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

Thaise Yano[1], **Eliane Martins**[1], **Fabiano Luís de Sousa**[2]

[1]Instituto de Computação – Universidade Estadual de Campinas (Unicamp)
Caixa Postal 6176 – 13083-970 – Campinas – SP – Brasil

[2]Instituto Nacional de Pesquisas Espaciais (INPE)
São José dos Campos – SP – Brasil

tyano@ic.unicamp.br, eliane@ic.unicamp.br, fabiano@dem.inpe.br

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.*

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 sobre 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.*

1. Introdução

A atividade de teste tem sido apontada como uma das mais onerosas no desenvolvimento de software [Harrold 2000]. Um relatório preparado para o NIST (*National Institute of Standards and Technology*) [Gallaher and Kropp 2002] quantifica os altos impactos econômicos de uma infra-estrutura de teste de software inadequada e como melhoria cita, por exemplo, a geração automática de teste.

Bertolino [Bertolino 2007] descreve os desafios mais relevantes na pesquisa de teste de software, entre eles a geração de dados 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 baseada em busca [McMinn 2004, Harman 2007] na geração de dados de teste. Outro desafio citado é o teste baseado em modelos, sendo que a idéia principal é usar modelos no desenvolvimento de software para auxiliar o processo de teste, particularmente, na geração automática de casos de teste.

Dessa maneira, um dos problemas importantes na atividade de teste de software é o desenvolvimento de dados de teste, que consiste em identificar dados de entrada que satisfaçam um determinado critério de teste. Automatizar a geração de dados de teste contribuiria para a redução de custo e esforços ao processo de desenvolvimento de software. Nesse sentido, as principais abordagens para a geração automática de dados de teste são [Michael et al. 2001]: aleatória, simbólica e dinâmica. No teste aleatório, dados de teste são selecionados aleatoriamente do domínio de entrada, mas a probabilidade de se selecionar dados que pertençam a uma pequena porcentagem do domínio de entrada é muito baixa. Por outro lado, é comumente utilizado na literatura como uma medida de comparação por ser um método simples e de fácil implementação. As abordagens que utilizam a execução simbólica [Offutt 1991] atribuem valores simbólicos às variáveis, ao invés de usar seus reais valores, e reduzem o problema de geração de dados 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 *loops*, vetores e referências a ponteiros que limitam seu uso nos dias de hoje [Michael et al. 2001, Korel 1990]. Já a abordagem dinâmica de geração de dados de teste baseia-se na execução dos valores reais das variáveis. Quando um requisito de teste não é satisfeito, uma função objetivo é associada a ele e métodos de otimização de funções são utilizados para buscar dados que mais se aproximam de satisfazer o requisito. Com base nessas informações, os dados de teste são incrementalmente modificados até que um deles satisfaça o requisito. Assim a geração de teste é reformulada como um problema de otimização. Como métodos de otimização utilizados nessa abordagem incluem o recozimento simulado [Tracey et al. 1998], método do gradiente [Korel 1990], *hill climbing* [Harman and McMinn 2007] e algoritmos evolutivos [McMinn 2004].

A área de pesquisa que investiga o uso de algoritmos evolutivos na atividade de teste é denominada como testes evolutivos. Nos últimos anos houve um grande crescimento no interesse por essa área, sendo aplicada para diferentes técnicas e critérios de teste. Muitos trabalhos focam o teste estrutural [Abreu 2006, Harman and McMinn 2007, Michael et al. 2001, Pargas et al. 1999, Watkins and Hufnagel 2006], mas também abordam com menor intensidade o teste funcional [Derderian et al. 2006, Guo et al. 2005, Jones et al. 1995, Tracey et al. 1998]. Entre os algoritmos evolutivos mais utilizados na geração de dados de teste está o algoritmo genético (AG). Porém outros algoritmos têm sido propostos para o teste evolutivo que apresentam a vantagem de possuir menos parâmetros a serem ajustados que o AG, tal como o GEO (*Generalized Extremal Optimization*) [DeSousa 2002]. Isso facilita o processo de configuração do algoritmo para obter um melhor desempenho. O GEO foi competitivo com outros métodos estocásticos em funções-teste [DeSousa et al. 2003a] e tem sido aplicado com sucesso em problemas reais de projeto ótimo [Galski et al. 2007, DeSousa et al. 2003a, DeSousa et al. 2003b, DeSousa et al. 2004]. No trabalho de Abreu [Abreu 2006] foi explorado pela primeira vez o uso do GEO em uma atividade da engenharia de software. Os estudos de casos mostraram que o GEO também é competitivo com o algoritmo genético na geração de dados de teste estrutural, exigindo menos esforço computacional para tal. Isso motivou a aplicação do GEO como uma opção interessante a ser utilizada na geração de dados de testes funcionais. Então neste trabalho é realizada uma investigação inicial do algoritmo GEO no teste de especificações baseadas em modelos, particularmente, Máquina de Estados Finitos (MEF). A abordagem é comparada com o teste aleatório.

O artigo está organizado da seguinte forma. Na Seção 2 é descrito o teste baseado em modelo e é apresentada a MEF. Na Seção 3 é apresentado o conceito de testes evolutivos. Na Seção 4 é descrito o algoritmo GEO. Na Seção 5 é mostrada a proposta de aplicação do GEO no contexto de teste de conformidade. Finalmente na Seção 6 conclui-se o trabalho e são apresentados os trabalhos futuros.

2. Teste baseado em modelo

O teste de conformidade é um teste baseado em modelo que tem sido aplicado com o objetivo de investigar se a implementação se comporta de acordo com a especificação, buscando detectar discrepâncias entre uma especificação e suas implementações. O teste de conformidade baseia-se principalmente nas abordagens de caixa-preta. Então os casos de testes são gerados com base na especificação e são utilizados no teste da implementação. Portanto os testes são dependentes do formalismo da especificação. O uso de uma especificação descrita em linguagem natural passa a ilusão de ser facilmente entendida, porém induz especificações longas e informais que geralmente contêm ambigüidades e dificultam a verificação de completitude e consistência [Bochmann and Petrenko 1994]. Assim justifica-se o uso de métodos formais, tal como as MEFs, para especificação de sistemas que requerem alto grau de confiabilidade e segurança.

Geurts et al. [Geurts et al. 1998] relatam um estudo de caso em que se obtiveram benefícios na fase de teste com o uso de métodos formais. Neste estudo de caso, devido à precisão, consistência e completitude das especificações formais, casos de teste eficientes e efetivos puderam ser sistematicamente derivados dessas especificações.

2.1. MEF

Uma Máquina de Estados Finitos (MEF) [Gill 1962] é uma máquina hipotética 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 [Davis 1988].

Uma MEF pode ser definida como $M = (S, s_0, I, O, T)$ em que: S é um conjunto finito e não vazio de estados; $s_0 \in S$ é o estado inicial; I é um conjunto finito de entradas; O é um conjunto finito de saídas; e $T : D_s \rightarrow S \times O$ é a função de transição, onde $D_s \subseteq S \times I$ é o domínio da especificação. A função de transição representa as possíveis transições da máquina. A expressão $t(s_i, x) = (s_n, y)$ representa a transição do estado s_i para o estado s_n quando o símbolo de entrada x é recebido, produzindo a saída y.

Quando $D_s \subset S \times I$, T não é definida para cada par (s, x) e representam-se máquinas parcialmente especificadas. Para tais máquinas, uma suposição de completitude deve ser adotada a fim de especificar seu comportamento para entradas inesperadas. Por exemplo, pode-se ignorar as entradas inesperadas produzindo uma saída nula e permanecer no estado atual. Outra propriedade das máquinas é o determinismo, no qual para cada entrada existe no máximo uma transição definida em cada estado da máquina.

Uma MEF pode ser representada por um grafo direcionado. Um grafo $G(V, E)$ é uma estrutura definida por dois conjuntos não vazios: nós (V) e arestas (E). Um par ordenado de nós (v_i, v_n) de um grafo direcionado descreve uma aresta $e_k \in E$ que parte de v_i e termina em v_n. Um caminho em $G(V, E)$ é uma seqüência não vazia de aresta. Se

o nó inicial e final de um caminho são os mesmos, então o caminho forma um *loop*. Em particular, os nós representam os estados e as arestas representam as transições da MEF.

2.2. Geração de teste

O teste baseado em MEF é realizado usando-se seqüências de teste que consistem de uma seqüência de entradas (e eventualmente das saídas correspondentes) a partir do estado inicial da máquina.

Os métodos de geração de casos de teste que exploram o aspecto de controle das MEFs procuram encontrar falhas de saída e de transferência na implementação em teste. Vários métodos para a geração de seqüências de teste de MEF são encontrados na literatura, tais como [Bourhfir et al. 1996]: DS - *Distinguishing Sequence*, Método W, UIO - *Unique-Input-Output* e Método Wp. Tais métodos baseiam-se em propriedades e seqüências especiais das máquinas para a determinação de um conjunto de casos de teste que seja pequeno e, ao mesmo tempo, consiga a cobertura do maior número possível de falhas contidas em uma máquina [Fujiwara et al. 1991]. As propriedades das máquinas que são exigidas para a aplicação dos métodos, na prática, são muito difíceis de serem satisfeitas, limitando o uso desses métodos.

3. Testes evolutivos

Nesta seção são apresentados o conceito de testes evolutivos e os trabalhos relacionados de teste funcional deste contexto.

3.1. Princípio

Recentemente, houve um grande interesse sobre teste baseado em busca (em inglês, *Search Based Testing*, SBT), que procura aplicar técnicas de otimização na geração de dados de teste. A idéia geral é que o conjunto das entradas possíveis forma um espaço de busca e o critério de teste é codificado como uma função objetivo. Essa abordagem natural torna SBT bastante atrativa para tratar diferentes problemas de geração de dados de teste, simplesmente mudando-se o espaço de busca e a função objetivo. Para cobertura de instruções, por exemplo, a função objetivo avaliaria quão próxima uma entrada de teste está de executar uma instrução não coberta.

Independente da técnica de otimização utilizada, apenas dois aspectos são necessários para aplicá-la em problemas de engenharia de software [Harman 2007]:

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

Ênfase deve ser dada à função objetivo, pois captura as informações cruciais para a otimização. Através dos valores da função objetivo é que guia-se a busca, diferenciando uma boa solução de uma ruim [Harman 2007]. 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 [McMinn 2004].

Diferentes técnicas de otimização têm sido aplicadas no SBT, entre as mais utilizadas estão os algoritmos evolutivos. Essa aplicação de algoritmos evolutivos na geração

de dados de teste, referenciada normalmente na literatura como Testes Evolutivos, consiste em otimizar uma determinada entrada de acordo com o critério de teste expresso em uma função objetivo. 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. Os valores das funções objetivos são obtidos pela comparação das soluções da busca em relação ao seu objetivo geral, direcionando a busca para regiões promissoras do espaço de busca [McMinn 2004]. A seguir são apresentados alguns trabalhos relacionados aos testes evolutivos, no contexto de teste funcional.

3.2. Teste evolutivo: teste funcional

Algoritmos evolutivos têm sido aplicados na geração de dados de diferentes técnicas de teste, entre a mais investigada está o teste estrutural, mas também encontram-se esforços para o teste funcional. Nesse caso, os testes podem ser derivados de diferentes formas de especificação: notação Z [Jones et al. 1995], SPARK-Ada [Tracey et al. 1998] e MEF [Guo et al. 2004, Guo et al. 2005, Derderian et al. 2006].

Em relação aos trabalhos que derivam os teste a partir de MEFs, é investigada a construção de seqüência UIO (*Unique-Input-Output*) utilizando AG. As seqüências UIO são usadas no teste de MEF para verificar o estado final de uma seqüência de transição. Guo et al. [Guo et al. 2005] propõem uma função objetivo que guia a busca para potenciais seqüências UIO. Uma regra é definida para calcular o grau de similaridade entre os indivíduos, punindo 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 busca resolver o problema de encontrar ótimos locais[1] apresentado em seu trabalho anterior [Guo et al. 2004].

Derderian et al. [Derderian et al. 2006] também descrevem um método para gerar UIOs de MEF utilizando AG. Porém, em relação ao trabalho de Guo et al. [Guo et al. 2004], há a diferença da máquina poder ser parcialmente especificada, uma vez que assumia-se uma máquina completamente especificada. Para tanto, as transições que faltam são ignoradas, mantendo a máquina no mesmo estado e a próxima entrada na seqüência é considerada. Nesse caso o valor de aptidão é penalizado. Além disso, a função objetivo é mais simples e utiliza uma classificação do número de ocorrência dos pares entrada/saída da máquina. O valor de aptidão é calculado como a soma das posições na classificação dos pares de entrada/saída que compõem a seqüência.

Neste trabalho, os casos de teste também são derivados a partir de uma MEF. Porém diferentemente dos trabalhos descritos acima nos quais a função objetivo busca encontrar seqüências UIO, a abordagem proposta neste artigo tem como finalidade cobrir um conjunto de transições determinadas pelo usuário. A abordagem é melhor descrita na Seção 5. Outra diferença é o algoritmo evolutivo adotado, sendo utilizado neste trabalho o algoritmo GEO, ao invés do AG. Na seção a seguir é apresentado o GEO.

4. GEO

Um algoritmo evolutivo proposto recentemente é o de Otimização Extrema Generalizada (em inglês, *Generalized Extremal Optimization*, GEO) [DeSousa 2002,

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

DeSousa et al. 2003a]. Esse algoritmo foi desenvolvido como uma generalização do método da Otimização Extrema (em inglês, *Extremal Optimization*, EO) [Boettcher and Percus 2001], com o objetivo de tornar sua implementação independente do tipo de problema que está sendo atacado. Os processos internos tanto do EO quanto do GEO foram inspirados em um modelo simplificado de seleção natural para mostrar evidências da presença de Criticalidade Auto-Organizada (em inglês, *Self-Organized Criticality*, SOC) em ecossistemas naturais [Bak and Sneppen 1993].

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:

$$P(s) \sim s^{-\tau} \tag{1}$$

onde τ é um parâmetro positivo. Nota-se que perturbações de menor tamanho ocorrem mais freqüentemente 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, economia e biologia [Bak 1996].

Bak e Sneppen [Bak and Sneppen 1993] desenvolveram um modelo simplificado de um ecossistema no qual as espécies são representadas lado a lado e possuem uma relação de vizinhança. A Figura 1 mostra n espécies, sendo que e_n é vizinho de e_{n-1} e e_1, e_1 é vizinho de e_n e e_2, e assim sucessivamente. Para cada espécie é associado um valor de aptidão no domínio $[0, 1]$. Aquela com pior valor de aptidão é considerada a espécie menos adaptada e o processo de evolução força esta e seus vizinhos a mudar. 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, toda população apresenta 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 (1). Um método de otimização baseado nesse modelo poderia evoluir soluções rapidamente, sistematicamente modificando as espécies menos adaptadas e, ao mesmo tempo, poderia investigar regiões diferentes do espaço de projeto através das avalanches, escapando de mínimos locais.

Figura 1. População de espécies no modelo de Bak e Sneppen

Boettcher e Percus [Boettcher and Percus 2001] propuseram o método EO inspirado no modelo de Bak-Sneppen com o objetivo de tratar problemas difíceis de otimização

combinatória. O valor de aptidão é associado a cada variável de projeto, ao contrário do AG, no qual uma configuração de variáveis é associada a uma solução. Entretanto, os autores observaram que uma definição geral para o valor de aptidão para as variáveis individuais pode se mostrar ambígua ou mesmo impossível, sendo necessária uma nova definição do valor de aptidão 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 que incluem variáveis contínuas, discretas, inteiras ou uma combinação destas.

No GEO, cada bit é uma espécie e uma cadeia (*string*) binária é considerada uma população de espécies. A cadeia codifica as M variáveis de projeto e um valor de aptidão é associado a cada bit da cadeia. Esta é a diferença com relação ao EO, cujo valor de aptidão é associado a cada variável de projeto. A Figura 2 mostra as variáveis x e y codificadas em uma cadeia representada por quatro espécies no GEO e duas espécies no EO. Uma população de m bits que constituem uma cadeia representa uma solução para o problema.

Figura 2. Representação de uma população no GEO (4 espécies e 2 variáveis de projeto) e no EO (2 espécies e 2 variáveis de projeto)

Uma descrição resumida dos passos do GEO, adaptada de [DeSousa 2002], encontra-se a seguir:

1. Inicialize aleatoriamente a população de L espécies (bits) que codificam M variáveis de projeto.
2. Para cada bit atribua um índice de adaptabilidade que seja proporcional ao ganho ou perda que a função objetivo tem ao realizar uma mutação no valor do bit (de '0' para '1' ou de '1' para '0'), comparado com um dado valor de referência. O valor do bit retorna ao seu valor original.
3. Ordene os bits de acordo com seu índice de adaptabilidade, sendo o primeiro o menos adaptado. Se dois ou mais bits possuírem o mesmo índice de adaptabilidade, ordene-os aleatoriamente com distribuição uniforme.
4. Escolha com probabilidade uniforme um bit candidato i para sofrer mutação. Gere um número aleatório RAN com probabilidade uniforme no domínio [0,1]. Se a probabilidade de mutação para o bit i calculada por $P_i(k) = k^{-\tau}$, onde k é a posição de i na ordenação e τ um parâmetro ajustável e positivo, for igual ou menor que RAN, o bit i é modificado. Senão, escolha um novo bit candidato e repita o processo até que um bit seja modificado.
5. Repita os passos de 2 a 4 até que um critério de parada seja satisfeito.
6. Retorne a melhor configuração de bits (solução) encontrada durante a busca.

Em uma implementação prática do algoritmo descrito acima, a primeira decisão a ser tomada é definir o número de bits necessário para representar cada variável de projeto. Isso depende do domínio e precisão que se deseja para o valor de cada variável. Além

disso, também deve-se definir o valor do parâmetro τ antes de iniciar a busca. Esse é o único parâmetro ajustável do GEO, sendo que para cada problema existe um que torna a busca mais eficiente. Por exemplo, se $\tau \rightarrow \infty$, somente o primeiro bit da ordenação sofrerá mutação em cada iteração do algoritmo, o que é equivalente a uma busca totalmente determinística. Por outro lado, se $\tau \rightarrow 0$, qualquer bit escolhido como candidato (estando ou não em uma boa posição na ordenação) sofrerá mutação. A experiência tem mostrado que o melhor valor de τ para um dado problema varia normalmente entre 0.75 e 5.0. Ter apenas um único parâmetro livre a ser ajustado pode ser considerado *a priori* uma vantagem em comparação a outros algoritmos estocásticos, uma vez que estes normalmente possuem mais parâmetros livres, demandando mais esforço e tempo para serem ajustados.

5. Método proposto

O objetivo deste trabalho é gerar dados para o teste de conformidade, tendo MEF como especificação, através do algoritmo GEO. O critério de teste adotado é cobrir um determinado conjunto de transições, que podem ser não seqüenciais. O motivo pela escolha desse critério, ao invés de cobrir todas as transições, é que pretende-se gerar seqüências que cubram transições que sejam críticas ao sistema em teste. Por exemplo, pode-se testar as transições que correspondem às entradas inválidas para realizar o teste de robustez do sistema.

Na Figura 3 é apresentada uma ilustração do procedimento de geração dos dados para o teste de conformidade. É utilizada a ferramenta SMC[2] (*State Machine Compiler*) capaz de gerar um código que faz a simulação de uma máquina de estados nas linguagens Java, C++ e Perl, por exemplo. Como primeiro passo, uma MEF M é passada para a ferramenta SMC que, por sua vez, gera um código Java P que simula essa máquina. O código gerado é instrumentado para informar em quais transições uma seqüência de entradas passou. Em seguida, dado um conjunto de transições alvo T_{alvo}, o GEO começa a gerar dados de teste tentando cobrí-lo. Cada dado gerado é uma seqüência de entrada para M. O caminho coberto pela seqüência é obtido graças à instrumentação inserida em P e será entrada para a função objetivo que calculará a cobertura desse caminho em relação a T_{alvo}. O critério de parada do GEO é (i) quando se atinge um número máximo de avaliações da função objetivo ou (ii) quando todo conjunto T_{alvo} for coberto. Caso o critério de parada não seja satisfeito, o GEO reinicia o processo de geração de dados.

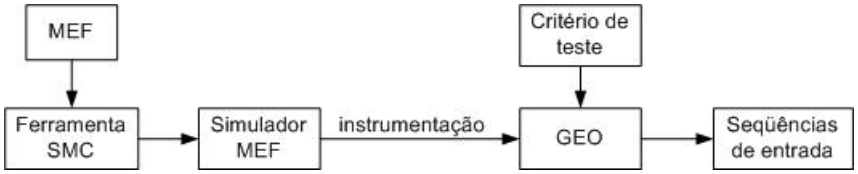

Figura 3. Ilustração da abordagem utilizada.

Antes de iniciar o processo para gerar dados de teste, deve-se definir em relação às variáveis de projeto: a quantidade, o tipo, o domínio que especifica o limite superior e inferior para seus valores e a precisão, como também deve-se definir o valor do único parâmetro ajustável do GEO τ e o número máximo de avaliações da função objetivo. É recomendável fazer um ajuste fino do parâmetro τ, visto que cada sistema pode ter uma ca-

[2]Disponível em http://smc.sourceforge.net.

racterística que faz com que valores diferentes deste parâmetro tenham grande influência na eficiência da geração dos dados de teste. Neste trabalho, cada variável de projeto representa um evento de entrada $x \in I$ da máquina e, consequentemente, a população é uma seqüência de eventos de entrada. O número de variáveis de projeto determina o tamanho da seqüência de teste gerada. O número de bits m necessário para representar cada variável de projeto depende do domínio das variáveis e da precisão p desejada, sendo dado pela inequação a seguir:

$$2^m \geq \left[\frac{max_j - min_j}{p} + 1 \right] \tag{2}$$

onde min_j e max_j são os limites inferior e superior da variável j, respectivamente. Caso o valor de m não seja um valor inteiro, então m passa a ser o próximo número inteiro imediatamente superior a ele. O valor real v_j da variável de projeto j é obtido pela equação abaixo:

$$v_j = min_j + (max_j - min_j) \cdot \left[\frac{Int_j}{2^m - 1} \right] \tag{3}$$

onde Int_j é o número inteiro obtido na transformação da variável j de sua representação binária para decimal.

5.1. Estudo de caso

Um estudo de caso da abordagem foi feito para o protocolo WTP (*Wireless Transaction Protocol*), uma das camadas do WAP (*Wireless Application Protocol*) cujo objetivo é oferecer acesso à Internet para dispositivos móveis, como telefone celular. WTP é um protocolo de transação confirmada que provê os serviços essenciais para aplicações interativas pedido/resposta. O protocolo permite balancear o grau de confiabilidade de um servidor de transação. O processo que inicia uma transação é denominado de *Initiator*, enquanto que o receptor é denominado de *Responder*.

As primitivas de serviço relativas à camada superior são: `TR-Invoke` (inicia uma nova transação), `TR-Result` (retorna um resultado de uma transação iniciada anteriormente) e `TR-Abort` (aborta uma transação existente). Os tipos de primitivas definidos são: (i) `req`, indica um pedido de um serviço da camada superior; (ii) `ind`, usada pela camada que provê o serviço para indicar à próxima camada superior as atividades relacionadas ao *peer* ou ao provedor do serviço; (iii) `res`, indica o recebimento do tipo de primitiva `ind` da próxima camada inferior; e (iv) `cnf`, informa que a atividade foi completada com sucesso.

Uma unidade de dado do protocolo (PDU) é composta por um cabeçalho e um dado (opcional). O cabeçalho contém uma parte fixa e uma variável. A parte fixa do cabeçalho contém o tipo de PDU e normalmente utiliza parâmetros. Os tipos básicos de PDU são: `invoke` (o *Initiator* inicia uma transação), `ack` (confirmação de uma mensagem recebida), `result` (o *Responder* responde a um `TR-Invoke` enviado pelo *Initiator* e `abort` (usado para abortar uma transação). Para maiores informações sobre o WTP, o protocolo [WAP Forum 2001] pode ser consultado.

A fim de testar o *Responder*, uma MEF M_1 foi construída considerando apenas suas operações. Embora exista um fluxo de dados no WTP, tais como variáveis e

parâmetros, somente o fluxo de controle foi levado em consideração até o momento na MEF. A máquina é determinística e parcialmente especificada, ignorando-se eventos de entrada que não foram especificados para um estado e mantendo-a no mesmo estado. A MEF M_1 possui 6 estados e 45 transições. Devido ao tamanho da máquina, esta não será apresentada neste artigo. O conjunto de transições a serem cobertas é aquele correspondente às exceções especificadas na documentação do protocolo. Por exemplo, pretende-se cobrir as transições correspondentes ao recebimento de um `abort` no *Responder*.

5.2. Função objetivo

Dado um conjunto de transições T_{alvo} a serem cobertas e uma seqüência de entrada seq da máquina, a função objetivo verifica as transições disparadas por seq e calcula o número de transições n comuns a T_{alvo}. O valor de aptidão de seq é definido como: $n/|T_{alvo}|$, em que $|T_{alvo}|$ é a cardinalidade do conjunto T_{alvo}. Assim os valores da função objetivo estão no domínio de [0,1]. Quando nenhuma transição de T_{alvo} é coberta por seq, o valor de aptidão é zero. Já seu valor máximo será 1 quando todas as transições de T_{alvo} forem cobertas por seq.

Deve-se ressaltar que outros critérios de adequabilidade poderiam ser utilizados, assim outras funções objetivo poderiam ser definidas para atender esses critérios.

5.3. Resultados

Como o desempenho do GEO varia significativamente com o valor do parâmetro ajustável τ, uma série de experimentos foi realizada para configurá-lo adequadamente ao problema do teste de conformidade. Foram realizados 10 experimentos para diferentes números de variáveis de projeto, que corresponde ao tamanho da seqüência de teste gerada, variando entre 4 e 24, com incremento de 4. Cada experimento é equivalente a 50 execuções com uma configuração de parâmetro. Para cada execução, o número máximo de avaliações da função objetivo é 25000. O valor do parâmetro τ inicia-se com zero e é incrementado de 0.25 em cada experimento até atingir valor 5, então $0.75 \leq \tau \leq 5$. O gráfico da Figura 4 mostra a influência de τ no desempenho do GEO para obter execuções com sucesso ao encontrar todas as transições de T_{alvo}. Pode-se observar que, com o valor de τ igual a 0.75, a maioria das execuções cobrem T_{alvo} independente do tamanho da seqüência. Também nota-se que conforme o número de variáveis de projeto aumenta, melhor é a cobertura de T_{alvo}, o que já era de se esperar, uma vez que há mais possibilidade da seqüência possuir transições de T_{alvo}.

Todos os experimentos foram realizados usando-se o tipo inteiro e apenas os inteiros positivos foram considerados. Em M_1, existem 23 eventos de entrada, assim as variáveis de projeto estão no intervalo [0, 22]. Com essas informações, o número de bits para representar cada variável de projeto é calculado pela relação (2), com precisão p igual a 1. As transições pertencentes a T_{alvo} são aquelas em que o evento de entrada equivale a uma exceção especificada na documentação do protocolo WTP e constitui um total de 25 transições em M_1. Considerando 4 variáveis de projeto, um dado de teste gerado é uma seqüência de 4 eventos de entrada, por exemplo: (`ack`, `invoke`, `TR-Result.req`, `abort`). Nesse caso, como `ack` não é esperado no estado inicial s_0, a máquina ignora esse evento e permanece em s_0. Em seguida, os demais eventos são disparados, visto que existe esse caminho na máquina. A transição correspondente ao evento `abort` está contida em T_{alvo}, assim o valor da função objetivo para essa seqüência é igual a $1/25 = 0,04$.

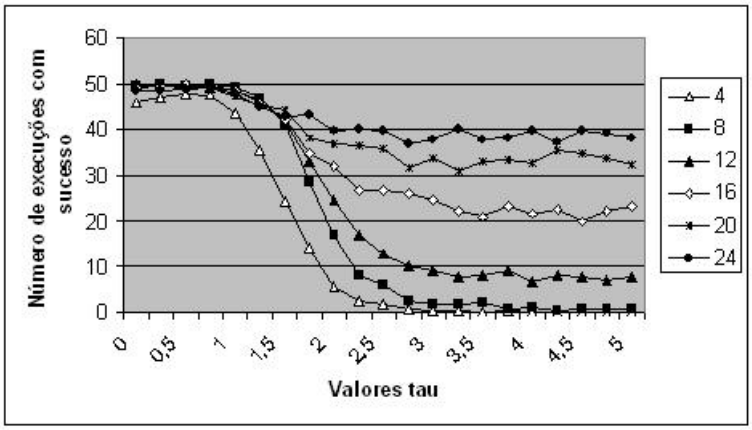

Figura 4. Configuração do parâmetro τ.

A fim de avaliar a aplicação da abordagem, o GEO foi comparado com o teste aleatório. O teste aleatório realiza uma busca na qual dados de teste são selecionados arbitrariamente do domínio de entrada para serem aplicados ao sistema em teste. A geração de dados de teste é feita atribuindo-se valores de '0' ou '1' aleatoriamente para cada bit necessário para representar um candidato a dado de teste. Em seguida, esse candidato é convertido para um valor inteiro e está pronto para ser aplicado no sistema em teste. A comparação do algoritmo GEO e o teste aleatório foi realizada com base na porcentagem média de cobertura em função dos dados de teste gerados. Um experimento de 10 execuções com máximo de 50000 avaliações da função objetivo foi feito para diferentes números de variáveis de projeto, iniciando-se de 4 até 20, com incremento de 4. No caso do GEO, o experimento foi executado com o melhor valor de τ ($\tau = 0.75$). O resultado dessa comparação com 8 variáveis de projeto é mostrado na Figura 5. Verifica-se que o teste aleatório alcançou rapidamente a cobertura total, devido principalmente ao fato do domínio de geração ser pequeno, que são valores do intervalo [0,22]. Embora o GEO necessite um número maior de execuções, o algoritmo também conseguiu obter um conjunto de casos de teste que cubra T_{alvo}. Lembrando que o algoritmo do teste aleatório é bastante simples e, possivelmente em um exemplo mais complexo, seu desempenho não deve se manter.

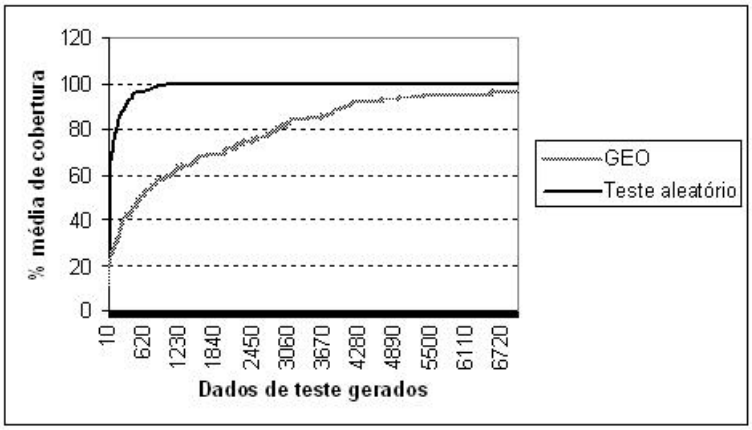

Figura 5. GEO X Teste aleatório.

A evolução dos melhores valores de aptidão no GEO com números de variáveis diferentes é mostrada na Figura 6. Observa-se que após uma média de 7000 avaliações da função objetivo, os valores de aptidão não variam muito.

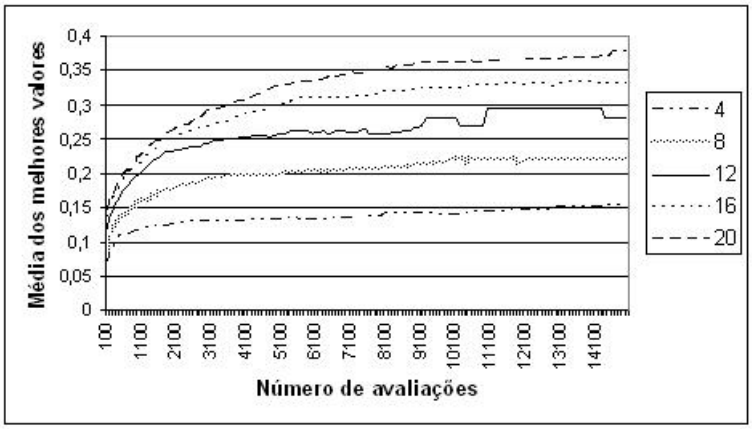

Figura 6. Evolução dos melhores valores da função objetivo no GEO.

6. Conclusões e trabalhos futuros

No contexto de testes evolutivos, o teste funcional é ainda pouco explorado. Assim neste trabalho busca-se aplicar um algoritmo evolutivo no teste de conformidade, possuindo MEF como especificação. Diferentemente da maioria dos trabalhos que aplicam o AG, foi utilizado o algoritmo GEO cujo processo de configuração é mais simples que o AG.

Visto que a função objetivo é bastante simples, deve-se investigar outras funções objetivos que utilizam o conhecimento da estrutura do modelo para melhorar o processo de busca por melhores soluções.

Este trabalho é o passo inicial para a investigação do algoritmo GEO no teste baseado em modelo. Pretende-se utilizar MEF estendida (MEFE), que permite representar o fluxo de dados de um sistema, ao invés do modelo clássico que apenas representa o fluxo de controle. Um problema a ser tratado nas MEFEs é que podem existir seqüências não executáveis. A questão de quais seqüências de entradas devem ser aplicadas para permitir que a implementação em teste execute uma determinada transição é indecidível, ou seja, é impossível encontrar um algoritmo que retorne tal seqüência de entrada ou retorne a mensagem como “esta transição não é executável” [Dssouli et al. 1999]. Assim, métodos de teste de conformidade que utilizam especificações baseadas em MEFE necessitam de heurísticas para tornar as seqüências executáveis. Essa é a razão pela qual foi utilizado um algoritmo evolutivo no teste baseado em modelo e não os métodos já tradicionalmente conhecidos na literatura.

Referências

Abreu, B. T. (2006). Uma abordagem evolutiva para a geração automática de dados de teste. Master’s thesis, IC/Unicamp, Campinas, SP.

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

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

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

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

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

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

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

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

DeSousa, F. L. (2002). *Otimização Extrema Generalizada: Um novo algoritmo estocástico para o projeto ótimo*. PhD thesis, INPE, São José dos Campos, SP, Brasil.

DeSousa, F. L., Ramos, F. M., Paglione, P., and Girardi, R. M. (2003a). New stochastic algorithm for design optimization. *AIAA Journal*, 41(9):1808–1818.

DeSousa, F. L., Vlassov, V., and Ramos, F. M. (2003b). Generalized extremal optimization for solving complex optimal design problems. In *GECCO '03: Proceedings of the 2003 Genetic and Evolutionary Computation Conference*, volume 2723 of *Lecture Notes in Computer Science*, pages 375–376.

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

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

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

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

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

Geurts, W., Wijbrans, K., and Tretmans, J. (1998). Testing and formal methods - BOS project case study. In 6^{th} *European International Conference on Software Testing, Analy-*

sis & (ReviewEuroSTAR'98), pages 215–229, Munich, Germany. Aimware, Mervue, Galway, Ireland.

Gill, A. (1962). *Introduction to the Theory of Finite-State Machines*. McGraw-Hill, New York.

Guo, Q., Hierons, R. M., Harman, M., and Derderian, K. (2004). Computing unique input/output sequences using genetic algorithms. In Petrenko, A. and Ulrich, A., 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.

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

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

Harman, M. and McMinn, P. (2007). A theoretical & empirical znalysis 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. ACM.

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

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

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

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

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

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

Pargas, R. P., Harrold, M. J., and Peck, R. R. (1999). Test-data generation using genetic algorithms. *The Journal of Software Testing, Verification and Reliability*, 9(4):263–282.

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

WAP Forum (2001). WAP wireless transaction protocol specification, wap-224-wtp-20010710-a.

Watkins, A. and Hufnagel, E. M. (2006). Evolutionary test data generation: a comparison of fitness functions: Research articles. *Software Practice and Experience*, 36(1):95–116.