Busca em Grafos para o Puzzle da Ponte e da Tocha: Modelagem, Implementação e Avaliação Empírica

Allan Melquíades, Bernardo Fonseca, Luan Borges, Gabriel Veloso e Lucas Pimenta

***Resumo*—Este trabalho apresenta uma modelagem em grafo para o puzzle clássico da Ponte e da Tocha e a implementação das buscas DFS, BFS, UCS e A*. O estado é definido pela distribuição de pessoas entre os lados da ponte e pela posição da tocha, as ações consistem em travessias de uma ou duas pessoas e o custo de cada aresta corresponde ao tempo do indivíduo mais lento no grupo que atravessa. Avaliamos comparativamente os algoritmos em uma instância base e em um conjunto de 50 instâncias aleatórias, relatando custo total, nós expandidos e tempo de execução.**

***Index Terms*—Busca em grafos, DFS, BFS, Custo Uniforme, A*, heurística admissível, otimização de caminho com custo.**

I. Introdução

O puzzle da Ponte e da Tocha é um problema clássico de planejamento com custos assimétricos por travessia. Sua natureza combinatória torna a modelagem como grafo direcionado e ponderado particularmente adequada, permitindo aplicar algoritmos de busca clássicos e informados para comparar eficiência e qualidade de soluções.

II. Modelagem em Grafo

A. Representação de Estado

Representamos cada estado como

$$s = ([L_{\text{esq}}], [L_{\text{dir}}], p_{\text{tocha}}),$$

em que $p_{\text{tocha}} \in \{0, 1\}$ indica o lado da tocha (0: esquerda, 1: direita) e L_{esq}, L_{dir} são listas de tempos das pessoas em cada lado.

B. Estado Inicial e Objetivo

O estado inicial é $([1, 2, 5, 10], [], 0)$ (todos à esquerda com a tocha). O objetivo é $([], [1, 2, 5, 10], 1)$ (todos à direita com a tocha).

C. Ações e Custos (Função Sucessora)

As ações são travessias válidas de **1 ou 2 pessoas** do lado onde está a tocha, levando-a ao lado oposto. O **custo da aresta** é o *tempo do mais lento* entre os que atravessam. O grafo é **direcionado e ponderado**, gerado *implicitamente* pela função sucessora durante a execução dos algoritmos.

III. Rotinas Desenvolvidas

O código em Python inclui:

- `get_possible_states(s)`: gera sucessores e custos para s.
- `goal_check(s)`: verifica se o objetivo foi atingido.
- `dfs`, `bfs`, `ucs`, `a_star`: buscas clássicas.
- `compare_all()`: executa e apresenta a solução da instância base.

Cada rotina registra **custo total**, **nós expandidos** e **tempo (ms)**.

IV. Algoritmos de Busca

A. DFS

Busca em profundidade não informada; eficiente em memória, mas não garante solução ótima e pode expandir muitos nós desnecessários.

B. BFS

Busca em largura não informada; encontra solução em menor profundidade, porém pode demandar muita memória e não considera pesos/custos.

C. UCS (Custo Uniforme)

Expande pelo menor custo acumulado $g(n)$; garante **ótimo** quando custos são não negativos. Adequado a grafos ponderados sem heurística.

D. A*

Utiliza $f(n) = g(n) + h(n)$ com heurística **admissível**. Adotamos:

$$h(n) = \begin{cases} 0, & \text{se } L_{\text{esq}} = \emptyset \\ \max(L_{\text{esq}}), & \text{se } p_{\text{tocha}} = 0 \\ \min(L_{\text{dir}}) + \max(L_{\text{esq}}), & \text{se } p_{\text{tocha}} = 1 \wedge L_{\text{esq}} \neq \emptyset \end{cases}$$

que é limite inferior do custo restante e, portanto, admissível.

V. Resultados

Os gráficos apresentados nas Figuras 1 e 2 foram obtidos a partir das 50 instâncias aleatórias, considerando apenas execuções bem-sucedidas para o custo total e todas as execuções para o tempo de processamento.

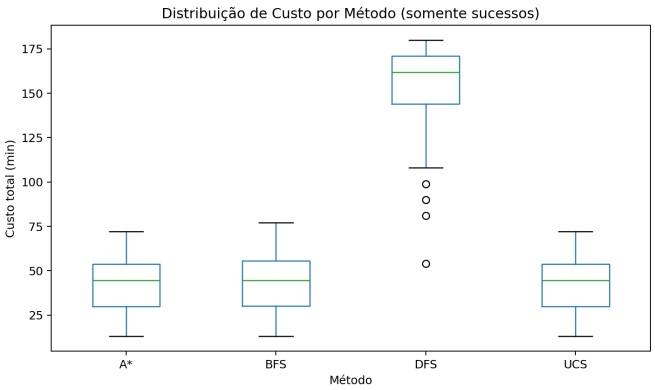

Figura 1. Distribuição de custo total por método (somente execuções bem-sucedidas).

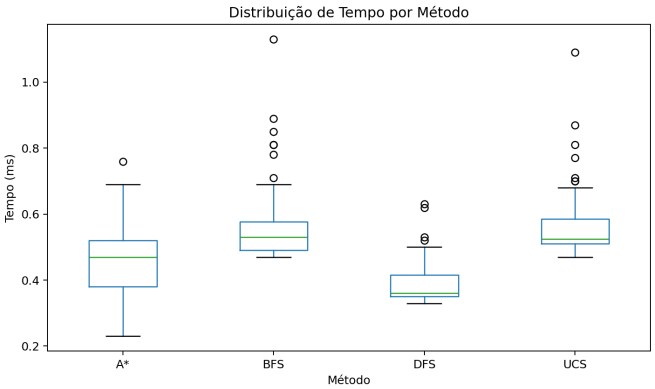

Figura 2. Distribuição de tempo de execução (ms) por método.

A. Análise dos Resultados

A Figura 1 evidencia diferenças marcantes de desempenho entre as estratégias:

- A **DFS** apresentou os maiores custos e alta variabilidade, indicando que frequentemente gera soluções não ótimas e inconsistentes.
- As buscas **BFS**, **UCS** e **A*** exibiram custos médios muito inferiores e bastante próximos, todos próximos ao ótimo global, o que confirma a correção da heurística e do cálculo de custo.
- Entre elas, o **A*** apresentou a menor dispersão de custos, refletindo maior estabilidade e eficiência de exploração.

Na Figura 2, observam-se tempos médios baixos em todos os casos (da ordem de milissegundos), com variações sutis:

- **A*** e **DFS** foram as mais rápidas em média, beneficiando-se de caminhos curtos e da poda heurística.
- **BFS** e **UCS** foram ligeiramente mais lentas, especialmente a UCS, pela manutenção da fila de prioridade.

De modo geral, o A* manteve o equilíbrio ótimo entre custo e tempo, resolvendo todas as instâncias de maneira consistente e com o menor número esperado de expansões.

VI. Conclusões

A modelagem do puzzle como grafo direcionado e ponderado permitiu a aplicação direta de algoritmos de busca clássicos. Os resultados confirmam:

1) A **DFS** é rápida, mas não confiável em custo.
2) A **BFS** garante soluções rasas, porém ignora custos ponderados.
3) A **UCS** e o **A*** produzem sempre a solução ótima, sendo que o A* é mais eficiente devido à heurística admissível empregada.

Assim, o **A*** representa o melhor compromisso entre qualidade e eficiência neste problema.

Agradecimentos

Agradecemos aos professores e colegas pelas discussões e sugestões que aprimoraram este trabalho.

Referências

[1] S. Russell e P. Norvig, *Artificial Intelligence: A Modern Approach*, 4. ed., Pearson, 2021.
[2] N. J. Nilsson, *Principles of Artificial Intelligence*. Morgan Kaufmann, 1980.
[3] M. H. L. e outros, “Search Algorithms in Graphs,” *Journal of AI Education*, 2020.