VIII ERMAC-R3

8º Encontro Regional de Matemática Aplicada e Computacional
20 a 22-Novembro-2008
Universidade Federal do Rio Grande do Norte - Natal/RN

Um modelo de otimização para o problema de design ótimo da programação N-Versões

Arthur G. Carvalho, Aluizio F.R. Araújo, Icamaan B.V. Silva, Christian D.A. Daniel,
Centro de Informática, UFPE,
50670-901, Recife, PE
E-mail: agc@cin.ufpe.br, aluizioa@cin.ufpe.br, ibvs@cin.ufpe.br, cdad@cin.ufpe.br.

Resumo

A programação N-versões é uma abordagem para a construção de softwares tolerantes a falhas que propõe a geração de componentes distintos, porém funcionalmente equivalentes. Tal redundância requer elevados custos adicionais, tanto em termos financeiros quanto em termos temporais.

Apresentamos neste artigo um modelo de otimização, com múltiplos objetivos, para o design ótimo de tais sistemas em que buscamos maximizar a confiabilidade dos mesmos e minimizar o custo e o tempo de desenvolvimento através da escolha dos melhores componentes existentes.

Para validar o modelo proposto, criamos uma base de dados para simular o desenvolvimento de um sistema crítico e aplicamos sobre a mesma o algoritmo evolucionário NSGA-II. Com isto, obtivemos um conjunto de soluções ótimas que ajudarão ao tomador de decisão escolher a melhor combinação de componentes através do controle do relacionamento entre confiabilidade, custo e tempo de desenvolvimento.

Palavras-chave

Otimização, Multicritério, Programação N-versões.

Introdução

Com o uso crescente de computadores na realização de tarefas críticas, tais como controle de tráfego aéreo e monitoramento de usinas nucleares, intensificaram-se os estudos sobre a confiabilidade de tais sistemas computacionais devido ao fato de que possíveis falhas nos mesmos podem acarretar em altas perdas econômicas ou em ameaças à vida humana. Logo, aspectos como disponibilidade, segurança e proteção são atributos essenciais neste contexto [13].

Para ser considerado confiável um sistema crítico deve ser tolerante a falhas, ou seja, o mesmo deve continuar em operação nos mais diversos cenários sem discrepâncias entre as saídas desejadas e a saídas obtidas [11]. Para se obter tal propriedade, propõe-se que sistemas altamente integrados possuam diversos componentes redundantes, operando em paralelo, com o intuito de se reduzir a probabilidade de falha [4].

Neste contexto, a programação N-Versões nasce como uma abordagem para construir sistemas tolerantes a falhas definida como a geração de dois ou mais componentes distintos de um software (versões), funcionalmente equivalentes, advindos da mesma especificação inicial, porém oriundos de diferentes projetos e desenvolvidos por diferentes equipes.

Estes softwares executarão uma dada tarefa (ou subtarefas da mesma) em paralelo, de forma independente, e terão suas saídas avaliadas por um terceiro componente, no esquema de votação, que elegerá a saída final do sistema baseado na resposta de maior freqüência [2].

Tal redundância requer elevados custos adicionais, quer seja em termos financeiros, como equipes de desenvolvimento e produção de hardware e software, quer seja em termos temporais despendidos no decorrer dos projetos [3]. Alguns modelos matemáticos de otimização foram propostos para o design ótimo destes sistemas que facilitariam a escolha de um conjunto das melhores versões disponíveis, como o proposto em [1]. Porém, tais modelos não consideram a otimização simultânea das funções relacionadas à confiabilidade, custo e tempo de desenvolvimento de cada versão, fatores essenciais no projeto de sistemas críticos.

Com isto em mente propomos neste artigo um modelo matemático de otimização multi-objetivo para o design de sistemas na programação N-Versões com estas três funções. Objetivamos com isto obter um conjunto de versões que possua a máxima confiabilidade e minimize o tempo de desenvolvimento e o custo financeiro.

Para exemplificar o uso do modelo, criamos um conjunto de dados para representar possíveis versões que estariam presentes em um sistema crítico. Para gerar as soluções, representamos estes dados de forma binária e utilizamos o algoritmo evolucionário NSGA-II [7] para obter um conjunto de soluções ótimas de Pareto que facilitariam a tomada de decisão.

Além desta seção introdutória o artigo apresentará outras seis seções. A segunda seção descreve em maiores detalhes a abordagem da programação N-Versões e os modelos de otimização previamente propostos para a mesma. Visto que nosso modelo matemático possui múltiplos objetivos, apresentamos na terceira seção uma breve introdução a este campo e à técnica utilizada para gerar soluções ótimas de Pareto chamada algoritmos evolucionários. Na quarta seção apresentamos nosso modelo matemático para a otimização do design de sistemas na programação N-Versões. Na seção quinta apresentamos os experimentos realizados com uma base de dados criada com o intuito de validar o modelo. Por fim, a sexta seção apresenta as considerações finais deste artigo.

Programação N-Versões

A programação N-Versões é uma técnica para construir softwares tolerantes a falhas que propõe a geração de dois ou mais componentes distintos, funcionalmente equivalentes e advindos da mesma especificação inicial.

Por componentes distintos entendemos que os grupos envolvidos na produção dos mesmos não compartilharão o mesmo processo de desenvolvimento e utilizarão, na medida do possível, ferramentas, metodologias e paradigmas diferentes.

Estas versões receberão entradas idênticas (tarefas) e produzirão saídas de forma independente em relação às outras. Um esquema de votação avaliará os resultados individuais obtidos e fornecerá como "saída correta" o que obtiver a maior freqüência dentre todos.

Uma tarefa geralmente consiste em *n* subtarefas seqüenciais. Com isto, cada versão possuirá características peculiares com relação à solução de cada subtarefa, por exemplo, a confiabilidade em resolvê-la. O sucesso de tal abordagem recai no grau de independência das versões produzidas e seu objetivo será o de minimizar a probabilidade de erro entre diferentes versões [10]. A figura 1 ilustra o cenário citado.

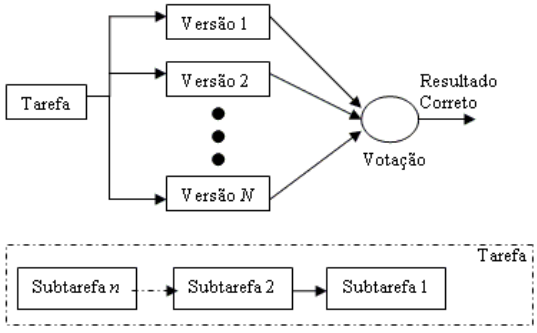

Figura 1: Programação N-Versões.

Devido aos custos adicionais intrínsecos a tal redundância, na fase de design de tais sistemas as versões devem ser escolhidas de forma ótima sob diversos aspectos. Um modelo de otimização, mono-objetivo, foi proposto em [1] almejando encontrar um conjunto de versões que produzam o sistema mais confiável. Sabendo que o orçamento disponível para a produção do mesmo é limitado, esta característica foi tratada como uma restrição no espaço de busca.

A pesquisa realizada em [14] propôs um modelo de otimização bi-objetivo tratando a soma dos custos das versões escolhidas como um objetivo a ser minimizado. Mais especificamente, o problema de design ótimo da programação N-Versões neste trabalho foi formulado matematicamente como um problema binário da programação inteira não-linear em que se busca o conjunto de versões que propicie a maior confiabilidade possível ao passo que reduza o custo associado ao desenvolvimento. Nesta configuração nasce o embate entre os objetivos visto que desejamos maximizar e, simultaneamente, minimizar objetivos conflitantes.

Uma característica fundamental não tratada nestes modelos é o tempo de desenvolvimento das versões. Acredita-se que exista uma relação entre tal característica, a qualidade e a confiabilidade de um software. Baseando-se na existência de técnicas da engenharia de software [13] para estimar o prazo de desenvolvimento de um software, este trabalho propõe um modelo de otimização baseado nos modelos acima citados com a inclusão de uma terceira função objetivo, a saber, o tempo de desenvolvimento.

Otimização Multi-Objetivo

A partir do momento que desejamos sistematicamente e simultaneamente otimizar variadas funções objetivo (geralmente conflitantes) teremos o processo conhecido como otimização multi-objetivo (ou otimização com objetivos múltiplos). De acordo com [6] podemos definir formalmente um problema de otimização com objetivos múltiplos nos seguintes termos:

$$\max / \min F(\overline{x}) = (f_1(\overline{x}), f_2(\overline{x}), ..., f_M(\overline{x})) \quad (1)$$

$$g_j(\overline{x}) \geq 0, \quad j = 1, 2..., J \quad (2)$$

$$h_k(\overline{x}) = 0, \quad k = 1, 2..., K \quad (3)$$

$$x_i^L \leq x_i \leq x_i^U, \quad i = 1, 2..., n \quad (4)$$

Ao contrário da otimização mono-objetivo, na otimização multi-objetivo existirão M funções que poderão ser, individualmente, maximizadas ou minimizadas representadas pelas funções em (1).

Associadas com o problema existem J restrições de desigualdade (2) e K restrições de igualdade (3) e a solução x que não satisfizer todas as $(K + J)$ restrições será considerada uma solução não-admissível. O conjunto de soluções que satisfazem estas restrições gera o espaço de busca. A solução x será formada por um vetor com n variáveis de decisão delimitadas pela restrição presente em (4).

Após este conjunto de definições é valoroso citar que tal família de problemas não possui soluções ótimas únicas, como no caso da otimização com uma única função objetivo. De modo antagônico, suas soluções são caracterizadas por uma família de alternativas que devem ser consideradas equivalentes na ausência de informações relevantes que venham a discriminá-las.

Logo, existirá um compromisso entre objetivos na escolha da solução final. Com isto, o procedimento ideal no uso da otimização com múltiplos objetivos será o de encontrar um conjunto de soluções ótimas e esparsas no espaço de busca e, posteriormente, escolher uma solução utilizando informações de alto nível [6]. Este segundo procedimento faz parte do campo da pesquisa operacional conhecido como *multiple criterion decision-making* (MCDM) e não será abordada no presente artigo.

Devido a esta multiplicidade de soluções, se faz necessário estabelecer critérios que venham a classificar

quais indivíduos seriam considerados ótimos. Para ajudar nesta discriminação usualmente é utilizado o conceito de dominância [12]. Considerando um conjunto com n funções objetivo a serem minimizadas, $\overline{x}$ e $\overline{x'}$ possíveis soluções, afirmamos que a solução $\overline{x}$ domina $\overline{x'}$ se:

$$f_i(\overline{x}) \leq f_i(\overline{x'}), \quad i = 1, 2..., n \tag{5}$$

$$\exists j \in \{1, 2..., M\} | f_j(\overline{x}) < f_j(\overline{x'}) \tag{6}$$

Logo, neste formato de otimização desejamos obter o conjunto de soluções ótimas, não-dominadas (chamadas soluções ótimas de Pareto), e utilizá-las para que a tomada de decisão possa ser mais eficaz.

O trabalho presente em [6] taxa de métodos clássicos os trabalhos pioneiros em algoritmos para geração de soluções ótimas nestes problemas. Os mesmos possuem a limitação de não tratar adequadamente as múltiplas funções existentes, usualmente convertendo-as em uma única função e utilizando técnicas da otimização mono-objetivo para gerar uma solução ótima por execução.

Com isto em mente, os chamados algoritmos evolucionários propiciaram um meio de extrema elegância para obter tais soluções, tornando-se, na atualidade, o principal método para solucionar problemas multi-objetivos.

Os algoritmos evolucionários constituem uma abordagem que utiliza modelos computacionais de processos evolutivos, relativos à seleção natural, para solucionar problemas de otimização [8]. Sua influência foi fortemente advinda de áreas como a genética, teoria evolucionária e biologia celular. Logo, seu funcionamento e nomenclatura são semelhantes com as das fontes inspiradoras. Assim, um candidato a uma solução é chamado indivíduo, enquanto um conjunto de soluções correntes é chamado de população. A atual representação de um indivíduo é chamada genoma (ou cromossomo). Cada genoma consiste de uma seqüência de genes (atributos) que descrevem um indivíduo. O valor de cada gene é chamado alelo.

Cada solução presente na busca será avaliada de acordo com uma dada função de aptidão (geralmente a função objetivo). Na busca pelo conjunto ótimo, as soluções serão modificadas por meio de operadores genéticos, como reprodução, permutação e mutação, para produzirem novos candidatos. Espera-se que no final do processo a população corrente esteja povoada com os melhores indivíduos.

No âmbito da otimização multi-objetivo existem ainda operadores específicos para se obter diversidade entre as soluções ótimas de Pareto. Um exemplo de tal conceito está presente em [9]. Conhecido como *fitness sharing function*, esta técnica compartilha a aptidão entre indivíduos presentes em uma mesma região do espaço. Com isto, quanto mais povoado uma região menor será a aptidão dos indivíduos presentes nela. Logo, esta técnica favorecerá regiões ótimas e pouco povoadas, contribuindo com a manutenção da diversidade.

Modelo Matemático

Para o perfeito funcionamento do nosso modelo assumimos que:

1. Diversas versões funcionalmente equivalentes estarão disponíveis;
2. O custo associado a cada versão estará disponível;
3. As falhas ocorridas em diversas versões serão independentes;
4. Os componentes do esquema de votação possuirão custos desprezíveis;
5. Os componentes do esquema de votação possuirão confiabilidade máxima;
6. Uma estimativa da quantidade de linhas de código para cada versão estará disponível;
7. Um valor que mede o fator de ajustamento de esforço para cada versão estará disponível.

Utilizaremos a seguinte notação neste artigo:

n	número de subtarefas requeridas pela tarefa principal
i	subtarefa $i = 1, 2, ..., n$
m_i	número de versões disponíveis para a subtarefa *i*
j	uma versão disponível para a subtarefa *i*, $j = 1, 2, ..., m_i$
N_i	coleção de todos os subconjuntos com cardinalidade maior ou igual a 1 para as versões $1, 2, ..., m_i$
N_i^*	um subconjunto $N_i^* \in N_i$
$\langle N_i^* \rangle$	cardinalidade de N_i^*
RN_i^*	confiabilidade de N_i^* na subtarefa *i*
R_i	confiabilidade da subtarefa *i*
R_{ij}	confiabilidade da versão *j* que realiza a subtarefa *i*
C_{ij}	custo da versão *j* que realiza a subtarefa *i*
E_{ij}	esforço despendido em meses para desenvolver uma versão *j* que realiza a subtarefa *i*
L_{ij}	Estimativa da quantidade de linhas de código (milhares) da versão *j* que realiza a subtarefa *i*
F_{ij}	fator de ajuste de esforço da versão *j* que realiza a subtarefa *i*

Consideramos que as variáveis de decisão x_{ij} são binárias. Uma versão *j* possuirá seu valor definido de acordo com o fato da mesma ser selecionada ou não para realizar a subtarefa *i*:

$$x_{ij} = \begin{cases} 1, & versão\ j\ selecionada \\ 0, & caso\ contrário \end{cases}$$

Com isto, nosso modelo matemático poderá ser descrito como:

$$\max F_1(\overline{x}) = \prod_{i=1}^{n} R_i \quad (1)$$

$$\min F_2(\overline{x}) = \sum_{i=1}^{n} \sum_{j=1}^{m_i} C_{ij} x_{ij} \quad (2)$$

$$\min F_3(\overline{x}) = \sum_{i=1}^{n} \sum_{j=1}^{m_i} 2,5 E_{ij}^{0,32} \quad (3)$$

$$\sum_{j=1}^{m_i} x_{ij} \geq 1, \quad i = 1, 2, ..., n \quad (4)$$

$$R_i = \sum_{N_i^* \in N_i} \left[\prod_{j \in N_i^*} x_{ij} \prod_{k \in N_i - N_i^*} (1 - x_{ik}) \right] RN_i^* \quad (5)$$

$$RN_i^* = \sum_{j=\lfloor \langle N_i^* \rangle / 2 \rangle \rfloor}^{\langle N_i^* \rangle} \left[\sum_{M \in N_i^* | \langle M \rangle = j} RM \right] \quad (6)$$

$$RM = \prod_{k \in M} R_{ik} \prod_{l \in N_i^* - M} (1 - R_{il}) \quad (7)$$

$$E_{ij} = 2,8 L_{ij}^{1,2} F_{ij} \quad (8)$$

$$x_{ij} = 0\ ou\ 1, \quad i = 1, 2, ..., n, \quad j = 1, 2, ..., m_i \quad (9)$$

A função objetivo (1) foi baseada no trabalho realizado em [1]. Assumimos que a confiabilidade dos componentes que realizam a votação será máxima e não influenciará a função. Seu alvo é maximizar a confiabilidade do sistema através das versões escolhidas.

A função objetivo (2) busca minimizar o custo associado à escolha das versões. A mesma foi baseada no trabalho [14] com a exceção de que consideramos o custo dos componentes que realizam a votação desprezível.

A função (3) busca minimizar o tempo gasto no desenvolvimento das versões e, conseqüentemente, do sistema. A mesma foi baseada no modelo de estimação de custo de softwares COCOMO (*COnstrutive COst MOdel*) intermediário, calculada através da estimativa do esforço necessário para desenvolver o projeto [5].

A restrição (4) garante que no mínimo uma versão será selecionada para cada subtarefa i. O produto dentro dos colchetes em (5) será unitário e a confiabilidade da subtarefa i será calculada através de (6) e (7). A equação (6) garante que pelo menos $\langle N_i^*/2 \rangle$ dentre $\langle N_i^* \rangle$ versões serão escolhidas.

O termo presente em (8) computa o esforço gasto no desenvolvimento de um software em função do tamanho do mesmo em linhas de código e de um fator formado pela análise subjetiva de atributos relacionados ao produto, hardware, projeto e pessoas. Estes atributos receberão uma pontuação baseada em sua relevância e o produto entre todas estas pontuações formará o fator de ajuste de esforço. Seus valores típicos variam entre $0,9$ a $1,4$.

Como pode ser visto em (9), nossa formulação torna o design ótimo de sistemas na programação N-Versões um problema binário, ou seja, as variáveis de decisão informarão quais versões estarão presentes ou ausentes no sistema.

Experimentos

Para validar o modelo proposto, criamos um sistema com 3 tarefas e 6 versões que poderão ser escolhidas para solucioná-las. Os valores para os dados foram baseados no trabalho realizado em [1] e a base pode ser vista na tabela 1.

Para gerar as soluções ótimas utilizamos o algoritmo NSGA-II [7] devido à sua reconhecida capacidade de gerar soluções ótimas, com a permutação em um único ponto com a probabilidade de 0.8, com o operador de mutação representado pela inversão de bits com uma probabilidade de 0.05 e com a reprodução realizada através de torneio binário. O tamanho da população foi de 100 indivíduos e o algoritmo foi executado por 25000 gerações.

Tabela 1: Conjunto de dados gerados

J	1	2	3	4	5	6
R_{1j}	0.90	0.80	0.70	0.60	0.95	–
C_{1j}	9	8	6	4	10	–
L_{1j}	50	65	35	75	50	–
F_{1j}	1.3	1.1	1.2	0.9	1.4	–
R_{2j}	0.95	0.80	0.70	0.85	–	–
C_{2j}	12	9	8	10	–	–
L_{2j}	55	67	70	40	–	–
F_{2j}	1.4	1.2	1.1	1.3	–	–
R_{3j}	0.90	0.95	0.85	0.80	0.75	0.83
C_{3j}	11	12	10	8	6	9
L_{3j}	45	40	56	60	60	55
F_{3j}	1.3	1.4	1.3	1.2	0.9	1.0

O vetor $\overline{x}$, que representa as variáveis de decisão, é composto por 3 variáveis, uma para cada subtarefa, que foram representadas de forma binária devido à natureza do problema. Para a subtarefa 1 a variável foi representada por um vetor de 5 bits, na subtarefa 2 por um vetor de 4 bits e na subtarefa 3 por um vetor de 6 bits. As soluções obtidas podem ser vistas na figura 2.

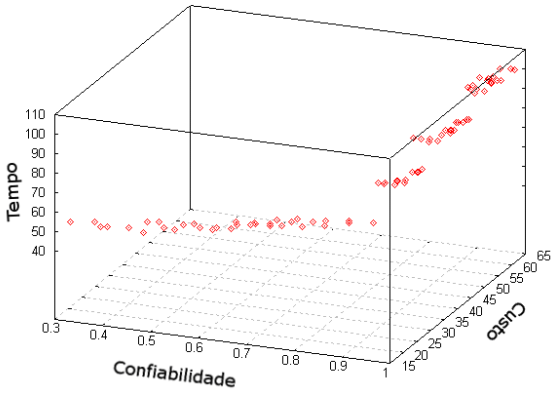

Figura 2: Soluções geradas.

Conclusões

Apresentamos neste artigo um modelo de otimização para o design ótimo de sistemas na programação N-Versões formulado como um problema com três funções objetivo (confiabilidade, custo e tempo) e com variáveis de decisão binárias.

Visto que o problema possui múltiplas funções objetivo, realizamos uma breve introdução à área e ao seu principal método de geração de soluções, os algoritmos evolucionários. Para validar o modelo criamos uma base de dados e utilizamos o algoritmo NSGA-II [7] sobre a mesma para obtermos um conjunto de soluções ótimas de Pareto, sobre a qual uma decisão com maior eficácia poderia ser tomada.

É válido ressaltar que assumimos muitas premissas neste trabalho que podem não ser válidas em algumas situações. Entretanto, o modelo proposto poderia fornecer um ponto de partida para uma especialização ao contexto de uso. Poderíamos, por exemplo, eliminar a premissa de que o custo seria fornecido para as versões em troca de uma função objetivo que calculasse este custo (como ocorre com o tempo).

Outro ponto valoroso de menção é o fato de que a definição de sistemas tolerantes a falha aqui utilizada é baseada no domínio dos dados e que a confiabilidade é mensurada independentemente do tempo. Como trabalho futuro poderíamos estender o atual modelo para aplicá-lo a sistemas cuja confiabilidade seja estimada com base no tempo de operação sem falhas.

Agradecimentos

Os autores são gratos ao CNPq/PIBIC (Conselho Nacional de Desenvolvimento Cientifico e Tecnológico - Programa Institucional de Bolsas de Iniciação Científica) pelo suporte.

Referências

[1] N. Ashrafi, O. Berman, e M. Cutler, Optimal design of large software-systems using N-version programming, *IEEE Transaction on Reliability*, 43 (1994) 344-350.

[2] A. Avizienis e L. Chen, On the implementation of N-version programming for software fault tolerance during execution, em "IEEE International Computer Science Applications Conference", pp. 149-155, 1977.

[3] F. Belli e P. Jedrzejowicz, Fault-Tolerant programs and their reliability, *IEEE Transaction on Reliability*, 39 (1990) 184-192.

[4] P.G. Bishop, D.G. Esp, M. Barnes, P. Humphreys, G. Dahll e J. Lahti, PODS: Project on diverse software, *IEEE Transaction on Software Engineering*, 12 (1986) 929-940.

[5] B. Boehm, "Software engineering economics", Prentice-Hall, New Jersey, 1981.

[6] K. Deb, "Multi-Objective Optimization Using Evolutionary Algorithms", John Wiley, New Jersey, 2001.

[7] K. Deb, A. Pratap, S. Agarwal e T. Meyarivan, A Fast and Elitist Multi-objective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2) (2002) 182-197.

[8] A.E. Eiben e J.E. Smith, "Introduction to Evolutionary Computing", Springer, New York, 1998.

[9] D. Goldberg e J. Richardson, Genetic algorithms with sharing for multimodal function optimization, em "International Conference on Genetic Algorithms on Genetic algorithms and their application", pp. 41-49, 1987.

[10] H. Hecht, Fault tolerant software, *IEEE Transaction on Reliability*, 28 (1979) 227-232.

[11] D.J. Iannino, A. Musa e K.O. Iannino, "Software Reliability: Measurement, Prediction, application", McGraw-Hill, New York, 1987.

[12] H.T. Kung, F. Luccio e F.P.Preparata, On Finding the Maxima of a Set of Vector, *Journal of the ACM*, 22 (1975) 469-476.

[13] I. Sommerville, "Engenharia de Software", Pearson Addison Wesley, São Paulo, 2003.

[14] H. Yamachi, Y. Tsujimura, Y. Kambayashi e H. Yamamoto, Multi-objective genetic algorithm for solving N-version program design problem, *Reliability Engineering & System Safety*, 91 (2006) 1083-1094.