Implementação de Algoritmos de Busca Cega e Heurística para Resolver Problemas de Caminhos Mínimos em Grafos

Renan Nagano

[1]Universidade Interdimensional Tuiuti do Paraná
Curitiba – PR

renan.nagano@utp.edu.br

Resumo. *Este trabalho visa abordar a solução de problemas de caminhos mínimos em grafos por meio da implementação e comparação de algoritmos de busca cega e heurística, bem como demonstrar a eficiência - em relação ao tempo de execução e uso de memória - e a melhor vantagem de cada algoritmo. Os resultados obtidos demonstram que o algoritmo ideal depende do modelo de problema e dos critérios da aplicação.*

1. Introdução

O problema de caminhos mínimos em grafos é um dos problemas clássicos na teoria dos grafos e ciência da computação, que consiste em encontrar o caminho de menor custo entre dois vértices em um grafo ponderado, onde os pesos representam custos associados a atravessar cada aresta. O objetivo é determinar a rota mais eficiente, seja em termos de distância, tempo ou qualquer outro critério quantificável representado pelos pesos das arestas.

A busca cega, como o próprio nome sugere, explora o espaço de busca sem utilizar informações adicionais sobre o problema, o que pode incluir métodos como a busca em largura e a busca em profundidade. Por outro lado, os algoritmos heurísticos utilizam conhecimentos prévios para otimizar a busca.

Será feita a implementação computacional de cada um desses algoritmos, sendo eles: Busca em Largura (BFS), Busca em Profundidade (DFS), Busca Gulosa e A*. Para a avaliação prática, o trabalho selecionará dez grafos diferentes, cada qual com as suas caracteríticas, como: tamanho, variedade de pesos e caminhos.

Todos os algoritmos serão testados a fim de avaliar a sua eficácia e eficiência em termos de tempo de execução e espaço de memória utilizado.

2. Descrição da implementação e testes

A implementação adotou uma estrutura orientada a objetos em Python, onde grafos são representados através da classe Grafo, utilizando dicionários de adjacência para armazenamento eficiente das conexões. A carga de dados é realizada a partir de arquivos CSV formatados como triplas (origem, destino, peso), permitindo fácil adaptação a diferentes conjuntos de dados. Para cada algoritmo, foi desenvolvida uma versão padronizada que registra simultaneamente: o caminho encontrado, o tempo de execução e o pico de utilização de memória.

O sistema de monitoramento emprega o módulo ”tracemalloc”para rastreamento preciso das alocações de memória, ideal para o estudo de caso. As medições são realizadas periodicamente (a cada 100 nós processados) para capturar a evolução do consumo durante a execução, enquanto uma medição inicial estabelece a linha de base. Esse esquema de amostragem balanceia a sobrecarga computacional com a necessidade de dados representativos.

Os dados de desempenho (tempo, memória, comprimento do caminho) são armazenados em um dicionário. Por fim, dois gráficos são gerados, tempo de execução (Em segundos) e memória ocupada (Em MB), através da biblioteca ”matplotlib”e exportado como JPEG.

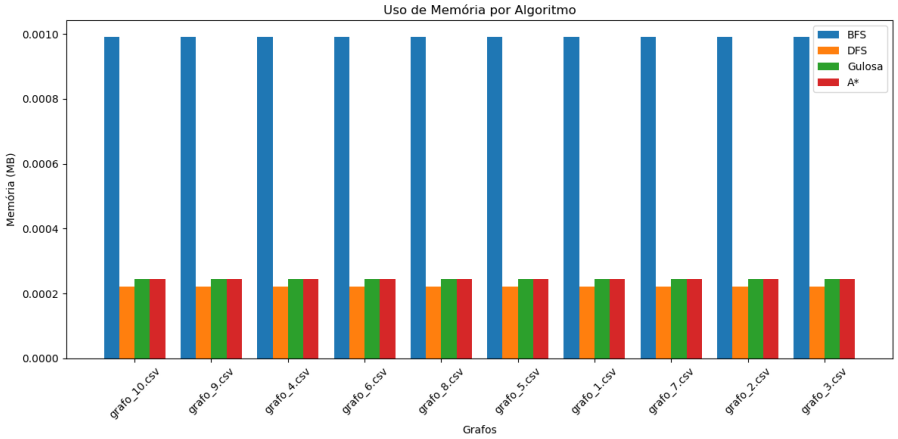

Figura 1. Gráfico comparativo de memória.

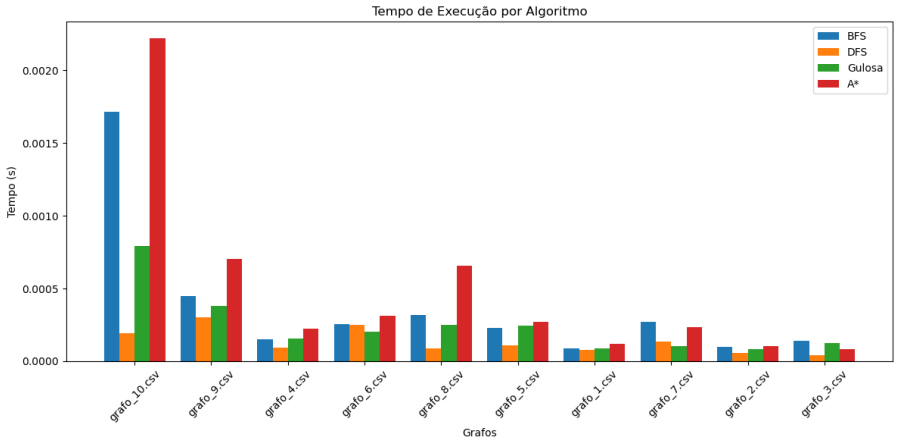

Figura 2. Gráfico comparativo de tempo.

3. Eficiência

3.1. Tempo de Execução por Algoritmo

Algoritmo A* teve, em geral, o maior tempo de execução, especialmente no maior grafo (grafo_10), indicando seu custo computacional elevado devido à combinação de custo real e heurístico.

BFS (Busca em Largura) também apresentou tempos altos em grafos maiores (como grafo 10), pois explora muitos nós, mesmo que não sejam promissores.

DFS (Busca em Profundidade) teve os menores tempos de execução na maioria dos casos, o que é esperado pela sua natureza de exploração rápida até o fundo da árvore de busca, mesmo que não seja a mais eficiente em encontrar o melhor caminho.

Busca Gulosa apresentou desempenho intermediário, com bons resultados em alguns grafos, mas também sendo superada pela DFS em rapidez.

O desempenho dos algoritmos parece estar diretamente ligado à estrutura do grafo, sendo mais impactante em grafos maiores ou mais ramificados, como grafo 10 e grafo 9

3.1.1. Uso de Memória por Algoritmo

BFS é consistentemente o algoritmo com maior uso de memória, o que é compatível com seu funcionamento: ele armazena muitos nós simultaneamente na fila.

DFS, Gulosa e A* têm consumo de memória muito similar entre si e significativamente menor que o BFS.

Isso indica que a escolha do algoritmo pode impactar drasticamente o consumo de memória, especialmente em aplicações com restrições de recursos.

3.1.2. Tabela Comparativa de Algoritmos

Critério	Melhor Algoritmo	Justificativa
Velocidade	DFS	Menor tempo na maioria dos grafos
Uso de Memória	DFS / Gulosa / A*	Menor uso consistente
Custo-benefício	Gulosa	Equilíbrio entre tempo e memória
Caminho Ótimo	A*	Algoritmo que retorna a solução ideal

Figura 3. Tabela Comparativa.

4. Conclusão

A análise comparativa entre os algoritmos de busca cega (BFS e DFS) e heurística (Busca Gulosa e A*) evidencia diferenças significativas em termos de desempenho computacional. Com base nos experimentos realizados em diversos grafos, é possível traçar conclusões relevantes quanto ao tempo de execução e uso de memória de cada técnica.

Em relação ao tempo de execução, o algoritmo DFS apresentou, de forma geral, os menores tempos, sendo o mais eficiente em cenários onde a profundidade da solução é baixa e o fator de ramificação é moderado. A Busca Gulosa também obteve bons desempenhos, mantendo uma boa relação entre tempo e precisão, especialmente em grafos maiores. Por outro lado, o algoritmo A*, apesar de fornecer soluções ótimas, foi consistentemente o mais lento, devido ao custo adicional de cálculo e manutenção das

heurísticas. O BFS teve desempenho intermediário em tempo, mas sofreu com maior lentidão em grafos mais densos ou profundos.

Quanto ao uso de memória, observou-se um padrão claro: o BFS apresentou o maior consumo em todos os cenários analisados, resultado esperado devido à necessidade de armazenar muitos nós simultaneamente em memória. Em contraste, DFS, Gulosa e A* mostraram consumos significativamente menores e similares entre si, tornando-os mais viáveis em ambientes com restrições de memória.

Com base nesses resultados, recomenda-se o uso do DFS quando o objetivo é a eficiência em tempo e não há necessidade de garantir o caminho ótimo. Para aplicações que exigem precisão e garantem a existência de uma heurística admissível, o A* continua sendo a melhor escolha, mesmo com o custo computacional mais elevado. A Busca Gulosa representa uma alternativa equilibrada, com boa performance geral e menor custo de implementação heurística em comparação ao A*.

Portanto, a escolha do algoritmo ideal deve considerar o contexto específico da aplicação, priorizando eficiência, qualidade da solução e restrições de recursos computacionais.

Referências

[1] Cormen, T. H. et al. Introduction to Algorithms. MIT Press, 2009. Acesso em: 30/03/2025.

[2] Python Software Foundation. Tracemalloc — Trace memory allocations. Python Documentation, 2023. Disponível em: https://docs.python.org/3/library/tracemalloc.html. Acesso em: 16/04/2025

[3] Russell, S.; Norvig, P. Artificial Intelligence: A Modern Approach. Pearson, 2020. Acesso em 30/03/2025.