Uma Abordagem Evolutiva Multiobjetivo Para Geração Automática de Casos de Teste a Partir de Máquinas de Estados

Este exemplar corresponde à redação final da Tese devidamente corrigida e defendida por Thaise Yano e aprovada pela Banca Examinadora.

Campinas, 15 de setembro de 2011.

Eliane Martins

Eliane Martins (Orientadora)

Tese apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Doutora em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA POR
MARIA FABIANA BEZERRA MÜLLER - CRB8/6162
BIBLIOTECA DO INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E
COMPUTAÇÃO CIENTÍFICA - UNICAMP

Yano, Thaise, 1979-
Y17a Uma abordagem evolutiva multiobjetivo para geração automática de casos de teste a partir de máquinas de estados / Thaise Yano. – Campinas, SP: [s.n.], 2011.

Orientador: Eliane Martins.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Engenharia de software. 2. Software - Testes. 3. Computação evolutiva. I. Martins, Eliane, 1955-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em inglês: A multi-objective evolutionary approach for automatic generation of test cases from state machines
Palavras-chave em Inglês:
Software engineering
Software - Testing
Evolutionary computation
Área de concentração: Ciência da Computação
Titulação: Doutor em Ciência da Computação
Banca examinadora:
Eliane Martins [Orientador]
Silvia Regina Vergílio
Maria de Fátima Mattiello-Francisco
Mario Jino
Eduardo Cândido Xavier
Data da defesa: 26-07-2011
Programa de Pós Graduação: Ciência da Computação

TERMO DE APROVAÇÃO

Tese Defendida e Aprovada em 26 de julho de 2011, pela Banca examinadora composta pelos Professores Doutores:

Profª. Drª. Silvia Regina Vergílio - membro
Departamento de Informática / UFPR

Dra. Maria de Fátima Mattiello-Francisco - membro
INPE

Prof. Dr. Mario Jino - membro
FEEC / UNICAMP

Prof. Dr. Eduardo Cândido Xavier - membro
IC / UNICAMP

Profª. Drª. Eliane Martins - orientadora
IC / UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Uma Abordagem Evolutiva Multiobjetivo Para Geração Automática de Casos de Teste a Partir de Máquinas de Estados

Thaise Yano[1]

Julho de 2011

Banca Examinadora:

- Eliane Martins (Orientadora)
- Eduardo Candido Xavier
  Universidade Estadual de Campinas
- Maria de Fátima Mattiello-Francisco
  Instituto Nacional de Pesquisas Espaciais
- Mario Jino
  Universidade Estadual de Campinas
- Silvia Regina Vergilio
  Universidade Federal do Paraná

[1]Suporte financeiro de: CAPES, CNPq e Serasa Experian.

Resumo

A geração automática de casos de teste contribui tanto para melhorar a produtividade quanto para reduzir esforço e custo no processo de desenvolvimento de software. Neste trabalho é proposta uma abordagem, denominada MOST (*Multi-Objective Search-based Testing approach from EFSM*), para gerar casos de teste a partir de Máquina de Estados Finitos Estendida (MEFE) com a aplicação de uma técnica de otimização. No teste baseado em MEFE, é necessário encontrar uma sequência de entrada para exercitar um caminho no modelo, a fim de cobrir um critério de teste (e.g. todas as transições). Como as sequências podem ter diferentes tamanhos, motivou-se o desenvolvimento do algoritmo M-GEO$_{vsl}$ (*Multi-Objective Generalized Extremal Optimization with variable string length*) que permite gerar soluções de diferentes tamanhos. Além disso, por ser um algoritmo multiobjetivo, M-GEO$_{vsl}$ também possibilita que mais de um critério seja usado para avaliar as soluções. Com a aplicação desse algoritmo em MOST, tanto a cobertura da transição alvo quanto o tamanho da sequência são levados em consideração na geração de casos de teste. Para guiar a busca, são utilizadas informações das dependências do modelo. O algoritmo gera as sequências de entrada, incluindo os valores de seus parâmetros. Em MOST, um modelo executável da MEFE recebe como entrada os dados gerados pelo M-GEO$_{vsl}$ e produz dinamicamente os caminhos percorridos. Uma vez que os aspectos de controle e dados do modelo são considerados durante a execução do modelo, evita-se o problema de geração de caminhos infactíveis. Um caminho pode ser sintaticamente possível, mas semanticamente infactível, devido aos conflitos de dados envolvidos no modelo. Para avaliar a abordagem proposta foram realizados vários experimentos com modelos da literatura e de aplicações reais. Os resultados da abordagem também foram comparados com os casos de teste obtidos em um trabalho relacionado.

Abstract

Automated test case generation can improve the productivity as well as reduce effort and cost in the software development process. In this work an approach, named MOST (Multi-Objective Search-based Testing approach from EFSM), is proposed to generate test cases from Extended Finite State Machine (EFSM) using an optimization technique. In EFSM based testing, an input sequence should be found to sensitize a path in the model, in order to cover a test criterion (e.g. all transitions). As the sequences can have different lengths, it motivates the development of the M-GEO$_{vsl}$ (Multi-Objective Generalized Extremal Optimization with variable string length) algorithm that makes possible the generation of solutions with different lengths. Moreover, as a multiobjective algorithm, M-GEO$_{vsl}$ also allows to use more than one criterion to evaluate the solutions. Using this algorithm in MOST, the coverage of the target transition as well as the sequence length are taken into account in the test case generation. To guide the search, the information about the model dependences is used. The algorithm generates the input sequences, including the values of their parameters. In MOST, an executable model of the EFSM receives as input the data generated by M-GEO$_{vsl}$ and produces the traversed paths dynamically. Since the control and data aspects are considered during model execution, the problem of infeasible path generation is avoided. A path can be syntatically possible, but semantically infeasible, due to the data conflicts in the model. In order to evaluate the proposed approach, experiments were performed with models of the literature and real-world applications. The results were also compared to the test cases obtained in a related work.

Agradecimentos

A Deus pela vida e pela fé.

Aos meus pais, irmãos e cunhados pelo carinho e incentivo.

Ao meu marido, Luciano, pelo amor, força e compreensão.

À Eliane Martins pela orientação e confiança.

Ao Fabiano Luis de Sousa pela orientação e cooperação.

Ao CNPq, CAPEs e Serasa Experian pelo apoio financeiro.

Ao IC pelo apoio institucional.

Sumário

Resumo **vii**

Abstract **ix**

Agradecimentos **xi**

1 Introdução **1**
1.1 Motivação 2
1.2 Objetivos 5
1.3 Organização do Trabalho 6

2 Teste Baseado em Modelo **9**
2.1 MEFE 10
2.2 Análise de dependência de MEFE 12
2.3 Processo de Teste 13
2.4 Trabalhos Relacionados 16
2.5 Considerações Finais 19

3 Teste Baseado em Busca **21**
3.1 Principais Aspectos 22
3.2 Algoritmos Evolutivos 23
3.3 Otimização Multiobjetivo 24
3.4 Trabalhos Relacionados 27
3.5 Considerações Finais 34

4 Algoritmo Evolutivo Multiobjetivo Proposto **35**
4.1 Origem do GEO 35
4.2 M-GEO$_{vsl}$: Algoritmo 37
4.2.1 Codificação da Solução 37
4.2.2 Operador de Mutação 39

4.2.3 Processo Evolutivo 39
4.3 Considerações Finais 43

5 Abordagem de Teste Proposta **45**
5.1 Modelo Executável 46
5.2 Dependências do modelo 47
5.3 Gerador de Casos de Teste 48
5.3.1 Codificação da Solução 49
5.3.2 Funções Objetivo 51
5.4 Considerações Finais 54

6 Experimentos **55**
6.1 Estudos de Caso 56
6.2 Configuração dos Experimentos 58
6.3 Métricas 60
6.4 Resultados 61
6.4.1 Geração de Casos de Teste 61
6.4.2 Comparação com Trabalho Relacionado 69
6.5 Considerações Finais 72

7 Conclusões **75**
7.1 Contribuições 76
7.2 Trabalhos Futuros 78

A Publicações **81**

Bibliografia **87**

Lista de Tabelas

2.1 Trabalhos para geração de casos de teste baseadas em MEFE. 19

3.1 Trabalhos relacionados em SBST: teste a partir de especificações. 31
3.2 Trabalhos relacionados em SBST: teste de programas orientados a objeto. . 32
3.3 Trabalhos relacionados em SBST: teste com otimização multiobjetivo. . . . 33

4.1 Exemplo numérico dos passos do M-GEO$_{vsl}$. 43

5.1 Funções de Distância. 54

6.1 Modelos utilizados nos experimentos. 56
6.2 M_1: Conjunto de Pareto para t_7. 62
6.3 M_1: legenda dos eventos de entrada. 62
6.4 Cobertura da geração dos casos de teste. 64
6.5 Médias e desvios padrões (σ) dos resultados da geração de casos de teste: número de soluções, número de avaliações, tamanho dos caminhos, tamanho das sequências e tempo de execução (segundos). 65
6.6 M_{11}: Número de soluções da fronteira de Pareto obtida pelo M-GEO$_{vsl}$ (pf) e porcentagem de soluções não-dominadas por s_k (nd). 71
6.7 M_{12}: Número de soluções da fronteira de Pareto obtida pelo M-GEO$_{vsl}$ (pf) e porcentagem de soluções não-dominadas por s_k (nd). 71
6.8 M_{13}: Número de soluções da fronteira de Pareto obtida pelo M-GEO$_{vsl}$ (pf) e porcentagem de soluções não-dominadas por s_k (nd). 72

Lista de Figuras

2.1 M_1: Sistema de caixa eletrônico representado em uma MEFE [96]. 11
2.2 Dependência de controle (NTSCD) e dados (DD) para a MEFE M_1. 13
2.3 Grafo de dependência. 13
2.4 Processo de teste baseado em modelo. 15
2.5 Exemplo de caminho infactível em uma MEFE: conflito de dados na ação de t_1 com a guarda de t_3. 16

3.1 Ilustração de um problema de otimização multiobjetivo [134]. 25
3.2 Fronteira de Pareto. 26

4.1 População de espécies no modelo de Bak e Sneppen. 36
4.2 População: GEO (4 espécies e 2 variáveis de projeto) × algoritmo genético (n indivíduos com 2 variáveis de projeto). 38
4.3 População: M-GEO × M-GEO$_{vsl}$. 38
4.4 Mutações no M-GEO$_{vsl}$. 39
4.5 Processo Evolutivo: M-GEO × M-GEO$_{vsl}$. 41

5.1 Abordagem MOST. 46
5.2 MEFE M_1: T_a (linha em negrito) e T_c (linha tracejada), considerando-se t_{18} como transição alvo. 48
5.3 Processo evolutivo do M-GEO$_{vsl}$ para a abordagem MOST. 49
5.4 População no M-GEO$_{vsl}$ para geração de teste baseado em MEFE. 51
5.5 Renomeação de parâmetros. 51

6.1 M_8 : Influência de τ em M-GEO$_{vsl}$. 59
6.2 M_1: Fronteira de Pareto para t_7. 62
6.3 M_1 a M_6: Número de soluções no conjunto de Pareto. 66
6.4 M_7 a M_{11}: Número de soluções no conjunto de Pareto. 67
6.5 M_{12} a M_{15}: Número de soluções no conjunto de Pareto. 68
6.6 M_{15}: Número de soluções no conjunto de Pareto. 69
6.7 M_{11}: Fronteira de Pareto. 70

Capítulo 1

Introdução

O teste de software é uma atividade crucial no processo de desenvolvimento de software, devido ao crescimento da complexidade do software e ao uso de sistemas computacionais ser cada vez maior. A atividade de teste consiste na análise dinâmica do produto de software, conduzida de maneira sistemática e criteriosa com o objetivo de detectar a presença de falhas [117]. Segundo o padrão IEEE 610.12-1990 [88], uma falha (em inglês, *fault*) quando ativada gera um erro (em inglês, *error*), o qual pode se propagar e levar a um defeito (em inglês, *failure*). Um teste é bem sucedido quando detecta falhas ao exercitar o software com os casos de teste. Assim, um ponto importante que deve ser considerado no teste de software é o projeto e/ou avaliação da qualidade de um determinado conjunto de casos de teste a fim de que esse tenha alta probabilidade de revelar a presença de falhas com um mínimo de tempo e esforço.

Como em geral é impraticável utilizar todo o domínio de dados de entrada para avaliar os aspectos funcionais e operacionais de um software, critérios de teste são utilizados tanto para auxiliar na geração de conjuntos de casos de teste quanto para auxiliar na avaliação da adequação desses conjuntos. Os critérios estabelecem os requisitos que devem ser testados e permite quantificar a atividade de teste, através da análise de satisfação desses requisitos denominada análise de cobertura ou análise de adequação. Em geral, os critérios de teste de software são classificados em três técnicas: funcional, estrutural e baseada em falhas. Tais técnicas se diferenciam basicamente pela origem da informação que é utilizada para avaliar ou construir conjuntos de casos de teste. Na *técnica funcional*, ou teste de caixa preta, utiliza-se a especificação do software para obter os requisitos de testes. Na *técnica estrutural*, ou teste de caixa branca, os critérios e requisitos de teste são derivados a partir dos aspectos de implementação do software. Já na *técnica baseada em falhas*, os requisitos para caracterizar a atividade de teste são baseados em falhas típicas que podem ser introduzidas durante o desenvolvimento do software. Vale ressaltar que as técnicas de teste possuem características complementares e devem ser aplicadas em

conjunto, buscando-se utilizar as vantagens de cada uma a fim de obter uma atividade de teste de melhor qualidade. Para tanto, o estabelecimento de estratégias de teste eficazes e de baixo custo é de grande importância, constituindo uma das principais direções para a pesquisa na área de teste de software [84]. Deve-se destacar também que para apoiar essa tarefa, é relevante a condução de estudos teóricos e empíricos dos critérios de teste.

Os testes podem ser conduzidos ao longo do processo de desenvolvimento de software em diferentes fases: teste de unidade, teste de integração e teste de sistema. O *teste de unidade* concentra-se na validação da menor unidade de projeto de software, que pode ser tanto um módulo quanto um procedimento do programa, dependendo da granularidade desejada. Como nessa fase é possível ter o acesso direto ao código fonte, geralmente são utilizadas técnicas estruturais ou baseadas em falhas. O *teste de integração* é aplicado na validação das interfaces entre os módulos. Pode-se aplicar técnicas funcionais para garantir que a entrada e saída de dados das interfaces estejam consistentes, bem como técnicas estruturais para validar estruturas de controle. O *teste de sistema* concentra-se na validação do software integrado em seu ambiente. Já que o software é visto como uma caixa preta nessa etapa, técnicas funcionais têm sido utilizadas para verificar os requisitos comportamentais e funcionais exigidos.

1.1 Motivação

A atividade de teste tem sido apontada como uma das mais onerosas no desenvolvimento de software [21, 84, 117, 122]. Um relatório preparado para o NIST (*National Institute of Standards and Technology*) [65] quantifica os altos impactos econômicos de uma infraestrutura de teste de software inadequada e como melhorias cita, por exemplo, gerar automaticamente os casos de testes.

Bertolino [22] descreve os desafios mais relevantes na pesquisa de teste de software, entre eles a geração de casos de teste. E nesse contexto, comenta a promissora relação entre o teste de software e outras áreas de pesquisa, tal como o uso de abordagens baseadas em busca na geração de casos de teste [111, 77, 81]. Outro desafio citado é o teste baseado em modelos, sendo que a ideia principal é usar modelos no desenvolvimento de software para auxiliar o processo de teste, particularmente, na geração automática de casos de teste.

Devido à busca por diferenciais no ambiente econômico, caracterizado por uma grande competitividade, automatizar a geração de casos de teste é um importante ponto a ser considerado na atividade de teste para a redução de custos e esforços. O desenvolvimento de casos de teste consiste em identificar dados de entrada que satisfaçam um determinado critério de teste. A geração automática de casos de teste tem sido alvo de muitos pesquisadores recentemente, uma vez que a seleção manual de casos de teste é um processo

extremamente caro, difícil e trabalhoso. Nesse sentido, as principais abordagens para a geração automática de casos de teste são [114]: aleatória, simbólica e dinâmica. Na *geração aleatória*, existe uma probabilidade muito baixa de selecionar dados de entrada que pertençam a uma pequena porcentagem do domínio de entrada [61]. Entretanto, por ser uma abordagem simples e de fácil implementação, é comumente utilizada na literatura como uma medida de comparação. As abordagens que utilizam a *execução simbólica* [38, 118] atribuem valores simbólicos às variáveis, ao invés de usar seus reais valores, e reduzem o problema de geração de casos de teste à resolução de expressões algébricas que envolvem um conjunto de restrições de igualdade e desigualdade sobre esses valores simbólicos. Uma desvantagem é que esse método apresenta problemas com laços, vetores e referências a ponteiros, limitando seu uso nos dias de hoje [114, 95]. Já a *abordagem dinâmica* de geração de casos de teste baseia-se na execução dos valores reais das variáveis utilizando técnicas de otimização. Nesse caso, a geração de casos de teste é reformulada como um problema de otimização. A aplicação de técnicas de otimização para gerar casos de teste consiste em otimizar um determinado dado de entrada de acordo com o critério de teste expresso em uma função objetivo [108]. Quando um requisito de teste não é satisfeito, técnicas de otimização de funções são utilizadas para buscar dados que mais se aproximam de satisfazer o requisito. Com base na avaliação da função objetivo, os casos de teste são incrementalmente modificados. Um exemplo seria a busca de casos de teste para a cobertura de instruções de um programa, em que uma função objetivo é definida para informar se o caso de teste gerado está próximo ou não de executar a instrução desejada.

A área de pesquisa que investiga o uso de técnicas de otimização na geração de casos de teste é conhecida como teste de software baseado em busca (em inglês, *Search Based Software Testing*, SBST). Nos últimos anos houve um grande crescimento no interesse por SBST, sendo este aplicado a diferentes técnicas e critérios de teste. As abordagens propostas focam principalmente o teste estrutural [111, 81]. Entretanto, existem trabalhos que investigam SBST para o teste baseado em modelo, inclusive para Máquina de Estado Finito Estendida (MEFE) [57, 93, 92]. Nesse contexto, os casos de teste são derivados a partir da especificação, ao invés do código fonte, e são utilizados para garantir que a implementação está de acordo com a especificação. O teste baseado em modelo por ser uma abordagem de caixa preta possui a vantagem de derivar os casos de teste mesmo quando o código não está disponível. Isso permite que a atividade de teste possa ser realizada no início do ciclo de desenvolvimento. Além disso, o teste baseado em modelo pode ser aplicado desde o teste de unidade até o teste de sistema, de acordo com o nível de abstração do modelo.

Um aspecto diferencial de abordagens SBST para teste baseado em modelo é o número de elementos a ser otimizado. Em testes estruturais, por exemplo, um caso de teste consiste nos valores dos argumentos de um programa. Portanto, o número de argumentos é

conhecido, tal como os três lados de um triângulo para determinar seu tipo (i.e., equilátero, isósceles ou escaleno). Já no teste baseado em modelo, em particular MEFE, busca-se uma sequência de entrada que deve ser fornecida para que um caminho no modelo seja percorrido, a fim de cobrir transições ou estados do modelo. Assim, a geração de casos de teste também precisa levar em consideração seu tamanho, que nesse caso corresponde ao comprimento da sequência de teste. Em alguns trabalhos SBST, é necessário estabelecer a priori o tamanho do caso de teste [57, 93, 92, 101, 113, 105, 18], mas também existem iniciativas que permitem gerar casos de teste de tamanho variável para programas orientados a objeto [127, 20, 15].

Uma futura direção apontada para abordagens baseadas em busca é a utilização de otimização multiobjetivo [77, 81]. Ao invés de utilizar apenas uma função objetivo, podem-se considerar diversos objetivos simultaneamente. Foram propostos trabalhos SBST multiobjetivo no contexto de teste que focam diferentes propósitos [119, 32, 121, 98, 150, 107, 31, 14, 140, 100, 18], mas a geração de casos de teste a partir de MEFE ainda não foi abordada. Uma abordagem multiobjetivo é interessante para o teste baseado em MEFE, avaliando-se, por exemplo, o critério de teste como também o tamanho da sequência de teste.

Em uma MEFE, pode-se representar não somente a parte de controle, referente à ordem dos eventos de entrada e saída do sistema, mas também o aspecto de dados, relativos aos parâmetros dos eventos e às variáveis do modelo. Para disparar uma transição, é necessário receber o evento de entrada correspondente no seu estado de origem e também satisfazer sua guarda (i.e., expressão condicional) que pode envolver valores de parâmetros, variáveis e constantes. Após o disparo da transição, ações podem ser executadas, como a atualização de variáveis. Uma vez que os valores das variáveis não podem ser controlados diretamente pela técnica de otimização, existe um problema de perda de informação desses valores na busca, tal como ocorre no teste estrutural [113]. Assim, é necessário fornecer informações que possam guiar a busca, por exemplo, para que uma variável tenha um determinado valor em uma expressão condicional.

Outro problema relacionado aos dados envolvidos nas MEFEs é a existência de conflitos entre os dados das transições de um caminho. Devido aos dados envolvidos nas guardas e ações das transições, podem haver conflitos que impossibilitam que um caminho seja factível. Dessa maneira, um caminho pode ser semanticamente infactível, embora possa ser sintaticamente possível. O fato é que o fluxo de controle das MEFEs não é independente dos dados. Determinar se um caminho é factível ou infactível é um problema indecidível [59]. A responsabilidade de descartar esses caminhos é da equipe de teste, dispendendo tempo e esforço visto que pode ser gerado um grande número de caminhos. Além disso, mesmo quando o caminho é factível, encontrar um conjunto de dados de entrada que o exercite também é uma tarefa difícil devido ao domínio de valores

das entradas.

Embora várias técnicas de geração de casos de teste a partir de modelos já tenham sido propostas [24], a automação da geração tem sido limitada pelo tamanho e complexidade do sistema em teste. Em geral, essas técnicas consistem em realizar uma busca exaustiva no espaço de estados para satisfazer os requisitos de teste. Entretanto, para evitar o problema de explosão de estados por causa dos dados nas MEFEs, as técnicas impõem restrições sobre o domínio de valores ou até mesmo tratam separadamente a parte de controle e dados do modelo [126, 26, 60]. Assim, motivou-se a investigação de técnicas baseadas em busca que pudessem ser aplicadas para automatizar a geração de casos de teste, ao invés da exploração exaustiva do espaço de estados.

1.2 Objetivos

O principal objetivo deste trabalho consiste em investigar uma abordagem SBST para a geração de casos de teste a partir de MEFE, que leve em consideração tanto os aspectos do fluxo de controle como de dados do modelo. Ênfase é dada aos algoritmos evolutivos como técnica de otimização.

Em relação ao fluxo de controle da MEFE, a técnica de otimização deve gerar sequências de teste para percorrer caminhos no modelo. Além disso, é interessante permitir a geração de sequências de tamanhos variáveis. Em geral, o tamanho tem sido fixo nos trabalhos existentes [57, 93, 92, 101, 113, 105, 18], restringindo possíveis soluções e exigindo que se pré-determine esse valor. Dependendo do modelo, pode não ser uma tarefa fácil conhecer a priori um tamanho de sequência que seja adequado para alcançar uma transição no modelo, por exemplo.

A fim de lidar com os dados envolvidos no modelo, deve-se não somente gerar as sequências de eventos de entrada mas também os valores de seus parâmetros. Em algumas abordagens SBST [93, 92], esses valores são obtidos de forma aleatória e após o processo de geração. Porém, como as partes de controle e de dados são tratadas separadamente, existe a possibilidade de gerar caminhos infactíveis. Dessa forma, é importante que ambas as partes sejam consideradas ao mesmo tempo na geração dos testes. Outro ponto a ser levado em conta é como resolver o problema de falta de controle das variáveis existentes no modelo.

Como resultado, é proposta uma abordagem denominada MOST (*Multi-Objective Search-based Testing approach from EFSM*) que aplica os conceitos de SBST no teste baseado em modelo. Na abordagem, é utilizada uma otimização multiobjetivo que avalia tanto a cobertura de uma determinada transição quanto o tamanho da sequência gerada. Para guiar a busca, são utilizadas informações sobre as dependências de controle e de dados dos modelos. A análise das dependências permite que o fluxo de dados das variáveis

do modelo seja verificado. O problema de geração de caminhos infactíveis também é evitado, obtendo-se dinamicamente os caminhos de transições através da execução do modelo. Os dados, tais como os valores dos parâmetros e das variáveis, são levados em consideração durante a execução do modelo para habilitar qualquer transição. Dessa maneira, as partes de controle e dados de uma MEFE são consideradas na geração de casos de teste, obtendo-se os caminhos de transições assim como os dados para exercitá-los. Isso garante que, pelo menos em nível de modelo, os caminhos sejam factíveis. Para apoiar a abordagem, foi desenvolvido o algoritmo evolutivo M-GEO$_{vsl}$ (*Multi-Objective Generalized Extremal Optimization with variable string length*). O algoritmo estende uma versão multiobjetivo do algoritmo de Otimização Extrema Generalizada (em inglês, *Generalized Extremal Optimization*, GEO) [46, 47] para possibilitar que sejam geradas soluções com tamanhos variáveis.

O uso do algoritmo GEO visa a dar continuidade a outros trabalhos no grupo SED (*Software Engineering and Dependability Research Group*) do Instituto de Computação da Unicamp que investigam o GEO na atividade de teste [3, 2, 4, 5, 30, 29]. Em outros contextos, o GEO tem sido aplicado com sucesso em problemas reais de projeto ótimo [69, 47, 51, 52, 135] e se mostrou competitivo com outros métodos estocásticos em funções-teste [47]. A escolha do modelo MEFE também está relacionada com a pesquisa realizada dentro do grupo [9, 8, 10, 11, 110], além de ser modelo amplamente utilizado para representar especificações de sistema.

Assim, em geral, os estudos teóricos deste trabalho compreendem principalmente a investigação de abordagens de teste baseado em MEFE, teste baseado em busca, geração automática de casos de teste e algoritmos evolutivos multiobjetivo. Como estudo empírico, devem ser conduzidos diversos estudos de casos a fim de avaliar a abordagem proposta. Além disso, deve-se comparar os resultados obtidos com outros trabalhos SBST relacionados. Para a realização dos experimentos neste trabalho, foram utilizados estudos de casos encontrados na literatura como também aplicações reais.

1.3 Organização do Trabalho

Neste capítulo, são apresentados o contexto no qual este trabalho está inserido, os fatores que motivam a sua realização e seus objetivos. No Capítulo 2 são apresentados a definição de uma MEFE, o processo de teste baseado em modelo e alguns trabalhos sobre teste baseado em MEFE. No Capítulo 3 são apresentados os principais conceitos envolvidos em abordagens de teste baseado em busca, tais como uma visão geral de um algoritmo evolutivo e da otimização multiobjetivo. Também são descritos os trabalhos em SBST mais relevantes. No Capítulo 4 é apresentado o algoritmo M-GEO$_{vsl}$, mostrando a codificação das soluções, o operador de mutação e o processo evolutivo do algoritmo. No Capítulo 5

é descrita a abordagem MOST, instanciando o algoritmo M-GEO$_{vsl}$ para a geração de casos de teste a partir de MEFE. Além disso, são mostrados o modelo executável e as dependências do modelo utilizadas. No Capítulo 6 são apresentados os resultados dos experimentos realizados para avaliar a abordagem proposta. E finalmente, no Capítulo 7 são apresentados as contribuições deste trabalho e alguns trabalhos futuros.

Capítulo 2

Teste Baseado em Modelo

Modelos têm sido utilizados no desenvolvimento de sistemas complexos, pois permitem abstrair o comportamento do sistema, tornando mais fácil sua representação, a comunicação entre os participantes, e também a validação, a avaliação e a documentação dos sistemas. Teste baseado em modelo (em inglês, *Model Based Testing*, MBT) vem sendo proposto de longa data, para permitir a automação da geração de casos de teste como também da análise dos resultados [116]. Em MBT, os casos de teste são gerados automaticamente a partir de modelos.

Como o modelo é a base para a geração dos casos de testes, a construção do modelo é uma parte essencial em MBT. O processo de modelagem envolve a concepção dos elementos importantes para o sistema, bem como o relacionamento entre eles e a sua representação usando linguagens específicas e bem definidas. Um modelo constitui uma representação tangível das propriedades conceituais e físicas do sistema. Métodos formais, por apresentarem uma sintaxe e semântica bem definidas para a descrição das propriedades e do aspecto comportamental do sistema, oferecem consistência e precisão à especificação, desenvolvimento e verificação do sistema. Tretmans e Belinfante [130] destacam que especificações formais são mais facilmente processadas por ferramentas, permitindo uma maior automação no processo de desenvolvimento de software. Geurts et al. [70] relatam um estudo de caso em que se obtiveram benefícios na fase de teste com o uso de métodos formais para sistematicamente derivar casos de teste a partir dessas especificações. Pode-se citar como técnicas de especificação formal: MEF [72], Statecharts [76], Estelle [90], SDL [34], Lotos [89] e Redes de Petri [120].

Máquina de Estado Finito (MEF) [72] está entre os mais populares métodos formais para representação de especificações de sistema, como protocolos de comunicação. Entretanto, MEFs tradicionais representam apenas o fluxo de controle e não fornecem mecanismos para modelar um importante aspecto comportamental do sistema como o fluxo de dados. Assim, MEFs estão propensas ao problema de explosão de estados para

representar dados do sistema, visto que o número de estados pode crescer rapidamente devido ao domínio dos dados. Dessa maneira tem-se utilizado Máquina de Estado Finito Estendida (MEFE) que permitem representar os aspectos de controle como de dados do comportamento de um sistema, descrevendo ações e condições com parâmetros e variáveis.

Nas seções a seguir, são apresentados a definição de uma MEFE (Seção 2.1), a análise de dependência de uma MEFE (Seção 2.2), o processo de teste em MBT e seus desafios (Seção 2.3) assim como os trabalhos relacionados ao teste baseado em MEFE (Seção 2.4).

2.1 MEFE

Uma Máquina de Estados Finitos Estendida (MEFE) é uma generalização do modelo clássico da MEF. Uma MEF é uma máquina composta por estados e transições. A cada instante, a máquina pode estar em somente um de seus estados. Em resposta a um evento de entrada, a máquina gera um evento de saída e muda de estado. Tanto o evento de saída gerado quanto o novo estado são definidos unicamente em função do estado atual e do evento de entrada [42]. Já a MEFE estende uma MEF no sentido de permitir interações com parâmetros, uso de variáveis e transições associadas com condições, que utilizam os parâmetros do evento de entrada recebido e os valores atuais das variáveis, e associadas com ações, que podem atualizar os valores das variáveis [59].

Uma MEFE pode ser formalmente definida como em seguida.

Definição 2.1 *Máquina de Estado Finito Estendida (MEFE) é uma 7-tupla:*

$$M =< S, s_0, I, O, V, P, T >$$

onde S, I, O, V, P e T são os conjuntos finitos de estados, eventos de entrada, eventos de saída, variáveis, parâmetros de entrada e transições, respectivamente, e $s_0 \in S$ é o estado inicial. Eventos de entrada podem conter parâmetros que pertencem a P. Cada transição $t \in T$ é uma 5-tupla:

$$t_x =< q_x^s, q_x^t, i_x, g_x, a_x >$$

onde q_x^s, q_x^t e i_x são o estado de origem, estado de destino e evento de entrada de t_x, respectivamente. g_x é a guarda de t_x, que consiste em uma condição de habilitação que precisa ser verdadeira para que t_x seja percorrida. a_x são as ações de t_x que devem ser executadas quando t_x é disparada.

Uma MEFE pode ser representada em um grafo direcionado, como ilustrado na Figura 2.1 [96] que mostra o comportamento de um sistema de caixa eletrônico (M_1). Os dados do cartão e a senha devem ser fornecidos para a utilização das funcionalidades

do caixa eletrônico, que incluem verificar saldo, fazer depósito e fazer saque da conta corrente e poupança, e também escolher a língua do sistema entre inglês e espanhol. Cada transição t_x é descrita por: $i_x[g_x]/a_x$. Para disparar t_x, a máquina deve receber a entrada i_x com o estado atual em q_x^s e satisfazer g_x. Se g_x é verdadeiro, a_x é executado, podendo mudar os valores das variáveis do modelo.

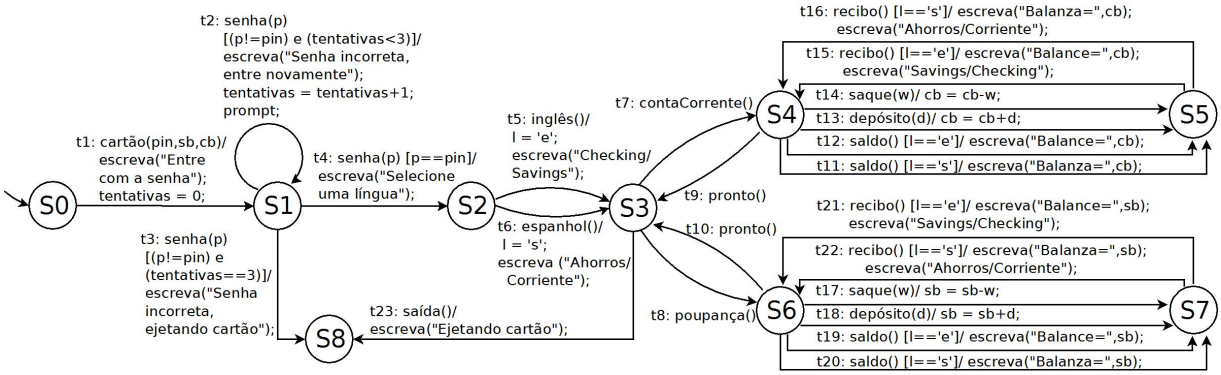


Figura 2.1: M_1: Sistema de caixa eletrônico representado em uma MEFE [96].

Um caminho de transições pode ser percorrido em uma MEFE dada uma sequência de entrada. Um **caminho de transições** é uma sequência de transições, tal que $q_j^t = q_{j+1}^s$, para cada par consecutivo de transições (t_j, t_{j+1}). Uma **sequência de entrada** é composta por eventos de entrada e os valores de seus parâmetros. Por exemplo, uma MEFE ao receber a sequência de entrada $seq_1 = \{cartão(442, 686, 12), senha(442), espanhol(), poupanca(), pronto(), poupanca(), depósito(654), recibo()\}$ percorre o seguinte caminho de transições $caminho = \{t_1t_4t_6t_8t_{10}t_8t_{18}t_{22}\}$. Essa sequência possui apenas **eventos de entradas nominais** que são eventos especificados em transições ao longo do caminho. Podem existir também **eventos de entrada inoportunos** (ou inesperados) que são aqueles não especificados em transições em um determinado estado. A máquina que possui eventos inoportunos é dita parcialmente especificada. Um evento que não tem tratamento em um estado, tanto por ser inoportuno quanto por não habilitar uma transição, é ignorado. Nessa situação, assume-se a semântica do modelo de estados da UML (*Unified Modeling Language*) em que a máquina permanece no estado atual e gera uma saída nula como resposta, visto que a MEFE representa um subconjunto desse modelo. Caso a sequência de entrada seja $seq_2 = \{cartão(442, 686, 12), saída()^\dagger, senha(442), saque(924)^\dagger, espanhol(), poupanca(), inglês()^\dagger, pronto(), poupanca(), depósito(654), recibo()\}$, sendo os eventos inoportunos indicados pelo símbolo †, o caminho de transições seria o mesmo disparado por seq_1.

2.2 Análise de dependência de MEFE

Análise de dependência tem sido utilizada em muitas atividades da Engenharia de Software, tais como manutenção de software e teste. Para programas, as dependências definem o relacionamento entre os comandos considerando o fluxo de controle e dados do programa. As dependências de controle dizem respeito à estrutura de controle do programa. As dependências de dados focam a relação de definição e uso de variáveis. Em geral, as dependências são representadas em um grafo que satisfaz o requisito de um único ponto terminal. Como modelos MEFE podem ter múltiplos nós de saída ou nenhum nó, abordagens de dependência de programa podem não ser aplicáveis.

Existem alguns estudos sobre análise de dependência para especificações baseadas em estados, tais como máquina de estado finito hierárquico [85], autômato I/O comunicantes [97], MEFE [96, 13, 12] e máquina de estado da UML [137]. Para lidar com a possibilidade de não terminação, Androutsopoulos et al. [13, 12] propuseram uma análise de dependência para modelos MEFE, cujas definições são utilizadas neste trabalho.

Androutsopoulos et al. definem as dependências em termos de transições de uma MEFE, ao invés de nós em dependências de programa. Uma noção genérica de dependência de controle entre transições é dada de acordo com uma função $CAMINHO$, que retorna um conjunto de caminhos com determinadas propriedades.

Definição 2.2 *Uma transição t_j é **dependente de controle** de uma transição t_i ($t_i \xrightarrow{DC} t_j$) se e somente se:* 1. *para todos os caminhos $\pi \in CAMINHO(t_i)$, t_j pertence a π;* 2. *existe pelo menos uma transição t_k com o mesmo estado de origem de t_i tal que existe um caminho $\pi \in CAMINHO(t_k)$ tal que t_j não pertence a π.*

A dependência de controle denominada NTSCD (*Non-Termination Sensitive Control Dependence*) é definida considerando $CAMINHO$ como uma função de caminho máximo.

Definição 2.3 *Um **caminho máximo** π é qualquer caminho que termina em uma transição final, ou é infinito. Uma transição final é aquela cujo estado de destino é um estado de saída no qual não partem transições.*

A dependência de dados (DD) relaciona as transições de acordo com as definições e usos das variáveis.

Definição 2.4 *Uma transição t_j é **dependente de dados** de uma transição t_i com relação a uma variável v ($t_i \xrightarrow{DD} t_j$) se:* 1. *$v \in D(t_i)$, onde $D(t_i)$ é um conjunto de variáveis definidas pela transição t_i, i.e., variáveis definidas pelas ações e pelo evento de t_i;* 2. *$v \in U(t_j)$, onde $U(t_j)$ é um conjunto de variáveis usadas na condição e ações da transição t_j;* 3. *existe um caminho na MEFE de t_i para t_j sendo que v não é modificada por qualquer transição intermediária.*

Um grafo de dependência pode ser construído mostrando as interdependências de controle e/dados entre as transições. A Figura 2.2 mostra as dependências de dados DD e de controle NTSCD obtida do modelo M_1 (Figura 2.1). A Figura 2.3 mostra o grafo de dependência (parcial) correspondente considerando ambos os tipos de dependência. Os nós representam as transições e as arestas direcionadas representam as dependências de dados (arestas sólidas) e controle (arestas tracejadas).

NTSCD	DD
$t_4 \rightarrow t_5, t_6, t_7, t_8, t_{23}$	$t_1 \rightarrow t_3, t_2, t_4, t_{19}, t_{22},$
$t_7 \rightarrow t_9, t_{11}, t_{12}, t_{13}, t_{14}$	$t_{17}, t_{18}, t_{21}, t_{20}, t_{11},$
$t_8 \rightarrow t_{10}, t_{17}, t_{18}, t_{19}, t_{20}$	$t_{14}, t_{13}, t_{15}, t_{16}, t_{12}$
$t_9 \rightarrow t_7, t_8, t_{23}$	$t_2 \rightarrow t_2, t_3$
$t_{10} \rightarrow t_7, t_8, t_{23}$	$t_3 \rightarrow t_3$
$t_{11} \rightarrow t_{15}, t_{16}$	$t_4 \rightarrow t_4$
$t_{12} \rightarrow t_{15}, t_{16}$	$t_5 \rightarrow t_{11}, t_{19}, t_{22}, t_{20},$
$t_{13} \rightarrow t_{15}, t_{16}$	$t_{15}, t_{21}, t_{16}, t_{12}$
$t_{14} \rightarrow t_{15}, t_{16}$	$t_6 \rightarrow t_{11}, t_{19}, t_{22}, t_{20},$
$t_{17} \rightarrow t_{21}, t_{22}$	$t_{15}, t_{21}, t_{16}, t_{12}$
$t_{18} \rightarrow t_{21}, t_{22}$	$t_{13} \rightarrow t_{13}, t_{14}, t_{11}, t_{12}, t_{15}, t_{16}$
$t_{19} \rightarrow t_{21}, t_{22}$	$t_{14} \rightarrow t_{14}, t_{13}, t_{11}, t_{12}, t_{15}, t_{16}$
$t_{20} \rightarrow t_{21}, t_{22}$	$t_{17} \rightarrow t_{17}, t_{19}, t_{22}, t_{20}, t_{18}, t_{21}$
	$t_{18} \rightarrow t_{18}, t_{19}, t_{22}, t_{17}, t_{20}, t_{21}$

Figura 2.2: Dependência de controle (NTSCD) e dados (DD) para a MEFE M_1.

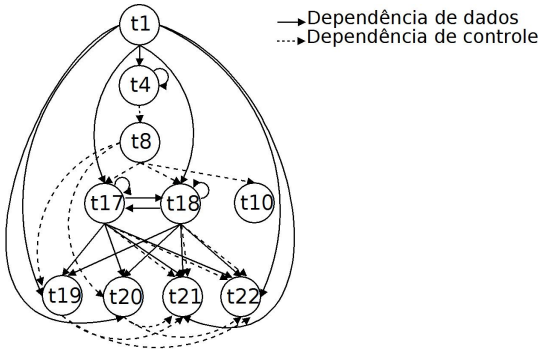


Figura 2.3: Grafo de dependência.

2.3 Processo de Teste

O processo de teste baseado em modelo envolve as seguintes atividades (Figura 2.4), baseando-se no processo de Utting e Legeard [132]:

1. Construir um modelo que represente o comportamento do sistema em teste a partir dos documentos de especificação. Deve-se focar nos principais aspectos a serem testados, omitindo detalhes do sistema em teste. É importante também validar o modelo, uma vez que a geração de casos de teste tem como base o modelo.

2. Gerar os casos de teste abstratos a partir do modelo. O critério de teste pode ser relacionado a funcionalidades do sistema ou estrutura do modelo, tal como cobertura de transições.

3. Transformar os casos de teste abstratos em executáveis. O objetivo desse passo é obter casos de teste em nível de implementação do sistema por meio de um adaptador que adiciona detalhes específicos, por exemplo, da linguagem de programação que não são abordados no modelo. Ao mudar as regras de transformação do adaptador, pode-se reusar o mesmo conjunto de casos de teste em diferentes plataformas de execução.

4. Executar os casos de teste no sistema em teste e atribuir um veredito: passou, não passou ou inconclusivo. Um teste passa quando se obtém a saída esperada, caso contrário, é revelada a presença de falha. O teste é inconclusivo quando a decisão sobre o teste não pode ser tomada.

5. Analisar os resultados dos testes e tomar as ações corretivas. Quando o teste não passa, é preciso verificar se a falha encontra-se no sistema em teste ou no próprio caso de teste, visto que pode-se ter falhas no adaptador, no modelo ou até mesmo na documentação.

Na abordagem MBT, os casos de testes podem ser obtidos mesmo quando o código fonte ainda não está ou não é disponível. Uma outra vantagem é que os casos de teste podem ser produzidos uma única vez para serem implementados em diferentes linguagens de programação e plataformas. Em caso de modificações no comportamento do sistema em teste, basta atualizar o modelo e gerar novamente os casos de teste de maneira automática, facilitando a manutenção dos testes [124, 132, 133].

Uma das limitações de MBT é que requer habilidade para modelar o sistema, implicando custos de treinamento e uma curva inicial de aprendizagem. Recomenda-se que o modelo seja mais simples do que o sistema em teste, senão o esforço de validar o modelo poderia ser equivalente ao de testar diretamente o sistema. Por outro lado, o modelo precisa ser suficientemente preciso para se poder gerar casos de teste. Há ainda a necessidade de mecanismos de transformar os casos de teste abstratos para o nível de implementação da plataforma de execução do sistema. A geração dos casos de teste também não abrange as partes omitidas na abstração do modelo [132].

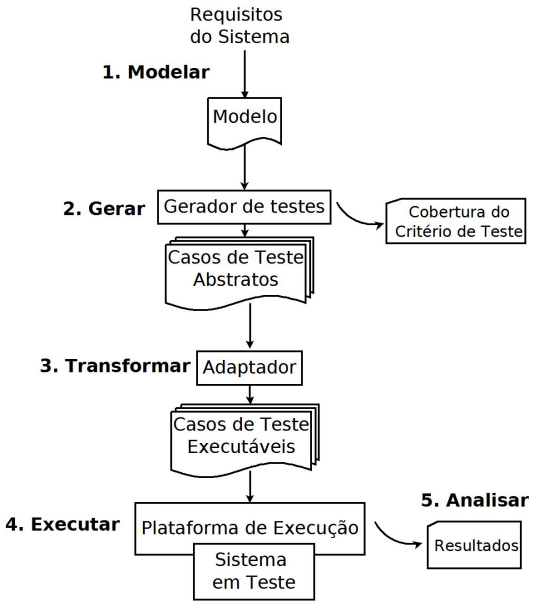


Figura 2.4: Processo de teste baseado em modelo.

Em particular no teste baseado em MEFE, a etapa de geração de casos de teste consiste em buscar sequências de eventos de entrada e seus parâmetros (i.e. sequência de entrada), e as saídas correspondentes, para exercitar caminhos de transições que satisfaçam um critério de teste. Pode-se citar como critérios de teste: *i*. cobertura de estados, em que cada estado deve ser visitado pelo menos uma vez; *ii*. cobertura de transições, em que cada transição deve ser exercitada pelo menos uma vez; *iii*. cobertura de caminhos, em que cada caminho possível no modelo deve ser exercitado pelo menos uma vez ou ainda pode-se restringir que cada transição no caminho não seja exercitada mais do que n vezes, para se evitar um número infinito de caminhos devido aos ciclos no modelo.

Devido aos dados envolvidos nas ações e guardas das transições das MEFEs, uma questão que deve ser considerada é o problema de geração de **caminhos infactíveis**. Um caminho pode ser sintaticamente possível, mas semanticamente infactível. Como uma transição t depende da sua guarda para ser disparada, a habilitação de t é restringida pelos valores atuais das variáveis e parâmetros de entrada. Por exemplo, um caminho infactível na MEFE da Figura 2.1 é obtido quando se requer um saldo da poupança em espanhol (transição t_{20}), mas seleciona-se o menu em inglês (transição t_5). Na Figura 2.5 existe um conflito de dados na ação de t_1 com a guarda de t_3. Devido aos valores de x, i e j, o caminho t_1t_3 é infactível. Note que t_3 é habilitada apenas após x execuções de t_2. Dessa maneira, a obtenção de caminhos de transições factíveis depende não apenas das interações de eventos de entrada, mas também do histórico das transições executadas anteriormente [138].

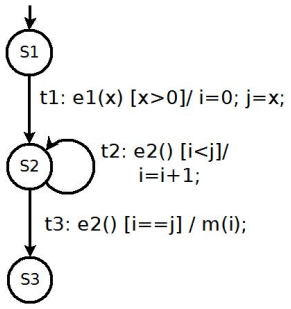


Figura 2.5: Exemplo de caminho infactível em uma MEFE: conflito de dados na ação de t_1 com a guarda de t_3.

2.4 Trabalhos Relacionados

Na literatura há diversas técnicas que já foram bem investigadas para o teste baseado em MEF, tais como: DS (*Distinguishing Sequence*) [73], Método W [36], UIO - *Unique-Input-Output* [125] e Método Wp [64]. Uma análise de alcançabilidade é realizada em um grafo que mostra todos os possíveis caminhos da MEF, a partir do estado inicial. Os caminhos são obtidos geralmente com uma busca em profundidade ou em largura nesse grafo. A fim de aplicar essas técnicas a MEFEs, pode-se abstrair a parte de dados da MEFE ou então converter a MEFE em uma MEF. No primeiro caso, como o fluxo de dados não é considerado, é mais propensa a produção de caminhos infactíveis. Já com a conversão, os caminhos derivados a partir da MEF são factíveis na MEFE original. Entretanto, pode ocorrer o problema de explosão de estados devido ao grande número de estados necessário para representar todo o domínio de entrada na MEF resultante. Além disso, as técnicas citadas baseiam-se em propriedades e sequências especiais das máquinas que, na prática, são muito difíceis de serem satisfeitas. Outro ponto é a análise exaustiva das técnicas que limitam o tamanho do modelo ou o domínio dos dados na geração dos casos de teste [132].

A seguir são descritos trabalhos que aplicam as técnicas baseadas em MEF apenas para testar o fluxo de controle da MEFE e propõem abordagens para o fluxo de dados, como também trabalhos que integram os aspectos de controle e dados da MEFE na geração de casos de teste.

A notação utilizada para a descrição dos predicados e ações das transições é geralmente baseada em alguma linguagem de programação de alto nível. Assim, as questões de sistematizar o teste baseado em MEFE são semelhantes às questões relativas ao teste estrutural [24]. Isso motivou diversos trabalhos a considerarem o teste do fluxo de controle e de dados separadamente. A ideia é selecionar casos de teste baseando-se nos critérios de fluxo de dados propostos no teste estrutural para a parte de dados, enquanto que as técnicas baseadas em MEF são aplicadas para a parte de controle. Porém, essa abordagem

possui os problemas mencionados anteriormente. A fim de separar claramente os aspectos de controle e dados, uma forma normal da especificação foi proposta por Sarikaya et al. [126]. Toda parte de controle é representada em MEF, enquanto que as ações são representadas em linhas de código sem instruções de condições e *loops*. Chanson e Zhu [35] também separam o teste do fluxo de controle e dados e utilizam a técnica de execução simbólica e satisfação de restrição para selecionar os dados de teste. Ural e Yang [131] realizam a análise de fluxo de dados de especificações em Estelle na forma normal. Estelle é uma técnica padrão de descrição formal baseada em MEFE. Identificam-se as definições e usos das variáveis e parâmetros de interação. Com base nessa informação, é definido o critério IO-df-chain (*Input-Ouput dataflow chain*) que requer que todas as associações entre as variáveis de uma saída e das entradas que influenciam essa saída sejam cobertas pelo menos uma vez. O trabalho não considera a parte de controle nem o problema de geração de caminhos infactíveis. Fantinato e Jino [62] propõem critérios de teste funcional para MEFE baseando-se nos critérios de fluxo de dados do teste estrutural. A seleção dos caminhos foca nas definições e usos das variáveis da MEFE.

Martins et al. [110] apresentam uma estratégia para desenvolver entradas de teste para protocolos de comunicação especificadas em MEFE. Possibilita a geração de casos de teste que cobrem tanto a parte de controle, com teste de transição, quanto a parte de dados da MEFE, através da combinação do teste baseado em sintaxe e particionamento de equivalência. A aplicação da estratégia é apoiada pela ferramenta ConDado, na qual é possível definir restrições que reduzem o número de testes gerados, tal como especificar quais transições devem ser exercitadas ou o número de vezes que um *loop* deve ser repetido. Além disso, gera valores aleatórios, com base no domínio de entrada das variáveis, mas não verifica se os caminhos são factíveis.

O problema de geração de caminhos infactíveis é abordado em alguns estudos, mas ainda os dados para exercitar os caminhos não são gerados. Bourhfir et al. [26] consideram esse problema na sua metodologia de teste baseado em MEFE que explora tanto o aspecto de controle como o de dados. Para análise do fluxo de controle usam UIO ou Wp e, para o fluxo de dados, o critério IO-df-chain [131]. A metodologia considera o problema da influência de *loops* e assume que a especificação é dada na forma normal. Duale e Uyar [60] propõem um método que permite a geração de casos de teste factíveis para uma classe de modelos MEFE. Por exemplo, assume-se que não existem ponteiros, funções recursivas e *loops* sem fim e todas as condições e ações são lineares. Nesse método, são utilizados algoritmos para detecção e eliminação de conflitos entre as variáveis de condições e ações nas MEFEs para obter apenas caminhos factíveis, mas sem perder qualquer propriedade do grafo original. Após a eliminação de todos os conflitos, geram-se os casos de teste por algum método baseado em MEF existente na literatura. Hierons et al. [86] tratam os caminhos infactíveis em dois passos. Primeiro derivam uma forma normal da MEFE

a partir de uma especificação SDL, uma linguagem baseada em MEFE. Em seguida, estendem essa forma normal a fim de auxiliar a testabilidade. Como resultado, apenas caminhos factíveis são obtidos.

Alguns trabalhos utilizam sequências de identificação de estado para lidar com o problema de geração de caminhos infactíveis. Essas sequências são utilizadas para verificar se a máquina está em um estado específico. Ramalingom et al. [123] apresentam um método de geração de casos de teste para MEFE usando um novo tipo de sequência de identificação de estado, chamado de CIUS (*Context Independent Unique Sequence*). O método analisa se os casos de teste são factíveis durante sua geração e abrange os aspectos de fluxo de controle e de dados de uma MEFE. No teste de fluxo de controle, CIUSs são úteis na confirmação dos estados finais das transições. No teste de fluxo de dados, CIUSs podem observar melhor os casos de teste para associações definição-uso de diferentes variáveis usadas na MEFE. Li et al. [104] também introduzem um novo tipo de sequência de identificação de estado que visa a factibilidade dos casos de teste, denominada de E-UIO (*Extended UIO*). O modelo MEFE utilizado é restrito ao tipo de dado inteiro e a condição das transições é descrita por uma expressão lógica composta de adição/subtração e comparação de inteiros. Porém, geram apenas casos de teste para o fluxo de controle.

Cavalli et al. [33] apresentam uma abordagem aleatória de geração de sequências em MEFE através do algoritmo denominado *Hit-or-Jump*. Essa abordagem possui mecanismos de evitar o problema de explosão de estados, embora realize uma busca exaustiva. Também é definido um tamanho máximo para a sequência. A abordagem pode demorar a encontrar partes não testadas, visto que a busca é aleatória e não possui informações que a guiem a soluções mais promissoras.

Existem algumas iniciativas que obtêm os casos de teste a partir de MEFE utilizando-se técnicas de otimização baseadas em busca [56, 57, 55, 93, 92], ao invés da exploração exaustiva do espaço de soluções como nos trabalhos mencionados acima. No próximo capítulo, cujo escopo é teste baseado em busca, são apresentados alguns desses trabalhos.

Na Tabela 2.1 é mostrada uma visão geral das soluções apresentadas anteriormente, verificando se abordam o fluxo de controle e de dados das MEFEs e o problema de geração de caminhos infactíveis. Na abordagem MOST proposta neste trabalho, uma técnica de otimização é utilizada para gerar os casos de teste a partir de MEFE, inclusive os parâmetros de entrada. Para a geração são considerados tanto os aspectos de controle quanto de dados. A fim de evitar caminhos infactíveis, é utilizado um modelo executável da MEFE que produz apenas caminhos factíveis, pelo menos no nível de modelo. Em geral, as técnicas citadas acima realizam apenas uma análise estática da estrutura do modelo, podendo ocorrer uma explosão de estados devido ao domínio de valores dos dados do modelo.

Tabela 2.1: Trabalhos para geração de casos de teste baseadas em MEFE.

Abordagem	*Fluxo de controle*	*Fluxo de dados*	*Caminhos factíveis*
Sarikaya et al. [126]	√	√	-
Chason e Zhu [35]	√	√	-
Ural e Yang [131]	-	√	-
Fantinato e Jino [62]	-	√	-
Martins et al. [110]	√	√	-
Bourhfir et al. [26]	√	√	√
Duale e Uyar [60]	√	√	√
Hierons et al. [86]	√	√	√
Ramalingom et al. [123]	√	√	√
Li et al. [104]	√	-	√
Cavalli et al. [33]	√	√	√

2.5 Considerações Finais

A geração de casos de testes a partir de modelos não exige que o código fonte do sistema em teste esteja disponível e também é independente de linguagem de programação e plataforma. Neste trabalho foco é dado à geração dos casos de teste, sendo que as demais etapas de MBT estão fora do escopo. No trabalho de mestrado da aluna Érika Regina Campos de Almeida do grupo SED são investigadas as regras de transformação dos casos de teste derivados a partir de MEFE em executáveis.

No teste baseado em MEFE, um problema em aberto é a geração de caminhos infactíveis, visto que pode haver conflitos de dados em condições e/ou ações ao longo do caminho. Além disso, a questão de quais dados devem ser fornecidos para que uma determinada transição seja executada é indecidível, ou seja, não é possível encontrar um algoritmo que retorne tais informações [59]. A explosão de estados também pode ocorrer em métodos exaustivos que produzem os possíveis caminhos em uma MEFE. Isso motivou o uso de técnicas de otimização nesse contexto de teste. No próximo capítulo são descritos os principais conceitos de teste baseado em busca.

Capítulo 3

Teste Baseado em Busca

Nos últimos anos houve um grande crescimento de trabalhos relacionados à Engenharia de Software Baseada em Busca (em inglês, *Search Based Software Engineering*, SBSE), na qual técnicas de otimização são aplicadas a várias atividades da engenharia de software. Podem ser encontrados trabalhos em [37, 77, 81]: engenharia de requisitos, planejamento de projeto, estimação de custo, teste, manutenção, engenharia reversa, engenharia de software orientada a serviço e avaliação de qualidade. Outra fonte de pesquisa nessa área é a lista de publicações mantida pelo projeto SEBASE[1] (*Software Engineering By Automated SEarch*) desde 2007. O termo SBSE surgiu apenas em 2001 [80], embora existam trabalhos de longa data neste contexto. Esse campo de pesquisa teve origem em trabalhos de teste de software baseado em busca (em inglês, *Search Based Software Testing*, SBST), para a geração automática de casos de teste. Como principais técnicas de otimização utilizadas estão os algoritmos evolutivos [108, 111, 81], mas também incluem a busca tabu [58], recozimento simulado [128, 102, 1], método do gradiente [95] e *hill climbing* [82]. A aplicação de algoritmos evolutivos na atividade de teste é denominada de teste evolutivo.

Os principais aspectos envolvidos tanto em SBSE quanto em SBST são apresentados na Seção 3.1. A estrutura geral de um algoritmo evolutivo é descrita na Seção 3.2 e os princípios de uma otimização multiobjetivo são apresentados na Seção 3.3, já que neste trabalho é proposto um algoritmo evolutivo multiobjetivo para a geração de casos de teste. Trabalhos relevantes em SBST relacionados com a abordagem proposta são apresentados na Seção 3.4.

[1] Disponível em: http://www.sebase.org/

3.1 Principais Aspectos

A ideia fundamental de SBSE é reformular problemas da engenharia de software como um problema de otimização. Independentemente da técnica de otimização utilizada, apenas dois aspectos são necessários para aplicá-la em problemas de engenharia de software [77]:

1. A escolha da representação do problema;
2. A definição da função objetivo.

Técnicas de otimização buscam soluções ótimas ou próximas de ótimas em um espaço de busca de soluções candidatas e são guiadas pelas avaliações da função objetivo, também conhecida por função de aptidão (em inglês, *fitness function*), função de avaliação ou função de custo. O papel da função objetivo é de grande importância visto que captura as informações cruciais para a otimização. Através dos valores da função objetivo é que se guia a busca, diferenciando uma boa solução de uma ruim [77]. As métricas existentes na engenharia de software podem ser consideradas como funções objetivo candidatas, por exemplo, cobertura estrutural em teste ou coesão e acoplamento em modularização [79]. Além disso, uma boa escolha da representação do problema garante que soluções similares sejam "vizinhas"no espaço representacional, permitindo que a busca mova-se mais facilmente de uma solução a outra que compartilha um conjunto similar de propriedades [111].

Pode-se destacar algumas vantagens para a aplicação de SBSE [77, 81]: *i*. escalável, visto a natureza paralela das técnicas de otimização; *ii*. genérica, devido à grande variedade de problemas da engenharia de software em que SBSE pode ser aplicada; *iii*. robusta, já que pode-se lidar com dados imprecisos e parciais (sub-ótimos); *iv*. perspicaz, dada a possibilidade de descobrir soluções não esperadas por especialistas; *v*. realista, visto que pode-se levar em consideração objetivos de engenharia conflitantes e competitivos. Além dessas motivações, outra razão que torna atrativo o uso de otimização baseada em busca em especial na engenharia de software é a propriedade virtual do software [78]. Os artefatos da engenharia de software podem ser avaliados e comparados diretamente pela técnica de otimização, não sendo necessário realizar simulações como na engenharia tradicional, para otimizar um motor de avião, por exemplo.

Segundo Harman et al. [81], os trabalhos na literatura de SBSE focam 59% em aplicações relativas à atividade de teste. Em SBST, o conjunto das entradas possíveis é considerado como o espaço de busca e as funções objetivo avaliam se os critérios de teste ou as caraterísticas do software foram alcançados. Considerando-se a aplicação de algoritmos evolutivos na geração automática de casos de teste, os indivíduos sobre os quais o processo de otimização ocorre constituem os casos de teste e estes são representados de maneira a permitir a aplicação de operadores genéticos. A função objetivo é utilizada como meio de se medir a cobertura que se deseja alcançar [37]. Os valores das funções

objetivo são obtidos pela comparação das soluções da busca em relação ao seu objetivo geral, direcionando a busca para regiões promissoras do espaço de busca [111]. Entre as principais referências em SBST pode-se citar o levantamento bibliográfico realizado por McMinn [111], Harman et al. [81] e Ali et al. [7]. Para o teste de requisitos não funcionais, tal como quantidade de memória utilizada, pode-se consultar o material de Afzal et al. [6].

A maioria dos trabalhos em SBST foca o teste baseado em código, principalmente teste estrutural. Os trabalhos procuram satisfazer vários critérios de teste, tais como cobertura de instruções, caminhos e predicados, aplicando diferentes métodos de otimização e funções objetivo. Harman e McMinn [82] apresentam uma análise teórica e empírica sobre a geração de casos de testes estruturais, respondendo questões como: "Quando e por que o teste evolutivo funciona?"e "Qual o desempenho do teste evolutivo em comparação às outras técnicas de busca". Também pode-se encontrar iniciativas para [111, 81, 6]: o teste funcional, teste não funcional, teste de robustez, teste de estresse, teste de mutação, teste de regressão, teste de integração e teste de exceção.

3.2 Algoritmos Evolutivos

O termo algoritmo evolutivo (AE) é uma denominação comum a todas as abordagens para sistemas baseados em princípios de evolução. Entre as principais abordagens propostas na literatura estão: algoritmo genético (AG), estratégias evolutivas (EE), programação evolutiva (PE) e programação genética (PG). Os sistemas baseados em computação evolutiva mantêm uma população de soluções potenciais, aplicam processos de seleção baseados na adaptação de um indivíduo e também empregam outros operadores "genéticos". Apesar das abordagens terem sido desenvolvidas de forma independente, seus algoritmos possuem uma estrutura comum. A estrutura de um algoritmo evolutivo é mostrada no Algoritmo 3.1 [115].

Uma população $P(t) = \{x_1^t, \ldots, x_n^t\}$ é mantida no algoritmo evolutivo na iteração (geração) t. Cada indivíduo representa um candidato à solução do problema e é implementado como alguma estrutura de dados S. Avalia-se cada solução x_i^t através da função objetivo e um valor de aptidão (em inglês, *fitness*) é produzido. Com base nessas informações, é realizada a seleção dos indivíduos mais adaptados formando uma nova população na iteração $t + 1$. Alguns indivíduos são submetidos a um processo de alteração por meio de operadores genéticos para formar novas soluções: i. transformações unárias m_i (mutação) que fazem pequenas modificações de atributos em um indivíduo ($m_i : S \rightarrow S$) e ii. transformações de ordem superior c_j (*crossover*) que combinam dois ou mais indivíduos ($c_j : S \times \cdots \times S \rightarrow S$). Após um certo número de gerações, a condição de parada deve ser alcançada, indicando geralmente um indivíduo da população que represente pelo menos um incremento na qualidade da solução para o problema, ou quando

Algoritmo 3.1 Estrutura de um algoritmo evolutivo [115].

Entrada: número de iterações t
Saída: População $P(t)$
  $t \leftarrow 0$
  Inicialize $P(t)$
  Avalie $P(t)$
  enquanto não condição de parada **faça**
    $t \leftarrow t + 1$
    Selecione $P(t)$ a partir de $P(t-1)$
    Altere $P(t)$
    Avalie $P(t)$
  fim enquanto
  retorne $P(t)$

o número máximo de gerações foi atingido [115].

As abordagens evolutivas compartilham o mesmo princípio comum: uma população de indivíduos sofre algumas transformações e durante a evolução os indivíduos competem pela sobrevivência. Entretanto, diferenciam-se em diversos aspectos, dentre os quais se destacam: estruturas de dados utilizadas para codificar um indivíduo, operadores genéticos empregados, métodos para criar a população inicial e métodos para selecionar indivíduos para a geração seguinte. Um algoritmo evolutivo também pode utilizar apenas uma função objetivo para avaliar as soluções (i.e. mono-objetivo), ou duas ou mais funções objetivo (i.e. multiobjetivo). Na seção a seguir são apresentados os conceitos envolvidos na otimização multiobjetivo.

3.3 Otimização Multiobjetivo

Uma das direções apontadas como trabalhos futuros em SBSE é a otimização multiobjetivo, visto que diversos problemas da engenharia de software envolvem múltiplos objetivos [81].

Formalmente, o problema multiobjetivo de minimização pode ser definido como [134]:

Definição 3.1 *Um problema multiobjetivo minimiza* $F(\vec{x}) = [F_1(\vec{x}), \ldots, F_{nf}(\vec{x})]$ *sujeito a* $g_j(\vec{x}) \leq 0, j \in \{1, \ldots, m\}$ *e* $h_k(\vec{x}) = 0, k \in \{1, \ldots, p\}$*, onde* $\vec{x} = x_1, \ldots, x_n$ *é um vetor n-dimensional de variáveis de projeto de algum universo* Ω *e* nf *é o número de funções objetivo.*

Na Figura 3.1 [134] é ilustrado um problema de otimização multiobjetivo, mostrando o mapeamento da solução $\vec{x} = x_1, \ldots, x_n$ para o vetor $\vec{y} = y_1, \ldots, y_{nf}$ no espaço das funções objetivo ($F : \Omega \rightarrow \Lambda$).

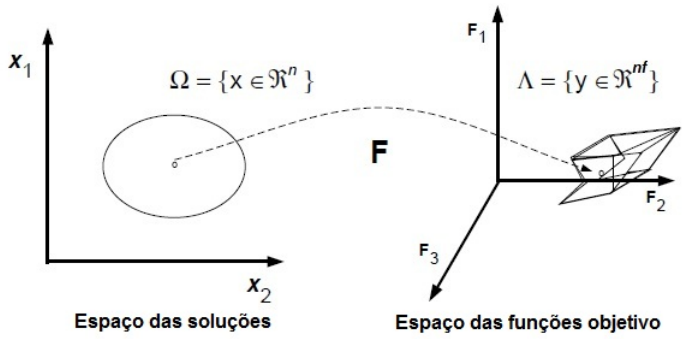


Figura 3.1: Ilustração de um problema de otimização multiobjetivo [134].

Uma abordagem convencional é combinar todos os objetivos em um único, denominada **agregação de funções**. Pode-se combinar um problema com nf funções objetivo $F_1(\vec{x}), \ldots, F_{nf}(\vec{x})$ sobre algum vetor $\vec{x}$ de acordo com um conjunto de coeficientes $c_1, \ldots, c_{nf}$:

$$F(\vec{x}) = \sum_{i=1}^{nf} c_i F_i(\vec{x}) \tag{3.1}$$

Assim o problema é tratado como uma otimização mono-objetivo. Em geral não é possível determinar precisamente os valores dos coeficientes; assim o ajuste dos objetivos com os coeficientes é tipicamente subjetivo e dependente do problema.

Para evitar o problema da determinação dos coeficientes, pode-se utilizar a noção de otimização de Pareto, na qual existe um conjunto de soluções de compromisso que procuram balancear todos os objetivos. Neste caso, as soluções de um problema multiobjetivo constituem o conjunto de Pareto (espaço das soluções) que leva em consideração todos os objetivos e os valores correspondentes das funções objetivo formam a fronteira de Pareto (espaço das funções objetivo). Existe uma relação de dominância entre as soluções, denotada por $\vec{u} \prec \vec{v}$ (solução $\vec{u}$ domina solução $\vec{v}$). Cada solução do conjunto de Pareto é não-dominada por todas as outras soluções. Isso significa que a solução não pode ser melhorada em qualquer objetivo sem causar uma degradação em no mínimo um outro objetivo. Assim, no conjunto ótimo de Pareto (P^*), nenhuma uma solução $\vec{x}'$ pertencente ao conjunto domina outra solução $\vec{x}$ de P^*. Os valores das funções objetivo de cada solução $\vec{x}$ de P^* ($F(\vec{x})$) são representados na fronteira de Pareto (FP^*).

Definição 3.2 *Um vetor* $\vec{u} = u_1, \ldots, u_{nf}$ *domina* $\vec{v} = v_1, \ldots, v_{nf}$ *(denotado por* $\vec{u} \prec \vec{v}$*) se e somente se* $\vec{u}$ *é parcialmente menor que* $\vec{v}$*, i.e.,* $\forall i \in \{1, \ldots, nf\}$*,* $u_i \leq v_i \wedge \exists i \in \{1, \ldots, nf\} : u_i < v_i$

Definição 3.3 *Para um determinado problema multiobjetivo* $F(\vec{x})$*, o conjunto ótimo de Pareto* (P^*) *é* $P^* = \{\vec{x} \in \Omega | \neg \exists \vec{x}' \in \Omega : F(\vec{x}') \prec F(\vec{x})\}$

Definição 3.4 *Para um determinado problema multiobjetivo* $F(\vec{x})$ *e conjunto ótimo de Pareto* P^**, a fronteira ótima de Pareto é* $FP^* = \{\vec{u} = F(\vec{x}) | \vec{x} \in P^*\}$

A Figura 3.2 ilustra a computação da otimalidade de Pareto para minimização de duas funções objetivo quaisquer F_1 e F_2. As soluções representadas pelos pontos p_1, p_2 e p_3 na figura pertencem à fronteira de Pareto. Já p_4 não faz parte do conjunto de Pareto, visto que os valores de F_1 para p_2 e p_4 são os mesmos, mas p_2 possui um valor melhor em relação a F_2 do que p_4. Portanto p_4 é uma solução dominada por p_2. Sob o ponto de vista de um problema de minimização, a solução p_1 possui o melhor valor para F_1 mas o pior para F_2. Analogamente, a solução p_3 possui o melhor valor para F_2 mas o pior para F_1. Como muitas soluções podem ser obtidas no conjunto de Pareto, o tomador de decisão é responsável por selecionar as soluções desejadas. Por exemplo, em um problema em que é dada prioridade para F_1, a solução p_1 poderia ser selecionada entre as soluções do conjunto de Pareto.

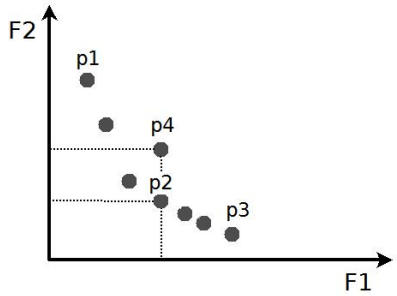


Figura 3.2: Fronteira de Pareto.

Gerar o conjunto de Pareto é computacionalmente caro, ou mesmo infactível; por esse motivo algoritmos evolutivos podem ajudar a encontrar uma solução aproximada [152]. De acordo com Coello Coello [41], os algoritmos evolutivos multiobjetivo podem ser divididos em duas gerações. Na primeira geração, os conceitos da otimalidade de Pareto são introduzidos diretamente no processo de seleção do algoritmo evolutivo. O critério de não-dominância é utilizado para definir a adaptação dos indivíduos e, consequentemente, a população torna-se uma aproximação direta do conjunto de Pareto. Existem diversos

tutoriais [39, 40, 134] que podem ser consultados sobre os detalhes de algumas dessas abordagens, tais como: NSGA (*Nondominated Sorting Genetic Algorithm*), NPGA (*Niched-Pareto Genetic Algorithm*) e MOGA (*Multi-Objective Genetic Algorithm*). Já na segunda geração, atual fase, é adotado o mecanismo de elitismo, no qual uma população externa (também denominada de população secundária) mantém os indivíduos não-dominados encontrados ao longo do processo evolutivo. Ao invés de verificar a não-dominância do indivíduo apenas na população atual, como nos trabalhos da primeira geração, verifica-se com relação a todas populações produzidas pelo algoritmo. Pode-se citar como os algoritmos mais representativos dessa geração [71, 152]: SPEA (*Strength Pareto Evolutionary Algorithm*), SPEA2 (*Strength Pareto Evolutionary Algorithm 2*), PAES (*Pareto Archived Evolution Strategy*) e NSGA-II (*Nondominated Sorting Genetic Algorithm II*).

Neste trabalho é proposto um algoritmo evolutivo multiobjetivo, denominado M-GEO$_{vsl}$ (Capítulo 4). O algoritmo não utiliza a noção de não-dominância para calcular as adaptações das soluções, como por exemplo em: NSGA-II, NPGA, MOGA e SPEA. É utilizada uma estratégia simples similar ao algoritmo VEGA (*Vector Evaluated Genetic Algorithm*) [41], no sentido de não considerar diretamente os conceitos de otimalidade de Pareto no processo de evolução das soluções. Entretanto, assim como nas abordagens da segunda geração, o conjunto de Pareto é armazenado em um arquivo (ou memória) externo. Além disso, o M-GEO$_{vsl}$ utiliza um esquema de reinicialização a fim de aumentar a possibilidade de encontrar soluções sobre toda a fronteira de Pareto. Uma característica particular desse algoritmo é a capacidade de lidar com soluções de tamanho variável.

3.4 Trabalhos Relacionados

Embora grande parte dos trabalhos em SBST diga respeito a teste baseado em código, podem-se encontrar esforços para o teste baseado em modelo. Como neste trabalho é proposta uma abordagem SBST a partir de MEFE, são descritos a seguir os principais trabalhos relacionados que derivam casos de teste a partir de modelos ou especificações (Tabela 3.1) Também são apresentadas abordagens que geram sequências de teste para programas orientados a objeto (Tabela 3.2) e algumas iniciativas de uso de otimização multiobjetivo no contexto de teste (Tabela 3.3).

Alguns trabalhos utilizam algoritmos evolutivos para obter automaticamente sequências de teste de abordagens tradicionais de teste baseado em MEF. Guo et al. [74] investigam a construção de UIOs (*Unique Input/Output sequences*) utilizando um algoritmo genético (AG). UIOs são usadas no teste de MEF para verificar o estado final de uma sequência de transição. A função objetivo guia a busca para potenciais UIOs, considerando dois aspectos: (i) recompensando-se soluções criadas de forma mais discreta e (ii) punindo-se

o tamanho das soluções. Um conjunto de experimentos foi realizado para comparar o AG e a busca aleatória. Os resultados da simulação mostraram que em sistemas simples o AG não teve vantagens significativas sobre a busca aleatória. Entretanto, em sistemas mais complexos, o AG conseguiu encontrar UIOs mais longas do que a busca aleatória. Guo et al. [75] também investigam a construção de UIOs com técnicas de compartilhamento com o objetivo de detectar efetivamente ótimos locais, um problema levantado no trabalho anterior. Como as soluções de UIOs formam regiões multi-modais (ótimos locais) no espaço de busca, uma busca pode encontrar apenas alguns desses ótimos locais. Uma regra é definida para calcular o grau de similaridade entre os indivíduos, sendo utilizada para punir aqueles que são muito similares a outros. Assim, esses indivíduos punidos têm menor probabilidade de serem selecionados para reprodução, ajudando a manter a diversidade da população. Essa abordagem utiliza AG e recozimento simulado para encontrar as UIOs. Não houve diferença entre ambos em termos de distribuição de UIOs e também superaram o desempenho da proposta anterior sem a técnica de compartilhamento, identificando-se mais UIOs.

Derderian et al. [54] também descrevem um método para gerar UIOs de MEF automaticamente utilizando um AG. Porém, em relação ao trabalho de Guo et al. [74], há a diferença da máquina poder ser parcialmente especificada, uma vez que assumia-se uma máquina completamente especificada. Além disso, a função objetivo é mais simples e tem complexidade $O(l)$, ao invés de $O(nl)$, onde l é o tamanho da sequência de entrada e n o número de estados da MEF. A função objetivo visa a recompensar sequências que são provavelmente uma UIO. Um processo de classificação da transição é feito antes do cálculo da função objetivo, observando quantas vezes o par entrada/saída da máquina ocorre na tabela de transição (uma tabela com todas as transições de uma máquina). O par que ocorre apenas uma vez é classificado na menor posição e assim sucessivamente, visto que quanto menor a repetição do par, mais chances da sequência formar uma UIO. Pares com o mesmo número de ocorrências ficam na mesma posição da classificação, assumindo então que esses pares têm habilidades similares para construir uma UIO válida. Com base nessas informações, o valor de aptidão é calculado como a soma das posições dos pares de entrada/saída que compõem a sequência. Para lidar com máquinas parcialmente especificadas, as transições que faltam são ignoradas, mantendo a máquina no mesmo estado e a próxima entrada na sequência é considerada. Nesse caso o valor de aptidão é penalizado, enquanto que sequências válidas não são penalizadas. O desempenho do método foi avaliado em grandes MEFs geradas aleatoriamente e não apenas em pequenas máquinas como no trabalho anterior. A abordagem foi comparada com a geração aleatória de UIO e os resultados mostraram que conforme o tamanho da MEF aumenta, melhora-se o desempenho do AG em relação à geração aleatória. Notou-se ainda que o tamanho médio da UIO tende a ser pequeno mesmo em grandes MEFs.

Lam et al. [99] investigam a geração de sequências de teste a partir de MEF baseada no método Wp usando a otimização de colônia de formigas. Busca-se encontrar o menor caminho para o problema do caixeiro viajante. A abordagem pode ser aplicada em MEFs minimais e que possuem um conjunto de caracterização, ou seja, conjunto de sequências de entrada capaz de distinguir quaisquer dois estados de uma máquina. As sequências de teste resultantes foram menores em relação aos trabalhos existentes de otimização de geração de sequências de teste.

Para testar aplicações web Ajax, Marchetto e Tonella [109] extraem dinamicamente uma MEF da aplicação e aplicam algoritmos baseados em busca para explorar sequências de teste. As sequências são analisadas de acordo com vários critérios de diversidade. Diferentes tamanhos de sequências pré-determinados foram aplicados nos experimentos.

Com relação a MEFEs, as abordagens existentes utilizam AG para buscar caminho de transições factível (CTF). No trabalho de Derderian et al. [57] uma função objetivo é definida para estimar quão provável o caminho de transições é factível e quão fácil é gerar uma sequência de entrada para dispará-lo. Para medir a factibilidade de um CTF, um gerador aleatório de parâmetros de entrada é usado. Quando o gerador encontra mais parâmetros de entrada para um CTF se tornar factível, esses são considerados como melhores. A função objetivo recompensa as transições que satisfazem com mais facilidade as guardas, a fim de minimizar o esforço de geração de sequência de entrada. Foram gerados caminhos de transições entre o estado inicial e final da máquina com o AG. Foram utilizados diferentes tamanhos de caminhos pré-determinados. Os resultados foram comparados com a geração aleatória de caminhos de transições, que não conseguiu encontrar nenhum CTF enquanto que o AG encontrou caminhos de tamanhos diferentes. Observou-se também que a busca torna-se mais difícil à medida que o tamanho do caminho cresce.

Kalaji et al. [93] também visam a geração de CTF a partir de uma MEFE utilizando um AG. O modelo é transformado para facilitar a busca por caminhos factíveis. Uma análise de dependência é realizada para determinar as transições que afetam t_i e que são afetadas t_j por outras transições. De acordo com essa análise, a função objetivo avalia os caminhos de transições gerados pelo AG. Para cada par de transições que possuem uma relação de dependência (t_i, t_j), são atribuídas penalidades na função objetivo levando-se em consideração três fatores: 1) o tipo de guarda de t_j, para determinar o quanto esta transição pode ser impactada por t_i, verificando-se se a guarda envolve parâmetros, variáveis e constantes; 2) o tipo do operador da guarda de t_j (e.g., operador relacionais e de igualdade), para verificar a complexidade de satisfazer um determinado tipo de operador; e 3) o tipo de atribuição de t_i, para verificar essa operação envolve parâmetros, variáveis e constantes. O valor da função objetivo reflete a dificuldade de percorrer um determinado caminho, em termos da complexidade das guardas e dados envolvidos no caminho. Os valores dos parâmetros são obtidos aleatoriamente após a geração dos caminhos; dessa

forma ainda existe a possibilidade de geração de caminhos infactíveis. É utilizado apenas um único tamanho de caminho. Essa abordagem foi estendida para lidar com guardas que referenciam um contador (variável) e seu valor depende de transições anteriores [92].

Zhao et al. [151] obtêm caminhos completos do estado inicial ao final em MEFE aplicando uma busca em largura. Os tamanhos dos caminhos obtidos variam de 3 a 50. Potenciais caminhos factíveis são selecionados, eliminando-se caminhos com conflitos nas condições de guarda. Um AG é utilizado para gerar os valores dos parâmetros de entrada dos caminhos selecionados, e não para gerar os caminhos em si. Através de uma análise empírica, verificou-se que os custos de geração dos dados cresce exponencialmente conforme o número de parâmetros de entrada e o tamanho do caminho quando existem operadores de igualdade nas guardas.

Em sistemas especificados na notação Z, Jones et al. [91] utilizam um AG na geração de casos de teste. A notação Z é uma linguagem de especificação formal baseada em notações matemáticas como a lógica de primeira ordem. A função objetivo avalia os indivíduos usando a abordagem de distância de ramo, recompensando aqueles que mais se aproximam de satisfazer as conjunções de cada disjunção. Esse trabalho limitou-se a um exemplo pequeno e utilizou apenas um subconjunto de Z.

Para o teste funcional, Bühler e Wegener [28] apresentam a aplicação de algoritmo evolutivo em um sistema de estacionamento automático. Os indivíduos são cenários do estacionamento que incluem áreas de colisão e posição inicial do carro. Os dados gerados são usados em um ambiente de simulação do sistema a fim de emular manobras para se estacionar e um caso de teste com sucesso é aquele que causa uma colisão. Assim o valor da função objetivo é calculado com base na avaliação da qualidade das manobras simuladas pelo sistema. Duas funções objetivo foram propostas e comparadas utilizando diferentes medidas de avaliação: *i*) distância entre o carro e a área de colisão, e *ii*) área entre o carro e a área de colisão. Embora ambas as funções identifiquem os mesmos cenários onde acontece uma colisão, avaliou-se que o segundo critério pode direcionar melhor uma busca para conseguir uma colisão.

Baresel et al. [18] introduzem abordagens SBST que visam a busca de sequências de entrada para os testes estrutural e funcional. Os testes são aplicados em um sistema de controle de carro Distronic, que é capaz de perceber veículos mais lentos e automaticamente diminuir a velocidade do carro a fim de manter uma distância segura. O caso de teste no teste estrutural consiste em uma sequência de entradas, sendo que cada entrada representa uma chamada da função em teste com seus parâmetros. Um caminho de execução é produzido para cada entrada. Para o valor de aptidão, são analisados todos os caminhos de execução e é selecionado aquele que mais se aproxima do requisito de teste. Para o teste funcional, são geradas sequências de entradas para modelos Simulink

e Stateflows[2]. Os casos de teste são sequências de sinais de entrada que visam violar o comportamento funcional do sistema. São verificadas as seguintes violações: dependência entre os sinais de saída do sistema, limites de amplitude dos sinais, variação máxima do sinal e tempo de estabilização do sinal. Para cada tipo de violação é produzido um valor de aptidão, avaliando-se as saídas geradas pela simulação do modelo com os casos de teste. Todos os valores de aptidão são analisados pela otimização multiobjetivo apoiada pela ferramenta GEATbx. Em ambas as abordagens o tamanho da sequência é determinado pelo usuário.

Tabela 3.1: Trabalhos relacionados em SBST: teste a partir de especificações.

Abordagem	*Técnica de Otimização*	*Contexto de Teste*
Guo et al. [74, 75]	AG	MEF: UIO
Derderian et al. [54]	AG	MEF: UIO
Lam et al. [99]	colônia de formigas	MEF: método Wp
Marchetto e Tonella [109]	recozimento simulado e *hill climbing*	MEF
Derderian et al. [57]	AG	MEFE
Kalaji et al. [93, 92]	AG	MEFE
Zhao et al. [151]	AG	MEFE (parâmetros de entrada)
Jones et al. [91]	AG	Notação Z
Bühler e Wegener [28]	AE (GEATbx)	teste funcional
Baresel et al.[18]	AE multiobjetivo (GEATbx)	teste estrutural e funcional (Simulink e Stateflows)

Outra direção de investigação encontrada é a geração de sequências de chamadas de métodos para o teste de programas orientados a objeto. Similarmente aos caminhos em MEFE, essas sequências são formadas por diversos elementos que também possuem parâmetros. Lefticaru e Ipate [101] buscam uma lista de valores de parâmetros de uma sequência de chamadas de métodos. A sequência é representada como um caminho em um modelo de estados. Assim uma solução corresponde aos dados dos parâmetros dos métodos que pertencem ao caminho a ser coberto. Nesse caso, como se busca cobrir um determinado caminho, o número de parâmetros é conhecido. Uma função objetivo é definida usando conceitos de teste estrutural, tais como nível de aproximação e distância de ramo. McMinn e Holcombe [113] também usam esses conceitos para gerar sequências de chamadas de métodos assim como seus parâmetros aplicando a abordagem da cadeia (em

[2]Simulink é uma ferramenta que fornece um ambiente gráfico interativo e um conjunto de bibliotecas de blocos que permitem projetar, simular, implementar e testar vários sistemas, tais como processamento de sinais, processamento de imagens e sistemas de comunicação. Stateflow estende Simulink para representar máquinas de estados e grafo de fluxo.

inglês, *chaining approach*) [63]. É utilizada uma codificação genérica de uma sequência de chamadas de métodos de tamanho pré-definido formada por duas partes. A primeira parte permite que o algoritmo evolutivo escolha livremente quais métodos chamar e a segunda é destinada para as chamadas de métodos que precisam ser executadas para alcançar o requisito de teste. Liaskos e Roper [105] introduzem uma abordagem que combina o algoritmo genético com conceitos de algoritmos de sistemas imunes artificiais (SIA) para obter casos de teste de classes Java. A função objetivo também é baseada no teste estrutural e visa à cobertura de ramos.

Existem trabalhos que permitem que o tamanho da sequência de chamadas de métodos seja variável, aspecto não considerado nos trabalhos descritos acima em que o tamanho é fixo e pré-determinado. Tonella [127] apresenta uma abordagem evolutiva que gera casos de teste para programas orientados a objeto e uma ferramenta de apoio denominada eToc. O caso de teste inclui informações sobre a ordem de criar os objetos, invocar métodos e utilizar valores dos parâmetros. Através dos operadores definidos pode-se modificar o tamanho da sequência, por exemplo, pela inclusão e remoção de métodos. Baresi et al. [20] geram sequências de teste para classes Java, apoiada pela ferramenta TestFul. Propõem uma abordagem evolutiva híbrida que realiza uma busca global no nível de classe e uma busca local no nível dos métodos. Também utilizam um algoritmo multiobjetivo que considera cobertura de ramo, tamanho do caso de teste e tempo de geração. Arcuri e Yao [15] também buscam sequências de teste para programas orientados a objeto com uma abordagem baseada em busca. A representação do caso de teste permite descrever uma sequência de chamadas de métodos, que pode ser modificada incluindo ou removendo-se as chamadas. A função objetivo leva em consideração a cobertura dos ramos, distância dos ramos e tamanho do caso de teste.

Tabela 3.2: Trabalhos relacionados em SBST: teste de programas orientados a objeto.

Abordagem	*Técnica de Otimização*	*Contexto de Teste*
Lefticaru e Ipate [101]	AG	seq. de chamadas
McMinn e Holcombe[113]	abordagem da cadeia	seq. de chamadas
Liaskos e Roper [105]	AG e SIA	seq. de chamadas
Tonella [127]	AG	seq. de chamadas
Baresi et al. [20]	AE híbrido	seq. de chamadas
Arcuri e Yao [15]	algoritmo memético	seq. de chamadas

Também pode-se encontrar trabalhos que utilizam otimização multiobjetivo no contexto de teste com diferentes propósitos. Oster e Saglietti [119] geram casos de teste para maximizar a cobertura de código minimizando o número de casos de teste requerido para programas procedimentais e orientados a objeto. Lima Pinto e Vergilio [121] propõem uma abordagem para o teste de programas orientados a objeto, levando-se em conta a

cobertura do critério de teste, habilidade de revelar falhas e tempo de execução. Carvalho et al. [32] obtêm um modelo de previsão de falhas com uma abordagem multiobjetivo, para o teste de classes e métodos. Lakhotia et al. [98] introduzem uma abordagem multiobjetivo para a cobertura de ramos e consumo de memória dinâmica de programas. Yoo and Harman [150] realizam a seleção de casos de teste para o teste de regressão combinando cobertura, custo e histórico de falhas. Maia et al. [107] propõem uma abordagem para priorizar casos de teste com base na sua volatilidade, complexidade e importância. Cabral et al. [31] determinam a ordem para o teste de integração de acordo com medidas de acoplamento. Assunção et al. [14] investigam quatro medidas de acoplamento para o teste de integração: atributos, métodos, número de tipos de retornos distintos e número de tipos de parâmetros distintos. Wang et al. [140] realizam alocação de recursos de teste, considerando a confiabilidade do sistema, custos dos testes e recurso total requerido nos testes. Langdon et al. [100] utilizam a otimização multiobjetivo no contexto de teste de mutação em que mais de uma alteração pode ser considerada. Outras abordagens multiobjetivo para a geração de sequências de teste são os trabalhos de Baresi et al. [20] e Baresel et al. [18], citados anteriormente.

Tabela 3.3: Trabalhos relacionados em SBST: teste com otimização multiobjetivo.

Abordagem	*Técnica de Otimização*	*Contexto de Teste*
Oster and Saglietti [119]	NSGA	teste estrutural
Carvalho et al. [32]	MOPSO-N	teste estrutural
Lima Pinto e Vergilio [121]	NSGA-II	teste estrutural
Lakhotia et al. [98]	NSGA-II	teste estrutural
Yoo and Harman [150]	NSGA-II	teste de regressão
Maia et al. [107]	MOCell, NSGA-II e SPEA2	priorização de teste
Cabral et al. [31]	colônia de formigas de Pareto	teste de integração
Assunção et al. [14]	NSGA-II e SPEA2	teste de integração
Wang et al. [140]	HaD-MOEA	alocação de recurso
Langdon et al. [100]	PG multiobjetivo	teste de mutação

Uma das diferenças da abordagem proposta com relação aos trabalhos que geram casos de teste a partir de MEFE [57, 93, 92] é que o algoritmo evolutivo utilizado neste trabalho gera sequências de eventos de entrada, ao invés de gerar diretamente os caminhos de transições. Os parâmetros de entrada também são obtidos durante a geração e não após o processo evolutivo. Assim nesses trabalhos ainda existe a possibilidade dos caminhos serem infactíveis, mesmo que o objetivo seja buscar caminhos factíveis. Na abordagem proposta MOST, esse problema é evitado utilizando-se um modelo executável que considera todos os dados envolvidos na MEFE para produzir os caminhos de transições. Além disso, o tamanho da sequência é determinado automaticamente

pelo algoritmo, assim como em alguns trabalhos que geram sequências de chamadas de métodos [127, 20, 15], não sendo necessário estabelecer a priori esse valor, ao contrário de trabalhos anteriores [57, 93, 92, 101, 113, 105, 18]. A fim de atender as necessidades do teste baseado em MEFE foi desenvolvido o algoritmo M-GEO$_{vsl}$, que além de ser capaz de gerar soluções de tamanho variável, também pode lidar com múltiplos objetivos. A maioria dos trabalhos utiliza os algoritmos genéticos como técnica de otimização. Seguindo uma das direções de pesquisa em SBST [77, 81], MOST é uma abordagem multiobjetivo que aborda um contexto de teste diferente dos trabalhos existentes [119, 32, 121, 98, 150, 107, 31, 14, 140, 100, 18].

3.5 Considerações Finais

Foram apresentados os conceitos de SBST, a estrutura genérica de um algoritmo evolutivo e os princípios de uma otimização multiobjetivo. Também foram descritos trabalhos em SBST relacionados com a abordagem proposta.

Embora existam diversas técnicas de otimização, não há uma regra geral que diga que uma técnica é melhor que todas as outras. A eficiência de uma técnica de otimização é dependente do tipo de problema em questão e uma determinada técnica é mais apropriada para um certo tipo de problema [143]. O desempenho da técnica está diretamente relacionado à qualidade da função objetivo, avaliando-se adequadamente as informações do processo de otimização, a fim de direcionar a busca a regiões mais promissoras. Dessa maneira, a investigação de novas técnicas de otimização, juntamente com a definição de representação e da função objetivo, para o problema de geração de casos de teste é de grande interesse. No próximo capítulo, é apresentado o algoritmo evolutivo multiobjetivo proposto, M-GEO$_{vsl}$, para lidar com as particularidades de geração de casos de teste a partir de MEFE.

Capítulo 4

Algoritmo Evolutivo Multiobjetivo Proposto

O problema de geração de sequências de teste motivou o desenvolvimento do M-GEO$_{vsl}$. O algoritmo é uma implementação multiobjetivo do algoritmo de Otimização Extrema Generalizada (em inglês, *Generalized Extremal Optimization*, GEO) [46, 47] capaz de lidar com tamanhos variáveis de cadeia de caracteres, tornando possível a geração de soluções com diferentes tamanhos. Como uma abordagem multiobjetivo, o algoritmo pode encontrar soluções que atendam a diversos objetivos simultaneamente. Neste capítulo são apresentados os fundamentos teóricos que originaram o GEO (Seção 4.1) e, em seguida, os passos do algoritmo M-GEO$_{vsl}$ (Seção 4.2) para um problema multiobjetivo genérico em que o tamanho da solução pode variar.

4.1 Origem do GEO

O algoritmo GEO é uma metaheurística de otimização global desenvolvida como uma generalização do método da Otimização Extrema (em inglês, *Extremal Optimization*, EO) [25], com o objetivo de tornar sua implementação independente do tipo de problema que está sendo atacado. O algoritmo EO foi inspirado em um modelo simplificado de seleção natural, proposto por Bak e Sneppen [17], para mostrar evidências da presença de Criticalidade Auto-Organizada (em inglês, *Self-Organized Criticality*, SOC) em ecossistemas naturais.

A teoria do SOC propõe que sistemas complexos com muitos elementos que interagem entre si evoluem naturalmente para um estado crítico onde uma simples mudança em um de seus elementos produz perturbações que podem atingir qualquer número de elementos do sistema. A probabilidade de ocorrer uma perturbação de tamanho s é descrita por uma lei de potência na forma:

$$P(s) \sim s^{-\tau} \tag{4.1}$$

onde τ é um parâmetro positivo. Nota-se que perturbações de menor tamanho ocorrem mais frequentemente que perturbações grandes, embora perturbações tão grandes quanto o sistema possam ocorrer com uma probabilidade não desprezível.

Essa teoria tem sido utilizada para explicar o comportamento de sistemas complexos em áreas como geologia (e.g. frequência de terremotos), economia (e.g. mercado financeiro) e biologia (e.g. evolução das espécies) [16].

Bak e Sneppen [17] desenvolveram um modelo simplificado para mostrar que um ecossistema possui características de SOC na evolução das espécies. As espécies que formam o ecossistema são representadas lado a lado e possuem uma relação de vizinhança. Na Figura 4.1 são mostradas n espécies, sendo que e_n é vizinha de e_{n-1} e e_1, e_1 é vizinha de e_n e e_2, e assim sucessivamente. Para cada espécie é associado um índice de adaptação no domínio $[0, 1]$. Aquela com pior índice de adaptação é considerada a espécie menos adaptada e o processo de evolução força esta e suas vizinhas a mudar. A ideia é que as vizinhas precisam se adaptar a um novo competidor local. A mudança é feita atribuindo-se aleatoriamente novos valores de aptidão a essas espécies, podendo ocasionar a evolução ou extinção de qualquer uma delas, mesmo que os valores de aptidão não sejam melhores que os anteriores. Uma vez que a espécie menos adaptada é constantemente forçada a mudar, a aptidão média do ecossistema aumenta e, após algumas iterações, todas as espécies apresentam um índice de adaptabilidade acima de um certo valor, denominado de valor crítico. Eventualmente os valores de aptidão de algumas espécies podem ficar abaixo do valor crítico por meio de avalanches (pertubações), cuja probabilidade de ocorrência também segue uma lei de potência na forma dada pela Equação 4.1. Uma técnica de otimização baseada nesse modelo poderia evoluir soluções rapidamente, sistematicamente modificando as espécies menos adaptadas e, ao mesmo tempo, poderia investigar regiões diferentes do espaço de busca através das avalanches, escapando de mínimos locais[1].

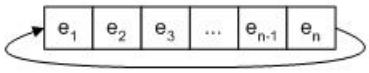


Figura 4.1: População de espécies no modelo de Bak e Sneppen.

Boettcher e Percus [25] propuseram o método EO inspirado no modelo de Bak-Sneppen com o objetivo de tratar problemas difíceis de otimização combinatória. O índice de adaptação é associado a cada variável de projeto (i.e. elemento a ser otimizado) e não à

[1]Mínimos locais são pontos que minimizam o valor de uma função, mas podem não ser o menor valor possível da função.

configuração do conjunto de variáveis. Entretanto, foi observado que uma definição geral para o índice de adaptação para as variáveis individuais pode se mostrar ambígua ou mesmo impossível, sendo necessária uma nova definição do índice de adaptação para cada novo problema de otimização abordado pelo método. Isso motivou o desenvolvimento do algoritmo GEO para resolver esse problema e possibilitar sua aplicação a uma ampla categoria de problemas de otimização contínua ou combinatória, que possuam espaço de busca complexo, inclusive não convexo ou disjunto, e que incluem variáveis contínuas, discretas, inteiras ou uma combinação destas.

Pode-se citar como problemas de otimização solucionados com o GEO: *i*. projeto de tubos de calor [50, 51, 53, 52, 135]; *ii*. projeto térmico de satélites [68, 69]; *iii*. constelações de satélites [67]; *iv*. perfis aerodinâmicos de planadores [43, 44]; *v*. identificação de propriedades óticas de materiais [45]; *vi*. otimização estrutural [49, 48]; *vii*. teste de software [3, 2, 4, 5, 30, 29, 144, 145, 146, 148, 149, 147]. No contexto de teste de software, existem iniciativas que aplicam o GEO para o teste estrutural [3, 2, 4, 5, 30, 29], utilizando-se diferentes implementações do algoritmo e diferentes funções objetivos [27, 106]. Neste trabalho, o algoritmo gera casos de teste a partir de modelos de estados, e não do código fonte de programas como nos trabalhos anteriores.

O algoritmo GEO pode ter diferentes implementações, como por exemplo: GEO canônico, GEO$_{var}$, GEOPar-1 e M-GEO. No GEO$_{var}$ [46, 47], as etapas de atribuição do valor de adaptação e da mutação são diferentes do GEO canônico. Quando o custo computacional da avaliação da função objetivo é expressivo, o GEOPar-1 [66] pode ser utilizado para realizar uma otimização paralela, ao invés de monoprocessada. Como uma versão multiobjetivo, o M-GEO [66] pode otimizar simultaneamente um conjunto de objetivos baseando-se nos conceitos de otimização de Pareto. Neste trabalho, é proposto o M-GEO$_{vsl}$ que estende o M-GEO a fim permitir a geração de soluções de tamanho variável. Os passos do algoritmo M-GEO$_{vsl}$ são descritos na próxima seção.

4.2 M-GEO$_{vsl}$: Algoritmo

Além de realizar uma otimização multiobjetivo, a principal característica do M-GEO$_{vsl}$ é a capacidade de gerar soluções com diferentes tamanhos. Como uma versão do GEO, também pode ser aplicado a problemas com miscelânea de variáveis reais, inteiras ou discretas.

4.2.1 Codificação da Solução

Nos métodos GEO, as espécies são dispostas ao longo de uma linha e um valor de adaptação é atribuído a cada espécie, analogamente ao modelo de Bak e Sneppen [17].

A população consiste na cadeia de espécies que codificam as variáveis de projeto. No GEO canônico, a população é uma sequência binária em que cada bit representa uma espécie. Cada variável de projeto é codificada por um conjunto de bits. Na Figura 4.2 são mostradas duas variáveis x_0 e x_1 codificadas em uma cadeia representada por quatro espécies no GEO, que é similar a um cromossomo binário que representa um indivíduo em um algoritmo genético (AG) canônico.

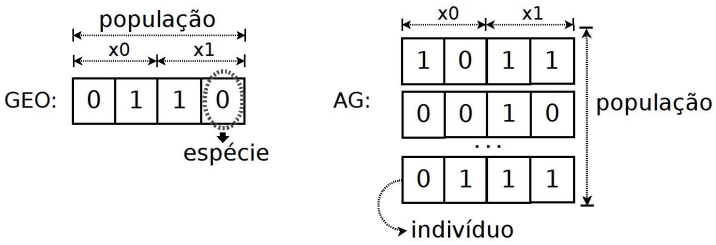


Figura 4.2: População: GEO (4 espécies e 2 variáveis de projeto) × algoritmo genético (n indivíduos com 2 variáveis de projeto).

A população no M-GEO segue a mesma codificação binária do GEO. Já no M-GEO$_{vsl}$ é utilizada uma codificação discreta, em que cada variável de projeto é tratada como uma espécie. Por exemplo, na Figura 4.3, a população com 6 variáveis de projeto tem 6 espécies no M-GEO$_{vsl}$, enquanto que há 18 espécies no M-GEO, considerando que cada variável de projeto é codificada com 3 bits.

A fim de permitir soluções com diferentes tamanhos durante o processo evolutivo, a população no M-GEO$_{vsl}$ consiste de duas partes (cadeias de espécies): uma com tamanho variável e outra com tamanho fixo. Na parte variável, a primeira espécie determina o tamanho dessa cadeia em um determinado momento do processo evolutivo. Cada elemento da cadeia variável possui o mesmo domínio de valores. As espécies da parte fixa podem ter domínio de valores diferentes e são opcionais, representando as variáveis de projeto do problema que não precisam de uma representação dinâmica de tamanho.

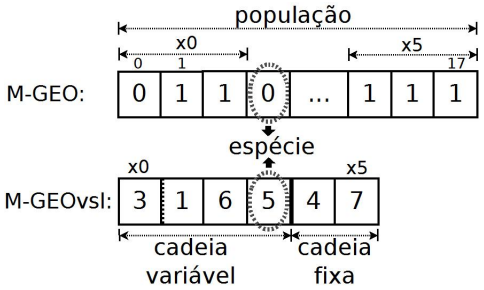


Figura 4.3: População: M-GEO × M-GEO$_{vsl}$.

4.2.2 Operador de Mutação

Para modificar as espécies no GEO, existe apenas o operador de mutação. Cada espécie é mutada de acordo com seu domínio. No GEO canônico ou M-GEO, a espécie pode assumir o valor 0 ou 1. A espécie no M-GEO$_{vsl}$ pode ser codificada por inteiros ou reais.

No M-GEO$_{vsl}$, um caso especial é a mutação da primeira espécie da parte variável da população. Duas situações devem ser consideradas quando essa espécie é mutada: o novo valor aumenta ou diminui o valor atual. No primeiro caso, novas espécies são adicionadas ao final da cadeia variável com valores aleatórios. No segundo caso, as espécies excedentes são ignoradas. Através dessas mutações, o algoritmo pode possuir populações com diferentes tamanhos. Mutações em ambas as partes da população são mostradas na Figura 4.4: *a*) mutação na cadeia variável: o valor da primeira espécie dessa parte é mutado de 3 para 4, assim uma espécie é adicionada nessa cadeia, e de 3 para 2, a quarta espécie é ignorada; *b*) mutação na cadeia fixa: uma espécie dessa cadeia é mutada para outro valor em seu domínio.

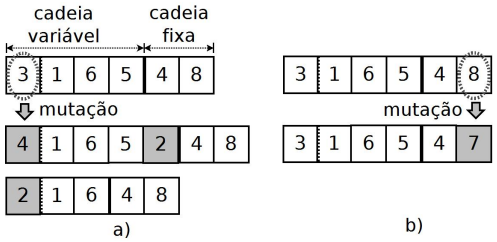


Figura 4.4: Mutações no M-GEO$_{vsl}$.

4.2.3 Processo Evolutivo

Uma visão geral do M-GEO e M-GEO$_{vsl}$ é apresentada na Figura 4.5 sob a forma de um fluxograma. As diferenças estão relacionadas ao tratamento da população pelos algoritmos, como na inicialização e mutação da população, mas os passos são os mesmos para ambos os algoritmos. Cada passo do M-GEO$_{vsl}$ é melhor descrito a seguir.

1. Inicialize a população: para a parte variável da população, gere um valor inteiro aleatório R para a primeira espécie e inicialize aleatoriamente uma cadeia de tamanho R; e para a parte fixa, se existir, inicialize aleatoriamente uma cadeia de tamanho S. Assim a população $\vec{x}$ consiste de $R+1$ espécies da parte variável, mais S espécies da parte fixa: $\vec{x} = x_0, x_1, \ldots, x_R, x_{R+1}, \ldots, x_{R+S}$.

2. Mute cada espécie x_i para outro valor em seu domínio (x_i'), uma a cada momento, gerando $\vec{x_i} = x_0, \ldots, , x_i', \ldots, x_{R+S}$. Todas as funções objetivo avaliam $\vec{x_i}$, calculando

$F(\vec{x_i}) = [F_1(\vec{x_i}), \ldots, F_{nf}(\vec{x_i})]$, onde nf é o número de funções objetivo. Após essa avaliação, o valor da espécie x_i retorna ao seu valor original. Esse processo é repetido para todas as espécies. Desse modo, todas as nf funções objetivo avaliam todas $\vec{x_i}$, onde $i \in \{0, 1, \ldots, R + S\}$. A aproximação da fronteira de Pareto é verificada e atualizada com $\vec{x_i}$, conforme necessário para conter apenas soluções não-dominadas, e é mantida em um arquivo separado. O conceito de dominância é apresentado na Seção 3.3.

3. Escolha aleatoriamente uma função objetivo F_c, onde $c \in \{1, \ldots, nf\}$ e nf é o número de funções objetivo.

4. Associe um índice de adaptação a cada espécie x_i como $\Delta_i = F_c(\vec{x_i}) - \rho$, onde ρ é um valor de referência (e.g., zero). A aptidão indica o ganho (ou perda) relativo à mutação da espécie, comparado com ρ.

5. Ordene as espécies de acordo com seu índice de adaptação. A primeira posição ($k = 1$) da ordenação indica a espécie menos adaptada. Para um problema de minimização, os menores valores de Δ_i estão nas menores posições da ordenação. Se duas ou mais espécies possuem a mesma aptidão, ordene-as aleatoriamente com distribuição uniforme.

6. Escolha com probabilidade uniforme uma espécie candidata x_i para mutar. Calcule $P_i(k) = k^{-\tau}$, onde k é a posição da espécie x_i no vetor ordenado e τ é um parâmetro livre configurado pelo usuário que controla o determinismo da busca. Gere um número aleatório RAN com distribuição uniforme no intervalo de [0,1]. Se $P_i(k) \leq RAN$, a espécie é confirmada para mutar, senão uma nova espécie é escolhida. Mude a população atual $\vec{x}$ para $\vec{x_i}$ obtida no Passo 2. Quando $\tau = 0$, qualquer variável possui a mesma probabilidade para ser mutada. Por outro lado, quanto maior o valor de τ, mais determinística a busca se torna para mutar a variável menos adaptada. Na prática, os valores usados para τ que maximizam a eficiência do algoritmo têm sido entre [1,5].

7. Verifique se o critério de parada é alcançado. A condição de parada é o número máximo de avaliações de todas as funções objetivo.

8. Verifique se a população deve reinicializar. A condição para uma reinicialização é $numAval = int(maxNumAval/numReInit)$, onde $numAval$ é o número atual de avaliações de todas as funções objetivo, $maxNumAval$ o número máximo de avaliações das funções objetivo e $numReInit$ o número de reinicializações desejado. Em caso de reinicialização, o algoritmo retorna ao Passo 1 e uma nova população

é gerada aleatoriamente, mantendo-se as soluções da fronteira de Pareto; senão o algoritmo retorna ao Passo 2.

9. Retorne o conjunto de Pareto e a fronteira de Pareto.

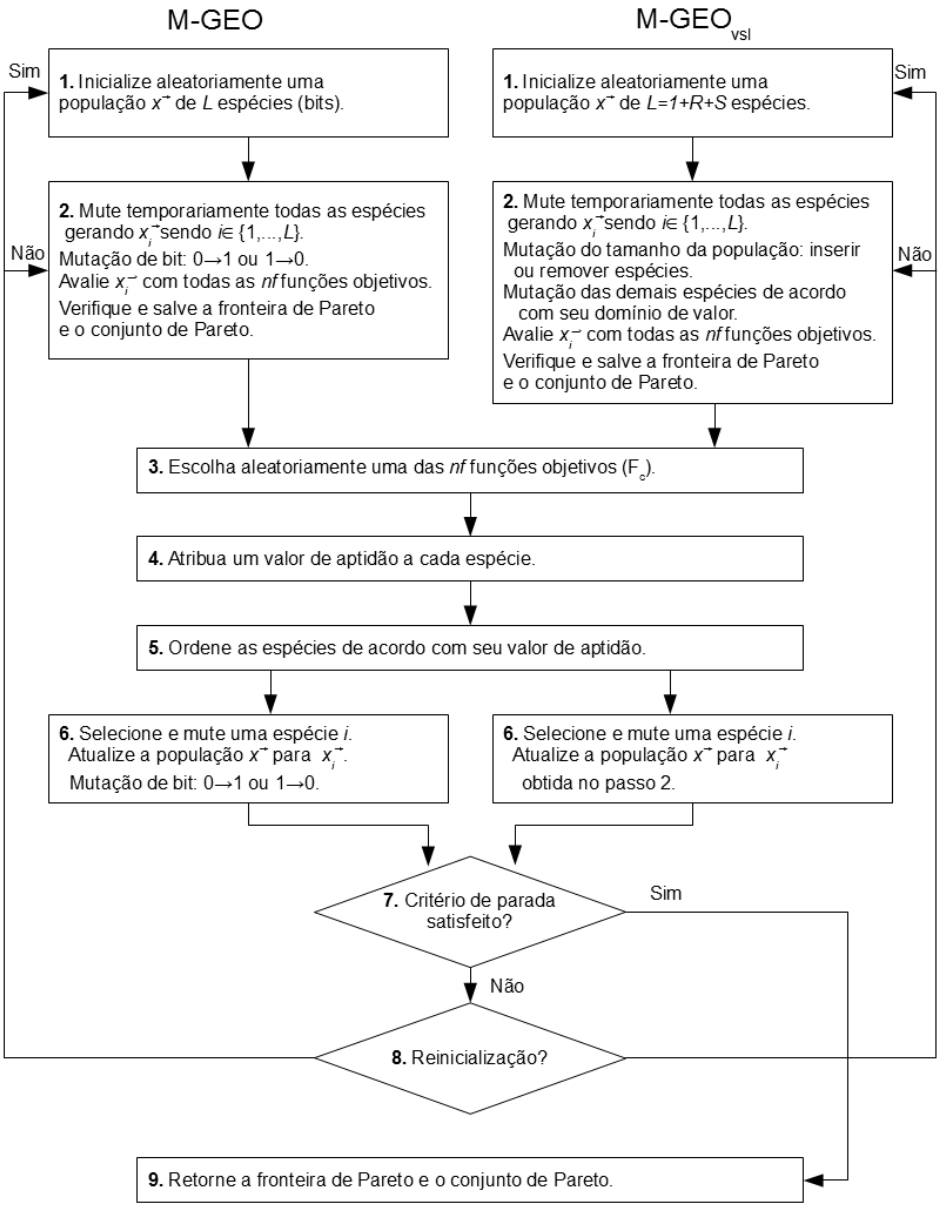


Figura 4.5: Processo Evolutivo: M-GEO × M-GEO$_{vsl}$.

Para ilustrar os passos do algoritmo, considere o problema de minimizar as seguintes funções objetivo com relação a um vetor de inteiros $\vec{x}$ de tamanho n e um inteiro y.

$$Minimizar\ \vec{x} : F_1(\vec{x}, y) = \sum_{j=1}^{n} x_j^y \tag{4.2}$$

$$F_2(\vec{x}, y) = \sum_{j=1}^{n} (x_j - 2)^y \tag{4.3}$$

$$Sujeito\ a : 0 \leq \vec{x} \leq 100 \tag{4.4}$$

$$Onde : \vec{x} = \{x_1, x_2, \ldots, x_n\} \tag{4.5}$$

Um simples exemplo numérico é mostrado na Tabela 4.1 para esse problema. Nesse caso, a população representa um vetor $\vec{x}$ (cadeia variável) e o número y (cadeia fixa). Primeiramente, um vetor de inteiros ($\vec{x}$) de tamanho 4 e um valor inteiro (y) são aleatoriamente gerados, assim a população possui 6 espécies nesse momento. No segundo passo, todas as espécies são mutadas temporiamente, uma de cada vez, para ordenar a aptidão das espécies. Note que a primeira espécie, que representa o tamanho do vetor, é mutado de 4 para 6. Assim dois valores inteiros são escolhidos aleatoriamente para completar o vetor. É importante observar que antes da próxima espécie mutar, a espécie anterior retorna a sua condição original. Cada nova configuração é avaliada por ambas as funções objetivo, obtendo-se uma solução candidata p à fronteira de Pareto. Se p é uma solução não-dominada, então p é incluída na aproximação da fronteira de Pareto e as soluções dominadas por p são removidas. No exemplo, a função objetivo F_2 é escolhida no Passo 3. Em seguida, a aptidão das espécies é determinada, calculando-se Δ_i com $\rho = 0$. No próximo passo, todas as espécies são ordenadas pelos seus valores de aptidão em relação a função objetivo escolhida F_2. Note que a posição de ordenação da primeira espécie é $k = 6$ e a última é $k = 1$. Isso indica que a primeira espécie é melhor adaptada que a última, uma vez que ao mutar a primeira espécie, F_2 melhora menos que mutando-se a última espécie.

No Passo 6, uma espécie é selecionada para mutar de acordo com a pressão de τ sobre a ordenação. No exemplo, $\tau = 1.0$ é utilizado e a terceira espécie com posição $k = 3$ (não é a menos adaptada) é escolhida para mutar. A mutação é confirmada com $RAN = 0.2$ pois a probabilidade $P(3) = \frac{1}{3} = 0.33$ é maior que RAN. Desse modo, o tamanho da população permanece com 6 espécies. No entanto, se a primeira espécie fosse selecionada para mutar, o tamanho da população aumentaria para 8 espécies. Quando o número máximo de avaliações de todas as funções objetivo é alcançado, o conjunto de Pareto e a fronteira de Pareto são retornados. Caso contrário, verifica-se a necessidade de uma reinicialização. Reinicialização é um recurso comum em abordagens multiobjetivo para evitar ótimos locais.

Tabela 4.1: Exemplo numérico dos passos do M-GEO$_{vsl}$.

Passo	População	F_1	F_2	F_c	Δ_i	k
1	*Inicialize a população*					
	$\vec{x} = 4 \mid 3\ 8\ 10\ 7 \parallel 2$	222	126	–	–	–
2	*Calcule F_1 e F_2 para cada mutação de espécie*					
	$\vec{x_0} = \mathit{6} \mid 3\ 8\ 10\ 7\ \mathit{9}\ \mathit{1} \parallel 2$	304	176	–	–	–
	$\vec{x_1} = 4 \mid \mathit{1}\ 8\ 10\ 7 \parallel 2$	214	126	–	–	–
	$\vec{x_2} = 4 \mid 3\ \mathit{3}\ 10\ 7 \parallel 2$	167	91	–	–	–
	$\vec{x_3} = 4 \mid 3\ 8\ \mathit{2}\ 7 \parallel 2$	126	62	–	–	–
	$\vec{x_4} = 4 \mid 3\ 8\ 10\ \mathit{5} \parallel 2$	198	110	–	–	–
	$\vec{x_5} = 4 \mid 3\ 8\ 10\ 7 \parallel \mathit{1}$	28	20	–	–	–
3	*Selecione uma função objetivo*					
	$\vec{x} = 4 \mid 3\ 8\ 10\ 7 \parallel 2$	222	126	F_2	–	–
4	*Encontre a aptidão de cada espécie*					
	$\vec{x_0} = \mathit{6} \mid 3\ 8\ 10\ 7\ \mathit{9}\ \mathit{1} \parallel 2$	304	176	F_2	176	–
	...					
	$\vec{x_5} = 4 \mid 3\ 8\ 10\ 5 \parallel \mathit{1}$	28	20	F_2	20	–
5	*Ordene as espécies de acordo com sua aptidão*					
	$\vec{x_5} = 4 \mid 3\ 8\ 10\ 5 \parallel \mathit{1}$	28	20	F_2	20	1
	...					
	$\vec{x_0} = \mathit{6} \mid 3\ 8\ 10\ 7\ \mathit{9}\ \mathit{1} \parallel 2$	304	176	F_2	176	6
6	*Escolha uma espécie candidata à mutação*					
	$\vec{x_2} = 4 \mid 3\ \mathit{3}\ 10\ 7 \parallel 2$	167	91	F_2	91	3
7	*Verifique se o critério de término é alcançado*					
8	*Verifique se é necessário uma reinicialização*					
9	*Retorne o conjunto de Pareto e a fronteira de Pareto*					

4.3 Considerações Finais

Devido às particularidades do M-GEO$_{vsl}$, é possível obter soluções de tamanhos variáveis utilizando múltiplos objetivos para guiar a busca. No próximo capítulo é apresentada a aplicação do M-GEO$_{vsl}$ na abordagem proposta de geração de casos de teste a partir de MEFE. Sequências de eventos de entrada, juntamente com os dados envolvidos, são geradas visando a minimizar o tamanho do caso de teste para a cobertura de uma transição alvo.

Capítulo 5

Abordagem de Teste Proposta

A abordagem de teste proposta neste trabalho, denominada MOST (*Multi-Objective Search-based Testing approach from EFSM*) combina o conceitos de teste baseado em modelo (Capítulo 2) e teste baseado em busca (Capítulo 3).

Os principais passos da abordagem MOST são (Figura 5.1):

1. modelar uma MEFE M a partir da especificação;
2. obter o modelo executável de M e validar M;
3. instrumentar o modelo executável de M;
4. analisar as dependências de M;
5. gerar sequências de entrada e avaliar o caminho de transições percorrido;
6. retornar os casos de teste.

Uma MEFE M, que representa o comportamento do sistema em teste, é manualmente obtida pela especificação do sistema e é modelada de acordo com a Seção 2.1. Na abordagem proposta uma versão executável da MEFE é utilizada, ao invés de considerar somente a estrutura estática da MEFE para gerar os casos de teste. O modelo executável implementa o comportamento da MEFE em uma linguagem de programação, sendo utilizado para produzir dinamicamente o caminho de transições disparado por uma sequência de eventos de entrada, considerando tanto o aspecto de controle quanto de dados do modelo. As características de um modelo executável são apresentadas na Seção 5.1. Uma vantagem do uso do modelo executável é a possibilidade de validar o modelo antes da geração dos casos de teste, observando-se o comportamento do modelo através de sua execução. Visto que os casos de teste são derivados a partir da MEFE, é importante a validação do modelo para verificar se o modelo está de acordo com a especificação do sistema em teste.

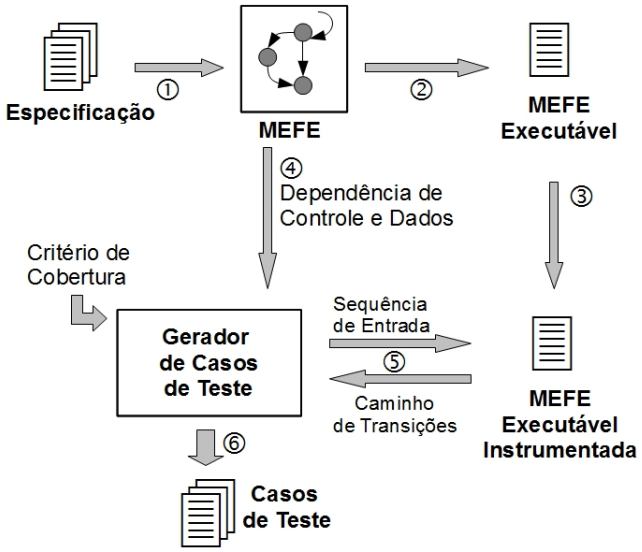


Figura 5.1: Abordagem MOST.

Uma vez validado, o modelo é preparado para a geração de casos de teste, instrumentando o código do modelo executável para produzir o caminho de transições disparado por uma sequência de entrada. Como gerador de casos de teste, é utilizado um algoritmo evolutivo, em particular, o M-GEO$_{vsl}$ (Capítulo 4). O algoritmo gera as sequências de entrada incluindo os valores dos parâmetros envolvidos no modelo, e avalia o caminho produzido pelo modelo executável. Na Seção 5.3, apresenta-se como o M-GEO$_{vsl}$ é utilizado, mostrando a codificação das soluções e as definições das funções objetivo para este problema. Na Seção 5.2, é apresentado como as informações da análise de dependência do modelo (Seção 2.2) são utilizadas nas funções objetivos. É importante notar que as dependências são obtidas em um passo anterior à geração de casos de teste. Em MOST, os casos de teste obtidos levam em consideração a cobertura da transição alvo e o tamanho da sequência de entrada gerada.

5.1 Modelo Executável

O principal objetivo do uso de um modelo executável é obter de forma dinâmica o caminho de transições disparado por uma sequência de entrada durante a execução do modelo. A ideia é controlar a habilitação das transições de acordo com os valores atuais dos dados

envolvidos no modelo.

O modelo executável deve se comportar de acordo com a Seção 2.1. Uma transição t_x é disparada apenas quando o evento de entrada correspondente i_x é recebido e a guarda g_x é satisfeita, considerando-se que o estado atual é q_x^s. Como resposta, a ação a_x é executada e o estado atual muda para q_x^t. A guarda utiliza os valores das variáveis e parâmetros de acordo com o histórico das transições que foram executadas anteriormente. Os valores desses dados podem ser modificados pelas ações das transições, por exemplo, em um comando de atribuição. O modelo permanece no mesmo estado e uma saída nula é produzida quando a guarda g_x é falsa ou um evento inesperado é recebido.

Como os aspectos de controle e de dados do modelo são considerados na sensibilização do caminho, o problema de geração de caminhos infactíveis é evitado.

O modelo executável pode ser obtido por qualquer ferramenta que dê apoio a modelos de estados da UML. O código fonte do modelo executável é instrumentado de modo a monitorar cada transição disparada e produzir o caminho percorrido ao final de sua execução.

5.2 Dependências do modelo

Dois conjuntos de transições são definidos na abordagem proposta baseados nas informações obtidas da análise de dependência: T_a e T_c. As informações desses conjuntos são usadas nas funções objetivo, descritas na Seção 5.3.2, para avaliar os casos de teste gerados. Percorrendo-se em direção para trás no grafo de dependência, $T_a(t_x)$ contém as transições que afetam t_x, incluindo t_x. $T_a(t_x)$ representa todas as transições em que t_x é diretamente ou indiretamente dependente de controle e/ou dados.

Definição 5.1 *Para uma determinada transição t_x, $T_a(t_x)$ contém t_x e as transições obtidas percorrendo-se em direção para trás a partir de t_x no grafo de dependência.*

Considerando-se o grafo de dependência da Figura 2.3 e a transição t_{18} como transição alvo, $T_a(t_{18}) = \{t_1, t_4, t_8, t_{17}, t_{18}\}$. As transições de T_a são indicadas com linhas em negrito na Figura 5.2. Na abordagem proposta, um caso de teste é gratificado quando as transições de $T_a(t_{18})$ são percorridas, direcionando a busca em direção à transição alvo t_{18}. Por exemplo, um caso de teste para cobrir t_{18} pode ser o caminho de transições $caminho = \{t_1, t_4, t_8, t_{18}\}$. Note que nem todas as transições de $T_a(t_{18})$ precisam ser parte do caso de teste.

Baseando-se em $T_a(t_{18})$, o conjunto $T_c(t_{18})$ pode ser obtido para identificar as transições críticas de t_{18}. O conceito de transição crítica estende a noção de nós críticos no teste baseado em código [112], no qual diz que ao sair deste nó, o caso de teste desvia do alvo.

Desse modo, o propósito é penalizar os casos de teste que passam por transições críticas. Uma transição crítica está relacionada com as transições irmãs de $T_a(t_{18})$.

Definição 5.2 *Uma transição t_c é **crítica** com relação a uma transição alvo t_x se $t_c \notin T_a(t_x)$ e $\exists t_a \in T_a(t_x)$ tal que o estado origem de t_c e t_a é o mesmo e existe um caminho π tal que $t_c \in \pi$ e $t_x \notin \pi$.*

Considerando-se t_{18} de M_1 como transição alvo, $T_c(t_{18}) = \{t_2, t_3, t_7, t_{10}, t_{19}, t_{20}, t_{23}\}$. As transições de T_c são indicadas com linhas tracejadas na Figura 5.2.

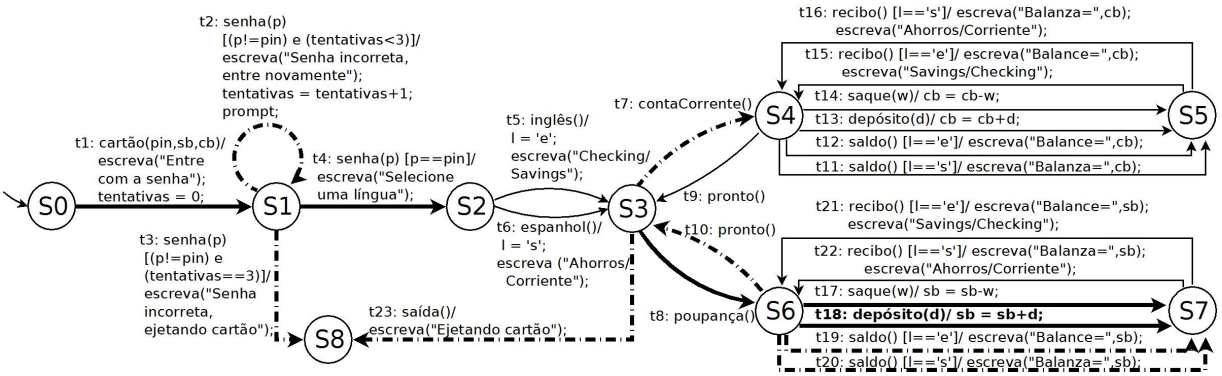


Figura 5.2: MEFE M_1: T_a (linha em negrito) e T_c (linha tracejada), considerando-se t_{18} como transição alvo.

Pode-se citar alguns trabalhos que também utilizam as informações de dependências em testes. A fim de cobrir transições de MEFE, Li e Wong [103] usam a análise de dependência para atribuir pesos às transições de acordo com o número de transições que dominam, ou seja, o número de transições que são cobertas se esta transição foi alcançada. No entanto, não é mencionado se foi utilizada técnica de otimização para gerar os casos de teste. McMinn et al. [112] utilizam as dependências de um programa para reduzir o espaço de busca de um algoritmo evolutivo no teste estrutural.

5.3 Gerador de Casos de Teste

Na abordagem proposta, a geração de casos de teste é reformulada como um problema de otimização. Os casos de teste são automaticamente gerados a partir de MEFE usando-se o M-GEO$_{vsl}$. O algoritmo gera uma sequência de entrada, composta por eventos de entrada e dados envolvidos nos eventos. O modelo executável recebe essa sequência como entrada e produz o caminho de transições percorrido pela sequência gerada (Figura 5.3). M-GEO$_{vsl}$ avalia o caminho de transições usando as funções objetivo F_1 e F_2 definidas

para o teste baseado em MEFE. Com base nas avaliações das funções objetivo, a sequência de entrada é incrementalmente modificada a fim de cobrir o critério de teste.

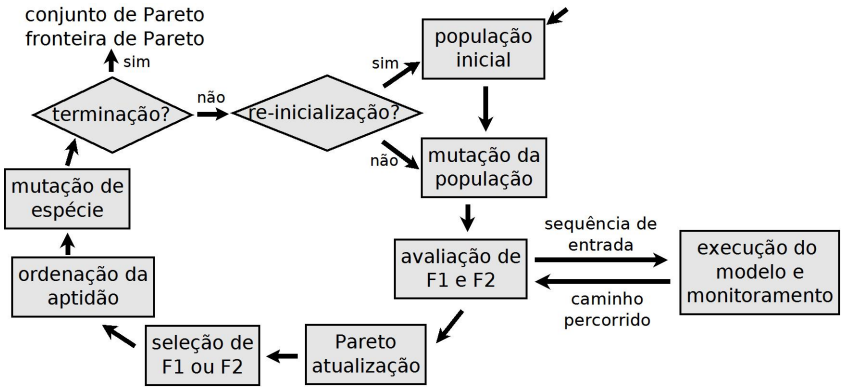


Figura 5.3: Processo evolutivo do M-GEO$_{vsl}$ para a abordagem MOST.

O tamanho da sequência de teste é um ponto a ser considerado no teste baseado em MEFE. Isso motivou o desenvolvimento e aplicação do M-GEO$_{vsl}$ (Capítulo 4) para gerar soluções com tamanhos variáveis. O Algoritmo 5.1 mostra como o M-GEO$_{vsl}$ é utilizado na abordagem MOST. As particularidades do algoritmo para o teste baseado em MEFE são apresentadas nas próximas seções: codificação da solução (Seção 5.3.1) e as duas funções objetivo definidas para guiar a busca (Seção 5.3.2).

5.3.1 Codificação da Solução

Para geração de casos de testes a partir de MEFE, a população em M-GEO$_{vsl}$ representa uma sequência de eventos de entrada e seus parâmetros. Como explicado na Seção 4.2.1, a população possui uma parte de tamanho variável e outra fixa. A parte variável contém a sequência de eventos de entrada, que constitui o aspecto de controle do caso de teste. A primeira espécie dessa parte determina o tamanho da sequência em um determinado momento do processo evolutivo. Assim, o tamanho da população é variável em relação ao tamanho da sequência de entrada. Já a parte fixa contém os parâmetros de todos os eventos de entrada do modelo, representando os dados do caso de teste. Cada parâmetro envolvido no modelo tem uma espécie na população. Na Figura 5.4 é ilustrada uma população considerando a MEFE M_1 do exemplo do caixa eletrônico (Figura 2.1).

Algoritmo 5.1 M-GEO$_{vsl}$ para a abordagem MOST.

Entrada: MEFE e transição alvo t

Saída: conjunto de Pareto e fronteira de Pareto

$R \leftarrow$ escolha aleatoriamente o tamanho da sequência de eventos de entrada
$S \leftarrow$ número de parâmetros envolvidos na MEFE
$\vec{x} \leftarrow x_0, x_1, \ldots, x_R, x_{R+1}, \ldots, x_{R+S}$ {inicialização da população}
enquanto not $maxNumAvaliacoes$ **faça**
  para cada espécie i de $\vec{x}$ **faça**
    $mut_i \leftarrow$ um valor do domínio de i
    $\vec{x_i} \leftarrow x_0, \ldots, mut_i, \ldots, x_{R+S}$
    $path \leftarrow$ obtenha o caminho de transições disparado por $\vec{x_i}$ {interação com o modelo executável}
    $F(\vec{x_i}) \leftarrow [F_1(\vec{x_i}), F_2(\vec{x_i})]$
    se $t \in path$ e $\vec{x_i}$ é não-dominada na fronteira de Pareto **então**
      atualize a fronteira de Pareto com $\vec{x_i}$
    fim se
    $\vec{x} \leftarrow x_0, \ldots, x_i, \ldots, x_{R+S}$ {população retorna à configuração original}
  fim para
  $F_c \leftarrow$ escolha aleatoriamente uma função objetivo
  para cada espécie i de $\vec{x}$ **faça**
    $\Delta_i \leftarrow F_c(\vec{x_i}) - ref$ {atribuição do valor de aptidão}
  fim para
  ordenarAptidão($\vec{x}$) {primeira posição pertence à espécie menos adaptada}
  repita
    $i \leftarrow$ escolha com probabilidade uniforme uma espécie candidata à mutação
    $k \leftarrow$ posição de i na ordenação
    $P_i(k) \leftarrow k^{-\tau}$
  até $P_i(k) \geq RAN$
  $\vec{x} \leftarrow \vec{x_i}$ {mutação}
  se not $maxNumAvaliacoes$ **and** $reinicializacao$ **então**
    $\vec{x} \leftarrow x_0, x_1, \ldots, x_R, x_{R+1}, \ldots, x_{R+S}$ {reinicialização da população}
  fim se
fim enquanto
retorne conjunto de Pareto e fronteira de Pareto

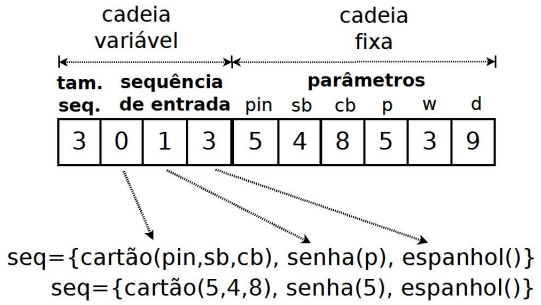


Figura 5.4: População no M-GEO$_{vsl}$ para geração de teste baseado em MEFE.

Assume-se que os parâmetros com nomes idênticos são os mesmos; dessa maneira, o número de parâmetros é constante. Por exemplo, se uma sequência contém os eventos $e_j(\ldots, p, \ldots)$ e $e_k(\ldots, p, \ldots)$, o valor do parâmetro p é usado em ambos os eventos. Por essa razão, se for necessário que p tenha valores diferentes em e_j e e_k, pode-se renomear p de acordo com cada evento, p_j e p_k (Figura 5.5(a)). Nesse caso, a população teria uma espécie correspondente a p_j e p_k. Uma outra situação a ser considerada é quando um evento $e_i(p_1, p_2)$ pode ocorrer em diferentes estados com diferentes valores, tal como na leitura de valores de um sensor em diferentes momentos. Uma alternativa para esse caso seria criar novos eventos de entrada com seus parâmetros análogos a $e_i(p_1, p_2)$ (Figura 5.5(b)): $e_{i1}(p_{11}, p_{21})$ e $e_{i2}(p_{12}, p_{22})$. Na população, o domínio dos eventos de entrada teria os eventos e_{i1} e e_{i2}, ao invés de e_i, e existiriam espécies para os parâmetros correspondentes.

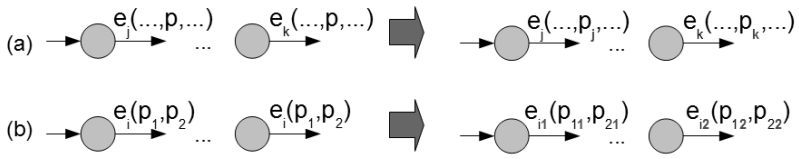


Figura 5.5: Renomeação de parâmetros.

5.3.2 Funções Objetivo

Duas funções objetivo foram definidas no M-GEO$_{vsl}$ para guiar a geração de teste a partir de MEFE: cobertura da transição alvo (F_1) e tamanho da sequência de entrada (F_2). As soluções geradas representam um balanço entre F_1 e F_2, visando-se a encontrar uma sequência de entrada de tamanho mínimo mas longa o suficiente para cobrir uma transição alvo.

$$Minimize : F_1 = AL + ND \quad (5.1)$$
$$F_2 = |seq| + (1 - 1.001^{-entradas_inoportunas}) \quad (5.2)$$

onde

$$AL = 2 * |T_a| - RW$$
$$ND = 1 - 1.001^{-d}$$
$$RW = \begin{cases} |T'_a| & \text{se não cobre } t; \\ |T'_a| + |T_a| & \text{se cobre } t. \end{cases}$$

A função objetivo F_1 é definida em termos de: nível de aproximação (AL) e distância normalizada (ND). Esses termos são bem conhecidos no teste baseado em código [111], no qual o nível de aproximação está relacionado com o número de estruturas de controle executado com sucesso para alcançar uma estrutura alvo. No ponto em que ocorre um desvio no fluxo de controle para executar a estrutura desejada, um valor de distância é calculado, usando-se os valores de variáveis e constantes envolvidos nos comandos condicionais do programa. A ideia é que a função objetivo receba melhores valores quando as soluções estão mais próximas da estrutura alvo em comparação com aquelas que desviam antes. Tradicionalmente, o valor da distância é normalizado no intervalo [0,1] [19, 141, 142], a fim de se evitar ótimo locais [111]. Uma fórmula para normalizar a distância d é: $1 - 1.001^d$ [113, 139, 83].

Neste trabalho, os termos AL e ND são modificados para considerar características da MEFE. F_1 foca a cobertura da transição alvo. Para seu cálculo são utilizados os conjuntos de transições T_a e T_c definidos na Seção 5.2. É importante notar que esses conjuntos são obtidos em um passo anterior ao processo evolutivo. Considerando-se um caminho de transições, o propósito é gratificar as transições que potencialmente podem afetar a transição alvo t usando-se $T_a(t)$ e penalizar aquelas transições que desviam de t usando-se $T_c(t)$.

O nível de aproximação AL mede quão próximo uma sequência de entrada está de alcançar um caminho que passa pela transição alvo. O valor de AL é minimizado em relação ao número de transições de T_a disparadas durante a execução do modelo, $|T'_a|$. Apenas quando a transição alvo é alcançada, a função objetivo é gratificada com $|T_a|$, a fim de claramente diferenciar essas soluções com aquelas que não cobrem com sucesso a transição alvo. Quando todas as transições de T_a são alcançadas, inclusive a transição alvo, obtém-se $RW = |T'_a| + |T_a|$. Nessa situação, é obtido o valor mínimo de AL (i.e, zero).

A distância normalizada ND calcula quão distante os dados de uma transição crítica estão de percorrer uma transição de T_a. Quando se passa por uma transição crítica t_c, significa que o caminho divergiu de uma transição $t_a \in T_a$. Assim analisam-se as transições com o mesmo estado origem de t_a, i.e. transições irmãs de t_a. Em uma MEFE, mais de duas transições podem ter o mesmo estado origem, ao contrário dos ramos verdadeiro e falso de um nó no teste estrutural. A fim de diferenciar, as transições críticas são divididas em relação aos seus eventos de entrada: *i*. os eventos de entrada de t_c e t_a são os mesmos mas as guardas são diferentes, ou *ii*. os eventos de entrada de t_c e t_a são diferentes. O Algoritmo 5.2 mostra como determinar T_c para uma determinada transição alvo t. De acordo com o tipo de transição crítica, a distância d é calculada. Na primeira situação, são utilizadas as funções de distância definidas por Tracey et al. [129] (Table 5.1). Por exemplo, se uma guarda $(x == y)$ precisa ser avaliada como verdadeira para alcançar a transição alvo, a função de distância é definida como $abs(x - y) + K$ para transições como o mesmo evento de entrada. O valor de K é uma constante positiva que sempre é adicionada se o termo não é verdadeiro. Para distinguir soluções com diferentes eventos de entrada, uma constante γ é aplicada para penalizar quando uma transição crítica é percorrida.

Algoritmo 5.2 Determinar as transições críticas (T_c).

Entrada: transição alvo t
Saída: $T_c(t)$

```
para ∀t_i ∈ T_a(t) faça
  S ← getIrmas(t_i)
  para ∀t_j ∈ S faça
    se t_j ∉ T_a então
      se evento(t_j) == evento(t_i) então
        adicioneTc(t_j, funcaoDistancia(t_i))
      senão
        adicioneTc(t_j, γ)
      fim se
    fim se
  fim para
fim para
retorne T_c(t)
```

Tabela 5.1: Funções de Distância.

Operador	Função de Distância d
Booleana	se $TRUE$ então 0 senão K
$a = b$	se $abs(a - b) = 0$ então 0 senão $abs(a - b) + K$
$a \neq b$	se $abs(a - b) \neq 0$ então 0 senão K
$a < b$	se $a - b < 0$ então 0 senão $(a - b) + K$
$a \leq b$	se $a - b \leq 0$ então 0 senão $(a - b) + K$
$a > b$	se $b - a < 0$ então 0 senão $(b - a) + K$
$a \geq b$	se $b - a \leq 0$ então 0 senão $(b - a) + K$
$\neg a$	Negação é propagada sobre a
$a \wedge b$	$d(a) + d(b)$
$a \vee b$	$min(d(a), d(b))$
a xor b	$d((a \wedge \neg b) \vee (\neg a \wedge b))$ $\equiv min((d(a) + d(\neg b)), (d(\neg a) + d(b)))$

A função objetivo F_2 busca minimizar o tamanho da sequência de entrada, devido ao custo do tempo de execução dos casos de teste. Um valor no intervalo [0,1] é adicionado ao tamanho da sequência, a fim de penalizar entradas inoportunas, visto que nenhuma transição é disparada com essas entradas. Isso permite reduzir o número de platôs formado no espaço das funções objetivo, que dificultam guiar a busca. O número de entradas inoportunas é normalizado, assim como ocorre com o valor de distância em F_1. Apenas não existe a penalização em F_2, quando a sequência de entrada não possui entradas inoportunas.

5.4 Considerações Finais

Neste capítulo foi apresentada a abordagem de teste MOST, que utiliza o algoritmo M-GEO$_{vsl}$ e um modelo executável para obter os casos de teste a partir de uma MEFE. Para a aplicação do M-GEO$_{vsl}$ neste contexto, foram apresentados como uma solução é codificada e quais as funções objetivo definidas para guiar a geração de casos de teste. No próximo capítulo são apresentados os resultados dos experimentos realizados para avaliar a abordagem proposta.

Capítulo 6

Experimentos

Foram realizados experimentos com diversos modelos a fim de avaliar a abordagem proposta MOST. Foram utilizados modelos da literatura como também modelos que representam aplicações reais (Seção 6.1). O objetivo da geração de casos de teste é cobrir cada transição do modelo, uma por vez, obtendo-se os dados necessários para percorrer um caminho que passe pela transição alvo.

Os experimentos visam a responder as seguintes questões:

Q_1**:** Qual o desempenho obtido da abordagem proposta na cobertura de transições no teste baseado em MEFE?

Q_2**:** Como guiar o testador para selecionar os casos de teste pela fronteira de Pareto?

Q_3**:** Qual o desempenho obtido da abordagem proposta em comparação com os resultados das abordagens existentes?

A questão Q_1 diz respeito à cobertura das transições alvo obtida com a aplicação da abordagem MOST. Também envolve os custos relacionados à geração medidos em termos de: número de avaliações das funções objetivo, tamanho do caso de teste e tempo de execução. Como os resultados produzidos pela abordagem MOST são baseados na otimalidade de Pareto, é necessário compreender a fronteira de Pareto para verificar o custo/benefício de cada caso de teste gerado. As questões Q_1 e Q_2 são tratadas na Seção 6.4.1.

A fronteira de Pareto também é utilizada para responder Q_3. É realizada uma análise de quais soluções são dominadas ou não pelas soluções obtidas por outra abordagem SBST que deriva casos de teste a partir de MEFE [93, 92] (Seção 6.4.2).

Devido à grande quantidade de informação, os dados dos experimentos estão disponíveis na página do grupo SED em: *http://www.sed.ic.unicamp.br/people-1/tyano/experiments*. Pode-se encontrar informação sobre cada modelo usado como estudo de caso, assim como os resultados dos experimentos.

6.1 Estudos de Caso

Os modelos utilizados como estudos de caso representam o comportamento de diferentes sistemas computacionais. A Tabela 6.1 apresenta algumas informações sobre os modelos, tais como número de estados, transições, eventos e parâmetros. A última coluna indica o valor de CCS (*Cyclomatic Complexity of State machine*) [136] dos modelos. CCS é uma adaptação das métricas que verificam a complexidade estrutural de modelos de estados, sendo dado por: $CCS = |T| + |I| + |A_G| + 2$, onde T é o conjunto de transições, I é o conjunto de eventos de entrada e A_G é o conjunto de expressões atômicas nas guardas.

Tabela 6.1: Modelos utilizados nos experimentos.

	Modelo	#Estados	#Transições	#Eventos	#Parâmetros	CCS
M_1 :	Caixa eletrônico 1	9	23	15	7	55
M_2 :	Pagamento em caixa	12	21	15	7	51
M_3 :	Controle de cruzeiro	5	17	9	3	47
M_4 :	Bomba de combustível	13	25	16	5	63
M_5 :	INRES 1	8	18	9	3	34
M_6 :	Elevador 1	6	12	11	0	28
M_7 :	Impressora	11	89	19	11	123
M_8 :	Máquina de venda	7	28	11	3	65
M_9 :	Segurança de voo	4	32	14	12	173
M_{10} :	Elevador 2	5	25	11	15	176
M_{11} :	Protocolo de transporte	6	21	13	14	57
M_{12} :	Caixa eletrônico 2	10	30	13	10	80
M_{13} :	INRES 2	4	16	7	2	37
M_{14} :	Serviço web 1	9	40	13	6	83
M_{15} :	Serviço web 2	26	235	57	29	376

As MEFEs são modelos obtidos da literatura, exceto os dois últimos modelos: M_1 a M_8 [13], M_9 e M_{10} [94], M_{11} [93], e M_{12} e M_{13} [92]. Os modelos M_{14} e M_{15} representam aplicações reais. Abaixo segue uma breve descrição de cada modelo:

- M_1: Especifica o comportamento de um sistema de caixa eletrônico [96], conforme descrito no Capítulo 2 (Figura 2.1). Provê as operações para saldo, depósito e saque em conta corrente ou poupança. Oferece a opção de menu em inglês e espanhol.

- M_2: Representa as ações para pagamento de compras em um caixa. Após calcular a soma total da compra, pode-se escolher o tipo de pagamento: dinheiro, cartão de crédito ou cheque. Para as duas últimas opções, são necessárias autenticações para realizar essas operações.

- M_3: Controla a velocidade de um cruzeiro para mantê-la constante. Existe um sensor de leitura da velocidade atual para informar se está entre a velocidade mínima e máxima. Também pode-se aumentar ou diminuir a velocidade do cruzeiro.

- M_4: Modela o comportamento para se utilizar uma bomba de combustível. Pode-se optar pelo tipo de combustível: regular, médio e super. O pagamento, que pode ser em dinheiro ou cartão de crédito, leva em consideração o tipo de combustível e se o carro foi lavado.

- M_5: Especifica o protocolo de comunicação INRES [87]. O protocolo é orientado a conexão composto por duas partes: uma parte que estabelece a conexão e envia dados, e a outra que recebe os dados e termina a conexão.

- M_6: Mostra diferentes estados de um elevador: quando a porta está aberta ou fechada e quando a porta está abrindo ou fechando. Existe um temporizador para manter a porta aberta por um certo tempo e um botão de interrupção que reinicia esse temporizador.

- M_7: Verifica cada caractere de um arquivo para impressão. Os caracteres podem ser números, símbolos e espaços em branco. Existe um contador para os caracteres "=>".

- M_8: Representa uma máquina de vendas de café e chá que aceita moedas de 25 centavos. Pode-se selecionar a bebida e o tamanho do copo entre grande e pequeno. A cada venda, verifica-se a disponibilidade do tipo de copo selecionado e, caso exista, atualiza-se o número de copos após a venda. Também é possível inserir copos na máquina.

- M_9: Monitora a cabine de um avião em termos de: vibração, pressão, temperatura e fumaça. De acordo com os valores desses fatores, o sistema pode estar no estado seguro, de alerta ou crítico. Algumas ações precisam ser tomadas dependendo da situação, tal como liberar as máscaras de oxigênio quando a pressão do ar estiver baixa.

- M_{10}: Representa um sistema de elevador para um prédio de três andares. Além do controle dos estados das portas do elevador, são verificados o peso, temperatura e fumaça. A fim de evitar que crianças utilizem o elevador, a carga da cabine precisa ser maior que 15 Kg para que o elevador funcione.

- M_{11}: Modela uma versão simplificada de um protocolo de transporte classe 2 [23]. O modelo representa o estabelecimento de conexão, transferência de dados, controle de fluxo e segmentação.

- M_{12}: Estende o modelo de caixa eletrônico descrito em [96], permitindo a operação de transferência, além de saldo, depósito e saque. O menu do sistema pode ser em inglês ou francês.

- M_{13}: Modela o protocolo de comunicação INRES [87]. Embora seja diferente do modelo M_5, também representa o estabelecimento de conexão, transferência de dados e término da conexão.

- M_{14}: Representa um serviço web que integra diversas fontes de informação, permitindo a interoperabilidade entre redes distintas e desenvolvimento em plataformas diferentes. Possibilita realizar consultas em diferentes bancos de dados ao mesmo tempo.

- M_{15}: Representa um serviço web que produz relatórios de acordo com uma classificação do dado de entrada. A quantidade e diversificação de informações no relatório são proporcionais ao tipo do dado de entrada.

6.2 Configuração dos Experimentos

Um passo anterior à geração de casos de teste em uma abordagem SBST é o ajuste da técnica de otimização baseada em busca para o problema a ser tratado. A eficiência do algoritmo M-GEO$_{vsl}$ está relacionada ao parâmetro τ que controla o determinismo da busca, conforme mencionado no Capítulo 4. Assim para ajustar o algoritmo, o M-GEO$_{vsl}$ foi executado com τ entre [1,5] com incremento de 0.25. Foram realizadas 10^5 avaliações das funções objetivo, como critério de parada, com 50 reinicializações. A sequência de entrada poderia conter no máximo 100 eventos de entrada.

A influência de τ em M-GEO$_{vsl}$ pode ser vista na Figura 6.1 para cobrir a transição t_{12} do modelo M_8. A fim de minimizar as funções objetivo F_1 e F_2, os melhores resultados são obtidos com um valor intermediário de τ entre [1, 5]. Assim, uma busca muito determinística ($\tau = 5.0$) ou aleatória ($\tau = 1.0$) é menos propensa a encontrar a fronteira de Pareto para este problema de geração de casos de teste.

No processo de ajuste, o desempenho do M-GEO$_{vsl}$ foi melhor com $\tau = 3.75$. Assim, para a geração dos casos de teste, o algoritmo M-GEO$_{vsl}$ foi executado 10 vezes com $\tau = 3.75$ para cobrir cada transição dos modelos. A configuração dos experimentos foi praticamente igual às informações do processo de ajuste, mas aumentando-se o número máximo de avaliações das funções objetivo, o número de reinicializações e o tamanho máximo da sequência de entrada, conforme a seguir:

- Parâmetro τ: $\tau = 3.75$;
- Número de avaliações das funções objetivo: $maxNumAval = 10^6$;
- Número de reinicializações : $numReIni = 100$;
- Valor de referência: $\rho = 0$;
- Tamanho máximo da sequência de eventos de entrada: $maxNumEventos = 500$;

- Eventos de entrada: inteiros de 1 a $|I|$, onde I é o alfabeto de entrada de cada modelo;
- Parâmetros de entrada: inteiros positivos;
- Constante K da função objetivo F_1: $K = 100$;
- Penalidade γ da função objetivo F_1: $\gamma = 1000$.

O número de avaliações das funções objetivo determina o critério de parada do algoritmo. Já as reinicializações visam a evitar ótimos locais. Um limite máximo para o tamanho da sequência de entrada foi dado a fim de evitar que sejam geradas sequências muito longas para os modelos utilizados. Com respeito ao domínio da população de M-GEO$_{vsl}$, o primeiro elemento que representa o tamanho da sequência de entrada é um inteiro positivo menor que 500. O domínio das espécies que representam os eventos e os parâmetros de entradas varia de acordo com cada modelo. Em relação a F_1, as constantes K e γ são utilizadas para o cálculo do termo d da função objetivo. Nos experimentos, foi utilizado um Pentium 4 com 3.00 GHz e 1 GB de memória RAM.

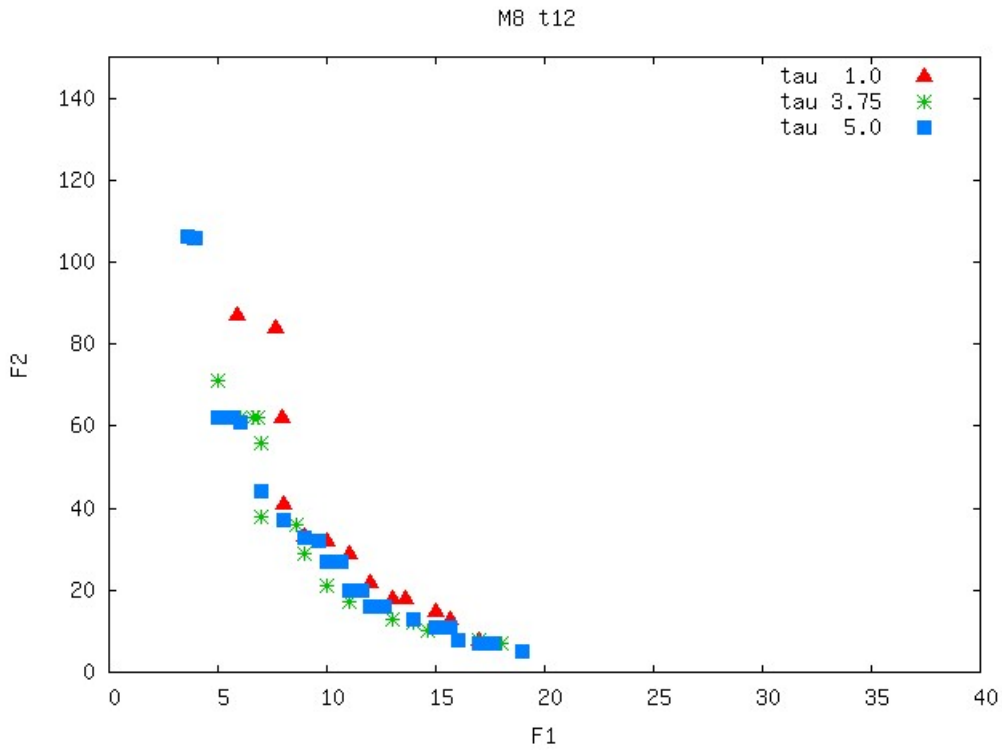


Figura 6.1: M_8 : Influência de τ em M-GEO$_{vsl}$.

A fim de automatizar alguns passos da abordagem MOST, duas ferramentas foram utilizadas: SMC[1] (*State Machine Compiler*) e SLIM (*SLIcing state based Model*) [13]. As informações de dependência de controle (NTSCD) e dados (DD) dos modelos foram obtidas pela ferramenta SLIM. Enquanto que SMC foi utilizada para obter os modelos executáveis. Essa ferramenta recebe como entrada uma descrição da MEFE e gera o código fonte correspondente ao comportamento do modelo em diversas linguagens de programação. Em particular, foi utilizado o modelo executável em Java, a fim de manter a compatibilidade de linguagem de programação usada no gerador de casos de teste. É importante notar que o modelo executável não corresponde ao código fonte do sistema em teste, pois é uma versão ainda independente de plataforma. O modelo executável foi devidamente instrumentado de forma manual para informar o caminho de transições percorrido, dados uma sequência de eventos de entrada e os valores dos parâmetros, e para coletar informações utilizadas pelas funções objetivo. Uma vez que a MEFE pode ser considerada como um modelo de estados da UML, porém sem algumas características como estados com histórico e estados paralelos, também poderiam ser utilizadas ferramentas que geram o código de modelos de estados da UML.

6.3 Métricas

Para avaliar os resultados, as soluções das fronteiras de Pareto de cada execução do M-GEO$_{vsl}$ para uma determinada transição alvo foram utilizadas para construir uma nova fronteira, mantendo-se apenas as soluções não-dominadas. Dessa maneira a análise de resultados dos experimentos leva em consideração apenas a fronteira de Pareto FP criada a partir dos resultados de cada execução do algoritmo.

A média do número de soluções de cada modelo M é calculada levando-se em consideração as soluções da fronteira de Pareto de todas as transições do modelo:

$$MédiaSolucões(M) = \frac{\sum_{\forall t \in T} Solucões(M,t)}{|T|} \tag{6.1}$$

Embora o critério de parada do M-GEO$_{vsl}$ seja o número máximo de avaliações das funções objetivo, foi coletado o número de avaliações necessário para encontrar cada solução. A média de avaliações para encontrar cada solução s na fronteira de Pareto FP de uma transição alvo t é calculada como a seguir:

$$MédiaAvaliacões(M) = \frac{\sum_{\forall t \in T} \sum_{\forall s \in FP} Avaliacões(M,t,s)}{|T| + |FP|} \tag{6.2}$$

[1]Available in http://smc.sourceforge.net.

Para o cálculo da média do tamanho dos caminhos obtidos em cada modelo M, é utilizado o tamanho do caminho de cada solução das fronteiras de Pareto:

$$MédiaTamCaminho(M) = \frac{\sum_{\forall t \in T} \sum_{\forall s \in FP} TamCaminhos(M, t, s)}{|T| + |FP|} \tag{6.3}$$

A média do tamanho das sequências também é calculada de acordo com as soluções das fronteiras de Pareto:

$$MédiaTamSequências(M) = \frac{\sum_{\forall t \in T} \sum_{\forall s \in FP} TamSequências(M, t, s)}{|T| + |FP|} \tag{6.4}$$

6.4 Resultados

Os dados resultantes da geração de casos de teste de cada modelo são apresentados na Seção 6.4.1. Além disso é descrito como a fronteira de Pareto pode ser analisada para selecionar os casos de teste. Já na Seção 6.4.2 são mostrados os resultados da comparação da abordagem proposta com um trabalho SBST relacionado.

6.4.1 Geração de Casos de Teste

Como MOST é uma abordagem multiobjetivo, pode-se produzir mais de uma solução bem-sucedida que alcance a transição alvo. O conjunto de soluções encontrado forma o conjunto de Pareto e cada solução é representada por um ponto na fronteira de Pareto. Na Figura 6.2 é mostrada a fronteira de Pareto para a transição t_7 de M_1. O conjunto de Pareto correspondente é apresentado na Tabela 6.2 com as seguintes informações: o caminho gerado, a sequência de eventos de entrada e os valores dos parâmetros. Os eventos de entrada estão codificados de acordo com a Tabela 6.3. As soluções diferenciam-se uma das outras no tamanho da sequência de entrada e no número de transições de T_a alcançadas pelo caso de teste de acordo as funções objetivo F_1 e F_2 (Seção 5.3.2). Observe pela Tabela 6.2 que as soluções possuem diferentes tamanhos. Além disso, o caminho percorrido em s_1 é formado por 4 transições, mas a sequência de entrada que habilita esse caminho possui 6 eventos de entrada. Então a sequência contém 2 eventos de entrada inoportunos.

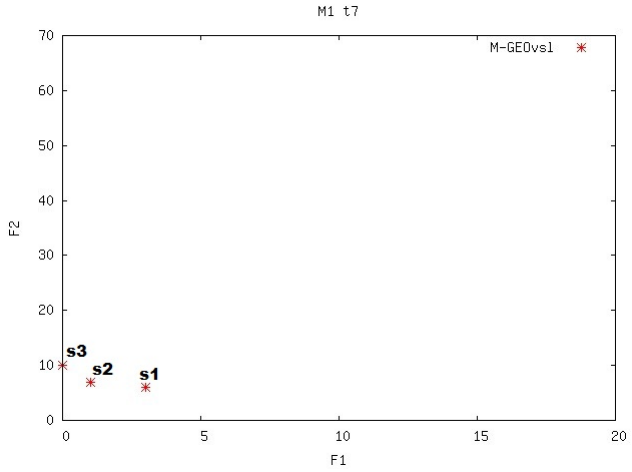


Figura 6.2: M_1: Fronteira de Pareto para t_7.

Tabela 6.2: M_1: Conjunto de Pareto para t_7.

Solução	Caminho	Sequência de eventos	Parâmetros					
			pin	sb	cb	p	w	d
s_1	$t_1t_4t_5t_7$	0 0 3 1 2 5	945	635	115	945	928	337
s_2	$t_1t_4t_6t_7t_9t_8$	0 4 1 3 5 7 6	208	298	410	208	942	143
s_3	$t_1t_4t_6t_7t_9t_8t_{10}$	4 0 1 4 3 5 7 6 7 10	70	929	907	70	110	605

Tabela 6.3: M_1: legenda dos eventos de entrada.

Número	Evento de entrada	Número	Evento de entrada
0	$cartão(pin, sb, cb)$	6	$poupanca()$
1	$senha(p)$	7	$pronto()$
2	$inglês()$	8	$saldo()$
3	$espanhol()$	9	$depósito(d)$
4	$saída()$	10	$saque(w)$
5	$contaCorrente()$	11	$recibo()$

Como mais de uma solução pode ser gerada, o testador tem o papel de decidir quais casos de teste serão aplicados. Para guiar o testador, a fronteira de Pareto e conjunto de Pareto devem ser analisados. Quanto mais soluções na fronteira, maior o número de soluções alternativas oferecidas para o testador. Ao minimizar F_2, as menores sequências de entrada podem ser obtidas enquanto que, provavelmente, poucas transições de T_a são alcançadas. Quando o tempo de execução dos casos de teste é alto, os menores casos de teste podem ser selecionados. Por exemplo, nessa situação, a solução s_1 da Tabela 6.2 poderia ser utilizada. Note que a abordagem foi capaz de encontrar o menor caminho do estado inicial a t_7 com a solução s_1. Por outro lado, mais transições de T_a podem ser percorridas minimizando-se F_1 mas uma sequência de entrada mais longa pode ser requerida. O testador pode optar pelos casos de teste mais longos quando o tempo de execução do caso de teste não for restrito. Uma vez que esses casos de teste possuem mais passos (interações de entrada), mais transições podem ser verificadas ao mesmo tempo. É possível, por exemplo, percorrer 7 transições diferentes usando o caminho de s_3. Observe que pela fronteira de Pareto (Figura 6.2) o valor de F_1 da solução s_3 é igual a zero. Isso significa que o caminho de s_3 percorre todas as transições que afetam t_7 ($T_a(t_7) = \{t_4, t_7, t_8, t_9, t_{10}\}$. Entretanto, s_3 possui o maior tamanho. Já s_1 possui o menor tamanho, mas poucas transições de $T_a(t_7)$ são percorridas.

Quando a fronteira de Pareto possui um ponto, significa que essa solução dominou todas as demais geradas pelo M-GEO$_{vsl}$. Dessa maneira, a solução apresentou melhores valores considerando ambas funções objetivo F_1 e F_2 do que as outras soluções geradas. Por exemplo, existe apenas uma solução para a transição t_4 de M_1:

Caminho	Sequência de eventos	Parâmetros					
		pin	sb	cb	p	w	d
t_1t_4	0 1	806	698	167	806	537	764

Nesse caso, foi encontrada a solução ótima global para t_4, visto que essa sequência de entrada não possui eventos inoportunos e habilita o menor caminho a partir do estado inicial a t_4, alcançando todas as transições de $T_a(t_4) = \{t_1, t_4\}$). Para quaisquer outras soluções, uma das funções objetivo ou ambas teriam valores piores que essa solução encontrada, e logo seriam soluções dominadas. Note também que foram gerados os valores dos parâmetros (p e *pin*) que permitem que t_4 seja habilitada. A guarda de t_4 é apenas satisfeita quando o valor de *pin* fornecido é a senha esperada p e o número de tentativas para entrar com o valor de *pin* correto é menor que 3.

Na Tabela 6.4 é mostrada a cobertura obtida pela geração de casos de teste. Para a maioria dos modelos, foi obtida uma cobertura de 100% das transições. Assim nesses casos foi encontrada pelo menos uma solução que cubra cada transição do modelo. Isso significa que a fronteira de Pareto correspondente a cada transição alvo possui pelo menos uma solução que cubra a transição alvo. Apenas em dois modelos não foram gerados casos

de teste para cobrir todas as transições, mas ainda foi obtida uma alta cobertura. As transições não cobertas também são apresentadas na Tabela 6.4. Pelas Figuras 6.3 a 6.6, pode-se verificar o número de soluções da fronteira de Pareto para cada transição. Foi analisado que qualquer caminho com a transição t_{83} de M_7 seria infactível. Essa transição precisa que o valor de um contador *count* seja menor ou igual a zero para ser habilitada, mas *count* sempre tem um valor maior ou igual a 1 no estado de origem de t_{83} independentemente do caminho tomado. Além disso, no estado de origem de t_{83}, a variável *count* não é redefinida. Assim não existem dados de teste que exercitem t_{83}. Em relação às transições de M_{15}, uma das razões pelas quais algumas transições não foram cobertas é que suas guardas envolviam diversos valores de parâmetros, limitando o domínio de dados capaz de satisfazê-las.

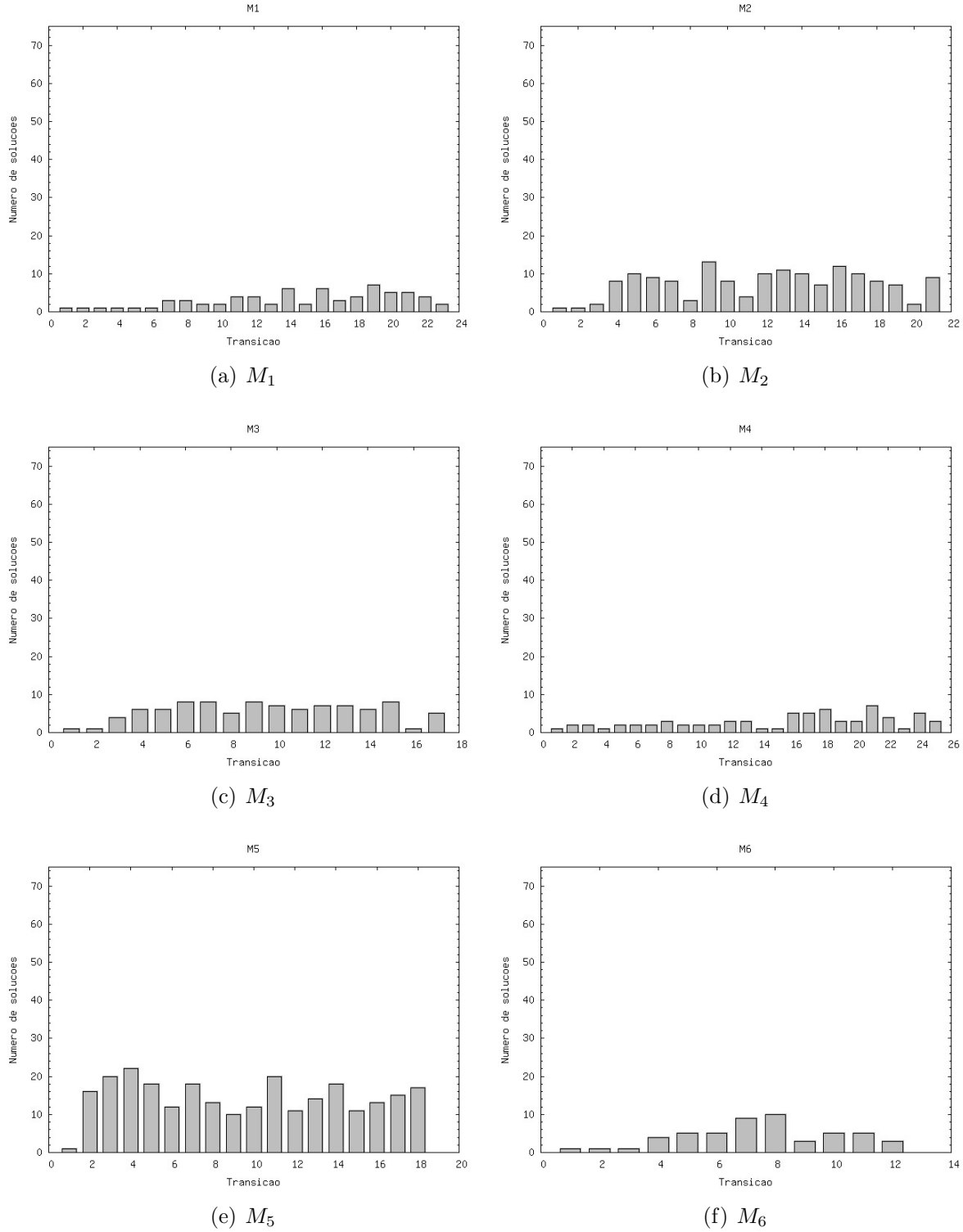
Tabela 6.4: Cobertura da geração dos casos de teste.

Modelo	Cobertura	Transições não cobertas
M_1 :	100%	
M_2 :	100%	
M_3 :	100%	
M_4 :	100%	
M_5 :	100%	
M_6 :	100%	
M_7 :	98.88%	t_{83}
M_8 :	100%	
M_9 :	100%	
M_{10} :	100%	
M_{11} :	100%	
M_{12} :	100%	
M_{13} :	100%	
M_{14} :	100%	
M_{15} :	97.45%	$t_{65}, t_{66}, t_{79}, t_{87}, t_{94}$ e t_{146}

De acordo com as métricas definidas na Seção 6.3, as médias dos resultados da geração de casos de teste são mostradas na Tabela 6.5: número de soluções, número de avaliações, tamanho dos caminhos, tamanho das sequências e tempo de execução (em segundos), e os desvios padrões correspondentes (σ). O tempo é referente à execução do algoritmo M-GEO$_{vsl}$ para uma transição do modelo. Em média, o algoritmo levou aproximadamente 26 segundos em cada execução. Note que pela Tabela 6.5, embora o critério de parada do M-GEO$_{vsl}$ fosse de 10^6 avaliações das funções objetivo, foi necessário aproximadamente metade desse valor para obter uma solução, em média. Os caminhos e sequências mais longos foram obtidos nos modelos com maior número de transições, M_7 e M_{15}.

Tabela 6.5: Médias e desvios padrões (σ) dos resultados da geração de casos de teste: número de soluções, número de avaliações, tamanho dos caminhos, tamanho das sequências e tempo de execução (segundos).

M	#Sol.	σ	#Aval.	σ	|Cam.|	σ	|Seq.|	σ	Tempo	σ
M_1 :	3.04	1.85	444297.28	215109.16	8.44	4.5	15.56	10.01	12.91	1.08
M_2 :	7.29	3.68	423318.67	183100.7	11.04	4.16	28.03	12.68	25.19	4.24
M_3 :	5.53	2.45	474713.55	191829.28	6.81	2.57	7.12	2.73	13.88	0.6
M_4 :	2.84	1.65	486978.85	247153.34	9.08	4.71	19.7	15.16	20.84	2.66
M_5 :	14.5	4.89	512107.85	157098.15	19.35	6.17	39.38	14.2	33.06	3.76
M_6 :	4.33	2.9	376056.57	172566.32	10.12	4.43	20.64	10.72	23.08	2.68
M_7 :	51.04	11.79	494939.44	76545.47	41.88	6.03	65.82	13.08	41.21	4.14
M_8 :	16.21	4.98	510913.61	61999.99	30.61	7.84	53.49	21.2	22.75	1.58
M_9 :	10.19	3.23	523125.42	117053.8	18.13	4.5	60.91	19.43	27.22	1.95
M_{10} :	6.16	1.91	511914.65	142526.83	10.41	3.63	56.39	23.01	16.76	1.01
M_{11} :	15.33	3.73	496579.78	104922.17	17.38	4.29	43.55	21.08	30.33	4.39
M_{12} :	6.73	3.08	501232.92	145421.33	16.52	8.15	61.7	40.95	20.23	1.77
M_{13} :	11.75	3.91	474617.1	157831.04	20.06	6.37	32.2	11.64	31.81	3.08
M_{14} :	16.45	3.83	473504.46	102092.43	22.86	4.54	40.88	9.86	34.95	3.8
M_{15} :	18.13	11.31	506794.44	149896.54	33.87	11.32	178.12	77.23	42.43	9.49
Médias	12.64	4.35	480739.64	148343.1	18.44	5.55	48.23	20.2	26.44	3.08

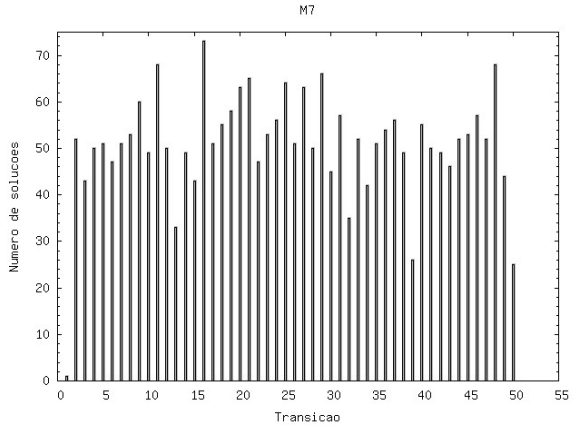


(a) M_1 (b) M_2

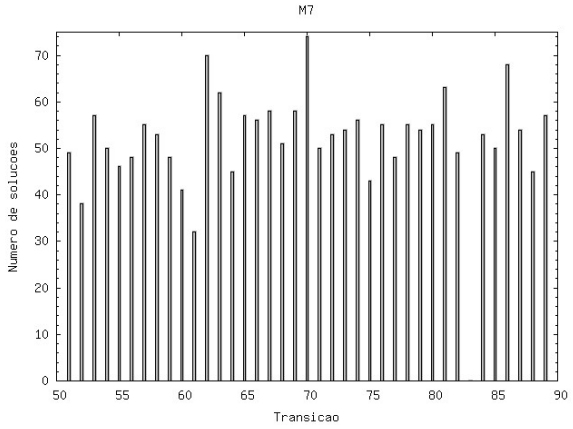
(c) M_3 (d) M_4

(e) M_5 (f) M_6

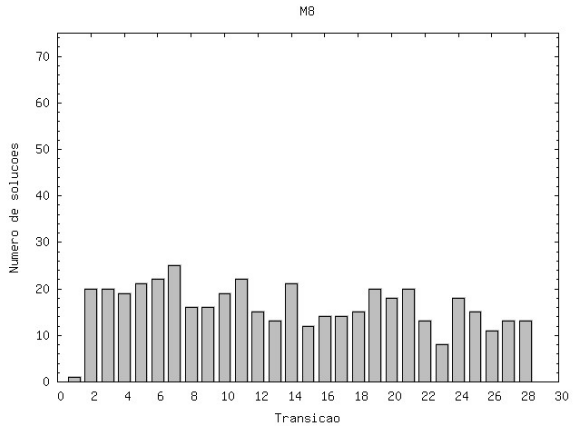
Figura 6.3: M_1 a M_6: Número de soluções no conjunto de Pareto.



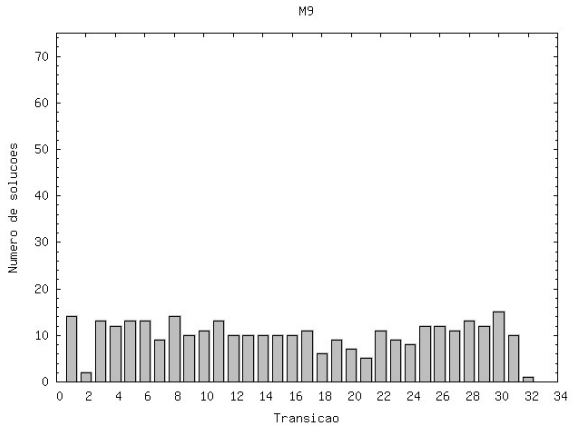
(a) M_7



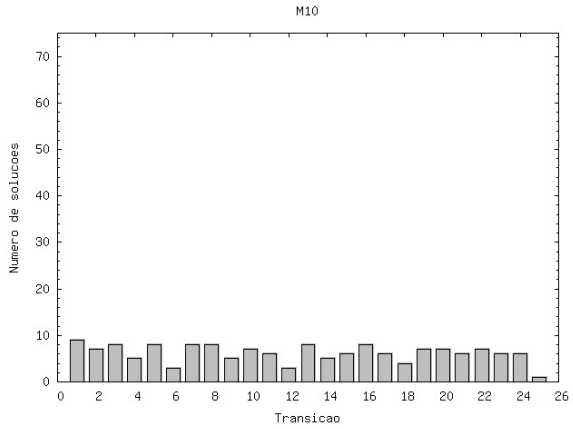
(b) M_7



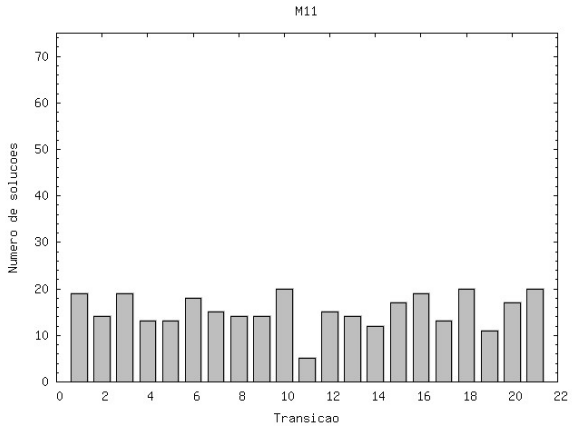
(c) M_8



(d) M_9

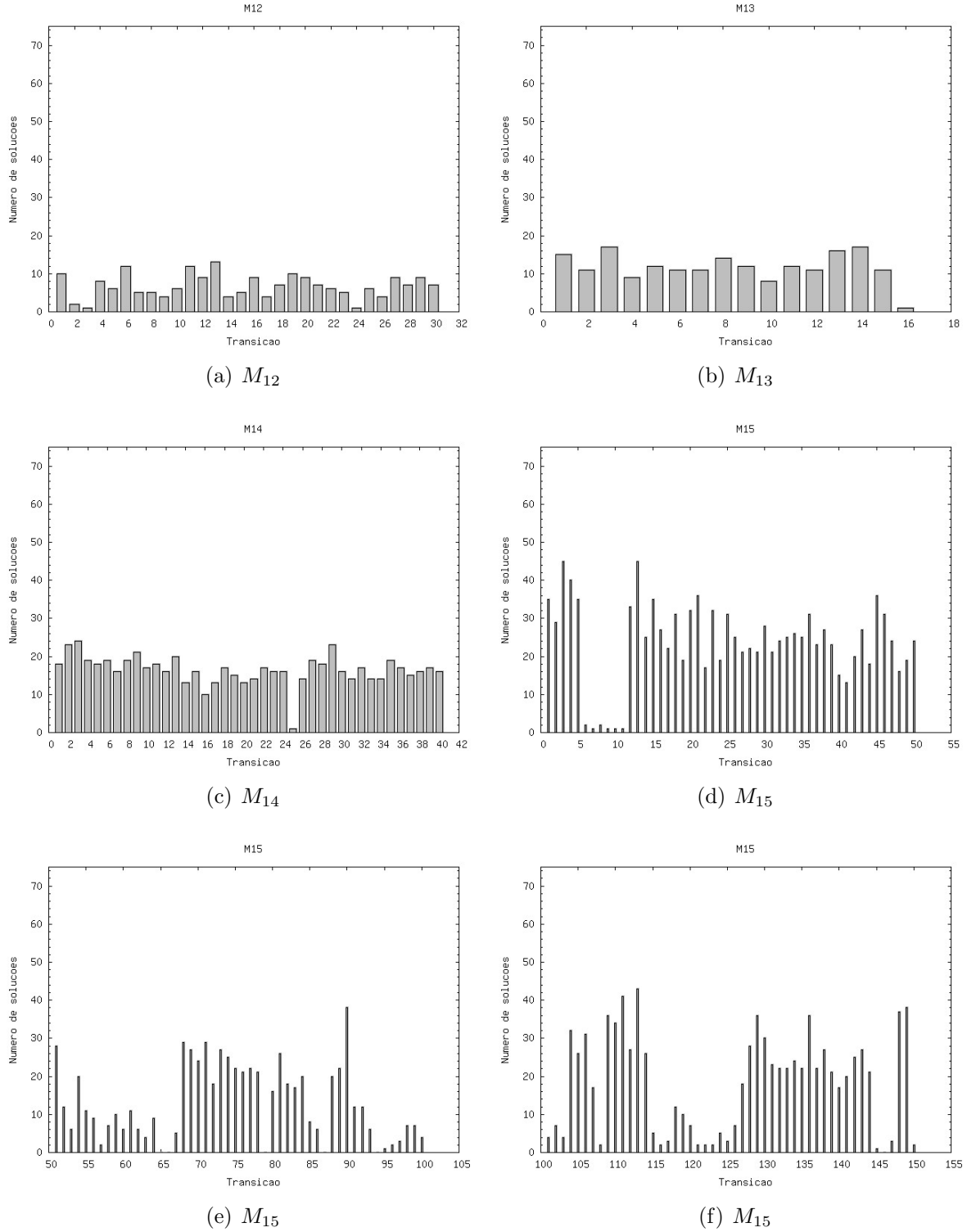


(e) M_{10}



(f) M_{11}

Figura 6.4: M_7 a M_{11}: Número de soluções no conjunto de Pareto.

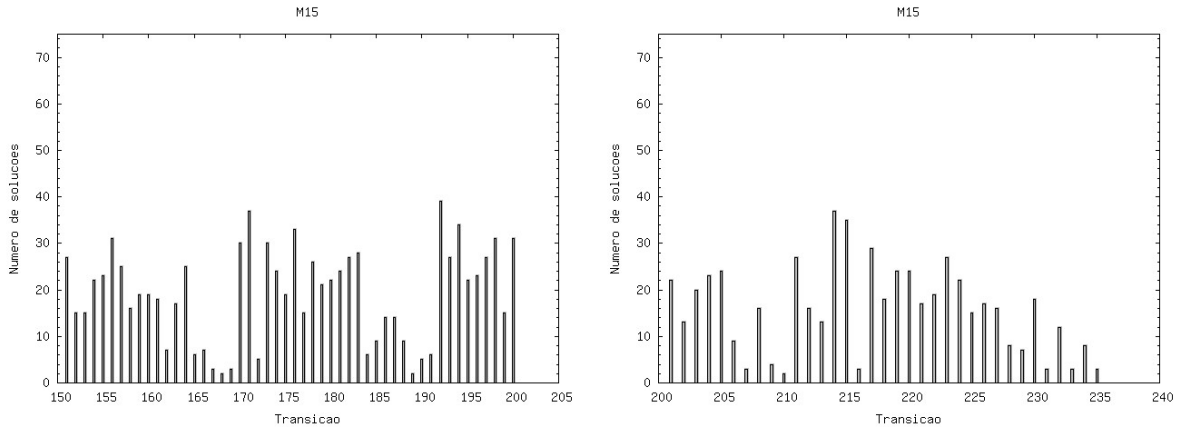


(a) M_{12} (b) M_{13}

(c) M_{14} (d) M_{15}

(e) M_{15} (f) M_{15}

Figura 6.5: M_{12} a M_{15}: Número de soluções no conjunto de Pareto.



(a) M_{15} (b) M_{15}

Figura 6.6: M_{15}: Número de soluções no conjunto de Pareto.

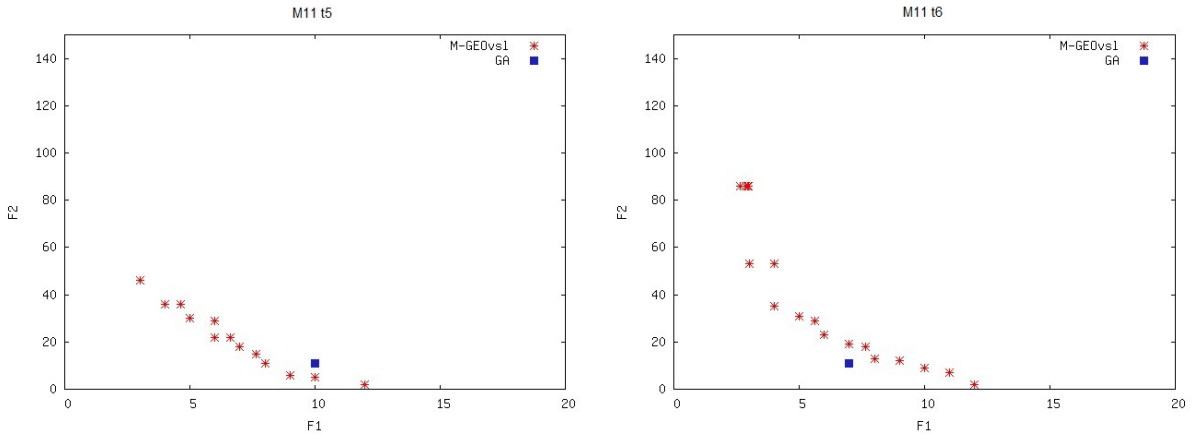
6.4.2 Comparação com Trabalho Relacionado

A fim de verificar o desempenho de MOST com trabalhos existentes, os resultados de MOST foram comparados com as abordagens SBST baseada em MEFE propostas por Kalaji et al. [93, 92]. A comparação foi dividida de acordo com os diferentes modelos aplicados em cada trabalho: M_{11} [93] e M_{12} e M_{13} [92]. As principais diferenças dessas abordagens com MOST são: *i*) um algoritmo genético foi utilizado com uma otimização mono-objetivo. Foram feitas 1000 gerações com uma população de 25 indivíduos. *ii*) o tamanho do caso de teste não era considerado parte do processo evolutivo. Foi fixado o tamanho de 11 para M_{11} e de 10 para M_{12} e M_{13}. *iii*) a geração dos casos de teste foi realizada em dois passos. Primeiramente, foram gerados os caminhos de transições com o algoritmo genético e, em seguida, foram gerados dados aleatoriamente para habilitar esses caminhos. Como os valores dos parâmetros dos eventos de entrada são gerados após a obtenção dos caminhos, existe a possibilidade de ter caminhos que não possam ser habilitados, por serem caminhos infactíveis. Embora utilizem uma outra função objetivo para guiar a busca, é importante destacar que o valor da função objetivo também relacionado com a análise de dependência das transições do caminho gerado, conforme descrito na Seção 3.4.

Cada caminho encontrado por Kalaji et al. foi avaliado por F_1 e F_2 definidas em MOST, obtendo-se a solução s_k no espaço das funções objetivo. A solução s_k foi analisada em relação à fronteira de Pareto produzida pelo M-GEO$_{vsl}$. Para ilustrar, na Figura 6.7(a) é mostrada a fronteira de Pareto obtida pelo M-GEO$_{vsl}$ para t_5 de M_{11} e a solução s_k (indicada por ■). Nesse caso s_k é uma solução dominada pelas soluções da fronteira,

assim não constitui uma solução ótima para o problema. Por outro lado, para a transição t_6, s_k é uma solução não-dominada no espaço das funções objetivo, como apresentado na Figura 6.7(b). Note que s_k domina 4 soluções da fronteira de Pareto encontrada pelo M-GEO$_{vsl}$. Para cada transição de M_{11}, na Tabela 6.6 é mostrado o número de soluções da fronteira de Pareto obtida pelo M-GEO$_{vsl}$ (pf), e a porcentagem de soluções da fronteira não-dominada pela solução s_k (nd). Observe que em 9 das 20 transições de M_{11}, as soluções da fronteira de Pareto do M-GEO$_{vsl}$ dominam as soluções encontradas por Kalaji et al.. Para as demais transições, s_k é uma solução não-dominada, assim pode ser considerada solução ótima sob a perspectiva multiobjetivo. Analogamente à Tabela 6.6, nas Tabelas 6.7 e 6.8 são mostradas as informações da comparação com os modelos M_{12} e M_{13}, respectivamente. Apenas em M_{12}, as soluções de MOST para t_{23} foram todas dominadas por s_k. Já em 10 das 15 transições de M_{13}, nenhuma solução de MOST foi dominada por s_k. Os detalhes dessa comparação também podem ser encontrados na página do grupo SED.

Um aspecto que deve ser considerado nessa comparação é o número de avaliações das funções objetivo. Embora diferentes algoritmos evolutivos tenham sido utilizados, foi necessário um número maior de avaliações no M-GEO$_{vsl}$, visto que os valores dos parâmetros são levados em consideração no processo evolutivo. Na abordagem de Kalaji et al. ainda é preciso demandar esforço para encontrar esses valores aleatoriamente após o término da execução do algoritmo genético. Além disso, as soluções de s_k não dominam as soluções de MOST que possuem tamanho menor que o tamanho fixo de s_k.



(a) t_5 (b) t_6

Figura 6.7: M_{11}: Fronteira de Pareto.

Tabela 6.6: M_{11}: Número de soluções da fronteira de Pareto obtida pelo M-GEO$_{vsl}$ (pf) e porcentagem de soluções não-dominadas por s_k (nd).

Trans.	pf	nd (%)	Trans.	pf	nd (%)
t_0	20	95.00	t_{11}	5	100.0
t_1	19	89.47	t_{12}	15	100.0
t_2	14	100.0	t_{13}	14	92.86
t_3	19	79.94	t_{14}	12	91.67
t_4	13	84.61	t_{15}	17	94.12
t_5	13	100.0	t_{16}	19	100.0
t_6	18	77.78	t_{17}	13	92.31
t_7	15	80.00	t_{18}	20	100.0
t_8	14	64.28	t_{19}	11	100.0
t_9	14	92.85	t_{20}	17	100.0
t_{10}	20	100.0			

Tabela 6.7: M_{12}: Número de soluções da fronteira de Pareto obtida pelo M-GEO$_{vsl}$ (pf) e porcentagem de soluções não-dominadas por s_k (nd).

Trans.	pf	nd (%)	Trans.	pf	nd (%)
t_1	10	60.00	t_{16}	9	100.0
t_2	2	100.0	t_{17}	4	75.00
t_3	1	100.0	t_{18}	7	85.71
t_4	8	75.00	t_{19}	10	70.00
t_5	6	16.67	t_{20}	9	55.56
t_6	12	75.00	t_{21}	7	100.0
t_7	5	60.00	t_{22}	6	100.0
t_8	5	40.00	t_{23}	5	00.00
t_9	4	50.00	t_{24}	1	100.0
t_{10}	6	16.67	t_{25}	6	33.33
t_{11}	12	75.00	t_{26}	4	75.00
t_{12}	9	77.78	t_{27}	9	100.0
t_{13}	13	84.62	t_{28}	7	57.15
t_{14}	4	75.00	t_{29}	9	100.0
t_{15}	5	80.00	t_{30}	7	85.71

Tabela 6.8: M_{13}: Número de soluções da fronteira de Pareto obtida pelo M-GEO$_{vsl}$ (pf) e porcentagem de soluções não-dominadas por s_k (nd).

Trans.	pf	nd (%)	Trans.	pf	nd (%)
t_0	1	100.0	t_8	14	85.71
t_1	15	100.0	t_9	12	91.67
t_2	11	100.0	t_{10}	8	100.0
t_3	17	100.0	t_{11}	12	100.0
t_4	9	88.89	t_{12}	11	100.0
t_5	12	100.0	t_{13}	16	93.75
t_6	11	90.91	t_{14}	17	88.24
t_7	11	100.0	t_{15}	11	100.0

6.5 Considerações Finais

Nos experimentos realizados, foi obtida uma alta cobertura das transições nos modelos com os casos de teste gerados pela abordagem MOST, sendo que na maioria houve uma cobertura de 100%. Devido às características do M-GEO$_{vsl}$, foi possível encontrar soluções com diferentes tamanhos, inclusive os menores caminhos em alguns casos. Na comparação com outra abordagem SBST, foi verificado que em muitos casos as soluções dessa abordagem não dominam as soluções de MOST. Isso significa que, em geral, as soluções de MOST apresentam melhores valores tanto para a função objetivo F_1 que verifica se as transições que afetam a transição alvo também foram percorridas, quanto para F_2 que verifica o tamanho da sequência gerada.

Uma das limitações de MOST é a questão de manter todos os parâmetros do modelo na geração. Isso deve ser tratado em um trabalho futuro para que apenas os parâmetros envolvidos na sequência de entrada sejam otimizados. Nos casos em que os nomes dos parâmetros eram iguais em diferentes transições, foi necessário renomeá-los para permitir que os eventos correspondentes pudessem ter valores de parâmetros diferentes. Em relação ao domínio de valores dos parâmetros, apenas valores inteiros foram utilizados; entretanto poder-se-ia utilizar uma combinação de valores inteiros e reais. Valores do tipo lógico poderiam ser representados por 0 (falso) e 1 (verdadeiro). Caso um parâmetro de entrada fosse uma cadeia de caracteres, poder-se-ia utilizar uma lista enumerando um conjunto de palavras do domínio de aplicação.

Como os casos de teste gerados por MOST estão no mesmo nível de abstração do modelo, ainda é preciso refiná-los para serem executados na implementação da aplicação em teste. As regras para essa transformação estão sendo definidas no trabalho de mestrado da aluna Erika Almeida, que também participa do grupo SED. Dois casos de teste de M_{15} (s'_1 e s'_2) foram transformados em executáveis e aplicados de fato na implementação da

aplicação web, a fim de verificar o tempo de geração e de execução dos casos de teste. Os caminhos dessas soluções cobrem a mesma transição mas possuem diferentes tamanhos: $|caminho(s'_1)| = 4$ e $|caminho(s'_2)| = 15$. Ambos os casos de teste poderiam ser usados para percorrer a transição alvo. No entanto, como s'_2 possui mais operações que s'_1, s'_2 requer mais tempo para ser executado, visto que M_{15} representa uma aplicação web e demanda tempo para a entrada de dados e para a espera de resposta do sistema. Os tempos de execução de s'_1 e s'_2 foram de 70 segundos e 140 segundos, respectivamente. Como o tempo é um aspecto importante na indústria, s'_1 poderia ser selecionado. Por outro lado, s'_2 representa um cenário de teste que envolve mais situações que s'_1. Com relação ao tempo de geração, os casos de teste para cada transição de M_{15} foram gerados em aproximadamente 42 segundos. Note então que para executar apenas um caso de teste no nível de implementação, por exemplo s'_1, leva mais tempo que toda a geração de casos de teste para uma transição alvo. De acordo com essas informações, é nítido que o desempenho de uma técnica SBST baseada em modelo seria melhor do que uma técnica baseada em código fonte em termos de tempo neste contexto de aplicações web. É importante enfatizar que técnicas de SBST geram e executam no mínimo na ordem de 1000 casos de teste. Embora seja necessário transformar os casos de teste de MOST em executáveis no nível de implementação, todos os dados de entrada para executar os casos de teste já foram gerados pela abordagem.

Capítulo 7

Conclusões

Um dos grandes desafios na atividade de teste consiste na geração automática de casos de teste, a fim de reduzir custo e esforço no processo de teste. Neste trabalho é proposta uma abordagem de teste baseado em busca (SBST), denominada MOST, para gerar automaticamente casos de teste a partir de modelos de estados, em particular Máquina de Estado Finito Estendida (MEFE). Em geral, as técnicas tradicionais de teste baseado em modelo realizam uma análise exaustiva para a derivação dos casos de teste, um fator que limita sua aplicação de acordo com o tamanho do espaço de estados. Já em SBST, técnicas de otimização são utilizadas para explorar o espaço de soluções a fim de cobrir um critério de teste. Na geração de casos de teste a partir de MEFE, um dos aspectos com que a técnica de otimização precisa lidar é como gerar sequências de teste cujo tamanho pode ser indeterminado. Isso motivou o desenvolvimento do algoritmo evolutivo M-GEO$_{vsl}$ que permite gerar soluções de tamanho variável. Com a aplicação desse algoritmo em MOST, foi possível obter sequências de eventos de entrada com diferentes tamanhos. Dessa maneira, não é necessário estabelecer a priori o número de elementos a ser otimizado.

Além da geração de sequências de entrada, uma outra vantagem da abordagem proposta é a geração dos dados envolvidos na sequência, tais como os valores dos parâmetros de entrada. Caso esses valores sejam obtidos após a geração dos casos de teste, existe a possibilidade de se obter caminhos infactíveis. Esse é um problema em aberto no teste baseado em MEFE devido aos conflitos nas guardas e ações ao longo dos caminhos do modelo. Em MOST, a sequência de eventos de entrada e os valores dos parâmetros gerados por M-GEO$_{vsl}$ são utilizados como entrada para um modelo executável da MEFE que, por sua vez, produz dinamicamente o caminho exercitado no modelo. Como os dados do modelo (parâmetros e variáveis) são considerados durante a execução do modelo, obtêm-se somente caminhos factíveis, pelo menos no nível de modelo.

Como MOST utiliza uma abordagem de otimização multiobjetivo, foram definidas

duas funções objetivo para guiar a busca: uma avalia se a transição alvo t e as transições que afetam t são alcançadas e a outra procura minimizar o tamanho da sequência de entrada. Para a primeira função objetivo, foi feita uma análise de dependência tanto de controle quanto de dados do modelo para identificar partes do modelo que afetam t.

É importante notar que os casos de teste estão no mesmo nível de abstração do modelo e não são derivados a partir do código fonte. Dessa maneira para executar os casos de teste gerados no sistema em teste, ainda é necessário transformá-los em executáveis, adicionando detalhes de implementação. Entretanto, as abordagens SBST baseadas em código requerem que o sistema em teste seja executado diversas vezes com os casos de teste gerados. Isso pode demandar muito tempo e esforço computacional para testar aplicações web, por exemplo. Outro ponto é que os dados de entrada gerados pela abordagem MOST também podem ser utilizados para executar os casos de teste no nível de implementação.

Para validar a abordagem proposta foram realizados experimentos com diversos estudos de casos encontrados na literatura como também aplicações reais. Também foi feita uma comparação do desempenho dos resultados de MOST com os casos de teste de outro trabalho SBST relacionado.

7.1 Contribuições

Pode-se citar como principais contribuições deste trabalho:

- Propõe a abordagem MOST para gerar de casos de teste a partir de MEFE, aplicando-se o algoritmo evolutivo multiobjetivo M-GEO$_{vsl}$. Como uma abordagem de teste baseado em modelo, permite gerar os casos de teste mesmo quando o código fonte não está disponível, além de ser independente de plataforma e linguagem de programação. A utilização de modelos também pode ser atrativa para a geração dinâmica de casos de teste, principalmente em SBST, visto que os modelos são uma abstração da implementação e não necessitam executar diversas vezes a implementação em teste.

- Integra os aspectos de controle e de dados de uma MEFE para a geração de casos de teste. Ambos os aspectos são considerados durante o processo evolutivo, obtendo-se não apenas os caminhos de transições como também os dados para exercitá-los. Em geral em abordagens SBST, os dados dos parâmetros dos eventos de entrada são obtidos aleatoriamente e/ou após a geração do caminho.

- Utiliza modelos executáveis na geração de casos de teste. Uma primeira vantagem desse uso é que os usuários podem simplesmente executar o modelo para validá-lo antes da geração dos casos de teste. Um outro ponto é que apenas caminhos

disparados durante a execução do modelo são considerados como soluções candidatas, lidando com o problema de geração de caminhos infactíveis.

- Propõe uma abordagem SBST multiobjetivo, definindo funções objetivo que consideram tanto a cobertura da transição alvo quanto o tamanho do caso de teste. Como pode ser gerada mais de uma solução, é descrito como guiar a seleção dos casos de teste pela fronteira de Pareto produzida na abordagem MOST.

- Utiliza análise de dependência de modelos para guiar a busca por soluções. O uso de dependência minimiza a perda de informações provenientes das variáveis de contexto nas MEFEs, visto que essas variáveis influenciam a execução de um caminho mas não são controladas pelas técnicas de otimização.

- Estende a versão multiobjetivo do algoritmo GEO, introduzindo o M-GEO$_{vsl}$. Além de lidar com múltiplos objetivos, o diferencial desse algoritmo é que permite a geração de soluções com tamanhos variáveis. Assim, o tamanho da solução é ajustado automaticamente durante o processo evolutivo, não sendo preciso fornecer previamente esse valor.

As publicações deste trabalho podem ser encontradas em: [144, 145, 146, 148, 149, 147]. No Apêndice A são apresentados os resumos desses artigos.

No início do doutorado foi investigada a aplicação do GEO no teste baseado em MEFE, mas considerando-se apenas o fluxo de controle do modelo [144, 145]. Foi utilizada uma codificação discreta do GEO, ao invés da binária como na versão canônica. O modelo executável foi adotado desde o princípio do trabalho.

Em [146], uma ideia inicial de uma abordagem multiobjetivo de geração de casos de teste com o M-GEO$_{vsl}$ foi proposta. Entretanto, foi utilizada a técnica de fatiamento (em inglês, *slicing*) para MEFE a fim de se obter as informações que guiassem a busca. Entre os passos dessa técnica, inclui-se a análise de dependência do modelo, além de envolver, por exemplo, a eliminação e junção de estados. Como as dependências já identificam as partes da MEFE que afetam a transição alvo, concluiu-se que não seria preciso realizar todos os passos da técnica de fatiamento. Dessa maneira, em [148] foi proposta a abordagem MOST, na qual funções objetivo foram definidas de acordo com as informações de dependência do modelo. Em [149] foi descrito de forma mais genérica o algoritmo M-GEO$_{vsl}$, e não apenas para teste baseado em modelo.

Em [147] foram apresentadas direções de um trabalho futuro da aplicação de MOST em teste de robustez.

7.2 Trabalhos Futuros

A seguir são apresentadas algumas alternativas de trabalhos futuros:

- A fim de automatizar os passos da abordagem MOST, é importante a especificação e desenvolvimento de uma ferramenta de apoio. Deve-se realizar não somente a geração automática dos casos de teste, mas também a automação da instrumentação do modelo executável. Além disso, essa ferramenta deve interagir com outras ferramentas, tais como SMC e SLIM, recebendo e tratando seus dados de forma adequada e automática.

- A avaliação da abordagem deve ser continuada, principalmente utilizando-se aplicações reais. É necessário verificar a eficiência, a escalabilidade e a praticidade da abordagem em diferentes contextos. Os estudos empíricos também devem avaliar a capacidade das funções objetivo propostas. Outras funções objetivo podem ser definidas para guiar melhor a busca. Seria interessante também encontrar aplicações que tenham o código fonte disponível, para investigar a eficiência dos casos de teste gerados no nível da implementação do sistema em teste.

- Ao invés de considerar todos os parâmetros do modelo, poder-se-ia otimizar apenas os parâmetros envolvidos no caso de teste gerado. Para tanto, é preciso investigar uma representação adequada para as soluções e verificar como o algoritmo evolutivo deve lidar com essa representação, por exemplo, no caso de uma operação de mutação.

- O requisito de teste poderia ser um cenário de teste, como um subcaminho no modelo, e não somente a transição alvo. Um aspecto que deve ser levado em conta é a dificuldade de verificar se o cenário se encontra no caminho obtido. Esse problema por si só pode exigir um grande esforço computacional para fazer essa verificação. Isso se torna ainda mais agravante com uso de algoritmo evolutivos, que requerem diversas iterações em suas execuções.

- A aplicação da abordagem MOST também poderia ser investigada em outro contexto de teste, tal como o teste de robustez. Isso implica em definir adequadamente uma representação de solução e funções objetivo para avaliar as soluções geradas.

- Para não restringir MOST ao algoritmo M-GEO$_{vsl}$, deve-se investigar outras técnicas de otimização para serem aplicadas na abordagem e, consequentemente, compará-las com o M-GEO$_{vsl}$. Podem ser propostas extensões ou adaptações de técnicas de otimização existentes para permitir uma representação adequada às soluções e

operações sobre essas soluções para o problema de teste baseado em modelos de estados.

- Outra direção é aplicar o algoritmo M-GEO$_{vsl}$ em um contexto diferente ao de geração de casos de teste. É importante avaliar a capacidade e versatilidade do algoritmo em gerar soluções para problemas que requerem soluções com diversos tamanhos.

Apêndice A

Publicações

- Artigo [144]

  Título: Aplicação de algoritmos evolutivos na geração automática de dados de teste de conformidade

  Autores: T. Yano, E. Martins e F. L. de Sousa

  Evento: WTF 2008: Worshop de Testes e Tolerância a Falhas

  Resumo: Com o objetivo de reduzir custo e esforço no desenvolvimento de software, a geração automática de dados de teste tem sido considerada como uma importante atividade para auxiliar o processo de teste. Recentemente, houve um grande interesse em teste baseado em busca que procura aplicar técnicas de otimização na geração de dados de teste. Nesse contexto, pode-se encontrar diversos trabalhos que utilizam algoritmos evolutivos para o teste estrutural, porém poucos abordam o teste funcional. Dessa maneira, neste artigo é feita uma avaliação inicial da eficácia do algoritmo GEO, um algoritmo evolutivo recente, para gerar dados de teste de conformidade.

  Abstract: In order to reduce the cost and effort in the software development, the automatic test data generation have been considered as an important activity to aid the testing process. Recently, there has been a great interest in search based testing that apply optimization techniques in the test data generation. In this context, several works can be found that use evolutionary algorithms to structural testing, however few of them focus on the functional testing. Then in this paper, an initial assessment is done to check the efficacy of GEO algorithm, a recent evolutionary algorithm, in dealing with data generation for conformance testing.

- Artigo [145]

Título: An Evolutionary Approach for Robustness Testing

Autores: T. Yano, E. Martins e F. L. de Sousa

Evento: ICEC 2009: International Conference on Evolutionary Computation

Resumo: Neste trabalho é apresentada uma abordagem de teste evolutivo para gerar automaticamente sequências de teste de robustez para testar um protocolo de comunicação, modelado como uma máquina de estado finito estendida (MEFE). O modelo representa a situação normal como também em presença de falhas, o que torna o modelo muito grande para ser tratado por abordagens de geração de casos de teste convencionais, devido ao risco de explosão combinatória. Para evitar esse problema, é utilizada uma abordagem de teste baseada em propósito de teste. A fim de buscar sequências que satisfaçam os propósitos de teste, dois algoritmos evolutivos foram utilizados: o algoritmo de otimização extrema generalizada (GEO) e o algoritmo genético (AG). Apenas a parte de fluxo de controle do modelo é levada em consideração. Os resultados mostram que a abordagem é viável e potencialmente útil para considerar a parte de fluxo de dados de modelos MEFE complexos.

Abstract: In this paper we present an evolutionary testing approach to automatically generate robustness test sequences to test a communication protocol, modeled as an extended finite state machine (EFSM). The model represents the normal situation as well as in presence of faults, which makes the model too large to be treated by conventional test case generation approaches, because of the risk of combinatorial explosion. To cope with this problem, we use a testing approach based on test purposes. To search sequences that satisfy the test purposes, we use two evolutionary algorithms: the Generalized Extremal Optimization (GEO) and a Genetic Algorithm (GA). For the moment, only the control flow part of the model is taken into account. Results show that the approach is viable and potentially useful to consider data flow part of complex EFSM models.

- Artigo [146]

  Título: Generating Feasible Test Paths from an Executable Model Using a Multi-objective Approach

  Autores: T. Yano, E. Martins e F. L. de Sousa

  Evento: SBST 2010: 3^{rd} Internationl Workshop on Search-Based Software Testing

  Resumo: Técnicas de teste baseado em busca que utilizam metaheurísticas, tais como algoritmos evolutivos, têm sido largamente utilizadas para geração de casos de teste, mas a maioria das abordagens são propostas para o teste estrutural. Neste trabalho é apresentada uma abordagem para geração de sequência de teste a partir de modelo de comportamento, em particular, máquina de estado finito estendida. Um problema em aberto é a geração de caminhos infactíveis, já que precisam ser detectados e descartados manualmente. Para evitar esse problema, um modelo executável é utilizado para obter caminhos factíveis dinamicamente. Um algoritmo evolutivo é utilizado para buscar soluções que cubram um determinado propósito de teste, no caso uma transição alvo. A transição alvo é usada como critério para obter informações de fatiamento, a fim de identificar as partes do modelo que afetam o propósito de teste. Uma busca multiobjetivo é utilizada: cobertura do propósito de teste e minimização do tamanho da sequência, já que sequências longas requerem mais esforço para serem executadas.

  Abstract: Search-based testing techniques using meta-heuristics, like evolutionary algorithms, has been largely used for test data generation, but most approaches were proposed for white-box testing. In this paper we present an evolutionary approach for test sequence generation from a behavior model, in particular, Extended Finite State Machine. An open problem is the production of infeasible paths, as these should be detected and discarded manually. To circumvent this problem, we use an executable model to obtain feasible paths dynamically. An evolutionary algorithm is used to search for solutions that cover a given test purpose, which is a transition of interest. The target transition is used as a criterion to get slicing information, in this way, helping to identify the parts of the model that affect the test purpose. We also present a multi-objective search: the test purpose coverage and the sequence size minimization, as longer sequences require more effort to be executed.

- Artigo [148]

  Título: MOST: a Multi-Objective Search-Based Testing from EFSM

  Autores: T. Yano, E. Martins e F. L. de Sousa

  Evento: SBST 2011: 4^{th} Internationl Workshop on Search-Based Software Testing

  Resumo: Neste trabalho é introduzida uma abordagem evolutiva multiobjetivo para a geração de casos de teste a partir de máquina de estado finito estendida (MEFE), denominada MOST. Testes a partir de MEF(E) geralmente envolve a execução de vários caminhos de transições, até que um critério de cobertura (e.g. cobertura de todas as transições) seja satisfeito. Como os métodos de geração de teste tradicionais a partir de MEF apenas consideram os aspectos de controle, caminhos infactíveis podem ser produzidos ao serem aplicados em MEFE, devido aos conflitos nas guardas e ações ao longo de um caminho. A fim de evitar a geração de caminhos infactíveis, uma abordagem é proposta para obter caminhos factíveis dinamicamente, ao invés de realizar estaticamente uma análise de alcançabilidade como normalmente nos métodos baseados em MEF. Trabalhos anteriores têm tratado a geração de casos de teste a partir de MEFE como um problema de otimização mono-objetivo. Na abordagem proposta dois objetivos são considerados: cobertura do critério e tamanho da solução. Desse modo, não é necessário estabelecer a priori o tamanho do caso de teste como nas abordagens anteriores. MOST retorna uma aproximação do conjunto de Pareto, i.e. um grupo de soluções ótimas, que permite que a equipe de testes possa selecionar as soluções que conseguem balancear entre ambos os objetivos. Neste trabalho são apresentados estudos empíricos para ilustrar os benefícios da abordagem e para comparar os resultados com os casos de teste obtidos em um trabalho relacionado.

  Abstract: This paper introduces a multi-objective evolutionary approach to test case generation from extended finite state machines (EFSM), named MOST. Testing from an (E)FSM generally involves executing various transition paths, until a given coverage criterion (e.g. cover all transitions) is met. As traditional test generation methods from FSM only consider the control aspects, they can produce many infeasible paths when applied to EFSMs, due to conflicts in guard conditions along a path. In order to avoid the infeasible path generation, we propose an approach that obtains feasible paths dynamically, instead of performing static reachability analysis as usual for FSM-based methods. Previous works have treated EFSM test case generation as a mono-objective optimization problem. Our approach takes two objectives into account that are the coverage criterion and the solution length. In this way, it is not necessary to

establish in advance the test case size as earlier approaches. MOST constructs a Pareto set approximation, i.e., a group of optimal solutions, which allows the test team to select the solutions that represent a good trade-off between both objectives. The paper shows empirical studies to illustrate the benefits of the approach and comparing the results with the ones obtained in a related work.

- Artigo [149]

  Título: A Multi-Objective Evolutionary Algorithm to Obtain Test Cases With Variable Lengths

  Autores: T. Yano, E. Martins e F. L. de Sousa

  Evento: GECCO 2011: Genetic and Evolutionary Computation Conference

  Resumo: Neste trabalho é apresentada uma implementação multiobjetivo do algoritmo de otimização extrema generalizada (GEO), denominada M-GEO$_{vsl}$. O algoritmo foi desenvolvido primeiramente para ser usado como um gerador de casos de teste para encontrar caminhos a partir de máquina de estado finito estendida (MEFE), levando-se em consideração a transição a ser coberta e o tamanho do caso de teste. M-GEO$_{vsl}$ possui a capacidade de lidar com cadeia de caracteres cujo número de elementos pode variar dinamicamente, tornando possível gerar soluções com diferentes tamanhos. Os passos do algoritmo são descritos para um problema multiobjetivo genérico em que o tamanho da solução é um elemento a ser otimizado. Experimentos foram realizados para gerar casos de teste a partir de MEFE com modelos da literatura usando M-GEO$_{vsl}$ e a abordagem foi comparada com um trabalho relacionado.

  Abstract: In this paper a new multi-objective implementation of the generalized extremal optimization (GEO) algorithm, named M-GEO$_{vsl}$, is presented. It was developed primarily to be used as a test case generator to find transition paths from extended finite state machines (EFSM), taking into account not only the transition to be covered but also the minimization of the test length. M-GEO$_{vsl}$ has the capability to deal with strings whose number of elements vary dynamically, making it possible to generate solutions with different lengths. The steps of the algorithm are described for a general multi-objective problem in which the solution length is an element to be optimized. Experiments were performed to generate test case from EFSM benchmark models using M-GEO$_{vsl}$ and the approach was compared with a related work.

- Artigo [147]

  Título: A Model-based Approach for Robustness Test Generation

  Autores: T. Yano, E. Martins e F. L. de Sousa

  Evento: RACI 2011: 1^{st} International Workshop on Resilience Assessment of Critical Infrastructures

  Resumo: Visto que infraestruturas críticas dependem fortemente de serviços de rede, teste de robustez é um passo importante para garantir a resiliência de tais sistemas na presença de situações anormais. Este trabalho propõe uma abordagem dinâmica de geração de casos de teste a partir de modelos de estado, usando uma versão executável do modelo. Um algoritmo de meta-heurística baseada em busca é utilizado para gerar os casos de teste que levam em consideração os aspectos de controle e dados do modelo.

  Abstract: Since Critical Infrastructures (CI) strongly rely on network services, robustness testing is a key step to assure the resilience of such systems in the presence of abnormal situations. This work proposes a dynamic robustness test cases generation from state models, using an executable version of the model. A meta-heuristics search based algorithm is used to generate test cases taking into account the control and data aspects of the model.

Referências Bibliográficas

[1] O. Abdellatif-Kaddour, P. Thevenod-Fosse, and H. Waeselynck. Property-oriented testing based on simulated annealing. Technical Report 01425, LAAS-CNRS - Laboratoire d'Analyse et d'Architecture des SystÃ¨mes, Toulouse, France, September 2001.

[2] B. T. Abreu. Uma abordagem evolutiva para a geração automática de dados de teste. Master's thesis, IC/Unicamp, Campinas, SP, 2006.

[3] B. T. Abreu, E. Martins, and F. L. Sousa. Automatic test data generation for path testing using a new stochastic algorithm. In *XIX Simpósio Brasileiro de Engenharia de Software*, volume 19, pages 247–262, Uberlândia, Brasil, 2005.

[4] B. T. Abreu, E. Martins, and F. L. Sousa. Generalized extermal optimization: An attractive alternative for test data generation. In *Proc. GECCO '07: Genetic and Evolutionary Computation Conference*, page 1138, 2007.

[5] B. T. Abreu, E. Martins, and F. L. Sousa. Generalized extremal optimization: a competitive algorithm for test data generation. In *21st Brazilian Symposium on Software Engineering*, João Pessoa, Brazil, 2007.

[6] W. Afzal, R. Torkar, and R. Feldt. A systematic review of search-based testing for non-functional system properties. *Inf. Softw. Technol.*, 51:957–976, June 2009.

[7] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege. A systematic review of the application and empirical investigation of search-based test case generation. *IEEE Trans. Softw. Eng.*, 36:742–762, November 2010.

[8] A. M. Ambrosio and E. Martins. Space services: textual to formal description. In *Proceedings of the 8th International Conference on Space Operations*, Rome, Italy, June 2006.

[9] A. M. Ambrosio, E. Martins, N. L. Vijaykumar, and S. V. Carvalho. Systematic generation of test and fault cases for space application validation. In *DASIA 2005: Proceedings of Data Systems in Aerospace*, Edinburg, Escócia, May 2005.

[10] A. M. Ambrosio, E. Martins, N. L. Vijaykumar, and S. V. Carvalho. A conformance testing process for space applications software services. *AIAA (American Institute of Aeronautics and Astronautics) Journal of Aerospace Computing, Information, and Communication*, 3(4):146–158, 2006.

[11] A. M. Ambrosio, F. Mattiello-Francisco, V. A. Santiago, W. P. Silva, and E. Martins. Designing fault injection experiments using state-based model to test a space software. In *Proceedings of Third Latin-American Symposium on Dependable Computing (LADC)*, Morelia, Mexico, September 2007.

[12] K. Androutsopoulos, D. Clark, M. Harman, Z. Li, and L. Tratt. Control dependence for extended finite state machines. In *Proc. FASE '09: 12th Int. Conf. on Fundamental Approaches to Software Engineering*, pages 216–230, 2009.

[13] K. Androutsopoulos, N. Gold, M. Harman, Z. Li, and L. Tratt. A theoretical and empirical study of EFSM dependence. In *Proc. ICSM'09: 25th IEEE Int. Conf. on Software Maintenance*, pages 287–296, September 2009.

[14] W. K. G. Assunç ao, T. E. Colanzi, A. T. R. Pozo, and S. R. Vergilio. Establishing integration test orders of classes with several coupling measures. In *Proc. GECCO '11: Genetic and Evolutionary Computation Conference*, GECCO '11, pages 1867–1874, New York, NY, USA, 2011. ACM.

[15] A. Arcuri and X. Yao. A memetic algorithm for test data generation of object-oriented software. In *IEEE Congress on Evolutionary Computation*, pages 2048–2055, 2007.

[16] P. Bak. *How nature works: the science of self-organized criticality.* Springer-Verlag, New York, 1996.

[17] P. Bak and K. Sneppen. Punctuated equilibrium and criticality in a simple model of evolution. *Physical Review Letters*, 71(24):4083–4086, December 1993.

[18] A. Baresel, H. Pohlheim, and S. Sadeghipour. Structural and functional sequence test of dynamic and state-based software with evolutionary algorithms. In *Proc. GECCO '03: Genetic and Evolutionary Computation Conference*, pages 2428–2441, 2003.

[19] A. Baresel, H. Sthamer, and M. Schmidt. Fitness function design to improve evolutionary structural testing. In *Proc. GECCO '02: Genetic and Evolutionary Computation Conference*, pages 1329–1336, 2002.

[20] L. Baresi, P. L. Lanzi, and M. Miraz. Testful: An evolutionary test approach for java. In *Proceedings of the 2010 Third International Conference on Software Testing, Verification and Validation*, ICST '10, pages 185–194, Washington, DC, USA, 2010. IEEE Computer Society.

[21] B. Beizer. *Software Testing Techniques*. Van Nostrand Eeinhold, New York, 2^{nd} edition, 1990.

[22] A. Bertolino. Software testing research: Achievements, challenges, dreams. In *FOSE '07: 2007 Future of Software Engineering*, pages 85–103, Washington, DC, USA, 2007. IEEE Computer Society.

[23] G. V. Bochmann. Specifications of a simplified transport protocol using different formal description techniques. *Comput. Netw. ISDN Syst.*, 18:335–377, June 1990.

[24] G. v. Bochmann and A. Petrenko. Protocol testing: Review of methods and relevance for software testing. In *International Symposium on Software Testing and Analysis (ISSTA'94)*, pages 109–124, 1994.

[25] S. Boettcher and A. G. Percus. Optimization with extremal dynamics. *Physical Review Letters*, 86:5211–5214, 2001.

[26] C. Bourhfir, R. Dssouli, and E. M. Aboulhamid. Automatic test generation for EFSM-based systems. Technical Report IRO 1043, University of Montreal, Canada, August 1996.

[27] P. M. S. Bueno and M. Jino. Automatic test data generation for paths testing using genetic algorithms. *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, 12(6):691–709, 2002.

[28] O. Bühler and J. Wegener. Evolutionary functional testing of an automated parking system. In *International Conference on Computer, Communication and Control Technologies: CCCT '03 and The 9th. International Conference on Information Systems Analysis and Synthesis: ISAS '03*, Florida, USA, August 2003.

[29] A. V. Buzzo. Estudo de algoritmo evolutivo com codificação real na geração de dados de teste estrutural e implementação de protótipo de ferramenta de apoio. Master's thesis, IC/Unicamp, Campinas, SP, 2011.

[30] A. V. Buzzo, E. Martins, and F. L. De Sousa. Using georeal algorithm for automated test data generation. In *SAST'2010: 4th Brazilian Workshop on Systematic and Automated Software Testing*, pages 109–124, 2010. (in Portuguese).

[31] R. V. Cabral, A. Pozo, and S. R. Vergilio. A pareto ant colony algorithm applied to the class integration and test order problem. In *Proceedings of the 22nd IFIP International Conference on Testing Software and Systems (ICTSS '10)*, volume 6435, pages 16–29, Natal, Brazil, 8-12 November 2010. Springer.

[32] A. B. Carvalho, A. Pozo, and S. R. Vergilio. A symbolic fault-prediction model based on multiobjective particle swarm optimization. *Journal of Systems and Software*, 83:868–882, May 2010.

[33] A. R. Cavalli, D. Lee, C. Rinderknecht, and F. Zaïdi. Hit-or-jump: An algorithm for embedded testing with applications to IN services. In *Proc. of the IFIP TC6 WG6.1*, pages 41–56, Deventer, The Netherlands, 1999. Kluwer, B.V.

[34] CCITT. Specification and description language, 1988. CCITT Z.100, International Consultative Committee on Telegraphy and Telephony, Geneva.

[35] S. T. Chanson and J. Zhu. Automatic protocol test suite derivation. In *INFOCOM '94: Proceedings of 13th IEEE Networking for Global Communications*, volume 2, pages 792–799, Toronto, Ont., Canada, June 1994.

[36] T. S. Chow. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, 4(3):178–187, May 1978.

[37] J. Clarke, J. J. Dolado, M. Harman, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, and M. Roper. Reformulating software engineering as a search problem. In *IEE Proceedings Software*, 2003.

[38] L. A. Clarke. A system to generate test data and symbolically execute programs. *IEEE Transactions on Software Engineering*, 2(3):215–222, 1976.

[39] C. A. Coello Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, 1(3):269–308, 1999.

[40] C. A. Coello Coello. An updated survey of ga-based multiobjective optimization techniques. *ACM Computing Surveys*, 32:109–143, June 2000.

[41] C. A. Coello Coello. Evolutionary multi-objective optimization: a historical view of the field. *IEEE Computational Intelligence Magazine*, 1:28–36, February 2006.

[42] A. M. Davis. A comparison of techniques for the specification of external system behavior. *Communications of the ACM*, 31(9), September 1988.

[43] F. L. De Sousa, , F. M. Ramos, P. Paglione, and R. M. Girardi. Laminar profile design using extremal dynamics optimization. In *WCCM V: Fifth World Congress on Computational Mechanics*, page 311, Vienna, Austria, 2002.

[44] F. L. De Sousa, , F. M. Ramos, P. Paglione, and R. M. Girardi. Algoritmo da otimização extrema generalizada aplicado à aerodinâmica de planadores. In *DINCON 2003: Congresso Temático de Aplicações de Dinâmica e Controle da Sociedade Brasileira de Matemática Aplicada e Computacional*, pages 133–148, São José dos Campos, Brasil, 2003.

[45] F. L. De Sousa, , F. M. Ramos, F. J. C. P. Soeiro, and A. J. Silva Neto. Application of the generalized extremal optimization algorithm to an inverse radiative transfer problem. In *5th International Conference on Inverse Problems in Engineering: Theory and Practice*, volume I, 2005.

[46] F. L. De Sousa. *Otimização Extrema Generalizada: Um novo algoritmo estocástico para o projeto ótimo*. PhD thesis, INPE, São José dos Campos, SP, Brasil, 2002.

[47] F. L. De Sousa, F. M. Ramos, P. Paglione, and R. M. Girardi. New stochastic algorithm for design optimization. *AIAA Journal*, 41(9):1808–1818, September 2003.

[48] F. L. De Sousa and W. K. Takahashi. Discrete optimal design of trusses by generalized extremal optimization. In *6th World Congress in Structural and Multidisciplinary Optimization*, Rio de Janeiro, Brasil, May 2005.

[49] F. L. De Sousa and W. K. Takahashi. Generalized extremal optimization applied to three-dimensional truss design. In *COBEM 2005: Proc. of the International Congress of Mechanical Eng.*, pages 375–376, Ouro Preto, MG, Brazil, November 2005.

[50] F. L. De Sousa, V. Vlassov, and F. M. Ramos. Heat pipe design through generalized extremal optimization. In *ENCIT 2002: IX Brazilian Congress of Engineering and Thermal Sciences*, Caxambu, MG, Brazil, 2002.

[51] F. L. De Sousa, V. Vlassov, and F. M. Ramos. Generalized extremal optimization for solving complex optimal design problems. In *Proc. GECCO '03: Genetic and Evolutionary Computation Conference*, volume 2723 of *Lecture Notes in Computer Science*, pages 375–376, 2003.

[52] F. L. De Sousa, V. Vlassov, and F. M. Ramos. Generalized extremal optimization: An application in heat pipe design. *Applied Mathematical Modelling*, 28(10):911–931, October 2004.

[53] F. L. De Sousa, V. Vlassov, and F. M. Ramos. Heat pipe design through generalized extremal optimization. *Heat Transfer Engineering*, 25(7):34–45, 2004.

[54] K. Derderian, R. Hierons, M. Harman, and Q. Guo. Automated Unique Input Output sequence generation for conformance testing of FSMs. *The computer Journal*, 49(3):331–344, 2006.

[55] K. Derderian, R. Hierons, M. Harman, and Q. Guo. Estimating the feasibility of transition paths in extended finite state machines. *Automated Software Engineering*, 17:33–56, March 2010.

[56] K. Derderian, R. M. Hierons, M. Harman, and Q. Guo. Input sequence generation for testing of communicating finite state machines (CFSMs). In *Proc. GECCO '04: Genetic and Evolutionary Computation Conference*, volume 3103 of *Lecture Notes in Computer Science*, pages 1429–1430, Seattle, WA, USA, June 2004. Springer.

[57] K. Derderian, R. M. Hierons, M. Harman, and Q. Guo. Generating feasible input sequences for extended finite state machines (EFSMs) using genetic algorithms. In *Proc. GECCO '05: Genetic and Evolutionary Computation Conference*, pages 1081–1082, Washington DC, USA, June 2005. ACM.

[58] E. Diaz, J. Tuya, and R. Blanco. Automated software testing using a metaheuristic technique based on tabu search. In *Proceedings of 18th IEEE International Conference on Automated Software Engineering (ASEÅ$\frac{1}{2}$03)*, pages 310– 313, October 2003.

[59] R. Dssouli, K. Saleh, E. Aboulhamid, A. En-Nouaary, and C. Bourhfir. Test development for communication protocols. *Computer Networks*, 31(17):1835–1872, 1999.

[60] A. Y. Duale and M. Ü. Uyar. A method enabling feasible conformance test sequence generation for EFSM models. *IEEE Trans. on Computers*, 53(5):614–627, 2004.

[61] J. Edvardsson. A survey on automatic test data generation. In *Proceedings of the Second Conference on Computer Science and Engineering in Linköping*, pages 21–28. ECSEL, October 1999.

[62] M. Fantinato and M. Jino. Applying extended finite state machines in software testing of interactive systems. In Joaquim Jorge, Nuno Jardim Nunes, and João Falcão e Cunha, editors, *Interactive Systems. Design, Specification, and Verification*, volume 2844 of *Lecture Notes in Computer Science*, pages 109–131. Springer Berlin / Heidelberg, 2003.

[63] R. Ferguson and B. Korel. The chaining approach for software test data generation. *ACM Trans. Softw. Eng. Methodol.*, 5(1):63–86, 1996.

[64] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi. Test selection based on finite state models. *IEEE Transactions on Software Engineering*, 17(6):591–603, June 1991.

[65] M. P. Gallaher and B. M. Kropp. Economic impacts of inadequate infrastructure for software testing. Technical Report RTI Project Number 7007.011 - NIST Planning Report 02-3, University of Montreal, May 2002.

[66] R. L. Galski. *Development of improved, hybrid, parallel and multiobjective versions of the generalized extremal optimization method and its application to the design of spatial systems*. PhD thesis, INPE, São José dos Campos, SP, Brazil, 2006. (in Portuguese).

[67] R. L. Galski, F. L. de Sousa, and F. M. Ramos. Application of a new evolutionary algorithm to the optimum design of a remote sensing satellite constelation. In *5th International Conference on Inverse Problems in Engineering: Theory and Practice*, volume II, Cambridge, UK, July 2005.

[68] R. L. Galski, F. L. de Sousa, F. M. Ramos, and I. Muraoka. Spacecraft thermal design with the generalized extremal optimization algorithm. In *Inverse Problems, Design and Optimization Symposium*, Rio de Janeiro, Brazil, 2004. (in CDROM).

[69] R. L. Galski, F. L. De Sousa, F. M. Ramos, and I. Muraoka. Spacecraft thermal design with the generalized extremal optimization algorithm. *Inverse Problems in Science and Engineering*, 15(1):61–75, January 2007.

[70] W. Geurts, K. Wijbrans, and J. Tretmans. Testing and formal methods - BOS project case study. In *6^{th} European International Conference on Software Testing, Analysis & (ReviewEuroSTAR'98)*, pages 215–229, Munich, Germany, November 1998. Aimware, Mervue, Galway, Ireland.

[71] A. Ghosh and S. Dehuri. Evolutionary algorithms for multi-criterion optimization: A survey. *International Journal of Computing & Information Sciences*, 2(1):38–57, april 2004.

[72] A. Gill. *Introduction to the Theory of Finite-State Machines*. McGraw-Hill, New York, 1962.

[73] G. Gonenc. A method for design of fault-detection experiments. *IEEE Transaction on Computer*, C-19(6):551–558, 1970.

[74] Q. Guo, R. M. Hierons, M. Harman, and K. Derderian. Computing unique input/output sequences using genetic algorithms. In Alexandre Petrenko and Andreas Ulrich, editors, *3rd International Workshop on Formal Approaches to Testing of Software (FATES 2003)*, volume 2931 of *Lecture Notes in Computer Science*, pages 164–177. Springer, 2004.

[75] Q. Guo, R. M. Hierons, M. Harman, and K. Derderian. Constructing multiple unique input/output sequences using metaheuristic optimisation techniques. *IEE Proceedings — Software*, 152(3):127–140, 2005.

[76] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[77] M. Harman. The current state and future of search based software engineering. In *Future of Software Engineering 2007*. IEEE Computer Society, 2007.

[78] M. Harman. Why the virtual nature of software makes it ideal for search based optimization. In *Proceedings of the 13th International Conference on Foundations of Software Science and Computational Structures/Joint European Conferences on Theory and Practice of Software (FASE '10)*, volume 6013, pages 1–12, Paphoe, Cyprus, 20-28 March 2010. Springer.

[79] M. Harman and J. Clark. Metrics are fitness functions too. *Software Metrics, IEEE International Symposium on*, 0:58–69, 2004.

[80] M. Harman and B. F. Jones. Search-based software engineering. *Information & Software Technology*, 43(14):833–839, 2001.

[81] M. Harman, S. A. Mansouri, and Y. Zhang. Search based software engineering: A comprehensive analysis and review of trends techniques and applications. Technical Report TR-09-03, Department of Computer Science, King's College London, April 2009.

[82] M. Harman and P. McMinn. A theoretical & empirical analysis of evolutionary testing and hill climbing for structural test data generation. In *ISSTA '07: Proceedings of the 2007 international symposium on Software testing and analysis*, pages 73–83, New York, NY, USA, 2007. ACM.

[83] M. Harman and P. McMinn. A theoretical and empirical study of search-based testing: Local, global, and hybrid search. *IEEE Trans. Softw. Eng.*, 36:226–247, March 2010.

[84] M. J. Harrold. Testing: a roadmap. In *ICSE - Future of SE Track*, pages 61–72, 2000.

[85] M.P.E. Heimdahl and M.W. Whalen. Reduction and slicing of hierarchical state machines. In *Proceedings of the Fifth ACM SIGSOFT Symposium on the Foundations of Software Engineering*, September 1997.

[86] R. M. Hierons, T.-H. Kim, and H. Ural. On the testability of sdl specifications. *Computer Networks*, 44:681–700, April 2004.

[87] D. Hogrefe. OSI formal specification case study: the inres protocol and service. Technical Report IAM-91-012, University of Bern, 1991.

[88] IEEE 610.12-1990. Ieee standard glossary of software engineering terminology, 1990. Institute of Electrical and Eletronics Engineers.

[89] ISO/IEC. Information processing systems - Open systems interconnection - Lotos - A formal description technique based on the temporal ordring of observational behaviour, 1989. ISO/IEC 8807, Geneva.

[90] ISO/TC97/SC21/WG1/DIS9074. Estelle: A formal description technique based on an extended state transition model, 1987. ISO/TC97/SC21/WG1/DIS9074.

[91] B. Jones, H. Sthamer, X. Yang, and D. Eyres. The automatic generation of software test data sets using adaptive search techniques. In *3rd International Conf. on Software Quality Management*, pages 435–444, Seville, Spain, 1995.

[92] A. S. Kalaji, R. Hierons, and S. Swift. Generating feasible transition paths for from an extended finite state machine (EFSM) with the counter problem. In *Proc. SBST'10: 3rd Int. Workshop on Search-Based Software Testing*, pages 232–235, 2010.

[93] A. S. Kalaji, R. M. Hierons, and S. Swift. Generating feasible transition paths for testing from an extended finite state machine. In *Proc. ICST'09*, pages 230–239, 2009.

[94] A. S. Kalaji, R. M. Hierons, and S. Swift. A search-based technique for testing from extended finite state machine model. Technical report, Brunel University, 2009.

[95] B. Korel. Automated software test data generation. *IEEE Trans. Softw. Eng.*, 16(8):870–879, 1990.

[96] B. Korel, I. Singh, L. Tahat, and B. Vaysburg. Slicing of state-based models. In *Proc. ICSM'03: Int. Conf. on Software Maintenance*, page 34, 2003.

[97] S. Labbé and J.-P Gallois. Slicing communicating automata specifications: polynomial algorithms for model reduction. *Formal Aspects of Computing*, 20(6):563–595, 2008.

[98] K. Lakhotia, M. Harman, and P. McMinn. A multi-objective approach to search-based test data generation. In *Proc. GECCO '07: Genetic and Evolutionary Computation Conference*, pages 1098–1105, 2007.

[99] C. P. Lam, J. Xiao, and H. Li. Ant colony optimisation for generation of conformance testing sequences using a characterising set. In *ACST'07: Proc. of the third conference on IASTED International Conference*, pages 140–146, Anaheim, CA, USA, 2007. ACTA Press.

[100] W. B. Langdon, M. Harman, and Y. Jia. Efficient multi-objective higher order mutation testing with genetic programming. *J. Syst. Softw.*, 83:2416–2430, December 2010.

[101] R. Lefticaru and F. Ipate. Functional search-based testing from state machines. In *Proc. ICST '08*, pages 525–528, 2008.

[102] B.-L. Li, Z.-S. Li, J.-Y. Zhang, and J.-R. Sun. An automated test case generation approach by genetic simulated annealing algorithm. In *Third International Conference on Natural Computation (ICNC 2007)*, volume 4, pages 106–111, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[103] J. J. Li and W. E. Wong. Automatic test generation from communicating extended finite state machine (CEFSM)-based models. In *Proc. ISORC '02: 5th IEEE Int. Symp. on Object-Oriented Real-Time Distributed Computing*, page 181, 2002.

[104] X. Li, T. Higashino, M. Higuchi, and K. Taniguchi. Automatic test case derivation for communication protocols in an extended FSM model. *Electronics and Communications in Japan (Part I: Communications)*, 82(10):50–60, 1999.

[105] K. Liaskos and M. Roper. Hybridizing evolutionary testing with artificial immune systems and local search. In *Proceedings of the 2008 IEEE International Conference on Software Testing Verification and Validation Workshop*, pages 211–220, Washington, DC, USA, 2008. IEEE Computer Society.

[106] J.-C. Lin and P.-L. Yeh. Automatic test data generation for path testing using GAs. *Information Sciences*, 131(1-4):47–64, 2001.

[107] C. L. B. Maia, F. G. de Freitas, and J. T. de Souza. Applying search-based techniques for requirements-based test case prioritization. In *Proceedings of the Brazilian Workshop on Optimization in Software Engineering (WOES '10)*, Salvador, Brazil, 30-30 September 2010.

[108] T. Mantere and J. T. Alanderb. Evolutionary software engineering, a review. *Applied Soft Computing*, 5:315–331, 2005.

[109] A. Marchetto and P. Tonella. Using search-based algorithms for ajax event sequence generation during testing. *Empirical Softw. Engg.*, 16:103–140, February 2011.

[110] E. Martins, S. B. Sabião, and A. M. Ambrosio. ConData: a tool for automating specification-based test case generation for communication systems. In *33rd Hawaii International Conference on System Sciences*, Maui, Hawaii, January 2000.

[111] P. McMinn. Search-based software test data generation: a survey. *Software Testing, Verification and Reliability*, 14(2):105–156, 2004.

[112] P. McMinn, M. Harman, D. Binkley, and P. Tonella. The species per path approach to search-based test data generation. In *Proc. ISSTA'06*, pages 13–24, 2006.

[113] P. McMinn and M. Holcombe. Evolutionary testing of state-based programs. In *Proc. GECCO '05: Genetic and Evolutionary Computation Conference*, pages 1013–1020, 2005.

[114] C.C. Michael, G. McGraw, and M. A. Schatz. Generating software test data by evolution. *IEEE Transactions on Software Engineering*, 27(12):1085–1110, December 2001.

[115] Z. Michalewicz, R. Hinterding, and M. Michalewicz. Evolutionary algorithms. In Witold Pedrycz, editor, *Fuzzy Evolutionary Computation*, pages 3–31. Kluwer Academic, Boston, 1997.

[116] E. F. Moore. Gedanken-experiments on sequential machines. *Automata Studies*, pages 129–153, 1956.

[117] G. Myers. *The Art of Software Testing*. Wiley, New York, 1979.

[118] A. J. Offutt. An integrated automatic test data generation system. *Journal of Systems Integration*, 1(3-4):391–409, November 1991.

[119] N. Oster and F. Saglietti. Automatic test data generation by multi-objective optimisation. In *Computer Safety, Reliability, and Security, LNCS 4166*, pages 426–438, 2006.

[120] J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[121] G. H. Lima Pinto and S. R. Vergilio. A multi-objective genetic algorithm to test data generation. In *Proc. ICTAI'10: International Conference on Tools with Artificial Intellingence*, pages 129–134, 2010.

[122] R. S. Pressman. *Software Engineering — A Practitioner's Approach*. McGraw-Hill, 5^{th} edition, 2000.

[123] T. Ramalingom, K. Thulasiraman, and A. Das. Context independent unique state identification sequences for testing communication protocols modelled as extended finite state machines. *Computer Communications*, 26(14):1622–1633, September 2003.

[124] H. Robinson. Intelligent test automation. *Software Testing & Quality Engineering*, pages 24–32, September 2000.

[125] K. K. Sabnani and A. Dahbura. A protocol test generation procedure. *Computer Networks and ISDN Systems*, 15(4):285–297, 1988.

[126] B. Sarikaya, G. v. Bochmann, and E. Cerny. A test design methodology for protocol testing. *IEEE Transactions on Software Engineering*, 13(5):518–531, May 1987.

[127] P. Tonella. Evolutionary testing of classes. In *Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, ISSTA '04, pages 119–128, New York, NY, USA, 2004. ACM.

[128] N. Tracey, J. Clark, and K. Mander. Automated program flaw finding using simulated annealing. In *ISSTA '98: Proc. of the ACM SIGSOFT international symposium on Software testing and analysis*, pages 73–81. ACM Press, 1998.

[129] N. Tracey, J. Clark, J. McDermid, and K. Mander. A search-based automated test-data generation framework for safety-critical systems. *Systems engineering for business process change: new directions*, pages 174–213, 2002.

[130] J. Tretmans and A. Belinfante. Automatic testing with formal methods. In *EuroSTAR'99: 7^{th} European International Conference on Software Testing, Analysis & Review*, Barcelona,Spain, November 1999.

[131] H. Ural and B. Yang. A test sequence selection method for protocol testing. *IEEE Transactions on Communications*, 39(4):514–523, April 1991.

[132] M. Utting and B. Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[133] M. Utting, A. Pretschner, and B. Legeard. A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability*, pages n/a–n/a, 2011.

[134] D. A. Van Veldhuizen and G. B. Lamont. Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. *Evolutionary Computation*, 8:125–147, June 2000.

[135] V. V. Vlassov, F. L. De Sousa, and W. K. Takahashi. Comprehensive optimization of a heat pipe radiator assembly filled with ammonia or acetone. *International Journal of Heat and Mass Transfer*, 49(23-24):4584 – 4595, 2006.

[136] S. Wagner and J. Jürjens. Model-based identification of fault-prone components. In *Proc. EDCC-5: Fifth European Dependable Computing Conference, volume 3463 of LNCS*, pages 435–452. Springer, 2005.

[137] T. Waheed, M. Iqbal, and Z. Malik. Data flow analysis of uml action semantics for executable models. In Ina Schieferdecker and Alan Hartman, editors, *Model Driven Architecture - Foundations and Applications*, volume 5095 of *Lecture Notes in Computer Science*, pages 79–93. Springer Berlin / Heidelberg, 2010.

[138] Y. Wang. Finding executable paths in protocol conformance testing. Master's thesis, Simon Fraser University, 1990.

[139] Y. Wang, Z. Bai, M. Zhang, W. Du, Y. Qin, and X. Liu. Fitness calculation approach for the switch-case construct in evolutionary testing. In *Proc. GECCO '08: Genetic and Evolutionary Computation Conference*, pages 1767–1774, New York, NY, USA, 2008. ACM.

[140] Z. Wang, K. Tang, and X. Yao. Multi-objective approaches to optimal testing resource allocation in modular software systems. *Reliability, IEEE Transactions on*, 59(3):563 –575, 2010.

[141] J. Wegener, A. Baresel, and H. Sthamer. Evolutionary test environment for automatic structural testing. *Information & Software Technology*, 43(14):841–854, 2001.

[142] J. Wegener, K. Buhr, and H. Pohlheim. Automatic test data generation for structural testing of embedded software systems by evolutionary testing. In *Proc. GECCO*

'02: Genetic and Evolutionary Computation Conference, pages 1233–1240, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[143] D. H. Wolpert and W.G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, The Santa Fe Institute, Santa Fe, NM, 1995.

[144] T. Yano, E. Martins, and F. L. De Sousa. Aplicação de algoritmos evolutivos na geração automática de dados de teste de conformidade. In *WTF 2008: Worshop de Testes e Tolerância a Falhas*, pages 43–56, 2008.

[145] T. Yano, E. Martins, and F. L. De Sousa. An evolutionary approach for robustness testing. In *Proc. ICEC'09: Int. Conf. on Evolutionary Computation*, pages 277–280, 2009.

[146] T. Yano, E. Martins, and F. L. De Sousa. Generating feasible test paths from an executable model using a multi-objective approach. In *Proc. SBST'10: 3rd Int. Workshop on Search-Based Software Testing*, pages 236–239, 2010.

[147] T. Yano, E. Martins, and F. L. De Sousa. A model-based approach for robustness test generation. In *Proc. RACI'11: 1st International Workshop on Resilience Assessment of Critical Infrastructures*, pages 33–34, 2011.

[148] T. Yano, E. Martins, and F. L. De Sousa. MOST: a multi-objective search-based testing from EFSM. In *Proc. SBST'11: 4th Int. Workshop on Search-Based Software Testing*, pages 164–173, 2011.

[149] T. Yano, E. Martins, and F. L. De Sousa. A multi-objective evolutionary algorithm to obtain test cases with variable lengths. In *Proc. GECCO '11: Genetic and Evolutionary Computation Conference*, pages 1875–1882, New York, NY, USA, 2011. ACM.

[150] S. Yoo and M. Harman. Pareto efficient multi-objective test case selection. In *Proc. ISSTA '07*, pages 140–150, 2007.

[151] R. Zhao, M. Harman, and Z. Li. Empirical study on the effiency of search based test generation for EFSM models. In *Proc. SBST'10: 3rd Int. Workshop on Search-Based Software Testing*, pages 222–231, 2010.

[152] E. Zitzler, M. Laumanns, and S. Bleuler. A Tutorial on Evolutionary Multiobjective Optimization. In X. Gandibleux et al., editors, *Metaheuristics for Multiobjective Optimisation*, Lecture Notes in Economics and Mathematical Systems. Springer, 2004.