Universidade Federal Do Rio Grande do Sul
Instituto de Informática

Lucas Dinesh Weber Miranda

Trabalho 2 (Fluxo s-t máximo)

Porto Alegre, 23 de abril de 2025

1. Introdução

Este relatório apresenta uma análise experimental de diferentes variantes do algoritmo de Ford-Fulkerson, aplicadas ao problema de fluxo máximo em grafos. Foram implementadas e testadas três abordagens para encontrar caminhos aumentantes: a busca em largura (BFS), a busca em profundidade com aleatoriedade (DFS Randomizado) e o caminho com maior gargalo (Fattest Path).

O objetivo foi entender, na prática, como cada uma dessas estratégias se comporta em termos de desempenho, número de iterações, tempo de execução e outras métricas relevantes. Além disso, buscamos comparar os resultados obtidos com as expectativas teóricas sobre a eficiência de cada algoritmo.

---

2. Implementação

Toda a implementação do código foi realizada em C++, diferente do trabalho anterior qual foi implementado em python, após algumas adversidades resolvi migrar para uma linguagem mais eficiente.

---

3. Ambiente de teste

Os testes foram realizados em uma maquina Dell G15, 8GB de memoria RAM, 1TB de armazenamento. Dual boot, e o SO foi Linux Ubuntu 22.04 LTS. A Ide foi vscode.

---

4. Metodologia

Para avaliar o desempenho dos algoritmos, utilizamos um conjunto de grafos gerados com estruturas diversas — malhas, linhas e grafos aleatórios. Para cada combinação de instância e algoritmo, foram registradas as seguintes métricas:

- Fluxo máximo encontrado
- Tempo total de execução (ms)
- Número de iterações
- Número de operações (arestas e vértices tocados)
- Comprimento médio dos caminhos aumentantes
- Tempo médio por iteração (ms)
- Uso máximo de memória (KB)

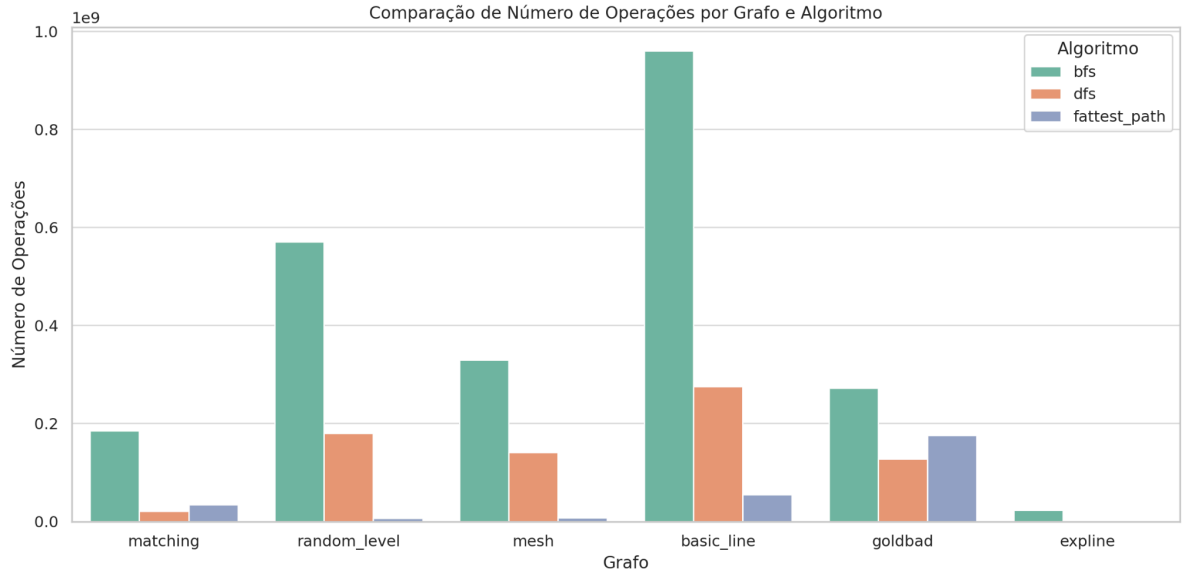
Os grafos tinham entre 3.000 e 6.400 vértices, permitindo testar os algoritmos em cenários de porte médio.

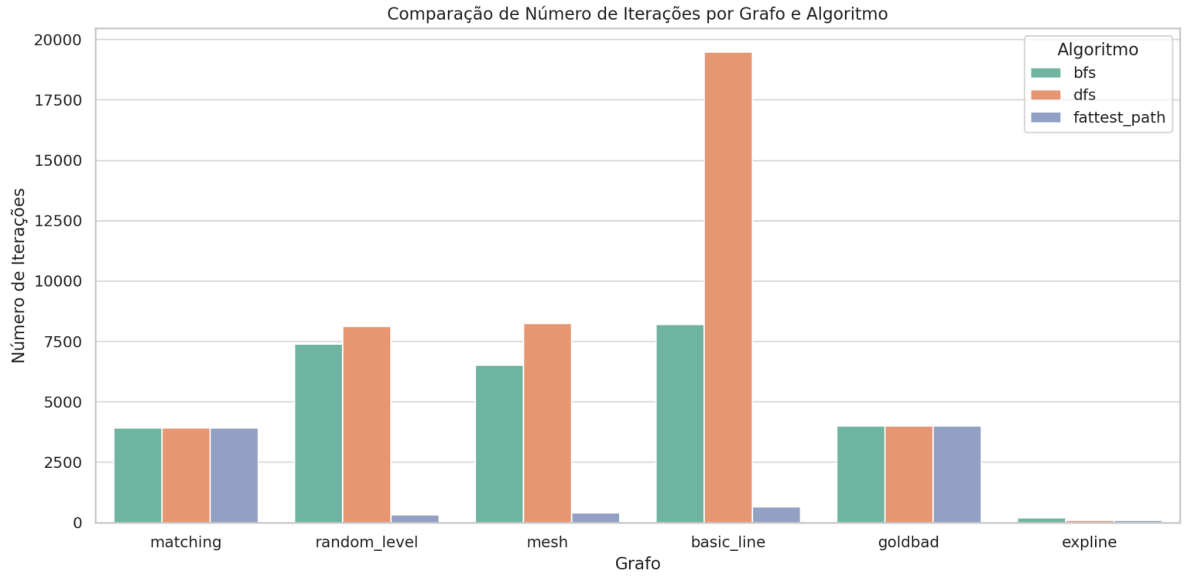
---

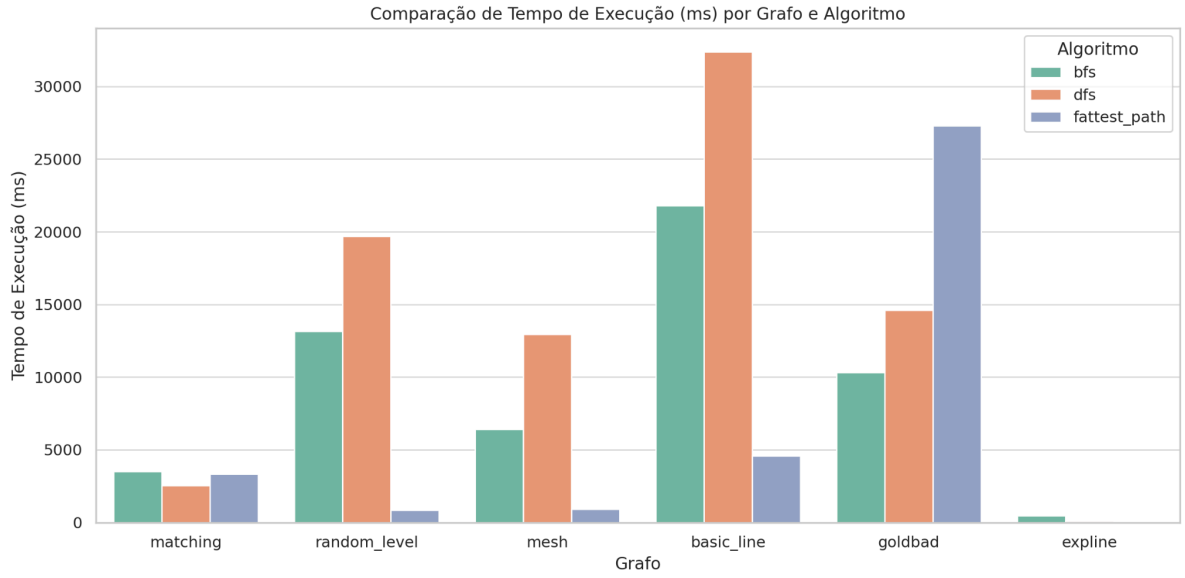
5. Características das Instâncias Utilizadas

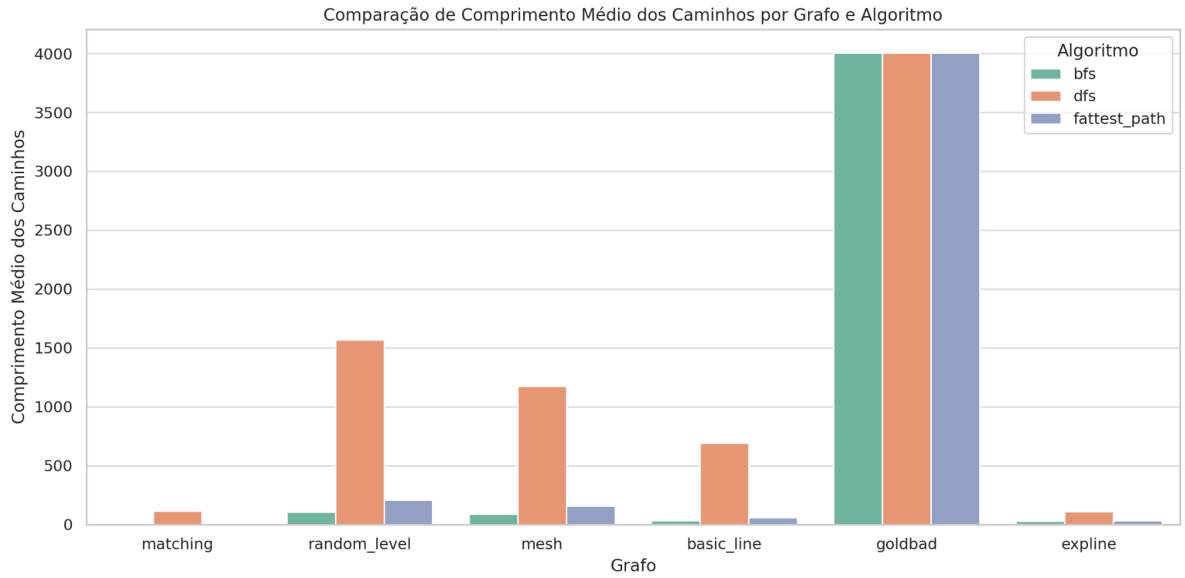
ID	Nome	Parâmetros	Nº de Vértices	Nº de Arestas	Justificativa
1	Mesh	r = 60, c = 60, C = 100	r × c + 2 = 3602	2r + 3(c − 1)r = 97720	Estrutura de malha regular para análise de algoritmos em topologias previsíveis.
2	Random Level	r = 80, c = 80, C = 100	r × c + 2 = 6402	2r + 3(c − 1)r = 206322	Gera grafos mais imprevisíveis com conectividade média.
4	Matching	n = 3000, d = 3, C = 100	2n + 2 = 6002	n(d + 2) = 15015	Representa muitos caminhos paralelos e balanceados.
6	Basic Line	n = 80, m = 80, d = 3, C = 100	nm + 2 = 6402	nmd + 2m = 193600	Modelo com múltiplos caminhos organizados em linhas paralelas.
7	Exp Line	n = 80, m = 80, d = 3, C = 100	nm + 2 = 6402	nmd + 2m = 193600	Variante com ramificações mais profundas, ideal para stressar BFS e DFS.
10	GoldBad	n = 3000	3n + 3 = 9003	4n + 1 = 12001	Instância projetada para o pior caso de desempenho em DFS.

6. Análise Experimental




Comparação de Número de Iterações por Grafo e Algoritmo
Número de Iterações
Grafo
Algoritmo
bfs
dfs
fattest_path
matching
random_level
mesh
basic_line
goldbad
expline


Comparação de Tempo de Execução (ms) por Grafo e Algoritmo
Tempo de Execução (ms)
Grafo
Algoritmo
bfs
dfs
fattest_path
matching
random_level
mesh
basic_line
goldbad
expline


Comparação de Comprimento Médio dos Caminhos por Grafo e Algoritmo
Comprimento Médio dos Caminhos
Grafo
Algoritmo
bfs
dfs
fattest_path
matching
random_level
mesh
basic_line
goldbad
expline

6.1 Desempenho Geral

De modo geral, os resultados ficaram dentro do esperado. O Fattest Path foi consistentemente o mais eficiente, resolvendo até os grafos mais complexos com poucas iterações e tempo reduzido. Isso se confirmou, por exemplo, na instância ExpLine, onde ele precisou de apenas 11 iterações.

O DFS Randomizado, como previsto, teve desempenho mais instável. Em grafos como BasicLine e GoldBad, fez um número enorme de iterações e operações, resultando em tempos de execução bastante altos. Isso é resultado da escolha aleatória dos caminhos, que frequentemente não são os mais curtos nem os mais vantajosos.

Já o BFS ficou num meio-termo. Apesar de fazer mais iterações que o Fattest Path, seu tempo por iteração foi estável, e o desempenho geral foi bastante sólido, especialmente em grafos mais estruturados, como o Matching.

6.2 Custo por Iteração

Analisando o tempo médio por iteração, vemos que o Fattest Path também leva vantagem aqui. Na instância Mesh, por exemplo, seu custo por iteração foi de apenas 0,983 ms. Em contraste, o DFS foi o mais custoso, chegando a quase 700 ms por iteração em BasicLine. O BFS manteve-se com custo moderado (por volta de 0,897 ms), equilibrando eficiência e simplicidade.

6.3 Relação com a Complexidade Pessimista

Os testes também serviram para comparar os resultados práticos com a complexidade pessimista prevista na teoria. O Fattest Path se comportou de forma próxima ao ideal, com baixa deficiência por iteração. Já o DFS, com caminhos mais longos e instabilidade, demonstrou deficiência elevada, especialmente em casos como GoldBad, onde foram mais de 14.000 iterações. O BFS apresentou desempenho estável e dentro do esperado, mesmo com mais iterações.

Em contrapartida, o DFS apresentou alta deficiência. Na instância GoldBad, foram necessárias mais de 14.000 iterações, valor significativamente maior do que o previsto mesmo nos piores casos. Isso evidencia que o DFS randomizado é fortemente impactado por escolhas ineficientes, agravando sua performance em topologias desfavoráveis.

O BFS, embora tenha realizado mais iterações que o Fattest Path, manteve uma relação aceitável com a complexidade teórica. Na instância Matching, por exemplo, executou 112 iterações com um tempo médio de 0,897 ms, o que está dentro do esperado para grafos com caminhos curtos e paralelos.

6.4 Validação de Resultados

Para garantir a correção dos algoritmos, o valor do fluxo máximo obtido foi comparado entre todas as estratégias aplicadas à mesma instância. Em todos os casos, os algoritmos chegaram ao mesmo valor de fluxo, indicando que, apesar das diferenças de desempenho, a implementação manteve a validade dos resultados.

7. Conclusão

Os resultados reforçam o que a teoria já apontava: o desempenho de cada variante do Ford-Fulkerson depende muito da estrutura do grafo. O Fattest Path foi o mais eficiente na maior parte das situações, lidando muito bem com grafos densos e ramificados. O DFS mostrou-se ineficiente em quase todos os cenários testados, e o BFS se manteve como uma opção intermediária segura e razoavelmente eficaz.

No geral, a experimentação confirmou as expectativas teóricas e ajudou a entender melhor como o comportamento prático dos algoritmos se manifesta em diferentes estruturas de grafo. Além disso, a análise da deficiência e do custo por iteração mostrou que, embora o Fattest Path mantenha-se eficiente mesmo em grafos desafiadores, algoritmos como o DFS podem apresentar um desempenho muito abaixo do esperado caso a estrutura do grafo não favoreça a sua estratégia de exploração.