Estudo de caso: Algoritmo Genético e função Eggholder

Alef C. de Sousa*, Caio C. S. Barbosa, Jorge L. C. Reis***,
Francisco A. B. Gabriel*****, Henrique S. de Andrade******.**

**Universidade Federal do Ceará, Fortaleza, CE
Brasil (e-mail: alef@alu.ufc.br)*
*** Universidade Federal do Ceará, Fortaleza, CE
Brasil (e-mail:caiocid@gmail.com)*
**** Universidade Federal do Ceará, Fortaleza, CE,
Brasil (e-mail: jlcr1997@gmail.com)*
***** Universidade Federal do Ceará, Fortaleza, CE,
Brasil (e-mail:alesonbg@alu.ufc.br)*
******Universidade Federal do Ceará, Fortaleza, CE,
Brasil (e-mail: henriquearaujoandrade@gmail.com)}*

Abstract: Um estudo de caso da utilização de Algoritmos Genéticos aplicado a função Eggholder, variando seus parâmetros para verificar qual seria as melhores utilizações e comparando sua eficiência com outros métodos de otimização.

Keywords: Algoritmo Genético, Otimização, Computação Evolucionária.

1. INTRODUCTION

Otimização de problemas é uma realidade constante dos ambientes científicos. Diferentes métodos foram desenvolvidos para otimizar problemas, entre eles, a computação evolucionária. Baseado na teoria de evolução de Charles Darwin, os Algoritmos Genéticos tentam reproduzir o processo evolutivo para chegar no melhor resultado para determinado problema.

Devido ao seu caráter populacional e aleatório, o Algoritmo Genético consegue explorar um maior espaço amostral do que outros métodos que não utilizam população (Por exemplo, o hill-climbing). Além disso, ele consegue lidar melhor com problemas mais complexos, ou seja, que sua função de aptidão seja mais irregular, como a função eggholder, que será explorado no artigo.

Outra vantagem da utilização dos Algoritmos Genéticos é sua simplicidade, o que o torna fácil sua implementação em diversas plataformas. Por fim, temos o aspecto multiobjetivo, que nos permite colocar diversas funções de aptidão, cada uma delas representando um diferente objetivo, e implementar de modo que todas sejam satisfeitas em uma proporção eficiente.

Nesse artigo, iremos analisar comparativamente o desempenho do algoritmo genético em relação a outros métodos de otimização.

2. METHODS

2.1 Algoritmos Genéticos e Otimização

Algoritmos Genéticos, AGs, são algoritmos de busca capazes de solucionar problemas de otimização para funções contínuas e discretas inspirados nos mecanismos de evolução dos seres vivos. Seguem o princípio da seleção natural do mais apto.

Essas técnicas geralmente apresentam um espaço de busca, onde estão as possíveis soluções do problema, e uma função objetivo, ou função de aptidão que é utilizada para avaliar as soluções produzidas.

Em termos matemáticos, a otimização consiste em achar a solução que corresponda ao ponto de máximo ou de mínimo da função de aptidão. Para resolver um problema de otimização um algoritmo genético primeiramente gera uma população inicial de cromossomos, que representam possíveis soluções para do problema a ser resolvido. Eles são feitos de unidades (genes) que controlam a hereditariedade de uma ou mais características.

Durante o processo evolutivo essa população é avaliada e cada cromossomo recebe uma nota refletindo a qualidade da solução que ele representa. Com isso, apenas os mais aptos se reproduzem passando suas características para as próximas gerações. Dessa forma o algoritmo seleciona os mais aptos reproduzindo o comportamento natural dos organismos num ambiente competitivo. O algoritmo 1 ilustra isso.

Duas características principais dos AGs são a criação de uma população diversa explorando o espaço de busca e a redução dessa diversidade com a seleção dos indivíduos mais aptos em cada geração. Geralmente uma estratégia chamada de Elitismo é adotada para garantir que os cromossomos mais

adaptados persistam nas próximas gerações caso eles não sobrevivam.

A diversidade de população é obtida por meio das operações de Crossover e Mutação. Crossover forma novos herdeiros de 2 cromossomos pais combinando a informação genética deles. Já mutação é a alteração randômica de valores dos genes de cromossomos pais.

ALGORITMO 1: Algoritmo Genético base:

Seja S(t) a população de cromossomos na geração t.
t<- 0
inicializar S(t)
avaliar S(t)
Enquanto o critério de parada não for satisfeito, faça:
  t<- t+1
  selecionar S(t) a partir de S(t-1)
  aplicar crossover sobre S(t)
  aplicar mutação sobre S(t)
  avaliar S(t)
fim enquanto

2.2 A função Eggholder.

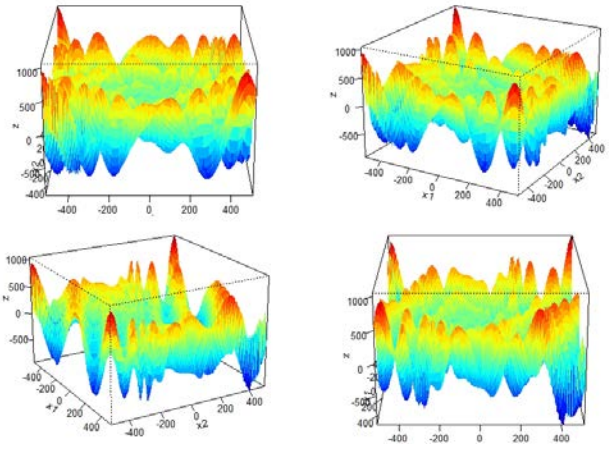

Fig.1. Impressão 3D dos valores da função *Eggholder* por x1 e x2

A função Eggholder é avaliada no domínio Xi "pertence" [-512,512] para todo i=1,2 e tem um mínimo global localizado em x*=(512,404.2319), f(x*)= -959.6407. Ela é uma função de teste para validar e comparar algoritmos de otimização que, segundo Jamil, 2013, é contínua, diferenciável, não-separável, escalável e multi-modal.

Um dos maiores desafios ao otimizar a função Eggholder é a existência de muitos mínimos locais, ou seja, a função atinge valores menores nesses pontos do que na vizinhança. Porém, eles não representam o menor valor que a função pode alcançar em todo o sistema dificultando a solução do processo de otimização. Este menor valor é chamado de Mínimo global.

Para fazer isso utilizaremos o pacote GA da linguagem R. A principal função do pacote é chamada 'ga' e possui os seguintes argumentos:

```
ga (type =c("binary ", "real-valued", "permutation"),
      fitness = …, min ,  max , nBits,
      population = gaControl(type)$population,
      selection = gaControl(type)$selection,
      crossover = gaControl(type)$crossover,
      mutation = gaControl(type)$mutation,
      popSize = 50,  pcrossover = 0.8, pmutation = 0.1,
      elitism = max(1, round(popSize * 0.05)),
      monitor = gaMonitor, maxiter = 100, run = maxiter,
      maxfitness = -Inf, names = NULL, suggestions, seed)
```

Os parâmetros alterados para melhorar o resultado do algoritmo são:

type: O tipo de algoritmo genético a ser rodado depende da natureza das variáveis. Aqui utilizaremos "real-valued", que é utilizado para problemas de otimização com variáveis tipo floating-point;

fitness: A função de fitness é qualquer função em R que receba como entrada uma string representando uma potencial solução e retorna um valor númerico descrevendo sua aptidão;

min = Um vetor de tamanho igual ao das variáveis que fornece o mínimo do espaço de busca para uma otimização;

max = Um vetor de tamanho igual ao das variáveis que fornece o máximo do espaço de busca para uma otimização;

population = Uma string ou função em R que gera uma população inicial randômica;

selection = Uma string ou função em R que realiza a seleção;

crossover = Uma string ou função em R que realiza o crossover;

mutation = Uma string ou função em R que realiza a mutação;

popSize = Tamanho da população;

pcrossover = Probabilidade do Crossover entre pares de cromossomos;

pmutation = Probabilidade da ocorrência de mutação em um cromossomo pai;

elitism = porcentagem de indivíduos mais aptos que sobrevivem a cada geração;

maxiter =Número máximo de iterações a ser rodadas antes da busca do AG ser parada;

run = número de gerações consecutivas sem nenhuma melhora no valor de aptidão antes do AG ser parado;

A função 'ga' retorna um objeto que contém os seguintes atributos;

iter: O valor numérico da iteração atual da busca;

population: uma matriz de dimensão object@popSize vezes o número de variáveis

fitness: a função fitness avaliada para a população atual;

best: o melhor valor de aptidão para cada iteração da busca do AG;

mean: o valor médio de aptidão para cada iteração da busca do AG;

fitnessValue: o melhor valor de aptidão encontrado pelo AG;

solution :Uma matriz de strings de solução com o número de linhas igual ao número de soluções encontradas e o número de colunas igual ao número de variáveis.

As funções escolhidas para realizar as operações de população, seleção, crossover e mutação são das configurações padrões da função “ga”, a “gaControl”:

População: gareal_Population- Cria uma população randômica de valores reais no intervalo [object@min , object@max];

Seleção: gareal_lsSelection - Realiza uma seleção proporcional adequando-se a uma escala de aptidão linear;

Crossover: gareal_laCrossover- Realiza um crossover aritmético local;

Mutação: gareal_raMutation- Realiza uma mutação randômica uniforme;

A seleção padrão utilizada na biblioteca ‘ga’ corresponde a uma seleção por regressão linear simples. Calcula-se um valor de ‘a’, correspondente ao quociente da aptidão média pela subtração da maior aptidão pela aptidão média, e um valor ‘b’, correspondente ao produto da aptidão média pela subtração da aptidão máxima pelo dobro da aptidão média, tudo dividido pela subtração da aptidão máxima pela aptidão média. Após isso, esses termos são multiplicados de modo que f = a * função de aptidão + b. A seleção é feita baseada nesse termo ‘f’.

3. RESULTS

Antes de tudo é necessário lembrar que os Algoritmos Genéticos trabalham com números aleatórios, para aplicar a mutação por exemplo, e por isso os valores obtidos pelo leitor provavelmente serão diferentes dos valores aqui mostrados. Os parâmetros utilizados foram: Tamanho da população = 50, taxa de Crossover = 80%, taxa de mutação = 10%, amostra para elitismo = 5, quantidade máxima de gerações = 100 e quantidade máxima de gerações com o mesmo melhor valor de aptidão = 50.

Para calcular a aptidão (fitness) foi utilizada a função -egg(x), pois queremos encontrar o valor de mínimo de egg(x) e a função ga() procura o valor máximo da função passada para o parâmetro fitness. Com isso, apesar de estarmos em busca do mínimo o gráfico mostra uma curva crescente, pois a avaliação da função é em relação a -egg(x).

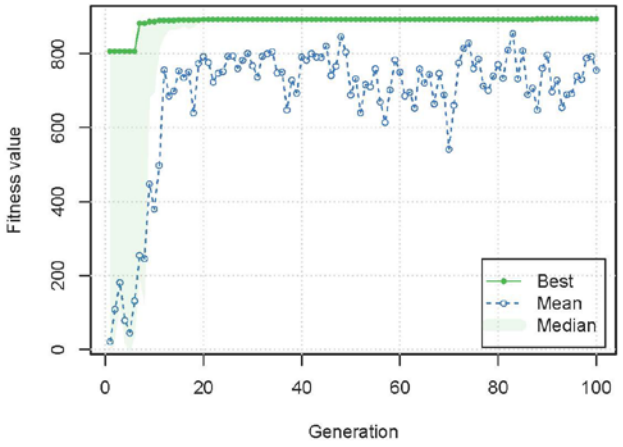

Fig. 1. Gráfico de maior, melhor e mediana dos valores de aptidão do Algoritmo Genético

Como é possível observar no gráfico há muita oscilação no valor Mean que representa o valor médio de aptidão para a população daquela geração. Isso acontece justamente pela aleatoriedade mencionada anteriormente. Entretanto isso não tira o mérito da técnica, por dois motivos:

1. Como é possível ver no gráfico os melhores valores de uma população é sempre crescente, o que acontece justamente pelo fato de que o melhor valor de uma população i, também é membro da população i+1;
2. Além disso, não é uma simples aleatoriedade que é aplicada. São realizadas técnicas, baseadas na teoria evolutiva de Darwin, para que os cromossomos da geração seguinte tenham uma maior tendência a chegar no resultado esperado.

	Melhor valor AG	Melhor valor Nelder-Mead
f(x1, x2)	-894,5383	-786,526
x1	-465,1106	-456,8875
x2	385,5744	-382,6215

Tabela. 1. Comparação entre desempenho do Algoritmo Genético em relação ao do método de Nelder-Mead

O resultado do Algoritmo Genético foram comparados com o resultado obtido pelo método de Nelder-Mead, obtido com a função optim(). Diferentemente de um AG, o método de Nelder-Mead não trabalha com aleatoriedade, logo o valor retornado é sempre o mesmo.

Comparando o resultado do Algoritmo Genético com o resultado obtido pelo método de Nelder-Mead é possível perceber que o AG conseguiu chegar em um resultado melhor, e isso é válido para grande parte das execuções. Um dos motivos para isso acontecer é o fato de que o método de Nelder-Mead possui uma maior tendência de parar em mínimo/máximo locais que o AG devido ao fato de trabalhar com um ponto de partida pré-determinado pelo método. O AG consegue "fugir" de pontos de mínimo/máximo locais devido a sua aleatoriedade, pois ainda que esteja perto de um mínimo/máximo local ele ainda apresenta chances de convergir para o mínimo/máximo real, já que os cromossomos da geração seguinte são definidos com um fator de aleatoriedade.

REFERENCES

Lacerda, E.G.M, Carvalho, A.C.P.L. Introdução aos algoritmos genéticos. In: Galvão, C.O., Valença, M.J.S. (orgs.) Sistemas inteligentes: aplicações a recursos hídricos e ciências ambientais. Porto Alegre: Ed. Universidade/UFRGS : Associação Brasileira de Recursos Hídricos. p. 99-150. 1999. (Coleção ABRH de Recursos Hídricos; 7.).

Momin Jamil and Xin-She Yang, A literature survey of benchmark functions for global optimization problems, Int. Journal of Mathematical Modelling and Numerical Optimisation, Vol. 4, No. 2, pp. 150–194 (2013).

Luca Scrucca, GA: A Package for Genetic Algorithms in R. Int. Journal of Statistical Software, Vol. 53, Issue 4 (2013).

Appendix A. FIRST APPENDIX

```
library(GA)

library(foreach)

egg <- function(x1, x2) {

  return (-(x2 + 47)*sin(sqrt(abs((x1/2) + x2 + 47))) -
x1*sin(sqrt(abs(x1 - (x2 + 47)))))

}


# Plotando eggholder

x1 <- seq(-512, 512, length.out=100)

x2 <- seq(-512, 512, length.out=100)

z <- outer(x1, x2, egg)

par(mar=c(1,1,1,1), mfrow=c(2,2))

lapply(c(0,30,60,90), function(t) persp3D(x1, x2, z, theta=t))

par(mfrow=c(1,1))

filled.contour(x1, x2, z, color.palette = jet.colors)



# Algoritmo Genético

egg <- function(x) {

  return (-(x[2] + 47)*sin(sqrt(abs((x[1]/2) + x[2] + 47))) -
x[1]*sin(sqrt(abs(x[1] - (x[2] + 47)))))

}


GA <- ga(type = "real-valued",

        fitness = function(x) -egg(x),

        min = c(-512, -512),

        max = c(512, 512),

        population = gaControl("real-valued")$population,

        selection = gaControl("real-valued")$selection,

        crossover = gaControl("real-valued")$crossover,

        mutation = gaControl("real-valued")$mutation,

        popSize = 50,

        pcrossover = 0.8,

        pmutation = 0.1,

        elitism = 5,

        maxiter = 100,

        run = 50)


par(1,1)

plot(GA)


opt <- optim(par = c(-512, -512), egg)

opt
```