# Relatório: Algoritmos Genéticos

**Universidade Estadual Paulista "Júlio de Mesquita Filho" (UNESP)**
**Programa de Pós-Graduação em Ciência da Computação**
**Disciplina: Computação Inspirada pela Natureza**
**Prof.: Fabricio Breve**

## Sumário



## 1. Introdução

Este relatório apresenta os resultados obtidos na implementação e análise de algoritmos genéticos aplicados a três problemas distintos: reconhecimento de padrões, maximização de função e minimização da função de Rosenbrock. Os algoritmos genéticos são métodos de busca e otimização inspirados nos princípios da seleção natural e evolução biológica, sendo particularmente úteis para problemas de otimização complexos onde métodos tradicionais podem não ser eficientes.

A implementação foi realizada em Python, utilizando uma abordagem orientada a objetos que permite a reutilização de código entre os diferentes problemas. Para cada problema, foram realizados experimentos com diferentes configurações de parâmetros, como taxas de crossover e mutação, tamanhos de população e métodos de seleção, a fim de analisar o impacto dessas configurações no desempenho dos algoritmos.

O relatório está estruturado da seguinte forma: para cada problema, apresentamos uma descrição detalhada, a implementação realizada, os experimentos conduzidos e os resultados obtidos, seguidos de uma análise. Ao final, realizamos uma análise comparativa dos três problemas e apresentamos as conclusões gerais do trabalho.

## 2. Reconhecimento de Padrões

### 2.1 Descrição do Problema

O problema de reconhecimento de padrões consiste em implementar um algoritmo genético capaz de reconhecer um padrão específico representado por uma bitstring. No caso deste trabalho, o objetivo é reconhecer o número 0, representado pela bitstring [1 1 1 1 0 1 1 0 1 1 1 1].

O algoritmo deve ser capaz de evoluir uma população de indivíduos (representados por bitstrings) até encontrar o padrão alvo. A função de fitness é definida como o número de bits que correspondem ao padrão alvo, normalizado para o intervalo [0, 1].

### 2.2 Implementação

Para este problema, foi implementada uma classe `PatternRecognitionGA` que estende a classe base `GeneticAlgorithm`. A implementação inclui:

- Representação dos indivíduos como bitstrings de comprimento 12
- Função de fitness que calcula a similaridade com o padrão alvo
- Operadores de crossover (ponto único, dois pontos e uniforme)
- Operador de mutação que inverte bits aleatoriamente
- Métodos de seleção (roleta, torneio e amostragem universal estocástica)
- Elitismo para preservar os melhores indivíduos entre gerações

Além disso, foram implementadas funções para executar experimentos com diferentes configurações de parâmetros e para visualizar os resultados.

### 2.3 Experimentos e Resultados

Foram realizados três conjuntos de experimentos:

1. **Comparação de diferentes taxas de crossover e mutação**: Foram testadas combinações de taxas de crossover (0.6, 0.7, 0.8, 0.9) e taxas de mutação (0.01, 0.05, 0.1, 0.2).
2. **Comparação de operadores genéticos**: Foram realizados experimentos utilizando apenas crossover, apenas mutação, e ambos os operadores.
3. **Execução de um único experimento detalhado**: Foi executado um experimento com a melhor configuração encontrada, registrando a evolução do fitness ao longo das gerações.

Os resultados dos experimentos são apresentados a seguir:

**Tabela 1: Resultados para diferentes taxas de crossover e mutação**

| Configuração | Gerações Médias | Desvio Padrão | Tempo Médio (s) |
|---|---|---|---|
| CR=0.6, MR=0.01 | 6.80 | 1.87 | 0.0131 |
| CR=0.7, MR=0.01 | 4.87 | 2.28 | 0.0097 |
| CR=0.8, MR=0.01 | 5.10 | 2.37 | 0.0101 |
| CR=0.9, MR=0.01 | 5.80 | 2.88 | 0.0112 |

**Tabela 2: Resultados para diferentes operadores genéticos**

| Operadores | Gerações Médias | Desvio Padrão | Tempo Médio (s) |
|---|---|---|---|
| Ambos | 6.90 | 3.38 | 0.0131 |
| Apenas Crossover | 5.03 | 3.43 | 0.0103 |
| Apenas Mutação | 6.60 | 4.22 | 0.0122 |

### 2.4 Análise

A análise dos resultados permite observar que:

1. **Taxas de crossover e mutação**: A combinação de taxa de crossover 0.7 e taxa de mutação 0.01 apresentou o melhor desempenho, com uma média de 4.87 gerações para convergência. Taxas de mutação mais altas (0.2) resultaram em um número significativamente maior de gerações para convergência, indicando que mutações excessivas podem prejudicar a busca.
2. **Operadores genéticos**: Surpreendentemente, o uso de apenas crossover apresentou o melhor desempenho, com uma média de 5.03 gerações para convergência. Isso pode ser explicado pelo fato de que, para este problema específico, o crossover é suficiente para explorar o espaço de busca e encontrar o padrão alvo, enquanto a mutação pode introduzir perturbações desnecessárias.

3. **Evolução do fitness**: A evolução do fitness ao longo das gerações mostra um aumento rápido nas primeiras gerações, seguido por uma estabilização até que o padrão alvo seja encontrado. O fitness médio da população também aumenta, mas de forma mais gradual, indicando que a diversidade da população é mantida durante a evolução.

Em geral, o algoritmo genético mostrou-se eficiente para o problema de reconhecimento de padrões, convergindo para o padrão alvo em poucas gerações. A escolha adequada dos parâmetros, especialmente as taxas de crossover e mutação, tem um impacto significativo no desempenho do algoritmo.

## 3. Maximização de Função

### 3.1 Descrição do Problema

O problema de maximização de função consiste em encontrar o valor de x no intervalo [0, 1] que maximiza a função:

g(x) = 2^(-2((x-0,1)/0,9)²) (sin(5*pi*x))^6

Esta função possui múltiplos máximos locais, sendo o máximo global em x = 0.1, onde g(0.1) = 1.0.

O algoritmo genético deve ser capaz de encontrar o valor de x que maximiza a função, utilizando uma representação por bitstring com precisão de pelo menos três casas decimais.

### 3.2 Implementação

Para este problema, foi implementada uma classe `FunctionMaximizationGA` que estende a classe base `GeneticAlgorithm`. A implementação inclui:

- Representação dos indivíduos como bitstrings de comprimento 20 (permitindo uma precisão de mais de 6 casas decimais)
- Função para converter a representação binária para um valor decimal no intervalo [0, 1]
- Função de fitness que calcula o valor da função g(x) para o valor de x representado pelo indivíduo
- Operadores de crossover e mutação
- Métodos de seleção (roleta, torneio e amostragem universal estocástica)
- Elitismo para preservar os melhores indivíduos entre gerações

Além disso, foram implementados os algoritmos de Subida da Colina e Recozimento Simulado para comparação com o algoritmo genético.

### 3.3 Experimentos e Resultados

Foram realizados três conjuntos de experimentos:

1. **Comparação de diferentes métodos de seleção**: Foram testados os métodos de seleção por roleta, torneio e amostragem universal estocástica (SUS).
2. **Comparação com outros algoritmos**: O algoritmo genético foi comparado com os algoritmos de Subida da Colina e Recozimento Simulado.
3. **Execução de um único experimento detalhado**: Foi executado um experimento com a melhor configuração encontrada, registrando a evolução do fitness ao longo das gerações.

Os resultados dos experimentos são apresentados a seguir:

**Tabela 3: Resultados para diferentes métodos de seleção**

| Método de Seleção | Gerações Médias | Tempo Médio (s) | Valor g(x) |
|---|---|---|---|
| Roleta | 1000.00 | 1.9699 | 1.000000 |
| Torneio | 1000.00 | 1.2389 | 1.000000 |

| Método de Seleção | Gerações Médias | Tempo Médio (s) | Valor g(x) |
|---|---|---|---|
| SUS | 1000.00 | 1.6775 | 1.000000 |

**Tabela 4: Comparação com outros algoritmos**

| Algoritmo | Valor g(x) | Desvio Padrão | Iterações | Tempo (s) |
|---|---|---|---|---|
| Algoritmo Genético | 1.000000 | 0.000000 | 1000.00 | 1.9595 |
| Subida da Colina | 0.752050 | 0.230197 | 206.90 | 0.0043 |
| Recozimento Simulado | 0.841810 | 0.165586 | 449.00 | 0.0022 |

### 3.4 Análise

A análise dos resultados permite observar que:

1. **Métodos de seleção**: Todos os métodos de seleção (roleta, torneio e SUS) foram capazes de encontrar o valor ótimo de x = 0.1, resultando em g(x) = 1.0. O método de torneio foi o mais eficiente em termos de tempo de execução, enquanto o método de roleta foi o mais lento.
2. **Comparação com outros algoritmos**: O algoritmo genético superou significativamente os algoritmos de Subida da Colina e Recozimento Simulado em termos de qualidade da solução, encontrando consistentemente o valor ótimo global. No entanto, os algoritmos de Subida da Colina e Recozimento Simulado foram muito mais rápidos, com tempos de execução cerca de 450 e 900 vezes menores, respectivamente.
3. **Evolução do fitness**: A evolução do fitness ao longo das gerações mostra um aumento rápido nas primeiras gerações, seguido por uma estabilização à medida que o algoritmo se aproxima do valor ótimo. O fitness médio da população também aumenta, mas de forma mais gradual, indicando que a diversidade da população é mantida durante a evolução.

Em geral, o algoritmo genético mostrou-se muito eficaz para o problema de maximização da função g(x), encontrando consistentemente o valor ótimo global. A representação por bitstring com 20 bits proporcionou uma precisão adequada para o problema, permitindo que o algoritmo explorasse eficientemente o espaço de busca.

## 4. Minimização da Função de Rosenbrock

### 4.1 Descrição do Problema

O problema de minimização da função de Rosenbrock consiste em encontrar os valores de x e y no intervalo [-10, 10] que minimizam a função:

$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$

Esta função, também conhecida como "banana function" devido à forma de seu contorno, é um problema clássico de otimização. O mínimo global está localizado em (x, y) = (1, 1), onde f(1, 1) = 0.

O algoritmo genético deve ser capaz de encontrar os valores de x e y que minimizam a função, utilizando uma representação adequada para as variáveis no intervalo especificado.

### 4.2 Implementação

Para este problema, foi implementada uma classe `RosenbrockGA` que utiliza uma abordagem similar à dos problemas anteriores, mas adaptada para lidar com duas variáveis. A implementação inclui:

- Representação dos indivíduos como bitstrings de comprimento 32 (16 bits para cada variável)
- Função para converter a representação binária para valores decimais no intervalo [-10, 10]

- Função de fitness que calcula o inverso do valor da função de Rosenbrock (para transformar o problema de minimização em maximização)
- Operadores de crossover e mutação
- Métodos de seleção (roleta, torneio e amostragem universal estocástica)
- Versões com e sem elitismo para comparação

Além disso, foram implementadas funções para testar diferentes tamanhos de população e para visualizar os resultados.

### 4.3 Experimentos e Resultados

Foram realizados três conjuntos de experimentos:

1. **Comparação com e sem elitismo**: Foram realizados experimentos com e sem elitismo para avaliar o impacto desta técnica no desempenho do algoritmo.
2. **Diferentes tamanhos de população**: Foram testados tamanhos de população de 20, 50, 100 e 200 indivíduos.
3. **Execução de um único experimento detalhado**: Foi executado um experimento com a melhor configuração encontrada, registrando a evolução do fitness ao longo das gerações.

Os resultados dos experimentos são apresentados a seguir:

**Tabela 5: Resultados para comparação com e sem elitismo**

| Configuração | Fitness | Valor x | Valor y | Tempo (s) |
|---|---|---|---|---|
| Com Elitismo | 0.999976 | 1.002152 | 1.004379 | 6.7188 |
| Sem Elitismo | 0.999465 | 1.005844 | 1.011765 | 6.9528 |

**Tabela 6: Resultados para diferentes tamanhos de população**

| Tamanho da População | Fitness | Valor x | Valor y | Tempo (s) |
|---|---|---|---|---|
| 20 | 0.892605 | 0.805707 | 0.819257 | 0.9752 |
| 50 | 0.949882 | 0.896071 | 0.892317 | 3.0590 |
| 100 | 0.999973 | 1.001755 | 1.003616 | 6.9266 |
| 200 | 0.999991 | 0.998489 | 0.997055 | 18.3566 |

### 4.4 Análise

A análise dos resultados permite observar que:

1. **Elitismo**: O uso de elitismo resultou em um desempenho significativamente melhor, com valores de x e y mais próximos do ótimo global (1, 1) e um valor de fitness mais alto. Isso confirma a importância de preservar os melhores indivíduos entre gerações, especialmente para problemas complexos como a minimização da função de Rosenbrock.
2. **Tamanho da população**: O aumento do tamanho da população resultou em uma melhoria consistente na qualidade da solução, com a população de 200 indivíduos alcançando resultados muito próximos do ótimo global. No entanto, o tempo de execução também aumentou significativamente, com a população de 200 indivíduos levando cerca de 19 vezes mais tempo que a população de 20 indivíduos.
3. **Evolução do fitness**: A evolução do fitness ao longo das gerações mostra um aumento gradual, refletindo a dificuldade do problema. O fitness médio da população também aumenta, mas com uma diferença maior em relação ao melhor fitness, indicando a manutenção de diversidade na população.

Em geral, o algoritmo genético mostrou-se eficaz para o problema de minimização da função de Rosenbrock, especialmente com uma população grande e utilizando elitismo. A melhor configuração testada (população de 200 indivíduos com elitismo) encontrou uma solução muito próxima do ótimo global, com f(x, y) aproximadamente 0.000009.

## 5. Análise Comparativa

A análise comparativa dos três problemas abordados neste trabalho permite observar diferenças significativas em termos de complexidade, desempenho e configurações ótimas.

**Tabela 7: Análise comparativa dos problemas**

| Problema | Melhor Configuração | Gerações Médias | Tempo Médio (s) | Qualidade da Solução |
|---|---|---|---|---|
| Reconhecimento de Padrões | CR=0.7, MR=0.01 | 4.87 | 0.0097 | 1.0 |
| Maximização de Função | Roleta/Torneio/SUS | 1000.00 | 1.9595 | 1.0 |
| Minimização de Rosenbrock | Pop=200, Com Elitismo | 1000.00 | 18.3566 | 0.999991 |

### Complexidade dos Problemas

O problema de reconhecimento de padrões mostrou-se o mais simples, com convergência em poucas gerações e tempo de execução muito baixo. Isso se deve ao espaço de busca relativamente pequeno (2^12 = 4096 possíveis soluções) e à natureza discreta do problema.

O problema de maximização de função apresentou uma complexidade intermediária, com um espaço de busca maior (2^20 = 1048576 possíveis soluções) e a necessidade de lidar com valores contínuos. Apesar disso, o algoritmo genético foi capaz de encontrar consistentemente o valor ótimo global.

O problema de minimização da função de Rosenbrock foi o mais complexo, com um espaço de busca ainda maior (2^32 = 4294967296 possíveis soluções) e a necessidade de lidar com duas variáveis contínuas. A função de Rosenbrock é conhecida por ser um desafio para algoritmos de otimização devido à sua forma de vale estreito e curvado.

### Configurações Ótimas

As configurações ótimas variaram significativamente entre os problemas:

- Para o reconhecimento de padrões, uma taxa de crossover moderada (0.7) e uma taxa de mutação baixa (0.01) proporcionaram os melhores resultados.
- Para a maximização de função, todos os métodos de seleção testados foram eficazes, mas o método de torneio foi o mais eficiente em termos de tempo de execução.
- Para a minimização da função de Rosenbrock, uma população grande (200 indivíduos) e o uso de elitismo foram cruciais para obter bons resultados.

Essas diferenças destacam a importância de ajustar os parâmetros do algoritmo genético de acordo com as características específicas do problema a ser resolvido.

### Desempenho Comparativo

Em termos de desempenho, o algoritmo genético mostrou-se mais eficaz para o problema de reconhecimento de padrões, seguido pela maximização de função e, por fim, pela minimização da função de Rosenbrock.

Para o problema de maximização de função, o algoritmo genético superou significativamente os algoritmos de Subida da Colina e Recozimento Simulado em termos de qualidade da solução, mas foi muito mais lento. Isso

ilustra o trade-off entre qualidade da solução e tempo de execução que frequentemente ocorre em problemas de otimização.

## 6. Conclusões

Este trabalho apresentou a implementação e análise de algoritmos genéticos aplicados a três problemas distintos: reconhecimento de padrões, maximização de função e minimização da função de Rosenbrock. As principais conclusões são:

1. **Eficácia dos algoritmos genéticos**: Os algoritmos genéticos mostraram-se eficazes para os três problemas, sendo capazes de encontrar soluções ótimas ou muito próximas do ótimo. Isso confirma a versatilidade desta abordagem para diferentes tipos de problemas de otimização.
2. **Importância dos parâmetros**: A escolha adequada dos parâmetros, como taxas de crossover e mutação, tamanho da população e método de seleção, tem um impacto significativo no desempenho do algoritmo. Os parâmetros ótimos variam de acordo com as características específicas do problema.
3. **Elitismo**: O uso de elitismo mostrou-se particularmente importante para problemas complexos, como a minimização da função de Rosenbrock, onde preservar os melhores indivíduos entre gerações contribui significativamente para a qualidade da solução final.
4. **Trade-off entre qualidade e tempo**: Foi observado um trade-off entre a qualidade da solução e o tempo de execução, especialmente na comparação entre o algoritmo genético e os algoritmos de Subida da Colina e Recozimento Simulado para o problema de maximização de função.
5. **Representação adequada**: A representação por bitstring mostrou-se adequada para os três problemas, permitindo uma implementação unificada e facilitando a aplicação dos operadores genéticos. Para os problemas contínuos, a escolha do comprimento da bitstring afeta diretamente a precisão da solução.

Em resumo, os algoritmos genéticos demonstraram ser uma abordagem poderosa e flexível para problemas de otimização, capaz de lidar com diferentes tipos de problemas e encontrar soluções de alta qualidade. No entanto, a eficácia desta abordagem depende fortemente da escolha adequada dos parâmetros e da representação, que devem ser ajustados de acordo com as características específicas do problema a ser resolvido.

## 7. Referências


1. Mitchell, M. (1998). An Introduction to Genetic Algorithms. MIT Press.
2. Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley.
3. Holland, J. H. (1992). Adaptation in Natural and Artificial Systems. MIT Press.
4. Eiben, A. E., & Smith, J. E. (2003). Introduction to Evolutionary Computing. Springer.
5. Rosenbrock, H. H. (1960). An automatic method for finding the greatest or least value of a function. The Computer Journal, 3(3), 175-184.
6. Whitley, D. (1994). A genetic algorithm tutorial. Statistics and Computing, 4(2), 65-85.
7. De Jong, K. A. (1975). An analysis of the behavior of a class of genetic adaptive systems. PhD thesis, University of Michigan.
8. Michalewicz, Z. (1996). Genetic Algorithms + Data Structures = Evolution Programs. Springer.