Projeto 3: Busca Complexa (Local Search) - Hill Climbing e Simulated Annealing

Autor: Marcos Antonio Teles de Castilhos

Disciplina: FGA0221 - Inteligência Artificial

Professor: Fabiano Araujo Soares, Dr.

1. Introdução

Este projeto explora a **Busca Local**, uma estratégia de Inteligência Artificial voltada para **problemas de otimização**, onde o objetivo é encontrar a melhor configuração final, independentemente do caminho percorrido. Foi utilizado o clássico problema das **N-Rainhas** como estudo de caso, buscando posicionar N rainhas em um tabuleiro N×N sem ataques mútuos.

Foram implementados dois algoritmos principais de Busca Local:

- **Hill-Climbing:** Uma abordagem gananciosa que sempre busca a melhoria imediata.
- **Simulated Annealing:** Uma metaheurística que permite movimentos probabilisticamente piores para escapar de ótimos locais.

O projeto consiste em três scripts Python:

1. **busca-complexa.py**: Implementação visual do Hill-Climbing.
2. **simulated_annealing.py**: Implementação visual do Simulated Annealing.
3. **hc-vs-sa.py**: Ferramenta para análise estatística comparativa (sem visualização).

2. Formulação do Problema (N-Rainhas)

- **Estado:** Lista onde **estado[i]** é a linha da rainha na coluna **i**.
- **Função Objetivo:** Minimizar o número de pares de rainhas em ataque (horizontal/diagonal), calculado por **calcular_ataques**. O ótimo global tem 0 ataques.

- **Vizinhança:** Mover uma única rainha para uma nova linha em sua coluna.

3. Implementações

3.1. busca-complexa.py: Hill-Climbing Visual

- **Algoritmo:** Implementa o Hill-Climbing padrão. A cada passo, avalia todos os vizinhos e move-se para o que tiver o menor número de ataques, desde que seja melhor que o estado atual.
- **Limitação:** Propenso a ficar preso em **mínimos locais** (estados onde nenhum vizinho é estritamente melhor, mas que não são a solução ótima).
- **Visualização (matplotlib, numpy):**
    - Mostra o tabuleiro 8x8.
    - Rainhas em conflito são destacadas em **vermelho**, rainhas seguras em **dourado**.
    - O título exibe o número de ataques atual.
    - A animação para ao encontrar uma solução (0 ataques) ou um mínimo local.

3.2. simulated_annealing.py: Simulated Annealing Visual

- **Algoritmo:** Implementa o Simulated Annealing. A cada passo:
    1. Escolhe um vizinho **aleatório**.
    2. Aceita o vizinho se for melhor.
    3. Se for pior, aceita com probabilidade p = e^(Delta E / T), onde Delta E é a piora e T é a temperatura.
    4. A temperatura T diminui gradualmente (**taxa_resfriamento**).
- **Vantagem:** A capacidade de aceitar movimentos piores permite **escapar de mínimos locais**. Utiliza parâmetros otimizados (**temperatura_inicial=1000.0, taxa_resfriamento=0.999**) para alta taxa de sucesso.
- **Visualização:** Similar ao Hill-Climbing, mas o título também exibe a Temperatura atual, mostrando o processo de "resfriamento".

3.3. hc-vs-sa.py: Análise Comparativa (Batch)

- **Propósito:** Comparar a **robustez** (taxa de sucesso) do Hill-Climbing vs. Simulated Annealing.

- **Funcionamento:** Executa cada algoritmo (**hill_climbing_batch, simulated_annealing_batch**) 100 vezes (ou **NUM_EXECUCOES**) em modo silencioso (sem gráficos).
- **Saída:** Imprime um relatório no terminal para cada algoritmo, mostrando:
    - Tempo total de execução.
    - Número de sucessos (soluções com 0 ataques encontradas).
    - Taxa de sucesso percentual.
- **Resultados Esperados:** Demonstra empiricamente a **maior taxa de sucesso** do Simulated Annealing em comparação com o Hill-Climbing puro.

4. Como Usar

1. **Pré-requisitos:** Python 3, matplotlib, numpy. Instale com pip install matplotlib numpy.
2. **Execução Visual:**

    python busca-complexa.py
    OU
    python simulated_annealing.py
    - Observe a animação e o resultado final. Feche a janela para terminar.
3. **Execução da Análise:**

    python hc-vs-sa.py
    - Aguarde a conclusão das 100 execuções de cada algoritmo e compare os relatórios impressos.

5. Conceitos de IA Demonstrados

- **Busca Local:** Foco na otimização do estado atual, baixo consumo de memória.
- **Função Objetivo:** Quantificar a qualidade de uma solução candidata.
- **Mínimos Locais:** Limitação fundamental de algoritmos puramente gananciosos como o Hill-Climbing.
- **Metaheurísticas:** Estratégias (como Simulated Annealing) para guiar a busca local de forma mais eficaz e escapar de ótimos locais.
- **Análise Empírica:** Importância de testar algoritmos estocásticos múltiplas vezes para avaliar seu desempenho médio.