Application programming interface Library para Algoritmo Genético com Operador Transgênico

Fabrício Alves Rodrigues[2], Stéfanne Alves de Souza[1], Renan Vinícius Aranha[1]
Ana Paula Freitas Vilela Boaventura[1]

[1]Curso de Ciência da Computação - Universidade Federal de Goiás/Jataí (UFG)
BR364 Km 192, Setor Industrial, Jataí - GO - Brasil

[2]Pós-Graduação em Ciência da Computação - Universidade Federal de São Carlos (UFSCar)
Rodovia Washington Luís, km 235 - SP-310 São Carlos - São Paulo - Brasil

fabricio1989@gmail.com, tefi70@hotmail.com

Abstract. *This paper presents features and operating mode API (Application programming interface) of the genetic algorithm (GA) of Holland in 1975, which is based on the natural selection process proposed by Charles Darwin with the aim of get results favorable to complex problems. The API Also addresses a new genetic operator proposed in 2011 for Amaral and Hruschka, which is a genetic algorithm with a transgenic operator. Such change allows the population be diversified, and have got faster and favorable results to the problem. The API has been tested to mining data from a dataset from car traffic. The result was a mining with a textit fitness 100 % and the API has correctly executed the steps described by AG.*

Resumo. *Este artigo apresenta características e o modo de funcionamento da API(Application programming interface) do algoritmo genético (AG) de Holland em 1975, que baseia-se no processo de seleção natural proposto por Charles Darwin com o intuito de se obter resultados favoráveis para problemas complexos. A API Ainda aborda a proposta feita em 2011 por Amaral e Hruschka, que propuseram um algoritmo genético com um operador transgênico. Tal alteração permite que a população seja diversificada, além de se obter mais rápido resultado favorável ao problema. A API foi testada para minerar dados de um dataset referente a trafego de carros. O resultado foi uma mineração com um fitness de 100% e a API executou corretamente os passos descritos pelo AG.*

1. Introdução

Nos últimos anos, as heurísticas têm tido destaque como uma das abordagens mais promissoras para solução de problemas de programação inteira e combinatórios. Dentre as técnicas heurísticas mais sofisticadas, existem as chamadas meta-heurísticas, com varias aplicações nas diferentes áreas da ciência [Rodrigues et al. 2004]. As principais meta-heurísticas que apresentam grande potencial para encontrar soluções de problemas são: Busca Tabu (BT) [Glover 1977], *Ant Colony Optimization* (ACO) [Dorigo et al. 1996], *GRASP (Greedy Randomized Adaptive Search Procedure)* [Feo and Resende 1989], *Particle Swarm Optimization* (PSO) [Kennedy and Eberhart 1995] e Algoritmo Genético (AG) [Holland 1975].

AG tem sido muito usado para resolver problemas combinatórios e, em 2011, os autores [Amaral and Hruschka 2011] propuseram um algoritmo genético com um operador transgênico. O presente trabalho tem como objetivo descrever uma *Application programming interface Library* (API) do Algoritmo Genético com operador transgênico para validar a API será feita uma mineração de dados.O presente trabalho tem como objetivo descrever um Application programming interface Library (API) do Algoritmo Genético com operador transgênico. Para validar a API, será demonstrado um exemplo utilizando-se a mineração de dados. Na seção 2, será descrito o funcionamento do algoritmo genético e do operador transgênico, enquanto a seção 3 conterá a descrição da API do AG com operador transgênico. Por último, serão apresentados resultados e conclusões.

2. Algoritmo Genético e Operador Transgênico

Os Algoritmos Genéticos são técnicas de busca que utilizam o processo da seleção natural proposto por Charles Darwin. Os AGs inicialmente foram descritos por John Holland [Holland 1975] e são bastante utilizados em problemas combinatórios.

Os Ags utilizam operadores para fazer sua busca, os principais operadores são seleção de indivíduos para o *crossover*, o mais simples é o método da roleta no qual cada indivíduo da população recebe uma fatia da roleta proporcional ao (*Fitness*) do indivíduo. Em seguida é feito o sorteio aleatório desses indivíduos e os selecionados faram parte do operador de *crossover* [Man et al. 2001].

Crossover é a troca de genes entre os indivíduos selecionados. Um ponto de cruzamento é escolhido aleatoriamente e a partir dele é feita a troca genética dos indivíduos. A informações genéticas anteriores a este ponto de um dos pais é ligado as informações posteriores do outro pai gerando a cada cruzamento dois novos filhos, que são colocados na população auxiliar [Rezende 2005]. O operador mutação é necessário para que a população seja diversificada, permite chegar mais rápido ao objetivo da busca e para fugir dos mínimos locais, alterando aleatoriamente um ou mais genes dos indivíduos gerados pelo cruzamento.

Neste trabalho será usando a estrategia mais equilibrada para reinserção, essa estrategia parece melhorar o desempenho do AG, ela é chamada de melhores pais e filhos onde são escolhidos N/2 melhores indivíduos geradores (pais) e N/2 melhores indivíduos criados (Filhos) no qual são inseridos na população gerando uma nova geração [Man et al. 2001].

Em 2011 foi proposto por Laurence [Amaral and Hruschka 2011] um algoritmo genético com um operador transgênico, esse novo operador para o algoritmo genético foi inspirado pela engenharia genética, onde permite manipular o material genético de indivíduos adicionando características que acredita-se ser a mais importante. Essa mesma atividade que é feita por engenheiros genéticos foi introduzido no AG, assim identificasse qual é o gene ou genes mais importante então esses genes dos N melhores indivíduos são passados para M outros inidivíduos da população.

3. *Application programming interface Library Transgenic Genetic Algorithm*

A API Java proposta neste artigo segue os princípios descritos por Holland em seu trabalho e também contêm o operador transgênico descritos por Amaral e Hruschka.

A API apresenta duas classes (*Population* e *GeneticAlgorithm*), duas interfaces (*GeneImpl* e *IndividualImpl*). A classe *Population* contêm sete métodos, são eles: *generateIndividuals()* que gera os indivíduos na população, *greaterIndividual()* que retorna o individuo com maior *fitness*, *smallestIndividual()* que retorna o individuo com menor *fitness*, *getIndividual(int pos)* retorna o individuo da população que está na posição requirida, *getIndividuals()* retorna todos os indivíduos da população, *setIndividual(IndividualInpl ind, int pos)* modifica o individuo na posição requirida pelo individuo passado por parâmetro e *setIndividuals(IndividualInpl ind[])* modifica toda a população pelo vetor de indivíduos passado no parâmetro. A classe *Population* será utilizada pela outra classe chamada *GeneticAlgorithm*.

A classe *GeneticAlgorithm* contêm os operadores descritos na seção 2, também apresenta o método *bestIndividual()* que retorna o melhor resultado encontrado pelo Algoritmo Genético, para utilizar essa classe requer que o usuário passe tamanho da população, um exemplo de individuo implementado, já que o mesmo é somente uma *interface* Java, também é necessário passar uma *long* que corresponde a *Seed*, caso o problema necessite executar mais de uma vez usando a mesma população inicial e um valor booleano onde (*True*: se o problema e de maximização e *False* se for de minimização).

O gene e o individuo da API são representados por *interface*, pois os genes e indivíduos são diferentes para cada problema que o usuário pretenda resolver. A *interface GeneImpl* contêm os seguintes métodos que os usuários terão que implementar, *makeMutation()* onde deve ser implementado como será feita a mutação no gene, *createGene(Random seed)* ou *createGene()* método no qual será gerado o valor aleatório do gene, *G getValue()* retorna o valor do gene (G representa o tipo genérico da variável valor deve ser dito na implementação do gene, exemplo *public GeneImpl<Integer>*) e *setValue(G valor)* que deve modificar o valor do gene.

A *interface IndividualImpl* contêm os seguintes métodos que os usuários terão que implementar *createGenes(Random seed)* esse método deve gerar os N genes que representa o individuo do problema á ser resolvido, *getGenes()* deve retornar todos os genes que representa o individuo, *setGenes(GeneImpl genes[])* deverá modificar todos os genes do individuo pelo vetor que será recebido por parâmetro, *getGene(int pos)* deverá retornar o gene que está na posição requirida, *setGene(int pos, GeneImpl g)* deverá modificar o gene na posição requirida pelo gene 'g' passado pelo parâmetro, *getFitness()* deverá calcular o valor do *fitness* e retorna-lo e *Mutation(int pos)* que deverá requisitar a classe Gene, que deve ser implementada pelo usuário, a execução de uma mutação no gene que é representado na posição 'pos'.

4. Resultados e Conclusões

Para validar a API, foi feita uma mineração de dados em um *dataset* que contém dados de transito, que pode ser encontrado na aba*Tools and Files* do link `www.fabricioarodrigues.wix.com/researches`. Neste *dataset* requer encontrar uma regra *IF-THEN* que melhor represente se houve ou não congestionamento com base nos dados do *dataset*. A avaliação (*Fitness*) do AG foi formulado como sendo $TP * 0.5 + FP * 0.5$, onde TP são registros classificados corretamente e FP são registros que foram ditos como não pertencentes à classe avaliada e realmente não pertenciam. O AG foi executado com taxa de cruzamento de 100% e taxa de mutação de 2%. Eram

sorteados 2 indivíduos para cruzamento e foi utilizado a reinserção balanceada, também conhecida como melhores pais e filhos. Os genes e cromossomo usados neste artigo são os mesmo do trabalho [Cunha et al. 2012].

A API executou corretamente cada passo descrito por Holland e com o operador transgênico descrito por [Amaral and Hruschka 2011]. Apresentou ainda um resultado satisfatório, com resultados de *fitness* de 100% foi executadas 180 gerações o AG foi executado 100 vezes e as regras encontradas foram *IF Fluxo >= 51 THEN Congestionado* e *IF Fluxo < 50 THEN Não Congestionado*. Pode se concluir que, com o uso de uma API do AG transgênico, como o descrito nesse trabalho, pode-se obter resultados satisfatórios que são capazes de ajudar pesquisadores a encontrar resultados para problemas combinatórios. Como trabalhos futuros iremos implementar outras funcionalidades de seleção e cruzamento, pois usamos aqui somente uma das estrategias para os mesmos. Ainda, será feita a documentação apropriada para a API descrita neste artigo.

Referências

Al-Hinai, N. and ElMekkawy, T. (2011). Robust and stable flexible job shop scheduling with random machine breakdowns using a hybrid genetic algorithm. *International Journal of Production Economics*, 132(2):279–291.

Amaral, L. and Hruschka, E. (2011). Transgenic, an operator for evolutionary algorithms. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 1308 –1314.

Cunha, L. C., Rodrigues, F. A., and de Souza Alencar, W. (2012). Semáforo inteligente com lógica fuzzy baseado na mineração de dados por algoritmo genético transgênico. In *EATI 2012*.

Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 26(1):29–41.

Feo, T. and Resende, M. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters*, 8(2):67–71.

Glover, F. (1977). Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, USA.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948. IEEE.

Man, K. F., Tang, K. S., and Kwong, S. (2001). *Genetic Algorithms: Concepts and Designs*. Spring, 3 edition.

Rezende, S. O. (2005). *Sistemas inteligentes: fundamentos e aplicações*. Manole Ltda, Barueri,SP.

Rodrigues, F., Leite, H., Santos, H., de Souza, A., and Silva, G. (2004). Metaheurística algoritmo genético para solução de problemas de planejamiento florestal com restrições de integridade. *Revista Árvore*, 28(2):233–245.