Relatório de Implementação: Algoritmo Genético para o Problema da Mochila

Autores: Maria Eduarda S. e Fredson Arthur

Resumo

Este relatório descreve o processo de desenvolvimento de um algoritmo genético para resolução do Problema da Mochila 0/1, implementado em Python. O trabalho visa aplicar operadores genéticos variados e técnicas de otimização para encontrar soluções eficientes, respeitando as restrições de capacidade da mochila. A estrutura do código foi organizada de forma modular e flexível, permitindo a avaliação do desempenho dos diferentes operadores de crossover.

Palavras-chave: Algoritmos genéticos. Otimização. Problema da Mochila. Python.

1. Introdução

O Problema da Mochila (Knapsack Problem) é um problema clássico de otimização combinatória, no qual se busca selecionar um subconjunto de itens com valor máximo, respeitando uma capacidade de peso limitada. O presente trabalho tem como objetivo desenvolver uma solução baseada em algoritmos genéticos (AG) para esse problema, seguindo os requisitos definidos na Atividade 4 da disciplina.

2. Metodologia

2.1 Estrutura do Código

O código-fonte foi desenvolvido no arquivo `codigo.py`, organizado em blocos lógicos para facilitar a leitura e a manutenção. A estrutura está dividida nas seguintes partes:

a) Configuração e Parâmetros

Inicialmente, foram definidos:

- Dados do Problema: Vetores de `WEIGHTS` (pesos), `VALUES` (valores) e `KNAPSACK_CAPACITY` (capacidade da mochila). Esses dados foram inseridos como placeholders, aguardando substituição pelos dados reais da Atividade 1.

- Parâmetros do AG: Variáveis que controlam o comportamento do algoritmo, como: `POPULATION_SIZE`, `GENERATIONS`, `CROSSOVER_RATE`, `MUTATION_RATE`,

`TOURNAMENT_SIZE` e `ELITE_SIZE`.

b) Funções do Algoritmo Genético

O núcleo da solução foi construído por meio de funções específicas:

- `create_individual()` e `create_initial_population()`: Geram, respectivamente, um indivíduo aleatório e a população inicial.
- `calculate_fitness()`: Avalia a qualidade (aptidão) dos indivíduos. Se o peso total de um indivíduo ultrapassa a capacidade, aplica-se penalização com valor de fitness zero.
- `tournament_selection()`: Realiza a seleção de pais com base no método de torneio.
- Funções de crossover (`crossover_one_point`, `crossover_two_points`, `crossover_uniform`): Implementam os três tipos de recombinação genética conforme exigido.
- `bit_flip_mutation()`: Realiza a mutação dos indivíduos, alterando bits com base na taxa de mutação.

2.2 Motor Principal do Algoritmo (`run_ga`)

A função `run_ga` executa o fluxo completo do algoritmo genético:

1. Geração da população inicial;
2. Iteração ao longo de `GENERATIONS`;
3. Aplicação de elitismo para preservar os melhores indivíduos;
4. Geração da nova população via seleção, crossover e mutação;
5. Armazenamento do melhor fitness por geração;
6. Retorno da melhor solução e histórico de evolução do fitness.

3. Desenvolvimento e Refatoração

3.1 Implementação Inicial

A primeira versão do código foi escrita em português, com uso extensivo de comentários explicativos e divisores visuais, com o intuito de facilitar a compreensão.

3.2 Refatoração

Com o objetivo de adequar o código aos padrões da comunidade de desenvolvedores, foram realizadas as seguintes mudanças:

- Tradução para o inglês: Identificadores, nomes de funções e comentários foram traduzidos;
- Adição de type hinting: Foram incluídas anotações de tipo para tornar o código mais robusto;
- Redução de comentários: Comentários excessivos foram removidos, mantendo apenas o essencial;
- Padronização de docstrings: As documentações internas foram reescritas seguindo o formato padrão e conciso.

4. Conclusão e Próximos Passos

O algoritmo genético está implementado de forma funcional e modular, possibilitando testes com diferentes operadores genéticos. Os próximos passos para conclusão da atividade são:

1. Inserção dos dados reais da Atividade 1 nos vetores `WEIGHTS`, `VALUES` e `KNAPSACK_CAPACITY`;
2. Execução de 30 testes para cada tipo de crossover;
3. Cálculo das métricas estatísticas (média e desvio padrão);
4. Geração dos gráficos de convergência e boxplots comparativos (com dados da Atividade 3).