Framework para execução distribuída de GAvaPS

Nataniel Pereira Borges Junior[1], **Ricardo Pereira e Silva**[1]

[1]Departamento de Informática e Estatística (INE) – Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88.037-000 – Florianópolis – SC – Brazil

{natanieljr,ricardo}@inf.ufsc.br

Abstract. *Genetic algorithms (GAs) are non-specific optimization methods that posses a well-defined architecture and execution flow and are widely used into areas like machine learning and design optimization. Object-oriented framework is a known approach for the reuse of application architecture and execution flow aimed to increase speed and quality on software development. Multi-computer environments, such as clusters and grids, are now widespread and available at low cost. The focus of this document is to introduce a framework under development to allow the execution of genetic algorithms with variable population size (GAvaPS) into multi-computer environments.*
Keywords*: framework; genetic algorithm; distributed systems.*

1. Introdução

Algoritmos genéticos (GAs) são "*técnicas de busca estocásticas baseadas em mecanismos de seleção natural e genética*" [Guo et al. 2010, p. 2990]. Segundo [Fogel 1994, p. 3] os princípios da teoria da evolução de Darwin são mecanismos robustos de busca e otimização onde cada bioma evoluído demonstra um comportamento complexo e otimizado. Os problemas que as espécies solucionaram durante o seu processo evolutivo envolvem fatores como caos, sorte e temporalidade que são, por sua vez, características de problemas que se mostraram intratáveis através dos mecanismos de otimização tradicionais. Visando aproximar o funcionamento do algoritmo original (*Simple Genetic Algorithm* - SGA) [Holland 1975] ao comportamento existente na natureza foram propostas alterações dentre as quais citam-se a inclusão de população variável (GAvaPS), proposta por [Arabas et al. 1994] e a subdivisão das populações em ilhas (DGA), apresentada em [Escuela et al. 2007]. Todas as variações, embora acrescentem ou adaptem conceitos do algoritmo, mantêm intactas a estrutura e o fluxo de execução do mesmo, que é composta por mecanismos de avaliação da população, seleção dos indivíduos mais aptos, processo de reprodução sexuada (*crossover*) e mutação.

Já *frameworks* são definidos por [e Silva 2000, p. 31] como esqueletos de uma implementação de aplicação ou um subsistema com interconexões entre classes e fluxo de execução preestabelecidos, cabendo ao desenvolvedor somente inserir a lógica da sua aplicação nos *hot spots*[1]. Essa característica torna os *frameworks* ferramentas adequadas ao desenvolvimento de artefatos reusáveis para implementação de GAs.

2. Problema identificado

Atualmente, diante do aumento dos recursos computacionais disponíveis, é possível tratar diversos problemas com uma quantidade maior de detalhes, resultando por sua vez em

[1]Hot spots são as partes flexíveis de um *framework*, as partes fixas do *framework* são chamadas de *frozen spots* [e Silva 2000, p. 2].

abstrações mais fiéis das situações existentes no mundo real. Entretanto essas melhores abstrações ampliam a quantidade de possíveis elementos no espaço de soluções candidatas de um problema, tornando a sua otimização um processo que consome cada vez mais recursos computacionais. Hoje, ambientes multi-computados, como clusters e grids computacionais, encontram-se disponíveis a baixo custo, entretanto modelos de programação distribuída não são amplamente dominados pelos desenvolvedores, resultando em aumento de custo de desenvolvimento e utilização de soluções ineficientes.

3. Solução proposta

Para alterar esse panorama é proposto um *framework* para a execução distribuída de GAs com população variável que, além de proporcionar um aumento de produtividade no desenvolvimento de aplicações quando comparado ao modelo de desenvolvimento tradicional [Markiewicz and de Lucena 2001, p. 2], remove do desenvolvedor a necessidade de conhecimento prévio sobre modelos de programação distribuída. O presente trabalho terá como resultado dois produtos: especificação do *framework* e o código fonte do *framework*.

A especificação do *framework* será elaborada utilizando a linguagem UML2, possuindo mapeamento direto com o código fonte e contemplando os requisitos apresentados por [e Silva 2009, p. 21]: modelagem estrutural de sistema, modelagem dinâmica de sistema, modelagem estrutural de parte e modelagem dinâmica de parte.

O código fonte será organizado em pacotes (*namespaces*) dentre os quais citam-se *operators.mutation*, *operator.crossover*, *operators.selection* e *cromossome*, cada qual engloba um conjunto de classes que representa um conceito do GA. Sua escrita segue os padrões de projeto *strategy* e *factory* propostos em [Johnson et al. 1995] assim como o princípio de vinculação de objetos somente a interfaces não a implementações específicas [Johnson et al. 1995, p. 18-20]. Essa abordagem de projeto permite a escolha dos operadores utilizados internamente pelo algoritmo, assim como o desenvolvimento de novos operadores, sem a necessidade de alteração no código do mesmo.

Para utilização do *framework* é necessária a implementação das interfaces *IFitnessEvaluationStrategy* e *ICromossomeFactory* onde serão respectivamente definidas a função de avaliação do *fitness*[2] e um mecanismo para a criação de novos cromossomos[3], entretanto é possível reescrever qualquer outra classe adequando-a às necessidades do problema a ser solucionado. Em sua configuração padrão o algoritmo é executado utilizando:

- *Crossover rate* [Beasley et al. 1993, p. 3] de 0,8;
- *Mutation rate* [Beasley et al. 1993, p. 3] de 0,01;
- Seleção elitista [Fogel 1994, p. 6];
- *Minimum lifetime*[4] [Arabas et al. 1994, p. 75] de 3;
- *Maximum lifetime*[5] [Arabas et al. 1994, p. 75] de 15.

[2]O *fitness* serve para mensurar numericamente a qualidade de uma solução candidata [Escuela et al. 2007, p. 437].

[3]Cromossomos são representações de uma solução candidata em um GA. [Beasley et al. 1993, p. 7]

[4]Número mínimo de gerações em que um cromossomo deve permanecer na população

[5]Número máximo de gerações em que um cromossomo pode permanecer na população

O resultado da compilação do código fonte do *framework* são dois módulos, um módulo "Solver" que contém as classes responsáveis pela avaliação do *fitness* dos indivíduos, podendo ser executado em múltiplos computadores, e um módulo "Control" responsável pelo fluxo de execução, definição do ponto de parada e pelas operações de seleção, *crossover* e mutação, em execução em um único computador.

Ao final do processo de desenvolvimento do *framework* o mesmo será validado desenvolvendo-se duas aplicações, uma maximizando a função de Schwefel [Laguna and Martí 2005, p. 15], uma identificando o valor de parâmetro utilizado na aplicação de um algoritmo de *thresholding* [Neves and Pelaes 2008, p. 3] em uma imagem. Uma vez desenvolvidas as aplicações de validação será avaliado o reuso proporcionado pelo *framework* e será mensurada a diferença de performance durante a execução das mesmas utilizando 1, 3 e 5 computadores.

Referências

Arabas, J., Michalewicz, Z., and Mulawka, J. (1994). Gavaps - a genetic algorithm with varying population size. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, volume 1 of *999*, pages 73–78, Orlando, FL.

Beasley, D., Bull, D. R., and Martin, R. R. (1993). An overview of genetic algorithms: Part 1, fundamentals. *University Computing*, 15:58–69.

e Silva, R. P. (2000). *Suporte ao desenvolvimento e uso de frameworks e componentes*. PhD thesis, Universidade Federal do Rio Grande do Sul.

e Silva, R. P. (2009). *Como Modelar com UML2*. Visual Books.

Escuela, G., Cardinale, Y., and González, J. (2007). A java-based distributed genetic algorithm framework. In *19th IEEE International Conference on Tools with Artificial Intelligence.*, volume 1, pages 437–441.

Fogel, D. (1994). An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 5(1):3–14.

Guo, P., Wang, X., and Han, Y. (2010). The enhanced genetic algorithms for the optimization design. In *3rd International Conference on Biomedical Engineering and Informatics*, volume 7, pages 2990–2994, Yantai.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, Michigan.

Johnson, R., Helm, R., Vlissides, J., and Gamma, E. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.

Laguna, M. and Martí, R. (2005). Experimental testing of advanced scatter search designs for global optimization of multimodal functions. *Journal of Global Optimization*, 33(2):235–255.

Markiewicz, M. E. and de Lucena, C. J. P. (2001). Object oriented framework development. *Crossroads*, 7:3–9.

Neves, S. C. M. and Pelaes, E. G. (2008). Estudo e implementação de técnicas de segmentação de imagens. *Revista Virtual de Iniciação Acadêmica da UFPA - Universidade Federal do Pará – Departamento de Engenharia Elétrica e de Computação*, 1(2).