AUTOMATIZAÇÃO DA DEFINIÇÃO DE CENÁRIOS PARA APOIAR A GERAÇÃO DE CASOS DE TESTE DE SISTEMA E ACEITAÇÃO BASEADOS EM MODELOS NO CONTEXTO DA METODOLOGIA SOLIMVA.

Relatório parcial PIBIC

Bolsista: Juliana Marino Balera
e-mail: juliana.balera@inpe.br
Responsável: Dr. Valdivino Alexandre de Santiago Júnior
e-mail: valdivino.santiago@inpe.br

Julho/2014
São José dos Campos

1 – Introdução

A National Aeronautics and Space Administration (NASA) adota um termo denominado Software Assurance (Garantia de Software) o qual engloba diversas disciplinas, entre elas: Garantia da Qualidade de Software, Controle de Qualidade de Software, **Verificação e Validação de Software**, e **Verificação e Validação Independente de Software**. Portanto, está claro que a Verificação e Validação é um dos pilares para garantir que produtos de software tenham alta qualidade, e essa disciplina é particularmente importante se sistemas críticos (como software embarcado em computadores de satélite) são considerados. Entre as atividades associadas à Verificação e Validação estão revisões técnicas, inspeção, verificação formal e todos os tipos de teste de software.

O INPE é uma instituição que desenvolve software internamente em diversos domínios de aplicação e, em alguns casos específicos, faz uso de software desenvolvido por instituições parceiras ou por empresas. Portanto, é essencial que o INPE possa pesquisar e aplicar os mecanismos da Verificação e Validação (testes de software, verificação formal de software, ...) nos produtos internamente e/ou externamente desenvolvidos para projetos do INPE.

A metodologia SOLIMVA [Santiago Júnior 2011] [Santiago Júnior e Vijaykumar 2012] foi desenvolvida em um trabalho de doutorado da CAP/INPE para alcançar duas metas:

a) Geração de casos de teste de sistema e aceitação baseados em modelos a partir de artefatos de requisitos elaborados em Linguagem Natural (LN). Para esse propósito, uma ferramenta, também denominada SOLIMVA, foi projetada e implementada, e tal ferramenta traduz automaticamente requisitos elaborados em LN em modelos Statecharts. Uma vez gerados os Statecharts, outra ferramenta, GTSC [Santiago Júnior et al. 2012], é usada para gerar Casos de Teste Abstratos os quais depois são transformados em Casos de Teste Executáveis. Entre as teorias usadas para alcançar esse objetivo estão testes de software baseados em modelos, designs combinatoriais, e Processamento em Linguagem Natural/linguística computacional (Part Of Speech Tagging [Toutanova et al. 2003], Word Sense Disambiguation [Navigli 2009]). Essa é a versão 1.0 tanto da metodologia como da ferramenta SOLIMVA. A metodologia SOLIMVA 1.0 está mostrada na Figura 1;
b) Detecção de não completude em especificações de software. Entre as teorias usadas para alcançar esse propósito estão Model Checking (método de verificação formal) combinado com arranjos simples de valores de variáveis e padrões de especificação [Dwyer et al. 1999].

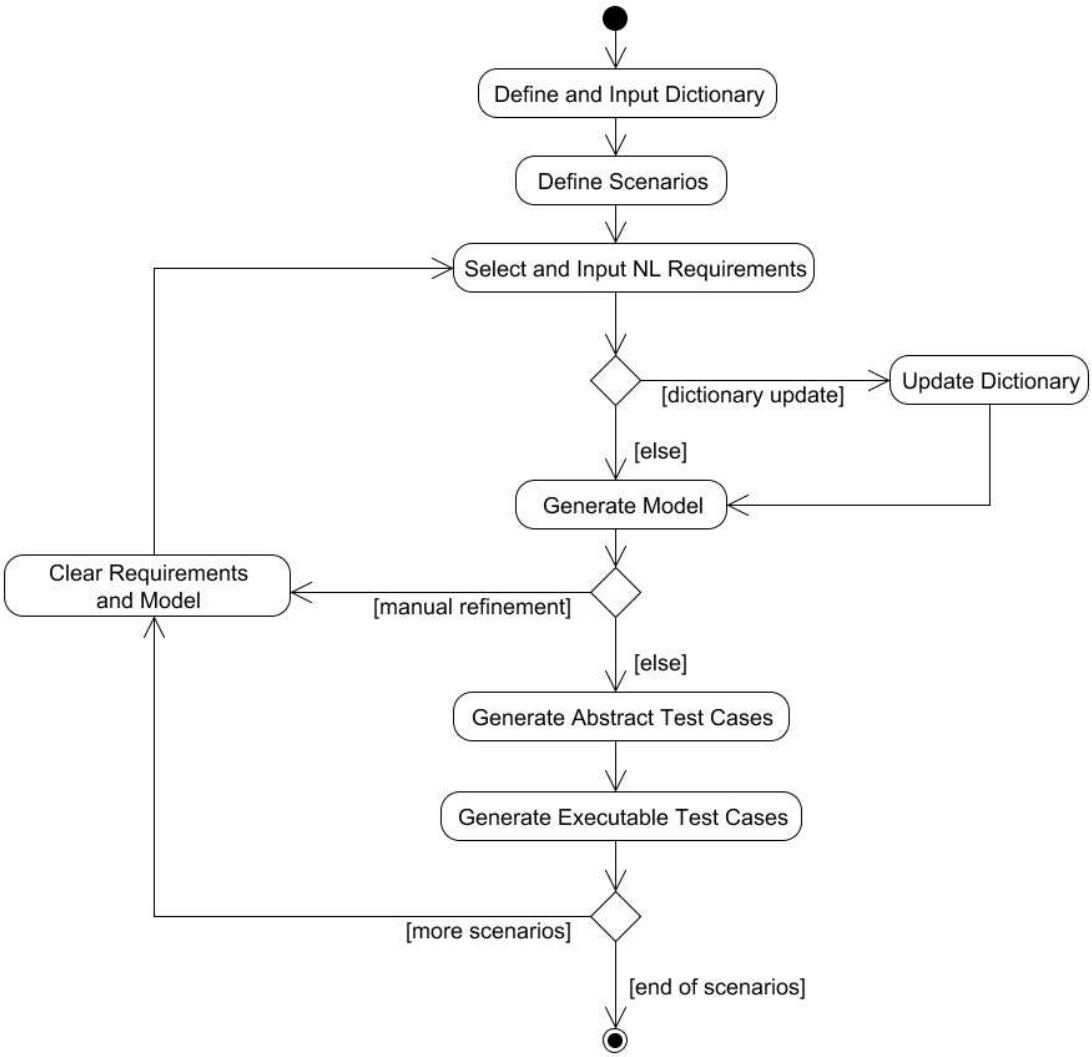

Figura 1 – Metodologia SOLIMVA 1.0

Para as duas metas citadas acima, a metodologia SOLIMVA foi aplicada a um estudo de caso da área espacial, Software for the Payload Data Handling Computer (SWPDC) [Santiago et al. 2007], desenvolvido no escopo do projeto de pesquisa, fomentado pela Financiadora de Estudos e Projetos (FINEP), denominado Qualidade do Software Embarcado em Aplicacões Espaciais (QSEE). Em relação à meta primária, também foram apresentadas diretivas de como aplicar a metodologia SOLIMVA a um segundo estudo de caso do domínio espacial, relacionado ao Segmento Solo: Satellite Control System (SATCS). O SATCS está sendo desenvolvido pela Divisão de Desenvolvimento de Sistemas de Solo (DSS/ETE) [Santiago Júnior 2011].

O SWPDC está sendo, atualmente, adaptado para ser o software do computador do Subsistema de Gestão de Bordo de um outro projeto de pesquisa e desenvolvimento financiado pela FINEP, o experimento científico protoMIRAX (em desenvolvimento na DAS/CEA com parceria do LAC/CTE). Além disso, parte da metodologia SOLIMVA também já está sendo aplicada ao projeto protoMIRAX.

A ferramenta de software SOLIMVA 1.0 foi desenvolvida na linguagem de programação Java usando o paradigma de Orientação a Objetos. A despeito da metodologia/ferramenta SOLIMVA 1.0 ter sido aplicada a um estudo de caso relevante da área espacial e do INPE (SWPDC), naturalmente, a ferramenta precisa de uma série de melhorias para que possa ser aplicada a outros projetos da área espacial do INPE. A metodologia SOLIMVA 1.0 (Figura 1), como um todo, necessitou de ferramentas externas livres para que pudesse ser aplicada. Em particular, foi necessário usar, externamente, a ferramenta **TConfig**, uma ferramenta livre que possui implementada o algoritmo In-Parameter-Order (IPO) para gerar Mixed-Level Covering Arrays (MCA) para dar apoio ao processo de definição de cenários (vide atividade Define Scenarios na Figura 1) por meio de designs combinatoriais. A despeito de outras atividades da metodologia SOLIMVA 1.0 já terem sido automatizadas pela ferramenta SOLIMVA 1.0, a implementação de algoritmo para gerar designs combinatoriais (no caso MCA) não está automatizada, fazendo com que o profissional da área espacial tenha que fazer uso de outra ferramenta (TConfig, externa à ferramenta SOLIMVA 1.0) para gerar os designs combinatoriais. Do ponto de vista prático, é muito mais conveniente se a própria ferramenta SOLIMVA pudesse gerar os MCAs.

Esse relatório parcial apresenta os resultados das atividades desenvolvidas no contexto do projeto **Automatização da definição de cenários para apoiar a geração de casos de teste de sistema e aceitação baseados em modelos no contexto da metodologia SOLIMVA**. O período a que se refere esse relatório parcial é de **01/04/2014 a 31/07/2014** (4 meses). Os trabalhos foram focados na realização das duas primeiras atividades previstas para a bolsa (vide comentários na seção 2 – Cronograma de Atividades e Etapas Concluídas).

2 – Cronograma de Atividades e Etapas Concluídas

Coforme mostrado no “Formulário para Solicitação de Bolsa PIBIC”, o cronograma de atividades do projeto está mostrado na Tabela 1 a seguir.

Tabela 1 – Cronograma de Atividades

Atividades da Metodologia	Tempo (em meses)
1. Estudar e comparar algoritmos existentes na literatura para gerar designs combinatoriais (Mixed-Level Covering Arrays - MCAs). Nesse caso, o bolsista deverá fazer uma análise dos algoritmos existentes para gerar designs combinatoriais	2
2. Implementar ao menos 1 algoritmo para gerar designs combinatoriais na ferramenta SOLIMVA 1.0. Dependendo da análise realizada, um novo algoritmo poderá ser desenvolvido pelo bolsista e implementado na ferramenta SOLIMVA 1.0	8

3. Evoluir a ferramenta SOLIMVA 1.0 de forma a permitir a visualização dos modelos Statecharts	5
4. Realizar a interoperabilidade entre a ferramenta SOLIMVA 1.0 e a ferramenta para geração automática de casos de teste GTSC [Santiago Júnior et al. 2012], para facilitar a geração de casos de teste	6
5. Estudar aspectos de usabilidade na interação humano-computador	1
6. Realizar melhorias de usabilidade da ferramenta SOLIMVA 1.0 utilizando técnicas de projeto de interfaces humano-computador	2
7. Atualizar as interfaces gráficas com o usuário (Graphical User Interfaces - GUIs) da ferramenta SOLIMVA 1.0 de acordo com novas necessidades que apareçam com o uso da ferramenta	3
8. Aplicar a nova versão da ferramenta SOLIMVA, após realizadas as tarefas anteriores, a estudo de caso da área espacial, tal como o SWPDC	4
9. Publicar artigo em conferência e/ou workshop e/ou simpósio na área de Engenharia de Software, e elaborar relatório final de atividades	5

Considerando as atividades previstas para serem desenvolvidas, mostradas na Tabela 1 acima, a Tabela 2 a seguir mostra as atividades concluídas considerando o período de vigência da bolsa, já mencionado anteriormente:

Início: 01/04/2014
Término: 31/07/2014
Total: 4 meses

Tabela 2 – Etapas Concluídas

Atividades da Metodologia	Tempo (em meses)	Execução no Período - Previsto	Execução no Período - Realizado
1. Estudar e comparar algoritmos existentes na literatura para gerar designs combinatoriais (Mixed-Level Covering Arrays - MCAs). Nesse caso, o bolsista deverá fazer uma análise dos algoritmos existentes para gerar designs combinatoriais	2	100%	100%
2. Implementar ao menos 1 algoritmo para	8	25%	25%

gerar designs combinatoriais na ferramenta SOLIMVA 1.0. Dependendo da análise realizada, um novo algoritmo poderá ser desenvolvido pelo bolsista e implementado na ferramenta SOLIMVA 1.0			

Na Tabela 1 acima, a coluna **Tempo (em meses)** mostra o tempo total previsto para a realização da atividade, a coluna **Execução no Período – Previsto** mostra a porcentagem prevista de execução da atividade, no período a que se refere esse relatório parcial, e a coluna **Execução no Período – Realizado** mostra a porcentagem realmente realizada da atividade, considerando o período a que se refere esse relatório. Desse modo, pode-se dizer que todas as atividades previstas para esse período da bolsa foram cumpridas: a atividade 1 foi totalmente concluída (100%), e 25% (2/8 meses) da atividade 2 foi também concluída.

Sobre a realização da atividade 1, foram estudados alguns dos principais algoritmos para gerar MCAs: IPO [Lei e Tai 1998], IPOG [Lei et al. 2007], IPOG-F [Forbes et al. 2008] e Tabu Search Approach (TSA) [Gonzalez-Hernandez et al. 2010]. Após o entendimento de como funcionava cada algoritmo, o foco maior foi dado aos artigos que descreviam o IPOG-G e o TSA. Iniciou-se, após essa leitura inicial dos artigos desses 2 algoritmos, uma análise de complexidade de cada algoritmo, no intuito de selecionar o algoritmo que traria mais benefícios a SOLIMVA.

Uma comparação detalhada, inclusive em termos de complexidade de algoritmos, também foi realizada. O resultado dessa análise foi que o algoritmo IPOG-F foi o selecionado para ser implementado na ferramenta SOLIMVA 1.0, pois o TSA realmente gera um conjunto menor de casos de teste, no entanto, em um espaço de tempo consideravelmente maior, o que ao final das contas, poderia ser prejudicial para a aplicação da SOLIMVA em projetos reais do INPE.

Sobre a realização da atividade 2, já foi implementado em linguagem Java, completamente, o algoritmo IPO, no entanto, essa implementação ainda não está incorporada na ferramenta SOLIMVA 1.0. A implementação do IPO será reaproveitada, pois o algoritmo IPOG-F é uma extensão do IPO, e todas as estruturas e métodos serão reaproveitados. Para esse período de análise, o que foi previsto para ser realizado dessa atividade 2 foi totalmente concluído.

3 – Algoritmos para gerar MCAs

Os algoritmos que foram analisados foram: o In-Parameter-Order, o In-Parameter-Order-General, o In-Parameter-Order-General-Fast e o *Tabu Search*

Approach. O *In-Parameter-Order* não foi analisado em função dele ser a base para a implementação do algoritmo principal, então independente de qualquer análise ele foi estudado e implementado. Todos foram estudados, no entanto o IPOGF e o TSA foram os únicos que foram análisados.

Em todos os algoritmos a ideia e a mesma: a geração de um conjunto de combinações entre todos os níveis de todos os fatores em grupos de tamanho *t.* O *t* é referenciado em todos os artigos como *strength,* e o conjunto de tamanho *t* como *t-tuple*. Então serão gerados um conjunto de casos de teste capaz de cobrir todas essas combinações. Como tal conjunto de casos de teste será gerado é o que diferencia um algoritmo do outro.

A teoria por trás desses artigos se baseiam na ideia de que a maioria das falhas podem ser causadas pela interação de um determinado número de fatores. No caso do *In-Parameter-Order* esse número é 2, o que corresponde á *t* igual 2, e por isso é chamado de *pairwise testing*. No entanto os outros 2 algoritmos, *t* vai assumir valores maiores ou iguais a 2, o que os torna um pouco mais complexos. Em função dessa complexidade o primeiro algoritmo a ser estudado é o *In-Parameter-Order*.

3.1 – Conceitos Importantes

3.1.1 – *Mixed-Level Covering Array* (MCA)

O MCA é uma matriz representada por MCA(*n*; *t*, *k*, *kkk*) de tamanho *n* x *k,* onde *n* corresponde ao número de linhas, *t* ao *strenght* e *k* á quantidade de fatores. Essa matriz é muito importante para o teste de software pois através de seu uso, é possível reduzir consideravelmente o número de testes. Sua ideia principal é de que suas linhas cubram um determinado número de vezes cada *t-tuple.*

3.2 – Algoritmo *In-Parameter-Order* (IPO)

Para um sistema com dois ou mais parâmetros de entrada, *In-Parameter-Order strategy* gera um conjunto *pairwise testing* para os primeiros dois parâmetros, estendendo o conjunto *pairwise testing* gerado para os primeiros três parâmetros, e continua a fazer isso para cada parâmetro adicional [Lei e Tai 1998].

Pairwise testing é uma técnica eficaz na geração de casos de teste que se baseia na observação de que a maioria das falhas são causadas por interações de no máximo 2 fatores. O conjunto *pairwise testing* gerado deve cobrir todas as combinações entre dois fatores, portanto, são muito menores do que os testes exaustivos e ainda muitos eficazes em encontrar defeitos. [Mathur 2008].

Como ponto de partida, esse algoritmo recebe como entrada o número de fatores assim como também a quantidade de seus respectivos níveis. Em seguida é gerado um conjunto com todas as combinações entre os níveis de todos os fatores, dois a dois. A esse par damos o nome de par de cobertura. Por fim, é gerado o conjunto *pairwise testing* dos primeiros dois fatores. A

construção a partir daí ocorre através de dois passos:

Crescimento Horizontal: a adição de um elemento a mais no caso de teste, chamada extensão;

Crescimento Vertical: a adição de um caso de teste a mais no conjunto de testes;

No crescimento horizontal o terceiro fator terá seus níveis acrescentados sequencialmente em cada caso de teste, e quando não houver mais níveis desse fator e ainda houver casos de teste a serem estendidos, será escolhido o nível, pertencente a esse mesmo 3° fator, que mais cobre pares de cobertura (um teste “cobrir” um par significa que esse teste contém os mesmos valores de níveis associados aos mesmos fatores). Todos os pares cobertos são retirados da lista de pares, e o procedimento será repetido para os próximos *n* fatores restantes.

Após a etapa anterior estar concluída, no crescimento vertical os pares de cobertura que não forem cobertos pelo crescimento horizontal serão transformados em casos de teste, com os valores dos níveis em seus respectivos fatores e nos lugares remanescentes serão adicionados valores “sem importância” representados por “_”. Para cada teste que for construído dessa forma, é necessário verificar se a fusão com outro teste não é viável. Todos os pares cobertos são retirados da lista de pares e esse procedimento será repetido para cada par de cobertura não coberto.

3.3 – Algoritmo *In-Parameter-Order-General-Fast* (IPOG-F)

O IPOG-F [Forbes et al. 2008] é uma melhoria do algoritmo IPO. Como ponto de partida, esse algoritmo recebe como entrada o número de fatores assim como também a quantidade de seus respectivos níveis e o tamanho do *strength*. Por fim, é gerada uma matriz, onde as colunas correspondem aos fatores, e cada linha a um caso de teste. Os primeiros elementos inseridos nessa matriz serão originados da combinação simples das primeiras t colunas em suas células correspondentes. A construção a partir daí ocorre através de dois passos:

Crescimento Horizontal: a adição de um elemento a mais no caso de teste, chamada de extensão do caso de teste;

Crescimento Vertical: a adição de um caso de teste a mais no conjunto de testes;

Para a construção de tal matriz, é necessário levar em consideração o número *t-tuples* não cobertas e o número de *t-tuples* que já foram cobertas. Assim sendo, considere a relação:

número de *t-tuples* não cobertas = número total de *t-tuples* – número de *t-tuples* já cobertas.

* Para simplificar, vamos atribuir *tn* ao número de t-tuplas não cobertas e *tc* ao número de t-tuplas já cobertas.

Como o intuito é maximizar a cobertura a cada extensão de linhas, *tn* é usado como métrica para avaliar quais pares linha/valor devem ser usados para estender a matriz. Para a solução proposta, *tc* será armazenado (e assim consequentemente *tn* também).

Para armazenar *tc*, serão necessários 2 arrays: *Tc*[*r*, *v*] e *Cov*[∧, *v*]. *Tc* é indexado pela linha e pelo valor e armazena *tc* para esse par linha valor.
Cov[∧, *v*] é um *array* booleano indexado pela combinação de colunas ∧ e o valor correspondente a essa combinação, os elementos dessa matriz indicarão se a *t-tuple* (∧, *v*) foi coberta.

Algoritmo em pseudocódigo para crescimento horizontal.

```
Tc[i, a] ← 0, ∀i, a
Cov[Λ, ν ] ← false, ∀Λ, ν
while some row is non-extended do
        Find non-extended row i and value a so that tn = (t–1 k–1) – Tc[i, a] is maximum
if tn = 0 then
        stop horizontal growth
end if
Extend row i with value a
        for all non-extended rows j do
                S←set of columns where row i and j have identical
                entries
                for all column tuples Λ contido S do
                        ν ←the value tuple in row i and column tuple Λ
                        if Cov[Λ, ν] = false then
                                Tc[ j, a] ← Tc[ j, a] + 1
                        end if
                end for
        end for
        for all column tuples Λ do
                ν ← the value tuple in row r and column tuple Λ
                if Cov[Λ, ν] = false then
                        Cov[Λ, ν] ← true
                end if
        end for
end while
```

Após a matriz ser “tratada” por esse algoritmo, ela passará pelo crescimento vertical:

```
for all column tuples Λ : k ∈ Λ do
        T∧ ← list of uncovered t-tuples with this Λ
        for all ν : (Λ, ν) ∈ T∧ do
                for all rows i with a don’t-care entry with a column in Λ do
                        if we can place (Λ, ν) then
                                place (Λ, ν)
                        end if
                end for
                if (Λ, ν) not placed yet then
                        add a new row with (Λ, ν) as the only entries
```

end if
end for
end for

3.4 – *Algoritmo Tabu Search Approach (TSA)*

O TSA [Gonzalez-Hernandez et al. 2010] é uma outra abordagem para gerar MCAs que foi analisada. Para explicar esse algoritmo, considere o exemplo de MCA(7; 2, 5, $3^1 2^4$);

Ele opera da seguinte maneira:

Como ponto de partida, esse algoritmo recebe como entrada o número de fatores assim como também a quantidade de seus respectivos níveis e o tamanho do *strength*. Por fim, é gerada uma matriz, onde as colunas correspondem aos fatores, e cada linha a um caso de teste. Os primeiros elementos inseridos nessa matriz serão originados da combinação simples das primeiras t colunas em suas células correspondentes, como representado na Figura 2. A construção a partir daí ocorre através de dois passos:

a	b	c	d	e
0	0	0	0	0
0	1	0	1	1
1	0	1	0	0
1	1	1	1	1
2	0	0	0	0
2	1	0	1	1
0	0	1	0	0

Figura 2 – Matriz Inicial

* Para criação dessa solução inicial não é necessária nenhuma lógica complexa, somente a combinação simples de *t* colunas.

Após a construção da matriz inicial, são gerados os conjuntos *C, A e R.* O conjunto *C* contém todas as combinações entre as colunas, sempre em grupos de tamanho *t.* O conjunto *A* contém todas as combinações de níveis correspondentes a cada combinação de colunas do conjunto *C*. E por fim, o conjunto *R* contém cada linhas da matriz principal. Todos esses conjuntos descritos serão utilizados pelas 3 funções que serão descritas a seguir.

Em seguida, a aproximação usa uma “mistura” de 3 funções auxiliares definidas no artigo. Cada função faz a mesma coisa, no entanto de formas diferentes: escolher o elemento mais vantajoso com relação ao número de pares de cobertura cobertos a ser escolhido para estender cada caso de teste. Essa mistura é feita a partir das probabilidades que cada função produz de acordo com cada caso.

4- Análise comparativa dos algoritmos

A análise comparativas dos algoritmos TSA e IPOG-F foi feita com base nos próprios relatos contidos nos artigos. Além da análise de complexidade de cada algoritmo.

Tabu Search Approach

Considere:

vj sendo o número de valores contidos em uma linha da matriz M até a coluna j (cada linha representa um teste na matriz M);

N é o número total de linhas da matriz M (número total de casos de teste produzidos).

Notação	**Descrição**
$O(N)$	Tempo necessário para que a função N_1 opere
$O((vj-1)\times N)$	Tempo necessário para que a função N_2 opere
$O((vj-1)\times N\times k)$	Tempo necessário para que a função N_3 opere
$O(\binom{k-1}{t-1}\times 2)$	Tempo necessário para que a função de avaliação opere

Tabela 3 – Complexidade do Algoritmo TSA

***In-Parameter-Order* (IPOGF)**

Considere:

r o número de linhas da matriz;
v o número níveis;
k o número fatores;
t o strength;

Notação	**Descrição**
$O(vr^2\binom{k-1}{t-1})$	Tempo total do algoritmo
$O(rv)$	Espaço necessário para o armazenamento de T_c
$O(v^t\binom{k-1}{t-1})$	Espaço necessário para o armazenamento de *Cov*

Tabela 4 – Complexidade do Algoritmo IPOG-F

Após essa análise, levando em consideração dados da Tabela 5, a tendência é

a escolha pelo algoritmo IPOG-F, pois o *Tabu Search Approach* realmente gera um conjunto menor de casos de teste, no entanto, em um espaço de tempo consideravelmente maior, o que no final das contas, poderia ser prejudicial para a aplicação da SOLIMVA em projetos reais do INPE.

Instance	N*	IPOG-F		TSA	
		N	Time [sec]	N	Time [sec]
$MCA(N;2,6,2^2 3^2 4^2)$	16	**16**	0.009	**16**	0.00202
$MCA(N;3,6,2^2 3^2 4^2)$	48	51	0.002	**48**	0.01647
$MCA(N;4,6,2^2 3^2 4^2)$	144	146	0.019	**144**	0.11819
$MCA(N;5,6,2^2 3^2 4^2)$	288	295	0.014	**288**	0.17247
$MCA(N;6,6,2^2 3^2 4^2)$	576	**576**	0.004	**576**	0.00162
$MCA(N;2,8,2^2 3^2 4^2 5^2)$	25	**25**	0.003	**25**	0.00716
$MCA(N;3,8,2^2 3^2 4^2 5^2)$	100	107	0.009	**100**	17.50079
$MCA(N;4,8,2^2 3^2 4^2 5^2)$	400	433	0.035	**400**	94.88019
$MCA(N;5,8,2^2 3^2 4^2 5^2)$	1200	1357	0.201	**1200**	11379.21255
$MCA(N;6,8,2^2 3^2 4^2 5^2)$	3600	3743	0.995	**3600**	7765.91885
$MCA(N;2,10,2^2 3^2 4^2 5^2 6^2)$	36	**36**	0.004	**36**	0.06124
$MCA(N;3,10,2^2 3^2 4^2 5^2 6^2)$	180	207	0.034	185	991.70933
$MCA(N;2,12,2^2 3^2 4^2 5^2 6^2 7^2)$	49	50	0.006	**49**	0.42382
$MCA(N;3,12,2^2 3^2 4^2 5^2 6^2 7^2)$	294	356	0.061	330	528.76392
$MCA(N;2,14,2^2 3^2 4^2 5^2 6^2 7^2 8^2)$	64	67	0.002	**64**	1.53441
$MCA(N;2,16,2^2 3^2 4^2 5^2 6^2 7^2 8^2 9^2)$	81	86	0.012	**81**	26.93236
$MCA(N;2,18,2^2 3^2 4^2 5^2 6^2 7^2 8^2 9^2 10^2)$	100	107	0.016	**100**	702.30086
$MCA(N;2,20,2^2 3^2 4^2 5^2 6^2 7^2 8^2 9^2 10^2 11^2)$	121	131	0.017	122	3927.93448

Tabela 5 – Comparação de diferentes resultados produzidos a partir da execução das mesmas instâncias para os algoritmos IPOG-F e o TSA.

5 – Etapas a Concluir

Considerando as atividades previstas no cronograma de atividades (vide Tabela 1), a Tabela 3 a seguir mostra as etapas a concluir a partir de 01/08/2014, e considerando como término da bolsa a soma total de meses restantes.

Tabela 3 – Etapas a Concluir

Atividades da Metodologia	Tempo (em meses)	Execução no Período - Previsto
2. Implementar ao menos 1 algoritmo para gerar designs combinatoriais na ferramenta SOLIMVA 1.0. Dependendo da análise realizada, um novo algoritmo poderá ser desenvolvido pelo bolsista e implementado na ferramenta SOLIMVA 1.0	8	75%
3. Evoluir a ferramenta SOLIMVA 1.0 de forma a permitir a	5	100%

visualização dos modelos Statecharts		
4. Realizar a interoperabilidade entre a ferramenta SOLIMVA 1.0 e a ferramenta para geração automática de casos de teste GTSC [Santiago Júnior et al. 2012], para facilitar a geração de casos de teste	6	100%
5. Estudar aspectos de usabilidade na interação humano-computador	1	100%
6. Realizar melhorias de usabilidade da ferramenta SOLIMVA 1.0 utilizando técnicas de projeto de interfaces humano-computador	2	100%
7. Atualizar as interfaces gráficas com o usuário (Graphical User Interfaces - GUIs) da ferramenta SOLIMVA 1.0 de acordo com novas necessidades que apareçam com o uso da ferramenta	3	100%
8. Aplicar a nova versão da ferramenta SOLIMVA, após realizadas as tarefas anteriores, a estudo de caso da área espacial, tal como o SWPDC	4	100%
9. Publicar artigo em conferência e/ou workshop e/ou simpósio na área de Engenharia de Software, e elaborar relatório final de atividades	5	100%

Na Tabela 3 acima, a coluna **Tempo (em meses)** mostra o tempo total previsto para a realização da atividade, a coluna **Execução no Período – Previsto** mostra a porcentagem prevista de execução da atividade, a partir de 01/08/2014.

6 – Referências

[Lei et al. 2007] LEI, Y.; KACKER, R.; KUHN, D. R.; OKUN, V.; LAWRENCE, J. IPOG: A general strategy for t-way software testing. In: ANNUAL IEEE INTERNATIONAL CONFERENCE AND WORKSHOPS ON THE ENGINEERING OF COMPUTER-BASED SYSTEMS (ECBS), 14., 2007, Tucson, AZ, USA. Proceedings... Washington, DC, USA: IEEE Computer Society, 2007. p. 549-556.

[Santiago Júnior 2011] SANTIAGO JÚNIOR, V. A. SOLIMVA: A methodology for generating model-based test cases from natural language requirements and detecting incompleteness in software specifications. 2011. 264 p. Thesis (Doctorate at Post Graduation Course in Applied Computing) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, SP, Brazil, 2011. Avaliable

from: <http://urlib.net/8JMKD3MGP7W/3AP764B>. Access in: Feb. 06, 2014.

[Santiago Júnior e Vijaykumar 2012] SANTIAGO JÚNIOR, V. A.; VIJAYKUMAR, N. L. Generating model-based test cases from natural language requirements for space application software. Software Quality Journal, v. 20, n. 1, p. 77-143, 2012. DOI: 10.1007/s11219-011-9155-6.

[Santiago Júnior et al. 2012] Santiago Júnior, V. A.; Vijaykumar, N. L. ; Ferreira, E. ; Guimarães, D. ; Costa, R. C. GTSC: Automated Model-Based Test Case Generation from Statecharts and Finite State Machines. In: Sessão de Ferramentas do III Congresso Brasileiro de Software: Teoria e Prática (CBSoft), 2012, Natal-RN. Anais do III Congresso Brasileiro de Software: Teoria e Prática (CBSoft), 2012. p. 25-30.

[Toutanova et al. 2003] TOUTANOVA, K.; KLEIN, D.; MANNING, C. D.; SINGER, Y. Feature-rich part-of-speech tagging with a cyclic dependency network. In: CONFERENCE OF THE NORTH AMERICAN CHAPTER OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS ON HUMAN LANGUAGE TECHNOLOGY, 2003, Edmonton, Canada. Proceedings... [S.l.], 2003. p. 173-180.

[Navigli 2009] NAVIGLI, R. Word sense disambiguation: A survey. ACM Computing Surveys, v. 41, n. 2, p. 1-69, 2009.

[Dwyer et al. 1999] DWYER, M. B.; AVRUNIN, G. S.; CORBETT, J. C. Patterns in property specications for finite-state verication. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE), 21., 1999, Los Angeles, CA, USA. Proceedings... New York, NY, USA: ACM, 1999. p. 411-420.

[Santiago et al. 2007] SANTIAGO, V.; MATTIELLO-FRANCISCO, F.; COSTA, R.; SILVA, W. P.;
AMBROSIO, A. M. QSEE project: an experience in outsourcing software development for space applications. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING & KNOWLEDGE ENGINEERING (SEKE), 19., 2007, Boston, MA, USA. Proceedings... Skokie, IL, USA: Knowledge Systems Institute Graduate School, 2007. p. 51-56.

[Raskin 2000] RASKIN, J. The Humane Interface: New Directions for Designing Interactive Systems. USA: Addison Wesley, 2000.

[Lei e Tai 1998] LEI, Y.; TAI, K.-C. In-Parameter-Order: A test generation strategy for pairwise testing. In: IEEE INTERNATIONAL SYMPOSIUM ON HIGH-ASSURANCE SYSTEMS ENGINEERING (HASE), 3., 1998, Washington, DC, USA. Proceedings... Washington, DC, USA: IEEE Computer Society, 1998. p. 254-261.

[Forbes et al. 2008] FORBES, M.; LAWRENCE, J.; LEI, Y.; KACKER, R. N.; KUHN, D. R. Refining the In-Parameter-Order Strategy for Constructing Covering Arrays. Journal of Research of the National Institute of Standards and Technology, v. 113, n. 5, p. 287-297, 2008.

[Gonzalez-Hernandez et al. 2010] GONZALEZ-HERNANDEZ, L.; RANGEL-VALDEZ, N.; TORRES-JIMENEZ, J. Construction of mixed covering arrays of variable strength using a tabu search approach. In Proceedings of the 4th international conference on Combinatorial optimization and applications - Volume Part I (COCOA'10), Weili Wu and Ovidiu Daescu (Eds.), Vol. Part I. Springer-Verlag, Berlin, Heidelberg, 2010, p. 51-64.

[Mathur 2008] MATHUR, A. P. Foundations of software testing. Delhi, India: Dorling Kindersley (India), Pearson Education in South Asia, 2008. 689 p.