



UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

Tópicos Avançados em Inteligência Artificial
Função Ackley

MATHEUS CASA NOVA DA LUZ - MCNL
EDUARDO BARRETO BRITO - EBB2
JULIANA NASCIMENTO DAMURIE DA SILVA - JNDS
DANIEL CARRIÇO DE LIMA MENEZES - DCLM

RECIFE, 3 DE NOVEMBRO DE 2020

Professor: Paulo Salgado

Sumário

1	Introdução	3
2	Configuração	3
3	Algoritmos	4
3.1	Seleção	4
3.1.1	Melhores pais / piores indivíduos	4
3.1.2	Matar metade da população	4
3.1.3	Sobrevivência dos mais fortes	4
3.1.4	Melhores pais aleatórios com elitismo	4
3.2	Cruzamento	5
3.2.1	Cruzamento Discreto	5
3.2.2	Cruzamento Intermediário	5
3.2.3	Cruzamento BLX- α	5
3.2.4	Crossover modificado	5
3.3	Mutação	7
3.3.1	Mutação Uniforme	7
3.3.2	Mutação Não Uniforme	7
3.3.3	Mutação Gaussiana	7
3.3.4	Mutação BLX- α	7
3.3.5	Mutação Não Uniforme modificada	7
4	Esquemático do programa	9
5	Resultados	10
6	Justificativa e Conclusão	14
7	Referências	15

1. Introdução

A ideia do projeto é desenvolver um Algoritmo Evolucionário (Estratégia Evolutiva) para a determinação do ponto de mínimo global da função de Ackley, definida pela Equação 1, e considerando $c_1 = 20$, $c_2 = 0.2$, $c_3 = 2\pi$, $n = 30$ e $-15 < x_i < 15$.

$$f(x) = -c_1 \cdot \exp(-c_2 \sqrt{\frac{\sum_{i=1}^n x_i^2}{n}}) - \exp(\frac{\sum_{i=1}^n \cos(c_3 \cdot x_i)}{n}) + c_1 + \exp(1) \quad (1)$$

2. Configuração

Em termos de configuração geral do algoritmo, o grupo optou por:

- Representação: cada indivíduo é um *array* composto por 30 números reais.
- Função de *Fitness* definida pela Equação 2:

$$\frac{1}{1 + \text{ackley}(\text{individo})} \quad (2)$$

onde *ackley* é a função definida na Equação 1.

- População:
 - Tamanho: 100
 - Inicialização:
 - * Números reais aleatórios entre -15 e 15.
 - * Sem repetição.
- Processos de seleção, cruzamento e mutação definidos e explicados na Seção 3.
- Probabilidade de cruzamento: 90%.
- Probabilidade de mutação: 5%.
- Condições de parada: *fitness* > 0.999 ou 10000 gerações.

Além disso, foi feito um arredondamento de 8 casas decimais (*round(num, 7)* em *Python*) dos números dos indivíduos e do cálculo da função para cada indivíduo. A aproximação do arredondamento e do *fitness* > 0.999 foi feito devido ao fato o tipo original da linguagem possuir uma precisão de 18 casas decimais, fazendo com que fosse extremamente difícil atingir o *fitness* = 1, mas ainda assim mantendo um grau de precisão satisfatório para o objetivo do projeto.

3. Algoritmos

3.1. Seleção

3.1.1. Melhores pais / piores indivíduos

Breve alteração do método proposto pelo professor Paulo Salgado na primeira parte do projeto das 8 Rainhas ([1]), segue os seguintes passos:

1. Ordena a população inteira em ordem de *fitness*;
2. pega os 2 pais com melhores *fitness*;
3. gera 2 filhos;
4. substitui os 2 piores indivíduos.

3.1.2. Matar metade da população

Proposto como um dos métodos para o problema das 8 rainhas por J. Rico em [2], o método:

1. Pega a população inteira par a par aleatoriamente:
 - mais apto sobrevive;
 - menos apto morre;
2. gera filhos com a metade da população que sobreviveu até completar a população novamente.

3.1.3. Sobrevivência dos mais fortes

1. Gera o número da população de filhos usando pais aleatórios (par a par);
2. ordena a população baseado no *fitness*;
3. descarta um número suficiente dos piores indivíduos até atingir o tamanho original da população.

3.1.4. Melhores pais aleatórios com elitismo

1. Seleciona 5 indivíduos aleatoriamente;
2. escolhe os 2 indivíduos com maior *fitness*;
3. cruza os dois indivíduos escolhidos; e
4. se o filho gerado tiver maior *fitness* que o pior da população, ele o substitui;
5. caso contrário, o filho é descartado.

Esse método de seleção de sobreviventes foi proposto por Joelan et al. em [3] para minimização de funções.

3.2. Cruzamento

3.2.1. Cruzamento Discreto

Proposto em aulas pelo professor Paulo Salgado ([4]): cada X_i do filho é uma média aritmética dos $X_i(s)$ dos pais.

3.2.2. Cruzamento Intermediário

Proposto em aulas pelo professor Paulo Salgado ([4]): cada X_i do filho é uma média ponderada dos $X_i(s)$ dos pais. A ponderação é baseada no *fitness* dos pais

3.2.3. Cruzamento BLX- α

Proposto por Eshelman & Schaffer em [5], gera um filho onde cada X_i do filho é um número aleatório com mínimo e máximo baseado nos X_i s dos pais (vide Figura 1). O α é o passo de *blend* e varia entre 0 e 1, geralmente ficando por volta de 0.15 (valor usado neste experimento). Na imagem, teríamos os pais:

$$pai_a = [x_1, x_2, \dots, x_n] \quad (3)$$

$$pai_b = [y_1, y_2, \dots, y_n] \quad (4)$$

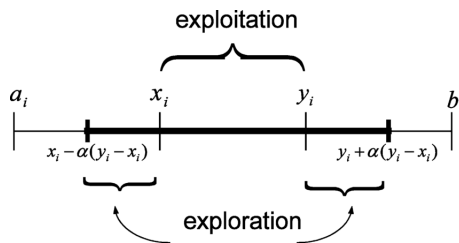


Figura 1: Funcionamento do cruzamento BLX (Blend). Abido, Mohamed. Retirado do URL: https://www.researchgate.net/figure/Blend-crossover-operator-BLX-a_fig1_222574063

3.2.4. Crossover modificado

Proposto por Joelan et al. em [3], funciona da seguinte forma:

Tendo os pais:

$$pai_a = [x_1, x_2, \dots, x_n] \quad (5)$$

$$pai_b = [y_1, y_2, \dots, y_n] \quad (6)$$

seleciona-se um ponto de corte (pc) aleatório entre 0 e o tamanho do indivíduo e a seguinte operação é feita:

1. seleciona-se um “fator de *crossover*” (f_c) aleatório entre 0 e 1, que muda a cada cruzamento;

2. cria-se dois genes resultantes de um *crossover* comum:

$$half_a = [x_1, x_2, \dots, x_{pc}, y_{pc+1}, \dots, y_n] \quad (7)$$

$$half_b = [y_1, y_2, \dots, y_{pc}, x_{pc+1}, \dots, x_n] \quad (8)$$

3. multiplica-se cada gene pelo fator de *crossover* f_c e divide-se por 2:

$$half_a = half_a \cdot f_c \cdot 0.5 \quad (9)$$

$$half_b = half_b \cdot f_c \cdot 0.5 \quad (10)$$

4. o filho, único, é a soma das duas metades resultantes de 9 e 10, elemento a elemento:

$$filho = half_a + half_b = [x_1 half_a + x_1 half_b, \dots, x_n half_a + x_n half_b] \quad (11)$$

3.3. Mutação

3.3.1. Mutação Uniforme

Proposto em aulas pelo professor Paulo Salgado ([4]): varia o X_i de cada indivíduo da população com uma certa chance (definido na Seção 2): o novo X_i é um novo número aleatório que varia entre LIM_MIN e LIM_MAX (no caso, de -15 a 15).

3.3.2. Mutação Não Uniforme

Proposto em aulas pelo professor Paulo Salgado ([4]): varia o X_i de cada indivíduo da população com uma certa chance (definido na Seção 2): o novo X_i é um novo número aleatório que varia entre $X_i - desvioPadrao$ e $X_i + desvioPadrao$, onde o desvio padrão é o desvio do próprio indivíduo.

3.3.3. Mutação Gaussiana

Também descrito por Heitzinger em [6], varia o X_i de cada indivíduo da população com uma certa chance (definido na Seção 2): o novo X_i é o próprio X_i acrescido de um dX provindo de uma distribuição gaussiana.

3.3.4. Mutação BLX- α

Baseada no cruzamento proposto por Eshelman & Schaffer em [5], varia o X_i de cada indivíduo da população com uma certa chance (definido na Seção 2) da seguinte forma:

- Semelhante ao cruzamento *BLX*, o novo X_i varia entre:
 1. $Min = X_i - (passo \cdot range)$
 2. $Max = X_i + (passo \cdot range)$
- Onde $range = 15$, definido na Seção 1, e passo é descrito pela Equação 12:

$$Passo = \frac{\alpha}{fitness_{indv} \cdot peso_{fitness}} \quad (12)$$

e α é o mesmo do Cruzamento BLX (descrito na Seção 3.2.3).

- O peso do *fitness* é o quanto o *fitness* influencia na mutação, que faz com que:
 - indivíduos com *fitness* baixos escolham um número novo com um *range* grande, fugindo do lugar onde estão;
 - indivíduos com *fitness* altos escolham um número novo com um *range* pequeno, mantendo a direção que estão seguindo.
- Neste experimento, o peso do *fitness* usado foi de 10.

3.3.5. Mutação Não Uniforme modificada

Proposto por Joelan et al. em [3], funciona da seguinte forma: varia o X_i de cada indivíduo da população com uma certa chance (definido na Seção 2) baseado em um fator aleatório:

1. Tendo o indivíduo: $indivíduo_x = [x_1, x_2, \dots, x_n]$;
2. gera-se um *array* de mesmo tamanho $[z_1, z_2, \dots, z_n]$ de *floats* aleatórios entre 0 e 1;
3. soma-se os dois *arrays* de forma que o novo indivíduo resulte em:

$$indivíduo_{novo_x} = [x_1 + z_1, x_2 + z_2, \dots, x_n + z_n] \quad (13)$$

4. Esquemático do programa

A estrutura do programa construído pelo grupo é a seguinte:

```
for i in range(0, NUM_AMOSTRAGEM):
    populacao = iniciarPopulacao()
    for metodo in METODOS:
        populacaoUsada = populacao.copy()
        resetarVarGlobais()
        numGeracoes = 1
        while (numGeracoes <= NUM_MAX_GERACOES
               and FITNESS_MELHOR_INDIVIDUO[-1] < FITNESS_DESEJADO):
            populacaoUsada = realizarCruzamento(populacaoUsada, metodo)
            populacaoUsada = realizarMutacao(populacaoUsada, metodo)
            numGeracoes += 1
```

Os algoritmos estão disponíveis nos arquivos, disponibilizados neste link do GitHub, da seguinte forma:

- As configurações que o programa roda em Config.py.
- Funções auxiliares (como calcular o *fitness* e desvio padrão) em Auxiliar.py.
- Algoritmos de seleção em Selecao.py.
- Algoritmos de cruzamento em Cruzamento.py.
- Algoritmos de mutação em Mutacao.py.
- O programa principal (controlador) em Ackley.py.

5. Resultados

A avaliação da combinação dos algoritmos foi feita baseada na avaliação dos seguintes critérios:

1. convergência (se convergiu);
2. média da quantidade de gerações até a convergência;
3. média de *fitness* dos indivíduos;
4. média de *fitness* dos melhores indivíduos;
5. média de tempo de execução;
6. média da soma dos *fitness* dos indivíduos a cada amostragem.

As várias combinações de algoritmos foram submetidas a mesma população inicial 20x e foram obtidos os gráficos exibidos nas Figuras 2, 3, 4, 5 e 6.

Nestes gráficos, os métodos são exibidos no Eixo X da seguinte forma:

- 1º número (centenas), método de seleção:
 1. Melhores pais / piores indivíduos
 2. Matar metade da população
 3. Sobrevivência dos mais fortes
 4. Melhores pais aleatórios com elitismo
- 2º número (dezenas), método de cruzamento:
 1. Cruzamento Discreto
 2. Cruzamento Intermediário
 3. Cruzamento BLX- α
 4. *Crossover* modificado
- 3º número (unidades), método de mutação:
 1. Mutação Uniforme
 2. Mutação Não Uniforme
 3. Mutação Gaussiana
 4. Mutação BLX- α
 5. Mutação Não Uniforme modificada

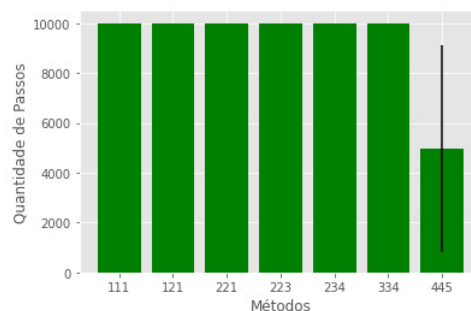


Figura 2: Média de gerações por estratégia de evolução após 20 execuções. Configuração geral descrita na Seção 2.

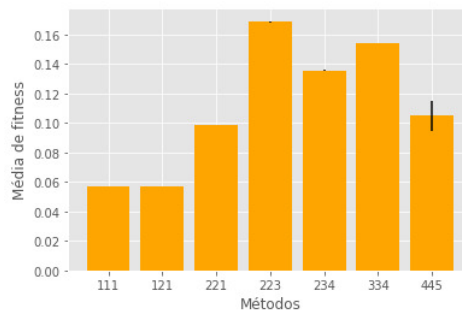


Figura 3: Média de *fitness* por estratégia de evolução após 20 execuções. Configuração geral descrita na Seção 2.

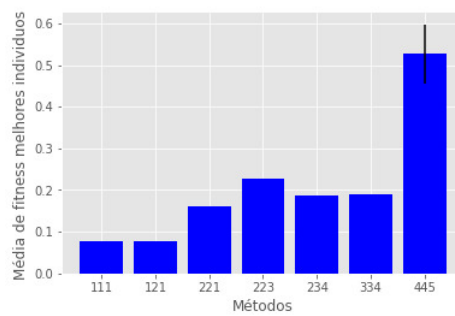


Figura 4: Média de *fitness* dos melhores indivíduos de cada geração por estratégia de evolução após 20 execuções. Configuração geral descrita na Seção 2.

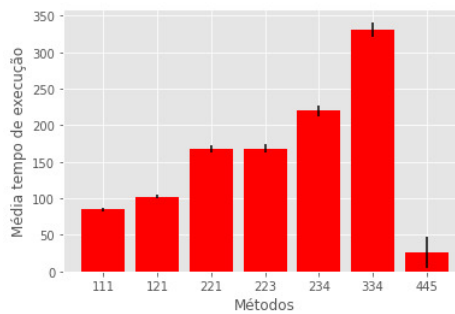


Figura 5: Média de tempo de execução (total) por estratégia de evolução após 20 execuções. Configuração geral descrita na Seção 2.

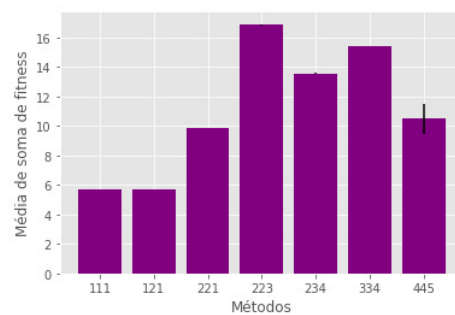


Figura 6: Média de soma de *fitness* dos indivíduos por estratégia de evolução após 20 execuções. Configuração geral descrita na Seção 2.

Um breve teste, feito com apenas 3 repetições de no máximo 10000 gerações cada, 1% de probabilidade de mutação e *fitness* desejado acima de 0.99, resultou nos gráficos das Figuras 7, 8, 9, 10 e 11.

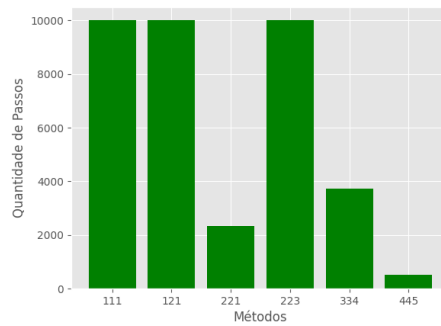


Figura 7: Média de gerações por estratégia de evolução após 3 execuções, com 1% de probabilidade de mutação e *fitness* > 0.99.

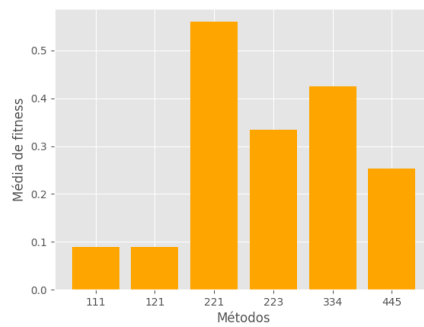


Figura 8: Média de *fitness* por estratégia de evolução após 3 execuções, com 1% de probabilidade de mutação e *fitness* > 0.99.

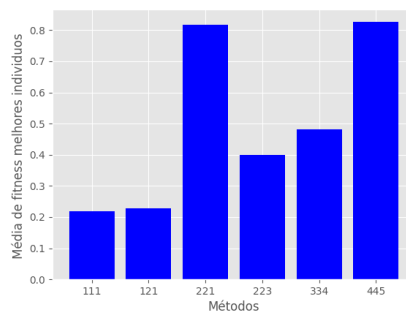


Figura 9: Média de *fitness* dos melhores indivíduos de cada geração por estratégia de evolução após 3 execuções, com 1% de probabilidade de mutação e *fitness* > 0.99.

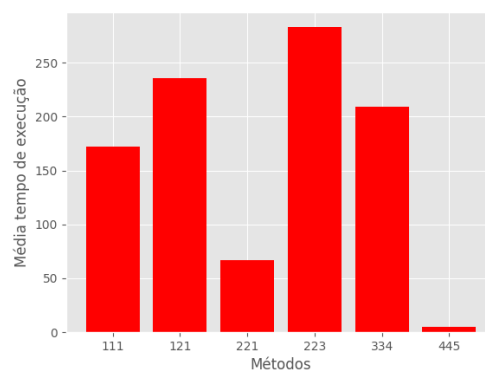


Figura 10: Média de tempo de execução (total) por estratégia de evolução após 3 execuções, com 1% de probabilidade de mutação e *fitness* > 0.99.

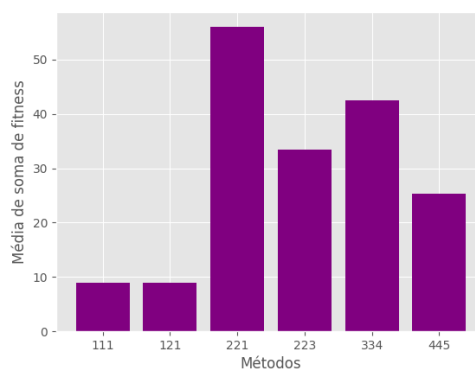


Figura 11: Média de soma de *fitness* dos indivíduos por estratégia de evolução após 20 execuções, com 1% de probabilidade de mutação e *fitness* > 0.99.

6. Justificativa e Conclusão

Por observação dos resultados na seção 5, podemos perceber que:

- em relação ao método de seleção, algoritmos muito elitistas como a seleção dos melhores pais de todos / piores indivíduos (Seção 3.1.1) e a sobrevivência dos mais fortes (Seção 3.1.3) não apresentam bons resultados;
- em relação ao cruzamento, os que utilizavam de aleatoriedade mais controlada foram os que mais se destacaram (BLX- α , Seção 3.2.3, e *Crossover* modificado, Seção 3.2.4), e os que eram mais fixos e baseados apenas nos *fitness* deixaram a desejar (Discreto, Seção 3.2.1, e Intermediário, Seção 3.2.2); e
- em relação à mutação, observamos um padrão um pouco oposto ao de cruzamento, onde aleatoriedade demais (mutações Uniforme, Seção 3.3.1, Não Uniforme, Seção 3.3.2 e Gaussiana, Seção 3.3.3) demonstraram um desempenho ruim, e mutações que não variavam tanto os números originais (BLX- α , Seção 3.3.4 e Não Uniforme modificada, Seção 3.3.5) trouxeram resultados melhores.

Acredita-se que esse comportamento se deve ao grande número de mínimos locais da função (vide Figura 12) e o fato de que temos um escopo muito grande de indivíduos, uma vez que números reais podem possuir muitas casas decimais.

Sendo assim, é necessário um balanceamento bom entre aleatoriedade (exploração) e controle (explotação). A ideia é sempre manter passos curtos baseados no que já temos no momento e por isso a junção da seleção com melhores pais aleatórios e substituição do pior indivíduo caso o filho seja melhor (seção 3.1.4), com o *crossover* modificado (seção 3.2.4) e a mutação não uniforme modificada (seção 3.3.5), que mantém um cuidado na exploração, sempre limitando o quão pode variar, conseguiu convergir com *fitness* ≈ 0.999 (Figura 2) e, mesmo com o *fitness* > 0.99 , convergiu com bem menos gerações que os outros algoritmos (Figura 7).

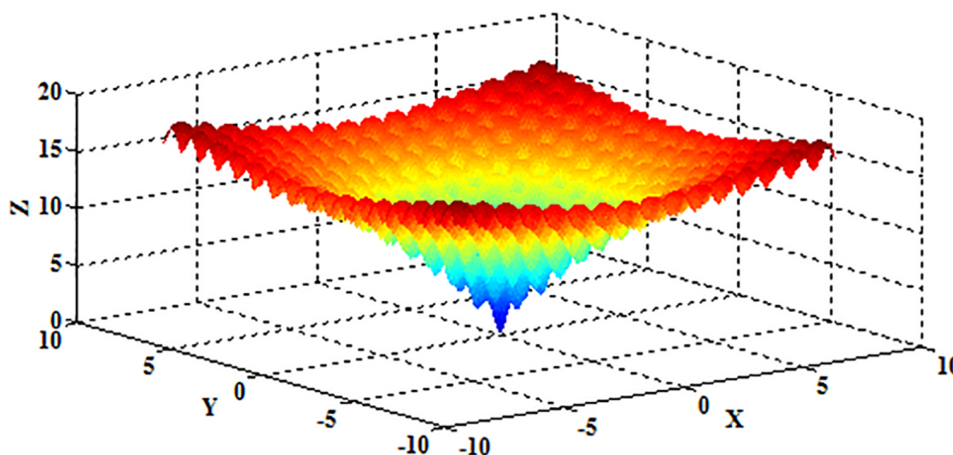


Figura 12: Distribuição dos valores da Função Ackley. Disponível em <https://journals.plos.org/plosone/article/figure?id=10.1371/journal.pone.0177666.g021>

7. Referências

- [1] Paulo Salgado Gomes de Mattos Neto. Primeira parte do projeto das 8 rainhas. Especificação Mini-Projeto.docx.
- [2] J Rico Alexander. Genetic algorithms: Solving the n-queens problem. <https://aljrigo.github.io/blog/genetic-algorithms/>. Accessed: 2020-11-02.
- [3] Joelan A. L. Santos, Márcio M. da Silva, José de A. da S. Júnior, André C. A. Firmo, and Tiago A. E. Ferreira. Algoritmo genético modificado para minimização de funções. *8º Encontro Regional de Matemática Aplicada e Computacional*, 2008.
- [4] Paulo Salgado Gomes de Mattos Neto. Slides usados em sala de aula. Aula06.pptx.
- [5] Larry J. Eshelman and J. David Schaffer. Real-coded genetic algorithms and interval-schemata. *Foundations of Genetic Algorithms*, pages 187–202, 1993. doi:10.1016/b978-0-08-094832-4.50018-0.
- [6] Clemens Heitzinger. <https://www.iue.tuwien.ac.at/phd/heitzinger/node27.html>. Accessed: 2020-11-02.