Projeto 4: Algoritmo Genético para o Problema das N-Rainhas

Autor: Marcos Antonio Teles de Castilhos

Disciplina: FGA0221 - Inteligência Artificial

Professor: Fabiano Araujo Soares, Dr.

1. Introdução

Este projeto implementa um **Algoritmo Genético (AG)**, uma técnica de **Busca Complexa** inspirada nos princípios da evolução natural, para resolver o problema das N-Rainhas. O objetivo, assim como nos projetos de busca local anteriores, é encontrar uma configuração de N rainhas em um tabuleiro N×N sem que nenhuma ataque a outra.

Diferente do Hill-Climbing e do Simulated Annealing, que operam sobre uma única solução candidata, os Algoritmos Genéticos trabalham com uma **população de soluções** que evolui ao longo de **gerações**. Através de operadores como seleção, crossover e mutação, a população tende a convergir para soluções de alta qualidade (alto "fitness").

O script inclui uma **visualização gráfica** (matplotlib) que mostra, a cada geração, o tabuleiro da melhor solução encontrada até o momento e um gráfico da evolução do melhor fitness da população.

2. Conceitos do Algoritmo Genético Implementado

O AG segue um ciclo iterativo, aplicando os seguintes conceitos e operadores:

- **Representação (Indivíduo/Cromossomo):** Cada solução candidata (um tabuleiro) é representada como uma lista de N inteiros, onde o índice i é a coluna e o valor **individuo[i]** é a linha da rainha. Ex: **[1, 3, 0, 2]** para 4 rainhas.
- **Função Fitness (calcular_fitness):** Avalia a qualidade de cada indivíduo. Neste projeto, o fitness é definido como o número máximo de pares de

rainhas possíveis *menos* o número de pares que se atacam. O objetivo é **maximizar** essa função. Uma solução perfeita terá o fitness máximo (igual a N*(N-1)/2).

- **População:** Um conjunto de indivíduos (soluções). O tamanho da população (**tam_populacao**) é um parâmetro importante.
- **Gerações:** O número de ciclos iterativos que o algoritmo executa (**geracoes**).
- **Seleção (selecionar_pais):** Processo de escolha dos indivíduos ("pais") que irão gerar a próxima geração. Foi implementado o método de **Seleção por Torneio**: K indivíduos são escolhidos aleatoriamente, e o melhor entre eles (maior fitness) é selecionado como pai.
- **Crossover (Recombinação) (crossover):** Combina o material genético de dois pais para criar um ou mais descendentes ("filhos"). Foi implementado o **Crossover de Ponto Único**: um ponto de corte aleatório é escolhido, e o filho é formado pela primeira parte do pai1 e a segunda parte do pai2.
- **Mutação (mutacao):** Introduz pequenas alterações aleatórias nos filhos com uma certa probabilidade (**taxa_mutacao**). Neste caso, a mutação consiste em escolher uma coluna aleatória e mover a rainha para uma linha aleatória. A mutação ajuda a manter a diversidade na população e a explorar novas áreas do espaço de busca.
- **Elitismo:** Uma estratégia onde o(s) melhor(es) indivíduo(s) da geração atual é(são) garantidamente copiado(s) para a próxima geração. Isso assegura que o melhor fitness encontrado nunca diminua. O script implementa um elitismo simples, preservando o melhor indivíduo.
- **Ciclo Evolutivo:** O processo se repete: Avaliação -> Seleção -> Crossover -> Mutação -> Nova Geração, até que uma solução ótima seja encontrada ou o número máximo de gerações seja atingido.

3. Implementação e Visualização Gráfica

- **Função Principal: algoritmo_genetico_visual** orquestra todo o ciclo evolutivo e as chamadas de visualização.
- **Visualização (matplotlib, numpy):**
    - A janela gráfica é dividida em duas partes:
        - **À Esquerda (ax_tabuleiro):** Mostra a configuração do tabuleiro do **melhor indivíduo** encontrado na geração atual.

        - **À Direita (ax_grafico):** Exibe um gráfico de linha mostrando a evolução do **melhor fitness** da população ao longo das gerações. A linha ascendente do gráfico representa o progresso do algoritmo em direção a uma solução melhor.
    - A animação é atualizada a cada geração, permitindo observar tanto a melhor configuração encontrada quanto a tendência geral de melhoria da população.

4. Como Usar o Programa

1. **Pré-requisitos:** Python 3, **matplotlib e numpy**. Instale com:

   ```
   pip install matplotlib numpy
   ```

2. **Execução:**

   ```
   python algoritmo_genetico.py
   ```

3. **Ajuste (Opcional):** Você pode alterar os parâmetros do AG na chamada da função **algoritmo_genetico_visual** no final do script:
    - **n_rainhas**: Tamanho do tabuleiro.
    - **tam_populacao**: Número de indivíduos por geração.
    - **geracoes**: Número máximo de iterações.
    - **taxa_mutacao**: Probabilidade de ocorrer uma mutação.

5. Interpretando a Saída

- Uma **janela gráfica** mostrará a evolução.
- Observe o **tabuleiro à esquerda**. Ele representa a melhor solução encontrada *até aquela geração*.
- Acompanhe o **gráfico à direita**. A linha deve tender a subir, indicando que a população está melhorando. O objetivo é atingir o valor máximo de fitness (linha superior do gráfico).
- A animação para quando a solução ótima (fitness máximo) é encontrada ou quando o número máximo de gerações é atingido.
- O resultado final (número de ataques do melhor indivíduo encontrado) é impresso no terminal.

6. Conclusão

Este projeto demonstra a implementação de um Algoritmo Genético para um problema de otimização combinatória (N-Rainhas). Diferente das buscas locais que operam em uma única solução, o AG explora o espaço de busca de forma paralela através de uma população, tornando-o mais robusto contra mínimos/máximos locais. A visualização da evolução do fitness ao longo das gerações ilustra o processo de convergência da população em direção a soluções de alta qualidade.