Tópico: Busca Local - Hill Climbing e Simulated Annealing

Objetivo

Aplicar os métodos estudados para solução de problemas

Referências

Capítulo 4.1 do livro Stuart Russell e Peter Norvig. Inteligência Artificial - Tradução da Terceira Edição. Elsevier Editora Ltda, 2013
Introdução à IA #16: Busca de Subida de Encosta - YouTube
How to Implement the Hill Climbing Algorithm in Python | by Hein de Haan | Towards Data Science
Subida da encosta (hill climb) em Python - YouTube
Cadernos IA - 3_1 - Busca Local - Subida da Encosta - YouTube
Inteligência Artificial - Aula 7 - Simulated Annealing - YouTube
Simulated Annealing - YouTube
Algoritmos de otimização: Hill Climbing e Simulated Annealing | by Alvaro Leandro Cavalcante Carneiro | Data Hackers | Medium
Capítulo 4.1.2 do livro Stuart Russell e Peter Norvig. Inteligência Artificial - Tradução da Terceira Edição. Elsevier Editora Ltda, 2013

Exercício: Implementação dos Métodos de Hill Climbing e Simulated Annealing para o Problema das 8 Rainhas

Descrição do Problema: O problema das 8 rainhas consiste em posicionar 8 rainhas em um tabuleiro de xadrez 8x8 de forma que nenhuma rainha possa atacar outra. Isso significa que não pode haver duas rainhas na mesma linha, coluna ou diagonal.

Objetivo: Implementar e comparar os métodos de Hill Climbing e Simulated Annealing para resolver o problema das 8 rainhas.

Instruções:

1. Representação do Problema em Python:

- Representar o tabuleiro como uma lista de 8 elementos, onde cada elemento indica a linha em que a rainha está posicionada na respectiva coluna. Por exemplo, o estado [0, 4, 7, 5, 2, 6, 1, 3] indica que a rainha na primeira coluna está na linha 0, a rainha na segunda coluna está na linha 4, e assim por diante.

2. Função para Retornar os Vizinhos:
   - Implementar uma função que gere todos os vizinhos de um estado atual. Um vizinho é definido como um estado onde uma única rainha foi movida para uma nova posição na mesma coluna.

Python

```python
import random

def generate_neighbors(state):
    neighbors = []
    for col in range(len(state)):
        for row in range(len(state)):
            if state[col] != row:
                neighbor = state.copy()
                neighbor[col] = row
                neighbors.append(neighbor)
    return neighbors

# Exemplo de uso:
initial_state = [random.randint(0, 7) for _ in range(8)]
neighbors = generate_neighbors(initial_state)
print("Estado inicial:", initial_state)
print("Vizinhos gerados:", neighbors)
```

3. Hill Climbing:
   - Implemente o algoritmo de Hill Climbing para encontrar uma solução para o problema das 8 rainhas.
   - Utilize uma função de avaliação que conte o número de pares de rainhas que se atacam.
   - A cada iteração, utilize a função de geração de vizinhos, calcule a função objetivo de cada solução e escolha o melhor vizinho
   - Se não houver melhorias possíveis, o algoritmo deve parar.
4. Simulated Annealing:
   - Implemente o algoritmo de Simulated Annealing para resolver o problema das 8 rainhas.
   - Utilize a mesma função de avaliação do Hill Climbing.
   - Defina uma temperatura inicial e uma taxa de resfriamento.
   - A cada iteração, explore a vizinhança e calcule a função objetivo de cada solução

   - Aceite movimentos que melhorem a função de avaliação ou, com uma certa probabilidade, movimentos que a piorem, dependendo da temperatura atual.
   - Reduza a temperatura gradualmente até que o algoritmo pare.
5. Comparação:
   - Execute ambos os algoritmos várias vezes e compare os resultados.
   - Meça o tempo de execução e o número de iterações necessárias para encontrar uma solução.
   - Discuta as vantagens e desvantagens de cada método com base nos resultados obtidos.

Entrega:

- Código fonte dos algoritmos implementados.
- Relatório com a descrição dos métodos, resultados dos experimentos e análise comparativa.