como observado nos outros experimentos, foi capaz de gerar conjuntos mais eficientes que todos os métodos, apresentando diferença estatística em todos os comparativos realizados (Tabela 6.7) e magnitude de superioridade grande.

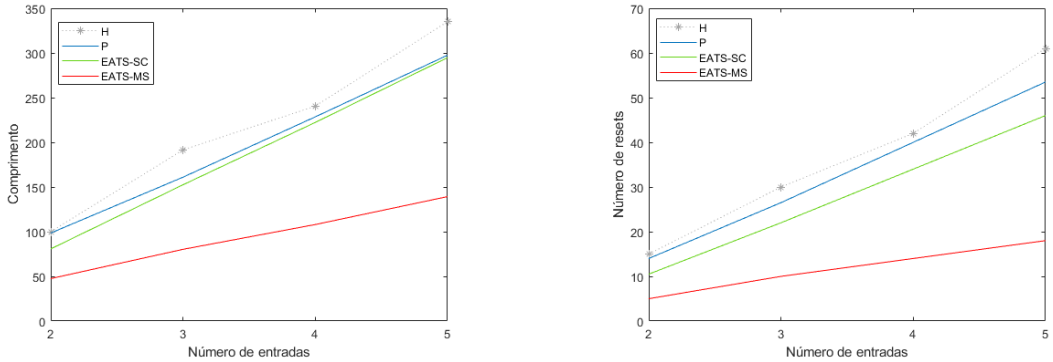


Figura 6.5: Detalhe da Figura 6.4 exibindo apenas os conjuntos gerados por H, P, EATS-MS e EATS-SC

Tabela 6.7: EATS-MS *versus* métodos da literatura - valores de *p-value* do teste de *Wilcoxon rank sum* variando o número de entradas

Método	**2 estados**		**3 estados**		**4 estados**		**5 estados**	
	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets
W	6,20E-10	2,30E-10	6,20E-10	1,20E-10	5,90E-10	2,00E-10	5,90E-10	3,30E-10
W_p	6,30E-10	4,70E-10	6,30E-10	5,60E-10	6,30E-10	5,40E-10	6,30E-10	5,90E-10
HSI	6,20E-10	2,90E-10	6,30E-10	5,60E-10	6,30E-10	5,30E-10	6,30E-10	5,90E-10
H	6,30E-10	4,70E-10	6,30E-10	5,50E-10	6,30E-10	5,20E-10	6,30E-10	6,00E-10
SPY	6,30E-10	3,80E-10	6,30E-10	5,40E-10	6,30E-10	5,30E-10	6,30E-10	5,90E-10
P	6,30E-10	4,50E-10	6,30E-10	5,50E-10	6,30E-10	5,30E-10	6,30E-10	5,90E-10

Tabela 6.8: EATS-SC *versus* métodos da literatura - valores de *p-value* do teste de *Wilcoxon rank sum* variando o número de entradas. Os valores em **negrito** representam que não houve diferença estatística

Método	**2 estados**		**3 estados**		**4 estados**		**5 estados**	
	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets
W	6,20E-10	2,60E-10	1,30E-10	6,10E-10	5,90E-10	2,20E-10	5,90E-10	3,40E-10
W_p	6,30E-10	5,50E-10	5,90E-10	6,30E-10	6,30E-10	5,80E-10	6,30E-10	6,00E-10
HSI	6,20E-10	3,30E-10	5,90E-10	6,30E-10	6,30E-10	5,70E-10	6,30E-10	6,10E-10
H	4,80E-06	3,80E-07	1,20E-09	1,40E-07	6,80E-04	4,80E-08	1,10E-04	2,60E-09
SPY	3,30E-09	7,00E-10	1,50E-09	8,90E-10	1,50E-09	1,90E-09	2,60E-09	1,50E-09
P	2,80E-04	2,10E-07	1,60E-04	1,20E-03	**1,00E+00**	3,70E-08	**1,00E+00**	2,50E-07

6.3.1.3 Variação do número de saídas

A Figura 6.6 apresenta como o custo dos conjuntos variam em relação ao número de saídas. Existe uma correlação negativa entre o comprimento/número de *resets* e o número de saídas (em média abaixo de -0.93/-0.94); portanto, os métodos tendem a gerar conjuntos de teste com menor custo ao adicionar mais saídas à máquina. Assim como

Tabela 6.9: EATS-SC *versus* métodos da literatura - valores de tamanho de efeito (Vargha-Delaney $\hat{A}_{12}$) variando o número de entradas. O valor em **negrito** representa que o método EATS-SC não foi superior.

Método	**2 estados**		**3 estados**		**4 estados**		**5 estados**	
	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets
W	1	1	1	1	1	1	1	1
W_p	1	1	1	1	1	1	1	1
HSI	1	1	1	1	1	1	1	1
H	0,89	0,92	0,99	0,94	0,81	0,95	0,84	0,98
SPY	0,98	1	0,99	1	0,99	0,99	0,98	0,99
P	0,83	0,93	0,84	0,80	0,63	0,95	**0,55**	0,93

observado nas análises anteriores com estados e entradas, o método W apresenta o maior custo e EATS-MS o menor custo.

O algoritmo EATS-MS se mostrou menos sensível à variação do número de saídas do que em relação ao número de estados e entradas, apresentando uma pequena variação no custo dos conjuntos à medida que o número de saídas aumenta. Ainda sim, apresentou diferença estatística em relação a todos os métodos da literatura (Tabela 6.10) e mostrou-se superior em 100% das vezes em que foi aplicado, uma vez que todos os valores de $\hat{A}_{12}$ foram iguais à 1.

Enquanto que o EATS-SC, similarmente ao outros experimentos, à medida que diminui-se a complexidade da MEF, nesse caso aumentando-se o número de saídas, observa-se um aumento da diferença do custo para o P. Essa diferença é ressaltada nas tabelas 6.11 e 6.12, onde observa-se diferença estatística e uma maior superioridade do método EATS-SC para MEFs com um maior número de saídas.

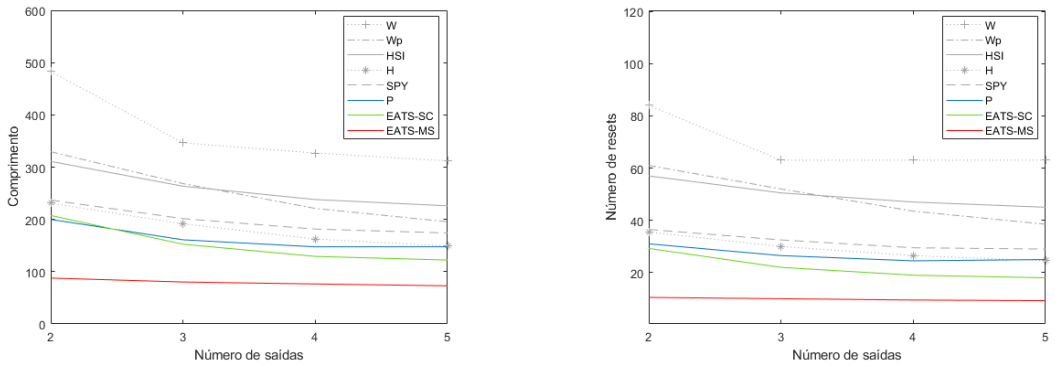


Figura 6.6: Custo variando o número de saídas

Tabela 6.10: EATS-MS *versus* métodos da literatura - valores de *p-value* do teste de *Wilcoxon rank sum* variando o número de saídas

Método	2 estados		3 estados		4 estados		5 estados	
	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets
W	6,30E-10	3,50E-10	6,20E-10	1,20E-10	6,00E-10	6,10E-11	5,60E-10	9,80E-11
W_p	6,30E-10	5,80E-10	6,30E-10	5,60E-10	6,30E-10	5,80E-10	6,30E-10	5,60E-10
HSI	6,30E-10	5,80E-10	6,30E-10	5,60E-10	6,30E-10	5,80E-10	6,30E-10	5,60E-10
H	6,30E-10	5,80E-10	6,30E-10	5,50E-10	6,30E-10	5,70E-10	6,20E-10	5,60E-10
SPY	6,30E-10	5,80E-10	6,30E-10	5,40E-10	6,30E-10	5,20E-10	6,30E-10	5,60E-10
P	6,30E-10	5,60E-10	6,30E-10	5,50E-10	6,20E-10	5,70E-10	6,30E-10	5,40E-10

Tabela 6.11: EATS-SC *versus* métodos da literatura - valores de *p-value* do teste de *Wilcoxon rank sum* variando o número de saídas. Os valores em **negrito** representam que não houve diferença estatística

Método	2 estados		3 estados		4 estados		5 estados	
	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets
W	6,30E-10	3,70E-10	6,20E-10	1,30E-10	6,10E-10	6,00E-11	5,60E-10	9,70E-11
W_p	6,30E-10	6,00E-10	6,30E-10	5,90E-10	6,30E-10	5,70E-10	6,30E-10	5,60E-10
HSI	6,30E-10	6,00E-10	6,30E-10	5,90E-10	6,30E-10	5,70E-10	6,30E-10	5,60E-10
H	2,10E-04	1,40E-07	2,10E-08	1,20E-09	1,40E-07	1,70E-08	7,20E-09	9,80E-10
SPY	5,00E-07	8,10E-08	5,40E-09	1,50E-09	8,90E-10	5,20E-10	7,30E-10	5,50E-10
P	**1,00E+00**	**1,15E-01**	**8,40E-01**	1,60E-04	1,20E-03	2,10E-08	2,60E-07	7,30E-10

Tabela 6.12: EATS-SC *versus* métodos da literatura - valores de tamanho de efeito (Vargha-Delaney $\hat{A}_{12}$) variando o número de saídas. O valor em **negrito** representa que o método EATS-SC não foi superior.

Método	2 estados		3 estados		4 estados		5 estados	
	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets
W	1	1	1	1	1	1	1	1
W_p	1	1	1	1	1	1	1	1
HSI	1	1	1	1	1	1	1	1
H	0,83	0,94	0,96	0,99	0,94	0,96	0,97	0,99
SPY	0,92	0,94	0,98	0,99	1	1	1	1
P	**0,39**	0,71	0,66	0,84	0,80	0,96	0,93	1

6.4 Quão eficazes são os conjuntos de teste gerados por EATS-MS e EATS-SC?

A análise dos resultados dos experimentos anteriores indicaram que os conjuntos gerados pelo método EATS-MS e, na maioria dos casos, os conjuntos gerados pelo método EATS-SC são mais eficientes que os conjuntos gerados pelos métodos do estado da arte, ou seja, possuem um menor custo.

No presente experimento, pretende-se avaliar qual a capacidade que os conjuntos gerados por EATS-MS e EATS-SC tem de revelar defeitos. Inicialmente, foi feita uma análise da completude dos conjuntos gerados, por ambas as variantes, no experimento com variação do número de estados. Relembrando, o experimento com variação do

número de estados considerou 30 MEFs para cada configuração do número de estados. Para cada MEF, os algoritmos EATS-MS e EATS-SC foram executados 30 vezes e considerada a melhor solução de cada execução, resultando em 30 conjuntos de teste gerados para cada MEF por cada algoritmo. Portanto, para cada configuração do número de estados foram analisados 900 conjuntos de teste para cada algoritmo.

Considerando a variante EATS-SC, para cada configuração do número de estados foi analisado quantos conjuntos dos 900 apresentavam valor 1 no objetivo de cobertura, ou seja, completo com base nas condições de suficiência definidas por Simão e Petrenko (2010a). A Tabela 6.13 apresenta o resultado desta análise.

Tabela 6.13: Percentual de conjuntos completos gerados por EATS-SC. Para cada configuração do número de estados foram analisados 900 conjuntos e a porcentagem representa quantos conjuntos, desses 900, são completos.

n	**Conjuntos completos (%)**
5	99,78
6	99,67
7	99,67
8	100
9	98,78
10	97,33
11	95
12	90
13	91,78
14	81
15	77,44

Para máquinas com um menor número de estados, a porcentagem de conjuntos que são completos é próxima de 100% ou é 100%. À medida que aumenta-se o número de estados, o percentual de conjuntos que são completos diminui.

Para os conjuntos de teste que não apresentavam valor 1 no objetivo de cobertura, foi feita uma análise empírica com base no *score* de mutação. Foram gerados 100 mil mutantes e analisado quantos desses mutantes tais conjuntos eram capazes de matar. Todos os conjuntos foram capazes de matar 100% dos mutantes, indicando que tais conjuntos possuem uma alta capacidade de detectar implementações defeituosas.

Com respeito ao método EATS-MS, assim como na análise do método EATS-SC, para cada configuração do número de estados foi analisado quantos conjuntos dos 900 apresentavam valor 1 no objetivo de cobertura. Porém, nesse caso, o valor 1 representa a completude do conjunto com base na análise de mutantes.

Foi observado que todos os conjuntos gerados apresentavam valor 1 no objetivo de cobertura. Contudo, como o cálculo do *score* de mutação é feito com base em uma estimativa, pode ocorrer do valor estimado não corresponder ao valor preciso, em

outras palavras, dada uma solução com *score* de mutação 1, considerada então completa, pode de fato não ser completa. Por este motivo, para confirmar a completude dos conjuntos gerados pelo métodos EATS-MS foi utilizada a ferramenta CheSCon (CUTIGI, 2010). Dados como entrada uma MEF e um conjunto de teste, a ferramenta verifica se determinado conjunto de teste satisfaz as condições de suficiência propostas por Simão e Petrenko (2010a). Contudo, nenhum dos conjuntos satisfizeram as condições de suficiência.

Como o fato de não satisfazer as condições de suficiência é inconclusivo, ou seja, não se pode afirmar se o conjunto é completo ou não, foi utilizada uma abordagem exaustiva para avaliar os conjuntos, uma vez que sua resposta é assertiva. A abordagem exaustiva consiste em executar todos os mutantes com o conjunto de teste a ser verificada a completude. Caso seja encontrado um mutante que não é morto pelo conjunto de teste e não está em conformidade com a especificação, a abordagem conclui que o conjunto de teste não é completo. Caso todos os mutantes que não estão em conformidade sejam mortos então o conjunto é completo. Por ser uma abordagem exaustiva, ela funciona para MEFs pequenas com apenas algumas unidades de estados. Considerando tal fato, foi feita a verificação de completude dos conjuntos gerados considerando as MEFs que possuíam 5, 6 e 7 estados, pois a partir de 8 estados o tempo de execução tornou-se impraticável. A Tabela 6.14 apresenta o resultado desta análise.

Tabela 6.14: Percentual de conjuntos completos gerados por EATS-MS

n	**Conjuntos completos (%)**
5	1,89
6	0,89
7	0,33

A partir do resultado da Tabela 6.14, é possível concluir que a técnica de verificação de completude utilizada no método EATS-MS gera muitos falso-positivos com respeito à completude devido ao cálculo estimado do *score* de mutação. Entretanto, apesar dos falso-positivos não poderem de fato detectar *todas* as máquinas de implementação defeituosas, ainda assim eles representam uma solução com a capacidade de detectar um alto número de defeitos. Ademais, os casos positivos evidenciam que há soluções, ou conjuntos de teste, com custo muito menor que os encontrados até então pelos métodos da literatura. Por exemplo, para uma MEF com 6 estados, EATS-MS encontrou um conjunto completo com comprimento 49 e 6 sequências, enquanto que o método P foi capaz de gerar um conjunto com comprimento 92 e 17 sequências. Ao mesmo tempo que, o menor conjunto completo encontrado por EATS-SC possui comprimento 53 e 8 sequências.

Considerando apenas os conjuntos completos gerados por EATS-MS e EATS-SC, a Tabela 6.15 apresenta a capacidade de redução, com respeito ao comprimento e

número de *resets*, de ambas as variantes em relação aos métodos do estado da arte.

Tabela 6.15: Média das taxas de redução do custo dos conjuntos de teste completos obtidos usando EATS-MS e EATS-SC variando o número de estados

n	Método	W (%)		Wp (%)		HSI (%)		H (%)		SPY (%)		P (%)	
		Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets	Comp.	Resets
5	EATS-MS	59,61	68,18	47,77	61,11	51,76	64,10	32,79	44	37,88	44,00	28,07	39,13
	EATS-SC	67,55	75,76	37,76	52,94	50,27	62,79	26,80	38,46	34,17	38,46	30,68	38,46
6	EATS-MS	72,73	82,05	55,26	72,00	54,87	72,00	43,33	58,82	42,05	53,33	47,42	63,16
	EATS-SC	60,20	71,79	38,99	56,00	45,69	60	19,01	33,33	30,40	38,89	19,94	31,25
7	EATS-MS	67,34	75,56	52,15	63,93	54,26	67,65	36,12	50,00	39,58	47,62	32,24	40,54
	EATS-SC	61,60	70,56	40,59	54,31	43,17	57,26	20,33	30,26	29,84	41,11	18,03	30,26
8	EATS-SC	60,12	68,63	39,88	52,94	45,04	57,89	20,54	30,43	28,82	33,33	13,50	20,00
9	EATS-SC	57,38	66,67	42,92	57,30	44,30	57,78	18,85	26,92	28,05	33,33	10,88	20,83
10	EATS-SC	55,99	65,08	43,20	57,69	42,13	56,44	20,37	26,67	24,32	32,31	5,28	16,98
11	EATS-SC	55,24	63,77	41,97	56,52	42,07	56,14	15,54	23,08	24,62	32,43	7,52	18,03
12	EATS-SC	53,22	61,33	42,11	56,39	40,79	55,38	13,83	20,55	24,50	32,56	6,85	19,44
13	EATS-SC	51,71	59,57	40,00	54,51	38,25	52,54	13,89	20,12	17,26	22,94	2,11	9,03
14	EATS-SC	50,91	59,77	42,05	57,32	39,87	54,55	13,62	17,65	21,82	29,29	3,86	14,63
15	EATS-SC	54,87	64,06	38,69	54,91	35,67	52,44	12,89	17,02	16,36	26,42	-2,26	10,34

Geralmente, o método W gera os maiores conjuntos de teste quando comparado com os outros métodos da literatura e, portanto, possui a maior taxa de redução com respeito ao EATS-MS e EATS-SC. Por outro lado, como o método P gera os menores conjuntos de teste o percentual de redução é menor. Considerando apenas as MEFs com 5, 6 e 7 estados, visto que para as demais MEFs não foi possível avaliar a completude dos conjuntos gerados por EATS-MS, o método EATS-MS possui uma maior taxa de redução média de comprimento (47,51%) e número de *resets* (61,14%) em relação aos métodos do estado da arte que o método EATS-SC (37,5% e 48,99%, respectivamente). Porém, com respeito à eficácia, como a variante EATS-SC se mostrou mais promissora em gerar conjuntos completos, constata-se que a variante EATS-SC é a mais recomendada para gerar conjuntos eficazes.

6.5 Estudo de caso: protocolo de comunicação INRES

Nesse estudo experimental, o método proposto foi avaliado considerando uma aplicação do mundo real. Para tanto foi considerada a entidade *Responder* do protocolo de comunicação INRES (*INitiator-REsponder*) (HOGREFE, 1991). O protocolo INRES é um serviço de comunicação orientado a conexão para a troca confiável de mensagens entre duas entidades de protocolo: *Initiator* e *Responder*. O *Initiator* estabelece uma conexão e envia dados. O *Responder* recebe dados e finaliza as conexões. A Figura 6.7 apresenta a MEF que modela o comportamento do *Responder* (TAN; PETRENKO; VON BOCHMANN, 1996; CUTIGI; SIMAO; SOUZA, 2016). A MEF é determinística, minimal, parcialmente especificada e contém 4 estados, 5 entradas, 8 saídas e 16 transições.

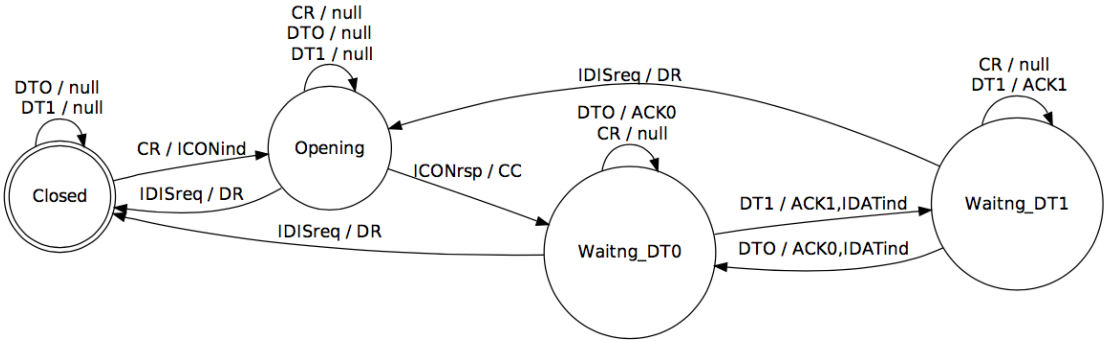


Figura 6.7: *Responder* especificado como MEF

A partir da MEF da Figura 6.7 foram aplicados os métodos HSI, H, SPY, P, EATS-MS e EATS-SC. A aplicabilidade dos métodos SPY, EATS-MS e EATS-SC é restrita à máquinas completamente especificadas. Como a MEF não é completa, foram adicionadas todas as transições ausentes em um determinado estado como *auto-loops*, ou seja, transição que inicia e termina no mesmo estado. A Tabela 6.16 mostra os conjuntos de testes gerados por todos os métodos. O símbolo '*' indica que o método foi aplicado considerando a MEF completa. Devido à estocasticidade dos métodos EATS-MS e EATS-SC ambos foram executados 30 vezes e apresentada a mediana dos valores obtidos. A configuração utilizada, em ambas variantes, foi pesos (0.7,0.15,0.15) para a função objetivo, população de tamanho 50 e $c = 10$.

Tabela 6.16: Conjuntos de teste gerados a partir da especificação da entidade *Responder*

Método	**Comprimento**	**Número de resets**
HSI	94	21
HSI*	117	27
H	65	13
H*	82	17
SPY*	95	16
P	70	14
P*	111	24
EATS-MS*	45	7,5
EATS-SC*	54	11

Como a máquina completa possui um maior número de transições é necessário um conjunto maior de teste para cobrir as transições adicionadas. Por este motivo, os conjuntos de teste gerados pelos métodos HSI, H e P para a MEF parcial foram menores que os conjuntos gerados quando considerada a MEF completa. Dentre os métodos da literatura, o método H foi o que apresentou o menor conjunto tanto para a MEF parcial quanto para a MEF completa. Os métodos EATS-MS e EATS-SC foram capazes de gerar

conjuntos ainda menores que os conjuntos do método H, mesmo tendo sua aplicação restrita à MEF completa. Ademais, é importante ressaltar que todos os conjuntos gerados pelo método EATS-SC são completos. Portanto, os conjuntos possuem a mesma eficácia do conjunto gerado por H e são mais eficientes. Com respeito ao método EATS-MS, apesar de apresentar um alto *score* de mutação, os conjuntos gerados não são completos.

6.6 Análise evolutiva

Com o objetivo de avaliar o método proposto sob uma perspectiva evolutiva, foi feita uma análise do método de inicialização e da curva de convergência do método proposto.

6.6.1 Método de inicialização

O método de inicialização gera a população inicial construindo soluções aleatórias aplicando diversas vezes o operador de mutação AUN, o qual adiciona um novo vértice ao indivíduo mutado. A escolha do tamanho de cada indivíduo a ser gerado, ou seja, o número de vértices é determinado pela escolha aleatória de um valor no intervalo $[n_i + 1, n \times n_i \times c]$, onde n_i e n representam o número de entradas e número de estados da MEF, respectivamente, e c é uma constante positiva.

Visando averiguar o comportamento do método de inicialização variando o valor da constante c, foi analisado o espalhamento das soluções da população inicial pelo valor de *fitness*. Para tanto, foi considerada uma máquina com 10 estados, 3 entradas e 3 saídas e os valores de c como 1, 5, 10 e 20. Devido à estocasticidade do método de inicialização, para cada valor de c foram geradas 10 populações iniciais cada uma com 1.000 indivíduos. O valor do objetivo de cobertura foi calculado com base nas condições de suficiência Simão e Petrenko (2010a), visto que o cálculo baseado na estimativa do *score* de mutação não fornece uma precisão com respeito à analise de completude das soluções.

Os resultados observados entre as 10 populações de cada valor de c foram similares e por este motivo serão apresentados apenas os resultados de uma única população para cada valor da constante. A Figura 6.8 apresenta como o espalhamento das soluções da população inicial pelo valor de *fitness* variou em relação ao valor da constante c. O intervalo das classes é de 0,01.

É possível observar que para valores menores da constante, há uma maior concentração de soluções com baixo valor de *fitness*. Ao mesmo tempo que, aumentando-se o valor da constante nota-se um aumento no número de soluções de alta qualidade, ou seja, com alto valor de *fitness*. Isso pode ser explicado pelo fato de que o aumento da constante permite gerar soluções com um maior número de vértices. Por sua vez,

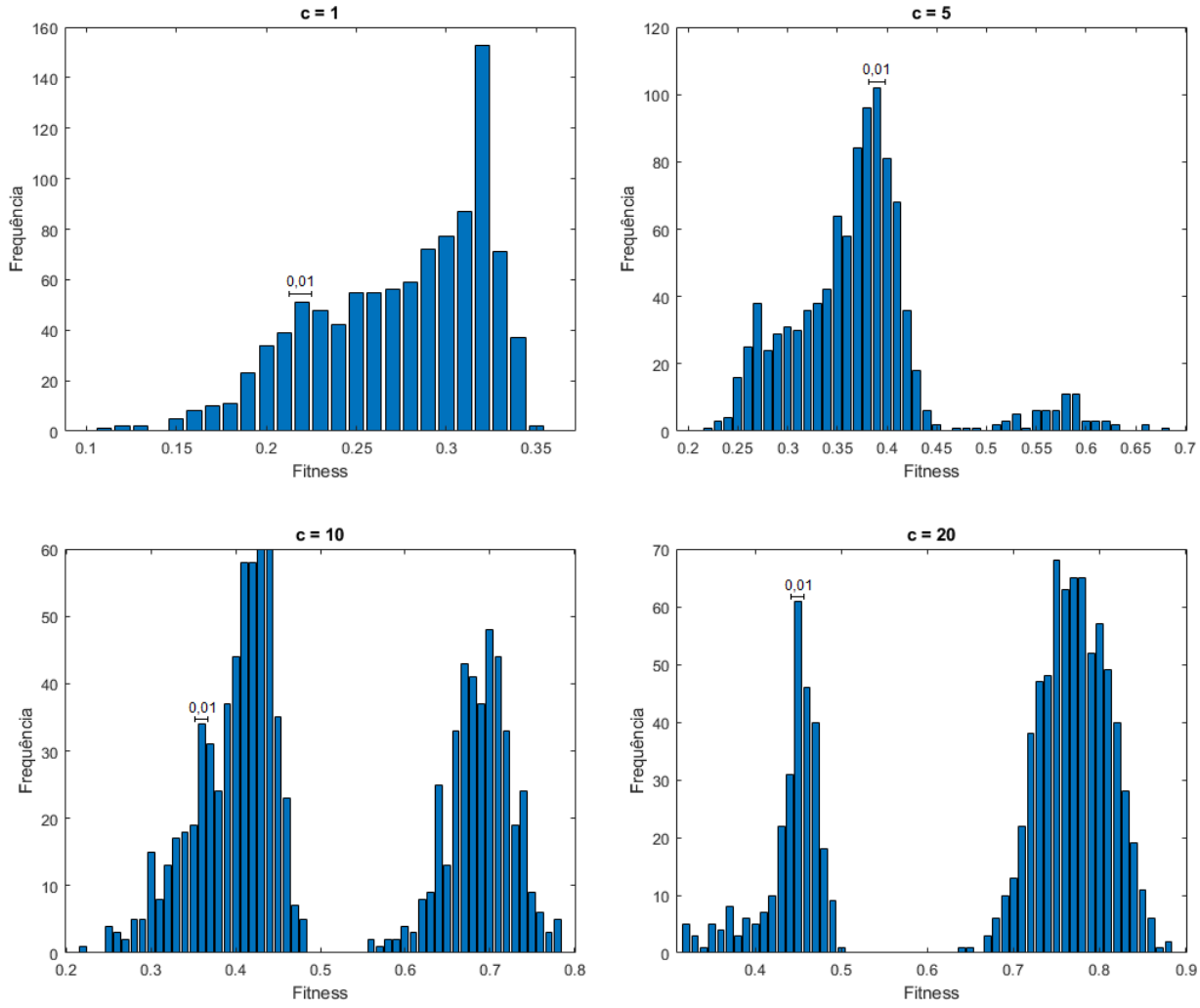


Figura 6.8: Espalhamento dos indivíduos da população inicial pelo valor de *fitness* considerando valores da constante c como 1, 5, 10 e 20

o aumento do número de vértices define soluções com maior comprimento, o que pode representar soluções com maior capacidade de cobertura de defeitos. Como o objetivo de cobertura possui um maior peso no cálculo da *fitness* (Equação 5-1), soluções que possuem alta cobertura contribuem para um maior valor da *fitness*, bem como soluções com baixa cobertura tendem a apresentar valores menores de *fitness*. Esse último fato pode ser melhor observado na Figura 6.9, a qual apresenta o espalhamento das soluções considerando cada componente da *fitness*. A população observada diz respeito à população gerada com $c = 10$ da Figura 6.8 .

O comportamento do espalhamento das soluções considerando a cobertura é bastante similar ao comportamento do espalhamento com respeito à *fitness*, evidenciando o que foi observado anteriormente de que o objetivo de cobertura tem um maior impacto na definição dos valores de *fitness*. Outro fato a ser observado, é uma maior frequência

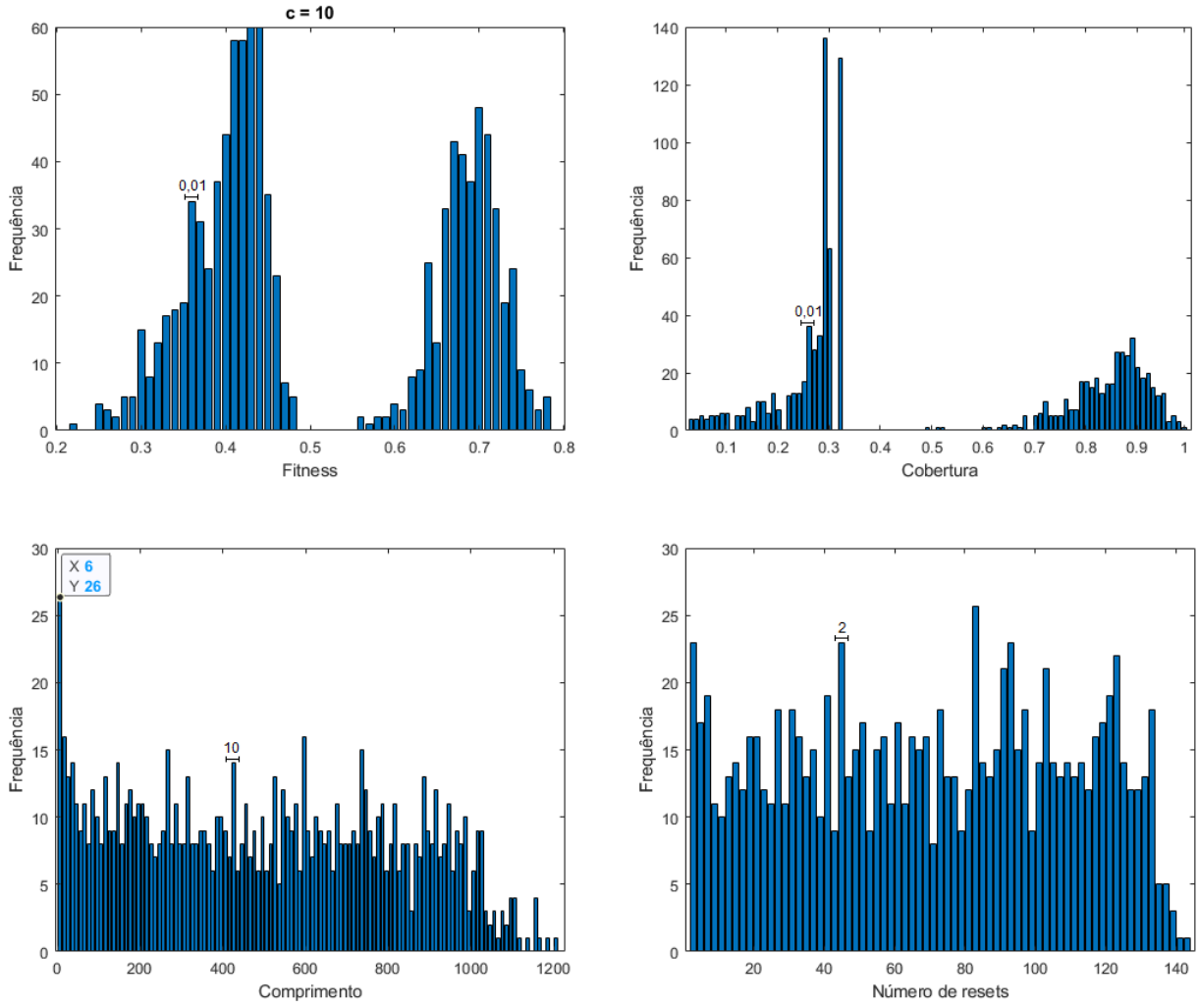


Figura 6.9: Espalhamento dos indivíduos da população inicial considerando cada componente da *fitness*

de soluções com cobertura entre 0.3 e 0.32 . Dado o cálculo do objetivo de cobertura (Algoritmo 5.8), quado uma solução não possui *transition cover* ou não há uma clique de tamanho n no grafo de distinção, o valor do objetivo é dado por $\frac{n_{tcTS}}{n_t \times 3}$ e $\frac{tamanhoCliqueMaxima}{n \times 3}$, respectivamente. Assim, soluções com essas características terão o valor de cobertura limitado à 0.32 e 0.3, respectivamente. Isso ocorre porque como o total de transições é $n \times n_i = 10 \times 3 = 30$, uma solução sem *transition cover* terá no máximo 29 transições cobertas e portanto o valor de cobertura máximo é $\frac{29}{30 \times 3} = 0.32$. Da mesma forma, uma solução que não possui clique de tamanho $n = 10$, terá uma clique de tamanho máximo 9, portanto o valor de cobertura máximo é $\frac{9}{10 \times 3} = 0.3$.

Com respeito à análise do espalhamento em relação à *fitness*, é possível notar que o método de inicialização foi capaz de incluir na população inicial soluções de alta qualidade, provendo para a busca pontos de partida em regiões promissoras. Além disso,

observando o espalhamento com respeito à cobertura, observa-se que há uma solução com cobertura 1, ou seja, completa. Apesar de ser uma solução ótima com respeito ao objetivo de cobertura, a solução apresenta alto custo (comprimento e número de *resets* igual à 979 e 126, respectivamente).

Analisando o espalhamento com respeito ao comprimento e número de *resets* é possível observar que o método de inicialização gera soluções bem espalhadas dentre as faixas de valores, indicando que o método de inicialização é capaz de gerar indivíduos bem diversos com respeito ao objetivo de comprimento e número de *resets*. Um fato a ser ressaltado é que soluções com comprimento no intervalo entre 6 e 15 apresentam uma alta frequência (26 ocorrências). Dado que um menor número de vértices possibilita um menor número de topologias possíveis, soluções com poucos vértices podem apresentar a mesma topologia ou topologias similares que implicam no mesmo valor de comprimento. Por exemplo, considerando uma MEF com 3 entradas ($X = \{a, b, c\}$), a Figura 6.10 exibe a única topologia possível para soluções que possuem 4 vértices, dado que cada indivíduo é formado pela "estrutura básica" que representa o vértice raiz e seus n_i filhos. Portanto, qualquer solução com 4 vértices sempre representará o mesmo conjunto de teste $TS = \{$a, b, c$\}$ que possui comprimento 6. Enquanto que a Figura 6.11 exibe as possíveis topologias para soluções com 5 vértices, no qual o símbolo "*" representa qualquer uma das 3 possíveis entradas. Independentemente da topologia, todas as soluções com 5 vértices irão possuir comprimento 7.

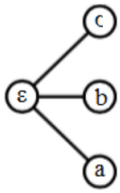
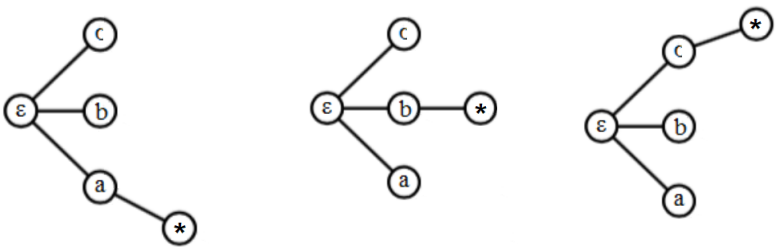


Figura 6.10: Única topologia possível de uma solução com 4 vértices



(a) (b) (c)

Figura 6.11: Possíveis topologias de soluções com 5 vértices

Ainda com respeito à analise do método de inicialização, dada a normalização do número de *resets* e do comprimento é necessário a definição de valores máximos e mínimos, porém não se tem a priori um conhecimento desses valores. Portanto, os valores são definidos com base nos indivíduos da população inicial e à medida que o algoritmo evolui esses valores são ajustados com base nos indivíduos da população corrente. Em outras palavras, em uma dada geração, se um indivíduo é incluído na população e os valores de seu comprimento e/ou *resets* são maiores que os máximos e/ou menores que os mínimos, esses valores são atualizados e é necessário atualizar o valor da *fitness* de todos os indivíduos da população corrente. Portanto, quanto menor a variação desses valores, menor o custo associado à execução do algoritmo.

Com o objetivo de avaliar a variação de tais valores foi feita uma análise dos valores mínimos e máximos utilizados na normalização ao longo das gerações. Os algoritmos EATS-MS e EATS-SC foram aplicados à máquinas com 5, 10 e 15 estados, 3 entradas e 3 saídas visando analisar o impacto da variação da complexidade da máquina na variação dos valores de normalização. Para cada máquina os algoritmos foram executados 10 vezes. Como foi observado um comportamento similar entre os resultados de ambos os algoritmos e entre os resultados da variação do número de estados, será apresentado a análise de apenas uma única execução. A Figura 6.12 exibe a variação dos valores mínimos e máximos utilizados na normalização ao longo das 100 primeiras gerações, pois não houve variação após esse momento. A referida figura apresenta os resultados de uma execução da variante EATS-SC considerando uma máquina com 15 estados.

Os valores mínimos de comprimento e *resets* definidos na população inicial não sofreram alterações, enquanto que os valores máximos sofreram poucas alterações e os valores atualizados são bem próximos dos valores definidos inicialmente. Esse fato endossa a afirmação de que o método de inicialização é capaz de gerar indivíduos bem diversos com respeito ao objetivo de comprimento e número de *resets*.

6.6.2 Análise de convergência

Com o objetivo de analisar a evolução da *fitness* ao longo das gerações, o presente experimento determina a curva de convergência dos algoritmos EATS-MS e EATS-SC. Foram consideradas máquinas com 5, 10 e 15 estados, 3 entradas e 3 saídas visando analisar o impacto da variação da complexidade da máquina na evolução das soluções. Para cada máquina os algoritmos foram executados 10 vezes.

Como foi observado um comportamento similar entre as variantes EATS-MS e EATS-SC, será apresentado o resultado de apenas uma delas e a análise vale para ambas. Além disso, o comportamento da curva de convergência foi similar entre máquinas com diferentes estados, portanto a mesma análise vale para todas as configurações. A

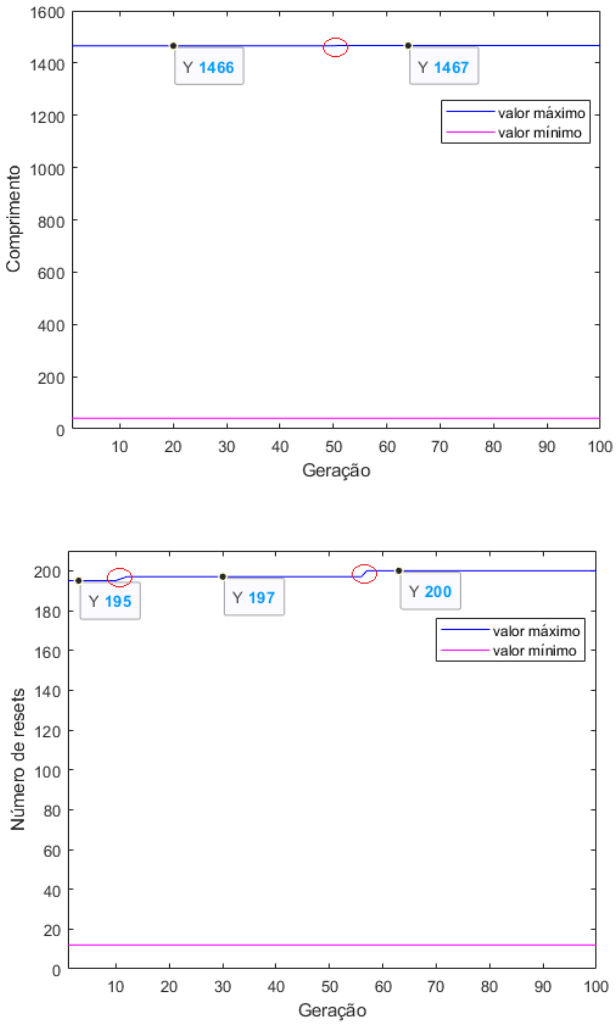


Figura 6.12: Variação dos valores máximos e mínimos utilizados na normalização

variação do número de estados impactou apenas no número de gerações necessárias para o algoritmo convergir, máquinas com menor número de estados exigem menos gerações que máquinas com maior número de estados. A Figura 6.13 exibe o gráfico de evolução da *fitness* de uma execução da variante EATS-SC considerando uma máquina com 15 estados.

É possível observar uma curva de evolução mais acentuada nas primeiras gerações, indicando que a função *fitness* tem uma boa capacidade de guiar o processo de busca para soluções de alta qualidade, bem como, os operadores de mutação tem a capacidade de gerar soluções em regiões promissoras do espaço de busca.

No momento que o algoritmo começa a convergir, nota-se que há várias gerações consecutivas sem melhora da *fitness* e em um dado momento ocorre um "salto"que indica a inclusão de um melhor indivíduo na população. O motivo desse comportamento não foi investigado, mas cabe como um trabalho futuro para identificar se é possível diminuir o número de gerações necessárias para a convergência. Suspeita-se que ao longo

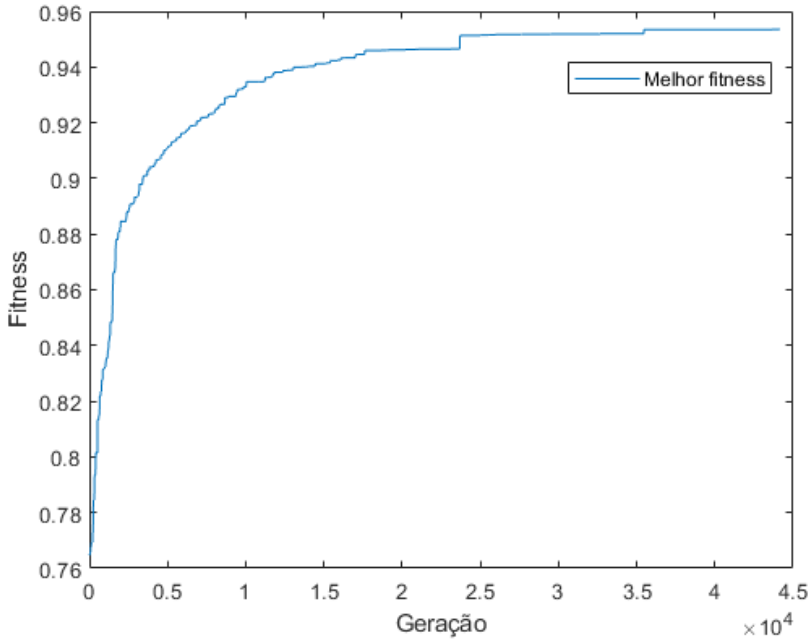


Figura 6.13: Evolução da *fitness* ao longo das gerações

da evolução a população começa a se tornar homogênea e seria interessante o uso de operadores de recombinação para intensificar a busca.

Outro aspecto que também deve ser levado em consideração é a inclusão de um critério de parada quando um conjunto completo é encontrado. O algoritmo foi capaz de encontrar o primeiro conjunto de teste completo com 276 de comprimento e 42 sequências com 23.701 gerações. Foram necessárias mais 19.913 gerações para o algoritmo convergir e alcançar um conjunto completo com 263 de comprimento e 41 sequências. Comparando com os métodos do estado da arte, o menor conjunto encontrado, gerado pelo método P, possui comprimento 331 e 49 sequências. Assim, o primeiro conjunto completo encontrado pelo algoritmo já representaria uma solução melhor do que as encontradas pelos métodos da literatura.

6.7 Análise da heurística de clique máxima em grafos k-partidos

Com o objetivo de avaliar a capacidade da heurística proposta para clique máxima em grafos k-partidos de encontrar soluções ótimas, foi realizado um experimento para verificar a quantidade de vezes que a heurística encontra a clique máxima. Caso não encontre a clique máxima, é verificado a diferença entre o tamanho da solução encontrada e o tamanho da solução ótima. A solução ótima foi determinada com base no algoritmo exato de clique máxima em grafos gerais proposto por Petersen, Eiter e Shin (2015).

Dado $k = \{5,6,7,8,9,10\}$, foram considerados 300 grafos para cada k, totalizando 1800 grafos. Os grafos foram obtidos a partir dos grafos de distinção dos conjuntos de teste gerados pelo método EATS-MS. A escolha do método EATS-MS e não do método EATS-SC, se deve ao fato observado nos experimentos de completude de que as soluções geradas por EATS-SC satisfazem as condições de suficiência de Simão e Petrenko (2010a), portanto apresentam clique de tamanho k. Com o objetivo de prover grafos nos quais a clique máxima não necessariamente tenha tamanho k foi escolhido o algoritmo EATS-MS. Foram consideradas 30 MEFs com 3 entradas, 3 saídas e variando o número de estados entre 5 e 10. Para cada MEF o método EATS-MS foi executado 10 vezes e considerada a melhor solução de cada execução. A Tabela 6.17 apresenta o resultado desse experimento. Para cada tamanho de partição, é apresentado o tamanho dos grafos considerados com respeito ao número de vértices e arestas.

Tabela 6.17: Análise da capacidade da heurística de encontrar soluções ótimas

k	**Quantidade de vértices**			**Quantidade de arestas**			**Total de clique máxima encontrada (%)**	**Distância média para o ótimo**
	Mediana	Min.	Max.	Mediana	Min.	Max.		
5	25	19	36	61	31	116	99,67	1
6	35	26	46	113	64	228	99	1
7	38	31	52	148	98	288	99	1
8	45	36	55	204	135	303	98,67	1
9	52	40	78	285	170	540	96	1
10	61	48	81	394	261	619	97.67	1

Os resultados evidenciam que a heurística se mostrou ser capaz de encontrar a clique máxima em mais de 98% dos grafos considerados e para os casos em que não foi encontrado o ótimo, a distância média do tamanho da clique encontrada para o ótimo é 1 (um). Portanto, o algoritmo soluciona o problema de encontrar clique máxima em grafos k-partidos de forma eficaz, visto que encontra soluções ótimas, na maioria dos casos, ou bem próximas do ótimo.

Com respeito à complexidade de tempo do algoritmo, foi feita uma análise empírica do tempo médio de execução considerando os grafos do experimento anterior. A Figura 6.14 exibe os resultados dessa análise.

O algoritmo apresenta um crescimento aproximadamente linear, porém mesmo para as maiores instâncias consideradas é capaz de encontrar a solução em aproximadamente 20 milissegundos. Os resultados provêm indícios que a complexidade de tempo do algoritmo é baixa. Contudo, seria interessante explorar o comportamento assintótico do algoritmo para determinar sua complexidade de tempo.

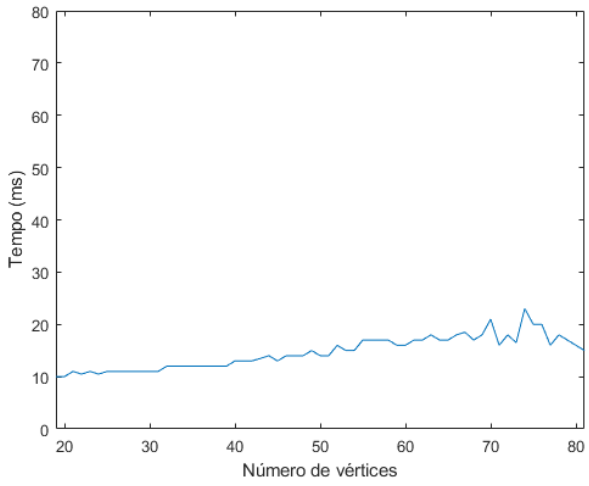


Figura 6.14: Tempo médio de execução da heurística proposta para encontrar clique máxima em grafos k-partidos

6.8 Considerações finais

Neste capítulo foram realizados experimentos com o objetivo de avaliar as soluções geradas pelo método proposto com respeito à eficiência e a eficácia, bem como avaliar o projeto da metaheurística sob uma perspectiva evolutiva. Ademais, foi feita uma análise da eficiência e eficácia da heurística proposta para encontrar clique máxima em grafos k-partidos.

Os resultados apresentados evidenciaram que o método proposto alcança a motivação inicial, a qual é gerar conjuntos de testes com baixo custo garantindo uma alta cobertura de defeitos. Ademais, a variante baseada em *score* de mutação se mostrou mais promissora em gerar conjuntos com menor custo, enquanto que a versão baseada em condições de suficiência beneficia mais o objetivo de cobertura de defeitos.

A análise evolutiva permitiu gerar indícios que o método de inicialização proposto é capaz de gerar uma população diversa e com soluções de alta qualidade. A função de *fitness* proposta se apresentou capaz de explorar o conhecimento específico do problema e guiar a busca para soluções promissoras, bem como os operadores de mutação foram capazes de gerar tais soluções.

Por fim, a heurística proposta para encontrar clique máxima em grafos k-partidos se mostrou ser capaz de encontrar soluções ótimas em quase todos os grafos considerados e em um tempo de execução inferior à 25 milissegundos.

CAPÍTULO 7

Conclusões

O uso de MEFs no contexto de teste de software como uma estratégia para automatizar a geração de casos de teste resultou no desenvolvimento de diversos métodos. Apesar de muitos destes métodos possuírem a importante característica de gerar conjuntos de testes com cobertura de defeitos garantida, muitas vezes os conjuntos gerados são muito grandes e com um elevado número de sequências. Visando tornar a atividade de teste menos onerosa, é desejável um método capaz de gerar um conjunto de teste com um custo menor. No entanto, para que a qualidade do teste não seja afetada, é necessário garantir um nível aceitável de confiabilidade.

Dito isso, esta tese investiga a geração de sequência de teste como um problema de otimização. Nesse sentido, um novo método de geração, baseado em busca, foi proposto e os estudos experimentais mostraram ganhos significativos indicando a relevância do estudo. As principais contribuições deste trabalho e atividades futuras são apresentadas nas seções 7.1 e 7.2, respectivamente.

7.1 Contribuições

Esta seção revisita as principais contribuições desta tese.

Método de geração de sequências de teste a partir de MEFs baseado em busca: o problema de gerar um conjunto de teste com custo mínimo e com alta probabilidade de revelar defeitos foi modelado como um problema de otimização multiobjetivo e o método EATS foi proposto para a solução deste problema no Capítulo 5. O método EATS é um algoritmo evolutivo, baseado em população, para geração de sequências de teste a partir de MEFs que consiste em otimizar três objetivos: maximizar a quantidade de defeitos revelados, minimizar o comprimento e minimizar a quantidade de sequências (*resets*) do conjunto gerado. O método EATS foi projetado considerando os conhecimentos específicos do problema resultando nas contribuições que seguem.

Representação Nó-Profundidade-Grau de saída: foi proposta uma nova representação para árvores na Seção 5.3.1. A NDOE foi inspirada na NDDE, que tem sua eficiência comprovada em PPRs, permitindo armazenar o grau de saída de cada nó e a repetição de rótulos dos nós.

Operadores RCS, CCS, AUN, RUN e MUN: foram propostos na Seção 5.4 cinco operadores de mutação para garantir o princípio de ergocidade. Os operadores foram projetados considerando o tamanho variável dos indivíduos e a repetição dos rótulos dos vértices, bem como a restrição de repetição de entradas entre filhos de um mesmo nó pai.

Método de inicialização: foi definido na Seção 5.5 um método para a construção dos indivíduos da população inicial que consiste em gerar indivíduos de forma aleatória fazendo o uso do operador de mutação AUN.

Avaliação da qualidade dos indivíduos: considerando a modelagem do problema como um problema de otimização multiobjetivo, foi proposto na Seção 5.8 o uso da técnica de soma ponderada para converter o problema em um problema mono-objetivo. Para o cálculo do objetivo de cobertura foram utilizadas duas técnicas: análise de mutantes e condições de suficiência.

Algoritmo para verificar cobertura de defeitos baseado nas condições de suficiência de Simão e Petrenko (2010a): tais condições foram utilizadas para a construção de abordagens de redução, as quais comprovaram que é possível encontrar conjuntos menores com a mesma eficácia do que os até então encontrados pelos métodos de geração do estado da arte. Até o momento deste trabalho, nenhum método de geração tinha sido proposto considerando tais condições. Para tanto, foi proposto um algoritmo capaz de avaliar a cobertura de defeitos e diferenciar os indivíduos com base nas referidas condições de suficiência. A avaliação atribui um valor entre 0 e 1, indicando que o conjunto não revela nenhum defeito quando o valor é 0 ou que o conjunto é completo quando o valor é 1.

Heurística para encontrar clique máxima em grafos k-partidos: para avaliar a cobertura de defeitos com base nas condições de suficiência é necessário o conhecimento da clique máxima do grafo de distinção. Tal grafo tem a propriedade de ser k-partido. Portanto, foi proposta uma heurística para encontrar clique máxima em grafos k-partidos, uma vez que na literatura só foram encontradas soluções que resolvem o problema de clique máxima em grafos gerais.

Avaliação experimental do método EATS: os experimentos consideraram duas imple-

mentações do método proposto, EATS-MS (baseada em análise de mutantes) e EATS-SC (baseada em condições de suficiência). Os resultados dos experimentos mostraram que ambas as implementações são capazes de gerar conjuntos de teste com custo menor que os métodos do estado da arte. Com respeito à cobertura de defeitos, observou-se que o método EATS-MS não é capaz de garantir a completude dos conjuntos gerados, contudo provê alta cobertura de defeitos. O método EATS-SC se mostrou capaz de garantir a completude dos conjuntos gerados, porém com um maior esforço de tempo. A análise evolutiva do método EATS gerou evidências de que a função de *fitness* é capaz de guiar a busca para soluções de alta qualidade, bem como, os operadores de mutação tem a capacidade de gerar soluções em regiões promissora do espaço de busca. Ademais, os resultados geraram evidências de que o método de inicialização é capaz de prover uma população inicial diversa e com indivíduos de alta qualidade. Por fim, a análise empírica da heurística proposta para encontrar clique máxima em grafos k-partidos mostrou que o algoritmo é capaz de encontrar, em poucos milissegundos, soluções ótimas em quase todos os casos considerados.

7.2 Trabalhos futuros

Como continuidade da pesquisa desenvolvida, propõem-se os seguintes trabalhos futuros:

MEFs parcialmente especificadas: ampliar a aplicação do método proposto para MEFs parciais, incluindo restrições nos operadores de busca de modo a garantir que apenas soluções factíveis sejam geradas. Dado um determinado nó da árvore, seus nós filhos só poderão ter como valor símbolos de entrada que representem transições presentes na MEF.

Inserir tendência no método de inicialização: considerando o fato de que um conjunto de teste para ser completo precisa cobrir todas as transições da MEF, é interessante investigar a modificação do método de inicialização proposto para permitir incluir na população inicial indivíduos que contenham um conjunto *transition cover*.

Remover estrutura básica do método de inicialização: essa estrutura foi pensada com base na hipótese de que as sequências básicas (de tamanho 1) eram necessárias para a construção das demais sequências dado que as sequências de teste representam a concatenação dos símbolos no caminho da árvore. Por exemplo, caso a raiz de um indivíduo não possuísse um filho com vértice b então nenhuma sequência de teste com prefixo b seria produzida. Contudo, com os experimentos foi possível observar que tal obrigatoriedade não é necessária, uma vez que é possível que o conjunto de teste seja completo mesmo

não possuindo sequências de teste que tenham como prefixo determinado símbolo de entrada.

Taxa variável de aplicação dos operadores: permitir que o conhecimento adquirido durante a busca defina a probabilidade de aplicação dos operadores, de modo que se a aplicação do operador gerou um indivíduo melhor sua taxa é incrementada, se for pior a taxa é decrementada.

Operadores de recombinação: desenvolver operadores de recombinação e analisar se a combinação destes operadores com os operadores de mutação propostos permite reduzir o número de gerações necessárias para convergência, aumentando o desempenho do algoritmo.

Análise assintótica da heurística de clique máxima em grafos k-partidos: o experimento empírico realizado forneceu indícios de que o algoritmo proposto é capaz de resolver o problema de clique máxima em alguns milissegundos, porém não pode-se afirmar se o algoritmo escala para entradas maiores. Portanto, é importante analisar o comportamento assintótico do algoritmo para definir o limite superior sobre o tempo de execução do algoritmo.

Comparativo da heurística proposta com a heurística de Rossi et al. (2014): a heurística de Rossi et al. (2014) se mostrou ser capaz de encontrar a clique máxima em mais da metade dos grafos considerados. Como os experimentos foram realizados com grafos gerais, seria interessante aplicar a heurística de Rossi et al. (2014) aos grafos k-partidos considerados no experimento da Seção 6.7. Assim seria possível gerar evidências de que considerar características específicas de grafos k-partidos permite que o algoritmo proposto tenha uma maior probabilidade de encontrar soluções ótimas.

Reduzir a necessidade de execução da tarefa de encontrar clique: identificar no momento da mutação quais mudanças foram realizadas no indivíduo e verificar se tais mudanças causam alterações no grafo de distinção. Caso não haja alterações, não será necessário encontrar o tamanho da clique no grafo de distinção do novo indivíduo gerado, pois será o mesmo do seu pai.

Cooperação entre os métodos EATS-MS e EATS-SC: dado que o método EATS-MS é mais promissor para gerar conjuntos com menor custo e o método EATS-SC para gerar conjuntos com maior cobertura de defeitos, a população da última geração do método EATS-MS poderia ser fornecida como população inicial do método EATS-SC.

Aplicação de algoritmos evolutivos multiobjetivo: aplicar uma técnica baseada em não-dominância para que o problema multiobjetivo não seja tratado como um problema de um único objetivo, eliminando assim a dependência com relação à escolha dos pesos de cada objetivo.

7.3 Publicações

O algoritmo EATS-MS e os resultados dos estudos empíricos referentes à esta variante foram publicados no seguinte artigo:

- RAMADA, M. S.; DE LIMA, T. W.; SIMAO, A.; SOARES, A. S. Generating Reduced Tests for FSMs using a Search-Based Testing Approach. In: 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI), Portland, OR, USA, 2019, pp. 400-407, doi: 10.1109/ICTAI.2019.00063.

Pretende-se ainda publicar:

- O algoritmo EATS-SC e os resultados dos estudos empíricos referentes à esta variante
- O algoritmo de clique máxima em grafos k-partidos e sua análise de eficiência e eficácia

Referências Bibliográficas

ABUALI, F. N.; SCHOENEFELD, D. A.; WAINWRIGHT, R. L. Designing telecommunications networks using genetic algorithms and probabilistic minimum spanning trees. In: *Proceedings of the 1994 ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 1994. (SAC '94), p. 242–246. ISBN 0-89791-647-6. Disponível em: <http://doi.acm.org/10.1145/326619.326733>.

ASAHIRO, Y.; MIYANO, E.; SAMIZO, K. Approximating maximum diameter-bounded subgraphs. In: *Proceedings of the 9th Latin American Conference on Theoretical Informatics*. Berlin, Heidelberg: Springer-Verlag, 2010. (LATIN'10), p. 615–626. ISBN 3642121993. Disponível em: <https://doi.org/10.1007/978-3-642-12200-2_53>.

ASOUDEH, N.; LABICHE, Y. Multi-objective construction of an entire adequate test suite for an efsm. In: *2014 IEEE 25th International Symposium on Software Reliability Engineering*. Toronto, Canada: IEEE, 2014. p. 288–299. ISSN 1071-9458.

BACK, T.; FOGEL, D. B.; MICHALEWICZ, Z. (Ed.). *Basic Algorithms and Operators*. 1st. ed. Bristol, UK, UK: IOP Publishing Ltd., 1999. ISBN 0750306645.

BEIZER, B. *Software testing techniques*. 2rd. ed. New York, NY, USA: Van Nostrand Reinhold Co., 1990. ISBN 0442206720.

BRANKE, J. et al. (Ed.). *Multiobjective Optimization: Interactive and Evolutionary Approaches*. Berlin, Heidelberg: Springer-Verlag, 2008. ISBN 978-3-540-88907-6.

CHOW, T. S. Testing software design modeled by finite-state machines. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 4, n. 3, p. 178–187, maio 1978. ISSN 0098-5589. Disponível em: <http://dx.doi.org/10.1109/TSE.1978.231496>.

COELLO, C. A. C.; LAMONT, G. B.; VELDHUIZEN, D. A. V. *Evolutionary Algorithms for Solving Multi-Objective Problems*. 2. ed. New York, NY, USA: Springer, 2007. ISBN 9780387332543.

COHON, J. L. Multicriteria programming: Brief review and application. In: GERO, J. S. (Ed.). *Design Optimization*. New York, NY: Academic Press, 1985. p. 163–191. ISBN 0486432637. Disponível em: <http://www.worldcat.org/isbn/0486432637>.

CUTIGI, J. F. *Uma Estratégia para Redução de Conjuntos de Sequências de Teste para Máquinas de Estados Finitos*. Dissertação (Mestrado) — ICMC/Universidade de São Paulo, São Carlos, SP, Brasil, 2010.

CUTIGI, J. F.; SIMAO, A.; SOUZA, S. R. Reducing fsm-based test suites with guaranteed fault coverage. *The Computer Journal*, v. 59, n. 8, p. 1129–1143, 2016. Disponível em: <http://dx.doi.org/10.1093/comjnl/bxv122>.

DAHL, O. J.; DIJKSTRA, E. W.; HOARE, C. A. R. (Ed.). *Structured Programming*. London, UK, UK: Academic Press Ltd., 1972. ISBN 0-12-200550-3.

DAVIS, L. et al. A genetic algorithm for survivable network design. In: FORREST, S. (Ed.). *Proceedings of the 5th International Conference on Genetic Algorithms*. San Mateo, CA, USA: Morgan Kaufmann, 1993. p. 408–415.

DEB, K.; KALYANMOY, D. *Multi-Objective Optimization Using Evolutionary Algorithms*. New York, NY, USA: John Wiley & Sons, Inc., 2001. ISBN 047187339X.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. *Introdução ao teste de software*. 2. ed. Rio de Janeiro, RJ: Elsevier, 2016. ISBN 9788535283525.

DELBEM, A. C. B.; CARVALHO, A. de. A forest representation for evolutionary algorithms applied to network design. In: CANTÚ-PAZ, E. et al. (Ed.). *Genetic and Evolutionary Computation — GECCO 2003*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. p. 634–635. ISBN 978-3-540-45105-1.

DELBEM, A. C. B. et al. Node-depth encoding for evolutionary algorithms applied to network design. In: DEB, K. (Ed.). *Genetic and Evolutionary Computation – GECCO 2004*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. p. 678–687. ISBN 978-3-540-24854-5.

DELBEM, A. C. B.; LIMA, T. W. de; TELLES, G. P. Efficient forest data structure for evolutionary algorithms applied to network design. *IEEE Transactions on Evolutionary Computation*, v. 16, n. 6, p. 829–846, Dec 2012. ISSN 1089-778X.

DEMILLO, R. A. et al. *Software Testing and Evaluation*. USA: Benjamin-Cummings Publishing Co., Inc., 1987. ISBN 0805325352.

DERDERIAN, K. et al. Generating feasible input sequences for extended finite state machines (efsms) using genetic algorithms. In: *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: Association for Computing Machinery, 2005. (GECCO '05), p. 1081–1082. ISBN 1595930108. Disponível em: <https://doi.org/10.1145/1068009.1068192>.

DERDERIAN, K. et al. Automated unique input output sequence generation for conformance testing of fsms. *Comput. J.*, Oxford University Press, Oxford, UK, v. 49, n. 3, p. 331–344, maio 2006. ISSN 0010-4620. Disponível em: <http://dx.doi.org/10.1093/comjnl/bxl003>.

DOROFEEVA, R. et al. Fsm-based conformance testing methods: A survey annotated with experimental evaluation. *Inf. Softw. Technol.*, Butterworth-Heinemann, Newton, MA, USA, v. 52, n. 12, p. 1286–1297, dez. 2010. ISSN 0950-5849. Disponível em: <http://dx.doi.org/10.1016/j.infsof.2010.07.001>.

DOROFEEVA, R.; EL-FAKIH, K.; YEVTUSHENKO, N. An improved conformance testing method. In: WANG, F. (Ed.). *Formal Techniques for Networked and Distributed Systems - FORTE 2005*. Berlin, Heidelberg: Springer, 2005. p. 204–218. ISBN 978-3-540-32084-5.

ENDO, A. T.; SIMAO, A. Experimental comparison of test case generation methods for finite state machines. In: *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*. [S.l.: s.n.], 2012. p. 549–558.

ENDO, A. T.; SIMAO, A. Evaluating test suite characteristics, cost, and effectiveness of fsm-based testing methods. *Inf. Softw. Technol.*, Butterworth-Heinemann, Newton, MA, USA, v. 55, n. 6, p. 1045–1062, jun. 2013. ISSN 0950-5849. Disponível em: <http://dx.doi.org/10.1016/j.infsof.2013.01.001>.

FONSECA, C. M.; FLEMING, P. J. Genetic algorithms for multiobjective optimization: Formulationdiscussion and generalization. In: *Proceedings of the 5th International Conference on Genetic Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. p. 416–423. ISBN 1-55860-299-2. Disponível em: <http://dl.acm.org/citation.cfm?id=645513.657757>.

FUJIWARA, S. et al. Test selection based on finite state models. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 17, n. 6, p. 591–603, jun. 1991. ISSN 0098-5589. Disponível em: <http://dx.doi.org/10.1109/32.87284>.

GAREY, M. R.; JOHNSON, D. S. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990. ISBN 0716710455.

GILL, A. *Introduction to the Theory of Finite-State Machines*. New York, NY, USA: McGraw-Hill, 1962. ISBN 0070232431.

GLOVER, F. Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.*, Elsevier Science Ltd., Oxford, UK, UK, v. 13, n. 5, p. 533–549, maio 1986. ISSN 0305-0548. Disponível em: <http://dx.doi.org/10.1016/0305-0548(86)90048-1>.

GONENC, G. A method for the design of fault detection experiments. *IEEE Transactions on Computers*, C-19, n. 6, p. 551–558, June 1970. ISSN 0018-9340.

GRUNERT, T. et al. Finding all k-cliques in k-partite graphs, an application in textile engineering. *Computers & Operations Research*, v. 29, n. 1, p. 13 – 31, 2002. ISSN 0305-0548. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0305054800000538>.

GUO, Q. et al. Computing unique input/output sequences using genetic algorithms. In: PETRENKO, A.; ULRICH, A. (Ed.). *Formal Approaches to Software Testing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. p. 164–177. ISBN 978-3-540-24617-6.

GUO, Q. et al. Constructing multiple unique input/output sequences using metaheuristic optimisation techniques. *IEE Proceedings - Software*, v. 152, n. 3, p. 127–140, June 2005. ISSN 1462-5970.

HAREL, D. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, Elsevier North-Holland, Inc., Amsterdam, The Netherlands, The Netherlands, v. 8, n. 3, p. 231–274, jun. 1987. ISSN 0167-6423. Disponível em: <http://dx.doi.org/10.1016/0167-6423(87)90035-9>.

HARMAN, M.; JONES, B. F. Search-based software engineering. *Information and Software Technology*, v. 43, n. 14, p. 833 – 839, 2001. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584901001896>.

HENNINE, F. C. Fault detecting experiments for sequential circuits. In: *Proceedings of the 1964 Proceedings of the Fifth Annual Symposium on Switching Circuit Theory and Logical Design*. Washington, DC, USA: IEEE Computer Society, 1964. (SWCT '64), p. 95–110. Disponível em: <http://dx.doi.org/10.1109/SWCT.1964.8>.

HIERONS, R. M.; URAL, H. Reduced length checking sequences. *IEEE Transactions on Computers*, v. 51, n. 9, p. 1111–1117, Sep 2002. ISSN 0018-9340.

HIERONS, R. M.; URAL, H. Optimizing the length of checking sequences. *IEEE Transactions on Computers*, v. 55, n. 5, p. 618–629, May 2006. ISSN 0018-9340.

HOGREFE, D. *OSI formal specification case study: the Inres protocol and service*. University of Bern, 1991. (Technical Report, 91-012).

HOLLAND, J. H. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press, 1975.

IEEE. *IEEE standard glossary of software engineering terminology*. Standard 610.12, Institute of Electric and Electronic Engineers, 1990.

ISHIBUCHI, H.; TSUKAMOTO, N.; NOJIMA, Y. Evolutionary many-objective optimization: A short review. In: *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. Hong Kong, China: IEEE, 2008. p. 2419–2426.

JONG, K. D. *Evolutionary Computation: A Unified Approach*. Cambridge, Massachusetts: MIT Press, 2006. ISBN 9780262041942. Disponível em: <https://books.google.com.br/books?id=OIRQAAAAMAAJ>.

JOURDAN, G.-V.; URAL, H.; YENIGüN, H. Reducing locating sequences for testing from finite state machines. In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2016. (SAC '16), p. 1654–1659. ISBN 978-1-4503-3739-7. Disponível em: <http://doi.acm.org/10.1145/2851613.2851831>.

KALAJI, A. S.; HIERONS, R. M.; SWIFT, S. Generating feasible transition paths for testing from an extended finite state machine (efsm). In: *Proceedings of the 2009 International Conference on Software Testing Verification and Validation*. USA: IEEE Computer Society, 2009. (ICST '09), p. 230–239. ISBN 9780769536019. Disponível em: <https://doi.org/10.1109/ICST.2009.29>.

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. *Science*, v. 220 4598, p. 671–80, 1983.

KONAK, A.; COIT, D. W.; SMITH, A. E. Multi-objective optimization using genetic algorithms: A tutorial. *Rel. Eng. & Sys. Safety*, v. 91, p. 992–1007, 2006.

LEE, D.; YANNAKAKIS, M. Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE*, v. 84, n. 8, p. 1090–1123, Aug 1996. ISSN 0018-9219.

LEFTICARU, R.; IPATE, F. Automatic state-based test generation using genetic algorithms. In: *Proceedings of the Ninth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. USA: IEEE Computer Society, 2007. (SYNASC '07), p. 188–195. ISBN 0769530788. Disponível em: <https://doi.org/10.1109/SYNASC.2007.47>.

LEHRE, P. K.; YAO, X. Runtime analysis of the (1+1) ea on computing unique input output sequences. *Inf. Sci.*, Elsevier Science Inc., New York, NY, USA, v. 259, p. 510–531, fev. 2014. ISSN 0020-0255. Disponível em: <https://doi.org/10.1016/j.ins.2010.01.031>.

LIMA, T. W. de. *Estruturas de Dados Eficientes para Algoritmos Evolutivos Aplicados a Projeto de Redes*. Tese (Doutorado) — ICMC/Universidade de São Paulo, São Carlos, SP, Brasil, 2009.

LU, G.; MIAO, H. An approach to generating test data for efsm paths considering condition coverage. *Electronic Notes in Theoretical Computer Science*, v. 309, p. 13 – 29, 2014. ISSN 1571-0661. Proceedings of the Sixth International Workshop on Harnessing Theories for Tool Support for Software (TTSS). Disponível em: <http://www.sciencedirect.com/science/article/pii/S1571066114000875>.

LUKE, S. *Essentials of Metaheuristics*. 2. ed. Lulu, 2013. Disponível em: <http://cs.gmu.edu/$\sim$sean/book/metaheuristics/>.

MALOD-DOGNIN, N.; ANDONOV, R.; YANEV, N. Maximum cliques in protein structure comparison. In: FESTA, P. (Ed.). *Experimental Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 106–117. ISBN 978-3-642-13193-6.

MEALY, G. H. A method for synthesizing sequential circuits. *The Bell System Technical Journal*, v. 34, n. 5, p. 1045–1079, Sept 1955. ISSN 0005-8580.

MOORE, E. F. Gedanken-experiments on sequential machines. *Annals of Mathematics studies*, v. 23, p. 129–153, 1956.

MYERS, G. J.; SANDLER, C.; BADGETT, T. *The Art of Software Testing*. 3rd. ed. New York, NY, USA: John Wiley & Sons, Inc., 2011. ISBN 1118031962.

PETERSEN, J.; EITER, E.; SHIN, J. *Clique Solver*. Oregon, USA: GitHub, 2015. <https://github.com/jaredpetersen/clique>.

PETERSON, J. L. Petri nets. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 9, n. 3, p. 223–252, set. 1977. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/356698.356702>.

PETRENKO, A.; BOCHMANN, G.; YAO, M. On fault coverage of tests for finite state specifications. *Computer Networks and ISDN Systems*, v. 29, n. 1, p. 81 – 106, 1996. ISSN 0169-7552. Protocol Testing. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0169755296000190>.

PETRENKO, A.; BOCHMANN, G. v. Selecting test sequences for partially-specified nondeterministic finite state machines. In: LUO, G. (Ed.). *7th IFIP WG 6.1 International Workshop on Protocol Test Systems*. London, UK, UK: Chapman & Hall, Ltd., 1995. (IWPTS '94), p. 95–110. ISBN 0-412-71160-5. Disponível em: <http://dl.acm.org/citation.cfm?id=236187.233118>.

PETRENKO, A.; BORODAY, S.; GROZ, R. Confirming configurations in efsm testing. *IEEE Transactions on Software Engineering*, v. 30, n. 1, p. 29–42, Jan 2004. ISSN 0098-5589.

PETRENKO, A.; YEVTUSHENKO, N. Testing from partial deterministic fsm specifications. *IEEE Transactions on Computers*, v. 54, n. 9, p. 1154–1165, Sept 2005. ISSN 0018-9340.

PEZZÈ, M.; YOUNG, M. *Teste e Análise de Software: Processos, Princípios e Técnicas*. Porto Alegre, RS: Bookman, 2008. ISBN 9788577802623.

PFLEEGER, S. L. *Engenharia de Software: Teoria e Prática*. 2. ed. São Paulo, SP: Pearson Prentice Hall, 2004. ISBN 8587918311.

POAGE, J. F.; MCCLUSKEY, E. J. Derivation of optimum test sequencies for sequential machines. In: *1964 Proceedings of the Fifth Annual Symposium on Switching Circuit Theory and Logical Design*. Princeton, NJ, USA: IEEE, 1964. p. 121–132.

PRESSMAN, R. *Software Engineering: A Practitioner's Approach*. 7. ed. USA: McGraw-Hill, Inc., 2009. ISBN 0073375977.

PRESSMAN, R. S. *Engenharia de Software*. 3. ed. São Paulo, SP: Makron Books, 2006. ISBN 8534602379.

R Core Team. *R: A Language and Environment for Statistical Computing*. Vienna, Austria, 2013. Disponível em: <http://www.R-project.org/>.

RADCLIFFE, N. J. Equivalence class analysis of genetic algorithms. *Complex Systems*, v. 5, p. 183–205, 1991.

RADCLIFFE, N. J. Forma analysis and random respectful recombination. In: *Proceedings of the Fourth International Conference on Genetic Algorithms*. San Mateo: Morgan Kaufmann, 1991. p. 222–229.

RADCLIFFE, N. J. Genetic set recombination. In: WHITLEY, L. D. (Ed.). *Foundations of Genetic Algorithms*. Elsevier, 1992, (Foundations of Genetic Algorithms, v. 2). p. 203 – 219. Disponível em: <http://www.sciencedirect.com/science/article/pii/B9780080948324500192>.

RADCLIFFE, N. J. Non-linear genetic representations. In: MÄNNER, R.; MANDERICK, B. (Ed.). *Parallel Problem Solving from Nature 2*. North Holland, Amsterdam: Elsevier Science Publishers, 1992. p. 259 – 268.

RADCLIFFE, N. J. Genetic set recombination and its application to neural network topology optimisation. *Neural Computing & Applications*, v. 1, n. 1, p. 67–90, Mar 1993. ISSN 1433-3058. Disponível em: <https://doi.org/10.1007/BF01411376>.

RADCLIFFE, N. J. The algebra of genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, v. 10, n. 4, p. 339–384, Dec 1994. ISSN 1573-7470. Disponível em: <https://doi.org/10.1007/BF01531276>.

RADCLIFFE, N. J.; SURRY, P. D. Fitness variance of formae and performance prediction. In: WHITLEY, L. D.; VOSE, M. D. (Ed.). Elsevier, 1994, (Foundations of Genetic Algorithms, v. 3). p. 51 – 72. Disponível em: <http://www.sciencedirect.com/science/article/pii/B9781558603561500078>.

RAIDL, G. R.; JULSTROM, B. A. Edge sets: an effective evolutionary coding of spanning trees. *IEEE Transactions on Evolutionary Computation*, v. 7, n. 3, p. 225–239, June 2003. ISSN 1089-778X.

RAMAMOORTHY, C. V.; BASTANI, F. B. Software reliability status and perspectives. *IEEE Trans. Softw. Eng.*, IEEE Press, v. 8, n. 4, p. 354–371, jul. 1982. ISSN 0098-5589. Disponível em: <https://doi.org/10.1109/TSE.1982.235728>.

ROCHA, A. da; MALDONADO, J.; WEBER, K. *Qualidade de software: teoria e prática*. São Paulo, Brasil: Prentice Hall, 2001. ISBN 9788587918543.

ROSSI, R. A. et al. Fast maximum clique algorithms for large graphs. In: *Proceedings of the 23rd International Conference on World Wide Web*. New York, NY, USA: Association for Computing Machinery, 2014. (WWW '14 Companion), p. 365–366. ISBN 9781450327459. Disponível em: <https://doi.org/10.1145/2567948.2577283>.

ROTHLAUF, F. *Design of Modern Heuristics: Principles and Application*. 1st. ed. Berlin, Heidelberg: Springer Publishing Company, Incorporated, 2011. ISBN 3540729615, 9783540729617.

ROTHLAUF, F.; GOLDBERG, D. E.; HEINZL, A. Network random keys: A tree representation scheme for genetic and evolutionary algorithms. *Evol. Comput.*, MIT Press, Cambridge, MA, USA, v. 10, n. 1, p. 75–97, mar. 2002. ISSN 1063-6560. Disponível em: <http://dx.doi.org/10.1162/106365602317301781>.

SABNANI, K.; DAHBURA, A. A protocol test generation procedure. *Comput. Netw. ISDN Syst.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 15, n. 4, p. 285–297, set. 1988. ISSN 0169-7552. Disponível em: <http://dx.doi.org/10.1016/0169-7552(88)90064-5>.

SALAM, A.; HIERONS, R.; SWIFT, S. Automatic generation of test sequences from efsm models using evolutionary algorithms. 2008.

SHU, T. et al. A heuristic transition executability analysis method for generating efsm-specified protocol test sequences. *Inf. Sci.*, Elsevier Science Inc., New York, NY, USA, v. 370, n. C, p. 63–78, nov. 2016. ISSN 0020-0255. Disponível em: <https://doi.org/10.1016/j.ins.2016.07.059>.

SIMÃO, A.; PETRENKO, A. Generating checking sequences for partial reduced finite state machines. In: *Proceedings of the 20th IFIP TC 6/WG 6.1 International Conference on Testing of Software and Communicating Systems: 8th International Workshop*. Berlin, Heidelberg: Springer-Verlag, 2008. (TestCom '08 / FATES '08), p. 153–168. ISBN 978-3-540-68514-2. Disponível em: <http://dx.doi.org/10.1007/978-3-540-68524-1_12>.

SIMÃO, A.; PETRENKO, A. Checking completeness of tests for finite state machines. *IEEE Transactions on Computers*, v. 59, n. 8, p. 1023–1032, Aug 2010. ISSN 0018-9340.

SIMÃO, A.; PETRENKO, A. Fault coverage-driven incremental test generation. *Comput. J.*, Oxford University Press, Oxford, UK, v. 53, n. 9, p. 1508–1522, nov. 2010. ISSN 0010-4620. Disponível em: <http://dx.doi.org/10.1093/comjnl/bxp073>.

SIMÃO, A.; PETRENKO, A.; YEVTUSHENKO, N. Generating reduced tests for fsms with extra states. In: NÚÑEZ, M.; BAKER, P.; MERAYO, M. G. (Ed.). *Testing of Software and Communication Systems*. Berlin, Heidelberg: Springer, 2009. p. 129–145. ISBN 978-3-642-05031-2.

SOMMERVILLE, I. *Engenharia de Software*. 9. ed. São Paulo, SP: Pearson Prentice Hall, 2011. ISBN 97885793610811.

SOUCHA, M.; BOGDANOV, K. Spyh-method: an improvement in testing of finite-state machines. *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, IEEE, Västerås, Sweden, p. 194–203, 2018. ISSN 978-1-5386-6352-3. Disponível em: <https://doi.org/10.1109/ICSTW.2018.00050>.

TAN, Q. M.; PETRENKO, A.; VON BOCHMANN, G. A test generation tool for specifications in the form of state machines. In: *Proceedings of ICC/SUPERCOMM '96 - International Conference on Communications*. Dallas,TX: IEEE, 1996. v. 1, p. 225–229.

TURNER, K. J. *Using Formal Description Techniques: An Introduction to Estelle, Lotos, and SDL*. 1st. ed. New York, NY, USA: John Wiley & Sons, Inc., 1993. ISBN 0471934550.

URAL, H.; WU, X.; ZHANG, F. On minimizing the lengths of checking sequences. *IEEE Transactions on Computers*, v. 46, n. 1, p. 93–99, Jan 1997. ISSN 0018-9340.

URAL, H.; ZHANG, F. Reducing the lengths of checking sequences by overlapping. In: UYAR, M. Ü.; DUALE, A. Y.; FECKO, M. A. (Ed.). *Testing of Communicating Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 274–288. ISBN 978-3-540-34185-7.

UTTING, M.; LEGEARD, B. *Practical Model-Based Testing: A Tools Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006. ISBN 0123725011.

WOLPERT, D. H.; MACREADY, W. G. *No Free Lunch Theorems for Search*. Santa Fe, NM, 1995. Tech. Rep. No. SFI-TR-95-02-010.

WOLPERT, D. H.; MACREADY, W. G. No free lunch theorems for optimization. *Trans. Evol. Comp*, IEEE Press, Piscataway, NJ, USA, v. 1, n. 1, p. 67–82, abr. 1997. ISSN 1089-778X. Disponível em: <http://dx.doi.org/10.1109/4235.585893>.

YANG, R. et al. Efsm-based test case generation: Sequence, data, and oracle. *International Journal of Software Engineering and Knowledge Engineering*, v. 25, n. 4, p. 633–668, 2015.

YANO, T.; MARTINS, E.; SOUSA, F. L. de. Most: A multi-objective search-based testing from efsm. In: *Proceedings of the 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*. USA: IEEE Computer Society, 2011. (ICSTW '11), p. 164–173. ISBN 9780769543451. Disponível em: <https://doi.org/10.1109/ICSTW.2011.37>.

YAO, M.; PETRENKO, A.; BOCHMANN, G. v. Fault coverage analysis in respect to an fsm specification. In: *INFOCOM '94. Networking for Global Communications., 13th Proceedings IEEE*. Toronto, Canada: IEEE, 1994. v. 2, p. 768–775.

YU, X.; GEN, M. *Introduction to Evolutionary Algorithms*. United Kingdom, London: Springer-Verlag London, 2012. ISBN 144712569X, 9781447125693.

ZHOU, G.; GEN, M. A note on genetic algorithms for degree-constrained spanning tree problem. *Networks*, v. 30, p. 91–97, 1997.

ZHOU, X.; ZHAO, R.; YOU, F. Efsm-based test data generation with multi-population genetic algorithm. In: *2014 IEEE 5th International Conference on Software Engineering and Service Science*. Toronto, Canada: IEEE, 2014. p. 925–928. ISSN 2327-0594.

ZITZLER, E. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. Tese (Doutorado) — Swiss Federal Institute of Technology Zurich, 1999.