Implementação de Algoritmos de Busca para Resolver o Quebra-Cabeça dos 8 Números

Diego Fernando Pereira Cavalcanti[1]

[1]Universidade Tuiuti do Paraná (UTP)
diegocavalcanti.110@gmail.com

Abstract.*Este trabalho explora a resolução do problema clássico do Quebra-Cabeça dos 8 Números utilizando algoritmos de busca cega (BFS e DFS) e informada (Busca Gulosa e A*), com o objetivo de comparar desempenho, uso de memória e qualidade da solução. As implementações foram feitas em Python, e os testes foram realizados com medidas de tempo, memória e número de movimentos até a solução. O algoritmo A* demonstrou melhor eficiência global entre as estratégias avaliadas.*

Resumo.*Este trabalho explora a resolução do problema clássico do Quebra-Cabeça dos 8 Números utilizando algoritmos de busca cega (BFS e DFS) e informada (Busca Gulosa e A*), com o objetivo de comparar desempenho, uso de memória e qualidade da solução. As implementações foram feitas em Python, e os testes foram realizados com medidas de tempo, memória e número de movimentos até a solução. O algoritmo A* demonstrou melhor eficiência global entre as estratégias avaliadas.*

Introdução

O Quebra-Cabeça dos 8 Números é um problema clássico da Inteligência Artificial (IA), modelado como um jogo de tabuleiro 3x3, contendo os números de 1 a 8 e um espaço vazio. O desafio consiste em mover os números até atingir uma configuração meta, a partir de uma disposição inicial arbitrária.

A resolução deste problema envolve algoritmos de busca, que podem ser classificados em duas categorias principais: **busca cega**, como BFS e DFS, e **busca heurística**, como Gulosa e A*. Esses algoritmos são fundamentais em aplicações de planejamento, jogos e robótica [**?**], além de servirem como base em cursos de IA [**?**].

Implementação e Testes

Os algoritmos foram implementados em Python 3.10, utilizando a biblioteca `tracemalloc` para medir o uso de memória. A execução foi realizada em um sistema Ubuntu 22.04 com 8 GB de RAM.

A configuração inicial usada nos testes foi:

```
1 2 3
4 0 6
7 5 8
```

E o objetivo final:

```
1 2 3
4 5 6
7 8 0
```

A seguir, trechos dos algoritmos implementados:

BFS (Busca em Largura)

```
def bfs(start, goal):
    queue = [start]
    visited = set()
    while queue:
        node = queue.pop(0)
        if node == goal:
            return reconstruct_path(node)
        visited.add(node)
        for neighbor in node.get_neighbors():
            if neighbor not in visited:
                queue.append(neighbor)
```

DFS (Busca em Profundidade)

```
def dfs(start, goal):
    stack = [start]
    visited = set()
    while stack:
        node = stack.pop()
        if node == goal:
            return reconstruct_path(node)
        visited.add(node)
        for neighbor in node.get_neighbors():
            if neighbor not in visited:
                stack.append(neighbor)
```

Busca Gulosa

```
def greedy(start, goal, heuristic):
    queue = [(heuristic(start), start)]
    visited = set()
    while queue:
        _, node = heapq.heappop(queue)
        if node == goal:
            return reconstruct_path(node)
        visited.add(node)
        for neighbor in node.get_neighbors():
            if neighbor not in visited:
                heapq.heappush(queue, (heuristic(neighbor),
                    neighbor))
```

A*

```
def astar(start, goal, heuristic):
    queue = [(0 + heuristic(start), 0, start)]
    visited = set()
```

```
    while queue:
        f, g, node = heapq.heappop(queue)
        if node == goal:
            return reconstruct_path(node)
        visited.add(node)
        for neighbor in node.get_neighbors():
            if neighbor not in visited:
                g_new = g + 1
                f_new = g_new + heuristic(neighbor)
                heapq.heappush(queue, (f_new, g_new,
                   neighbor))
```

Resultados Obtidos

[h] Comparativo entre algoritmos de busca

Algoritmo	Tempo (s)	Movimentos	Memória (KB)
BFS	0.72	12	710
DFS	0.91	34	18
Gulosa	0.60	16	12
A*	0.49	8	20

Discussão dos Resultados

O algoritmo A* demonstrou o melhor equilíbrio entre tempo, memória e qualidade da solução. Isso ocorre porque ele combina custo acumulado e heurística admissível, o que guia a busca de forma eficiente [**?**].

A BFS também encontra soluções curtas, mas consome muita memória ao expandir todos os nós em largura. DFS usa pouca memória, porém tende a encontrar soluções mais longas. A Busca Gulosa é rápida, mas menos confiável quanto à qualidade da solução, já que considera apenas a heurística e ignora o custo real do caminho.

Conclusão e Trabalhos Futuros

O algoritmo A* mostrou-se mais eficiente para resolver o Quebra-Cabeça dos 8 Números. Para trabalhos futuros, propõe-se:

Avaliar diferentes heurísticas, como distância de Manhattan;
Implementar o algoritmo IDA*, que reduz o uso de memória;
Paralelizar os algoritmos em ambientes multicore;
Criar visualizações gráficas interativas da busca.