Análise Comparativa de Estratégias de Busca no Algoritmo Ford-Fulkerson

Maximus Borges da Rosa
Universidade Federal do Rio Grande do Sul

5 de maio de 2025

Resumo

Este trabalho apresenta uma análise comparativa de diferentes estratégias de busca (DFS, BFS e Fattest Path) para o algoritmo Ford-Fulkerson de fluxo máximo em redes. Experimentos foram realizados com diversos tipos de grafos (basic_line, matching, random_level e square_mesh), coletando métricas de desempenho como tempo de execução, porcentagem de vértices e arestas visitados, e operações em estruturas de dados. Os resultados mostram as diferenças significativas entre os algoritmos em termos de eficiência e número de iterações, oferecendo insights para a escolha da melhor estratégia dependendo das características da rede.

1 Introdução

O problema de fluxo máximo é um dos problemas fundamentais na teoria dos grafos e tem aplicações em diversos campos como redes de transporte, telecomunicações, logística e sistemas distribuídos. Essencialmente, o problema consiste em determinar a quantidade máxima de fluxo que pode ser enviada de um vértice fonte para um vértice sumidouro em uma rede com capacidades limitadas nas arestas.

O algoritmo Ford-Fulkerson, proposto em 1956 por L. R. Ford Jr. e D. R. Fulkerson, é um método clássico para resolver este problema. O algoritmo funciona incrementalmente, encontrando caminhos aumentantes da fonte ao sumidouro através da rede residual e aumentando o fluxo ao longo desses caminhos até que nenhum caminho aumentante adicional possa ser encontrado.

A eficácia do algoritmo Ford-Fulkerson depende fortemente da estratégia utilizada para encontrar os caminhos aumentantes. Neste trabalho, investigamos três estratégias diferentes:

- **Busca em Profundidade (DFS)**: Explora o grafo seguindo um caminho até sua máxima profundidade antes de retroceder.
- **Busca em Largura (BFS)**: Explora o grafo em níveis, visitando todos os vizinhos de um vértice antes de avançar para o próximo nível (método de Edmonds-Karp).
- **Fattest Path**: Prioriza caminhos com maior capacidade residual, utilizando uma fila de prioridade.

O objetivo deste trabalho é analisar experimentalmente como estas diferentes estratégias impactam o desempenho do algoritmo em diversos tipos de grafos, fornecendo insights para a escolha da estratégia mais adequada dependendo das características da rede.

2 Metodologia

2.1 Implementação dos Algoritmos

O algoritmo Ford-Fulkerson foi implementado em C++ com três diferentes estratégias de busca para encontrar caminhos aumentantes:

Algorithm 1 Ford-Fulkerson Genérico

```
Inicialize fluxo f(e) = 0 para todas as arestas e
while existe caminho aumentante p da fonte s ao sumidouro t na rede residual do
    Encontre a capacidade residual mínima c_min em p
    for cada aresta e em p do
        Aumente o fluxo em e por c_min
        Atualize a rede residual
    end for
end while
return Fluxo total
```

Métodos de Busca Implementados:

- **DFS**: Implementado com uma pilha explícita e randomização na ordem de exploração dos vizinhos para evitar casos patológicos.
- **BFS**: Implementado com uma fila e garantindo que os caminhos encontrados tenham o menor número de arestas.
- **Fattest Path**: Implementado utilizando uma fila de prioridade (heap) que prioriza as arestas com maior capacidade residual.

2.2 Tipos de Grafos Analisados

Para avaliar o desempenho dos algoritmos em diferentes cenários, foram utilizados quatro tipos de grafos:

- **Basic Line**: Grafos com estrutura linear onde cada vértice está conectado aos vértices adjacentes.
- **Matching**: Grafos bipartidos que modelam problemas de emparelhamento.
- **Random Level**: Grafos com níveis aleatórios de conectividade entre camadas.
- **Square Mesh**: Grafos em formato de malha quadrada com conexões regulares.

2.3 Geração dos Parâmetros dos Grafos

Para cada tipo de grafo, foram gerados 24 grafos com parâmetros variados, seguindo as seguintes estratégias:

- **Basic Line**:
  - Número de linhas (n) e segmentos por linha (m): variados de 50 a 110, incrementando 2 unidades a cada grafo gerado (50, 52, 54, ..., 110)
  - Grau de conexão entre linhas (d): fixo em 25
  - Capacidade máxima das arestas (C): fixo em 1000
- **Matching**:
  - Número de nós em cada partição (n): variado de 2500 a 7465, incrementando 165 unidades a cada grafo gerado (2500, 2665, 2830, ..., 7465)
  - Grau (d): fixo em 25
  - Capacidade máxima (C): fixo em 1000
- **Random Level**:
  - Número de linhas (r) e colunas (c): variados de 50 a 110, incrementando 2 unidades a cada grafo gerado (50, 52, 54, ..., 110)
  - Capacidade máxima (C): fixo em 1000
- **Square Mesh**:
  - Tamanho do lado da malha quadrada (s): variado de 50 a 110, incrementando 2 unidades a cada grafo gerado (50, 52, 54, ..., 110)
  - Grau máximo (d): fixo em 25
  - Capacidade máxima (C): fixo em 1000

Para todos os tipos de grafos, os parâmetros foram escolhidos para criar uma progressão gradual no tamanho e complexidade dos grafos, permitindo avaliar o desempenho dos algoritmos em diferentes escalas. Os valores fixos para grau e capacidade máxima foram mantidos constantes para isolar o efeito do tamanho do grafo nos resultados. Cada grafo foi gerado usando um programa específico que recebe os parâmetros e produz o grafo no formato DIMACS.

2.4 Métricas Analisadas

Durante a execução dos algoritmos, foram coletadas as seguintes métricas:

- n: Número de vértices do grafo
- m: Número de arestas do grafo
- **max_flow**: Valor do fluxo máximo encontrado
- r: Razão entre o número de iterações realizadas e o número esperado teoricamente
- **exec_time_ms**: Tempo de execução em milissegundos
- s_bar: Porcentagem média de vértices visitados por iteração

- t_bar: Porcentagem média de arestas visitadas por iteração

Adicionalmente, para o algoritmo Fattest Path, foram coletadas métricas relacionadas às operações na fila de prioridade:

- i_bar: Fração do número médio de inserções por n
- d_bar: Fração do número médio de remoções de mínimo por n
- u_bar: Fração do número médio de atualizações por n

3 Resultados

3.1 Tabelas Comparativas

A seguir, apresentamos as tabelas com os resultados obtidos nas execuções dos três algoritmos para os diferentes tipos de grafos. Uma consideração muito relevante é que, devido ao elevado tempo de execução do algoritmo DFS em grafos do tipo basic_line e square_mesh, foram usadas as 3 e 6 primeiras instâncias de cada tipo, respectivamente.

Tabela 1: Resultados para grafos do tipo basic_line

Método	$\bar{\mathbf{r}}$	**tempo médio**	$\bar{\mathbf{s}}$	$\bar{\mathbf{t}}$	$\bar{\mathbf{i}}$	$\bar{\mathbf{d}}$	$\bar{\mathbf{u}}$
DFS	0.872701	8403770.00	0.42	0.43	-	-	-
BFS	0.000026	159152.50	0.99	0.51	-	-	-
FAT	0.001093	51331.17	0.00	0.00	0.84	0.56	0.03

Tabela 2: Resultados para grafos do tipo matching

Método	$\bar{\mathbf{r}}$	**tempo médio**	$\bar{\mathbf{s}}$	$\bar{\mathbf{t}}$	$\bar{\mathbf{i}}$	$\bar{\mathbf{d}}$	$\bar{\mathbf{u}}$
DFS	1.000245	5874.96	0.00	0.02	-	-	-
BFS	0.000009	83620.92	0.99	0.27	-	-	-
FAT	0.003078	6282.08	0.00	0.00	0.34	0.01	0.00

Tabela 3: Resultados para grafos do tipo random_level

Método	$\bar{\mathbf{r}}$	**tempo médio**	$\bar{\mathbf{s}}$	$\bar{\mathbf{t}}$	$\bar{\mathbf{i}}$	$\bar{\mathbf{d}}$	$\bar{\mathbf{u}}$
DFS	0.997759	386627.88	0.48	0.51	-	-	-
BFS	0.000154	36319.12	0.94	0.60	-	-	-
FAT	0.001238	2869.46	0.00	0.00	0.83	0.74	0.12

Tabela 4: Resultados para grafos do tipo square_mesh

Método	$\bar{\mathbf{r}}$	**tempo médio**	$\bar{\mathbf{s}}$	$\bar{\mathbf{t}}$	$\bar{\mathbf{i}}$	$\bar{\mathbf{d}}$	$\bar{\mathbf{u}}$
DFS	0.990060	3643628.33	0.49	0.50	-	-	-
BFS	0.000068	459886.33	1.00	0.52	-	-	-
FAT	0.000154	7199.79	0.00	0.00	0.98	0.70	0.06

3.2 Análise Gráfica

Para melhor visualização dos resultados, apresentamos os seguintes gráficos comparativos:

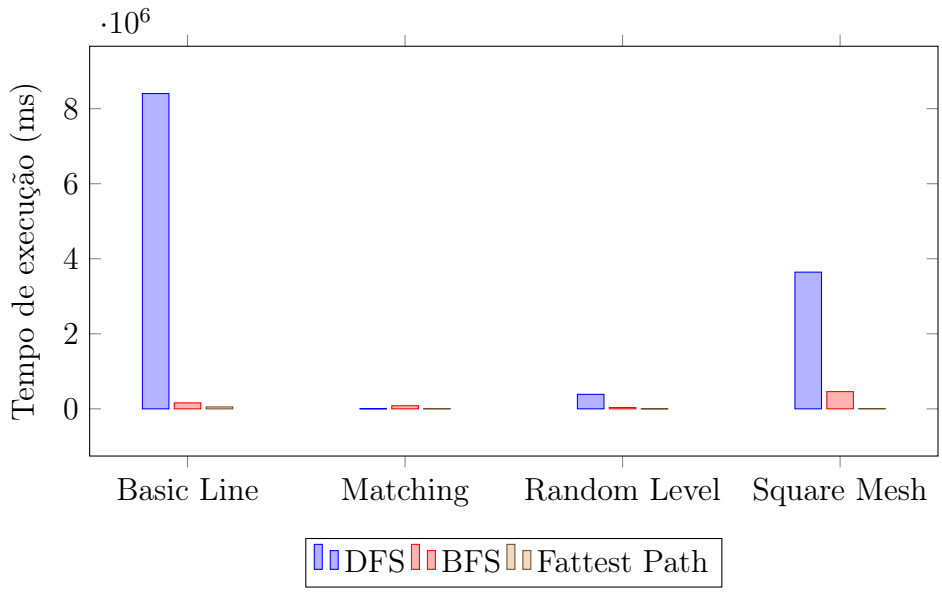

Figura 1: Comparação dos tempos de execução por tipo de grafo

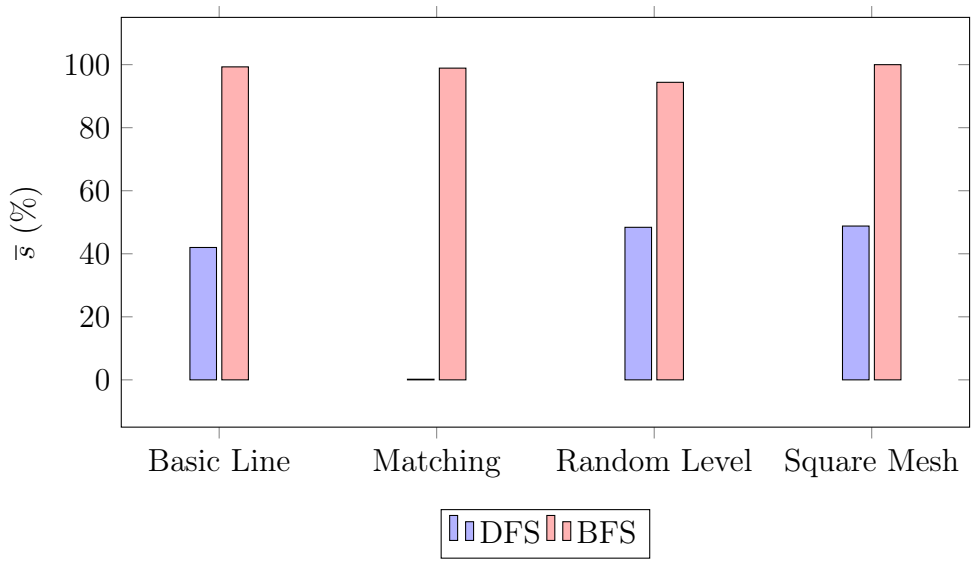

Figura 2: Comparação da porcentagem média de vértices visitados (DFS vs BFS) por tipo de grafo.

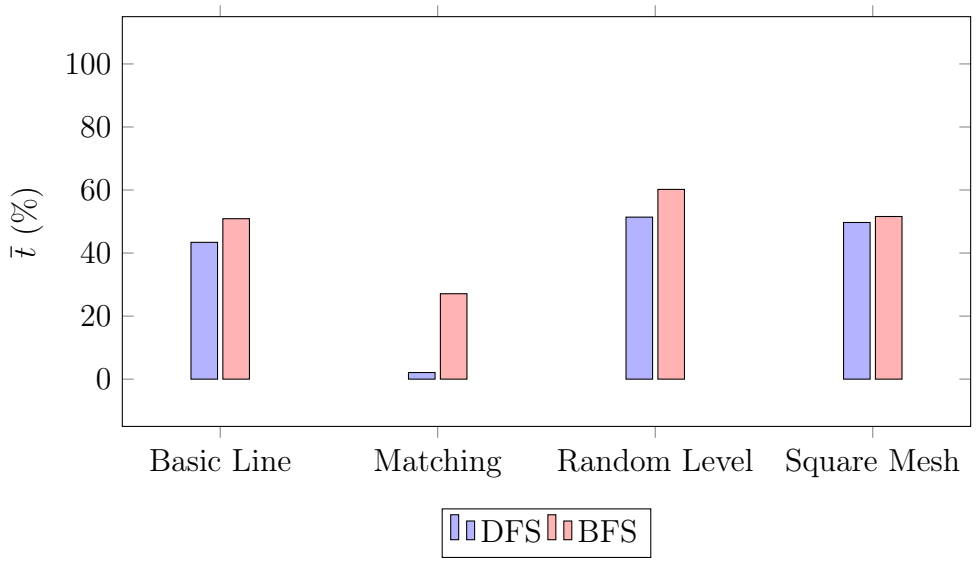

Figura 3: Comparação da porcentagem média de arestas visitadas (DFS vs BFS) por tipo de grafo.

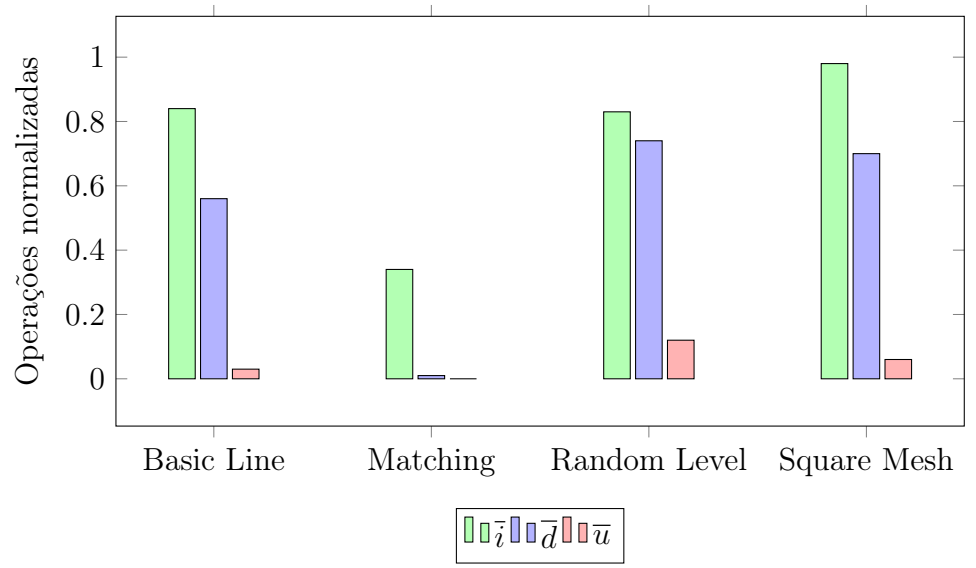

Figura 4: Comparação da contagem normalizada de operações no heap (Fattest Path).

4 Discussão

A análise dos resultados obtidos nos permite observar diferentes comportamentos dos algoritmos de busca no contexto do Ford-Fulkerson:

4.1 Desempenho por Tipo de Grafo

Basic Line: Neste tipo de grafo com estrutura linear, observamos que o DFS apresentou o pior desempenho com tempo médio de execução extremamente elevado (8.403.770 ms), enquanto o Fattest Path foi significativamente mais eficiente (51.331 ms). O BFS, embora melhor que o DFS, ainda foi consideravelmente mais lento que o Fattest Path. A visitação

de vértices foi de 42% para DFS e 99% para BFS, indicando que o BFS explora uma área muito maior do grafo em cada iteração.

Matching: Para grafos de emparelhamento, todos os algoritmos apresentaram desempenho relativamente bom, com o DFS (5.875 ms) e o Fattest Path (6.282 ms) mostrando tempos similares. O BFS foi significativamente mais lento (83.621 ms). Destaca-se o baixo percentual de vértices visitados pelo DFS (0,2%) em contraste com o BFS (98,9%), demonstrando a eficiência do DFS em encontrar caminhos de aumento sem explorar grandes porções do grafo neste tipo específico de estrutura.

Random Level: Nos grafos com níveis aleatórios de conectividade, o Fattest Path demonstrou superioridade clara com tempo médio de apenas 2.869 ms, seguido pelo BFS (36.319 ms) e com o DFS apresentando o pior desempenho (386.628 ms). A estratégia de busca pelo caminho mais "gordo"mostrou-se particularmente eficaz nesta topologia de grafo, possivelmente devido à variedade de caminhos disponíveis com diferentes capacidades.

Square Mesh: Para grafos em malha quadrada, novamente o Fattest Path apresentou desempenho excepcionalmente superior (7.200 ms) em comparação com BFS (459.886 ms) e DFS (3.643.628 ms). O BFS visita 100% dos vértices em cada iteração, enquanto o DFS visita aproximadamente 49%, mas leva muito mais tempo devido ao alto número de iterações necessárias para encontrar o fluxo máximo.

4.2 Comparação entre os Algoritmos

DFS vs. BFS: Comparando estas duas estratégias básicas, observamos que o BFS consistentemente supera o DFS em tempo de execução em todos os tipos de grafos testados, com diferença particularmente pronunciada nos grafos do tipo basic_line e square_mesh. O BFS visita significativamente mais vértices por iteração (94%–100%) em comparação ao DFS (0,2%–49%), o que resulta em menor número de iterações para encontrar o fluxo máximo. No entanto, o BFS também visita mais arestas em média, exceto no caso de matching, onde o DFS é ligeiramente mais eficiente tanto em tempo quanto em razão de complexidade.

Fattest Path: O algoritmo de caminho mais gordo apresentou desempenho superior em praticamente todos os cenários, com tempos de execução significativamente menores que as outras duas abordagens. Sua eficiência é particularmente notável em grafos complexos como random_level e square_mesh, onde supera o DFS por fatores de 135 e 506, respectivamente. As métricas de operações no heap mostram que embora o algoritmo realize um número significativo de inserções ($\bar{i}$ entre 0,34 e 0,98) e remoções ($\bar{d}$ entre 0,01 e 0,74), o número de atualizações ($\bar{u}$) é relativamente baixo (0,00 a 0,12), contribuindo para seu bom desempenho.

4.3 Análise de Complexidade

A complexidade teórica do Ford-Fulkerson depende da estratégia de busca utilizada:

- **DFS**: O pior caso é $O(|E| \cdot f_{max})$, onde $|E|$ é o número de arestas e f_{max} é o valor do fluxo máximo.
- **BFS (Edmonds-Karp)**: $O(|V| \cdot |E|^2)$, onde $|V|$ é o número de vértices.
- **Fattest Path**: Teoricamente $O(|E| \cdot |V| \cdot \log |V|)$ devido às operações de fila de prioridade.

Tabela 5: Razão média entre tempo real e complexidade teórica

Método	**Basic Line**	**Matching**	**Random Level**	**Square Mesh**
DFS	0.872701	1.000245	0.997759	0.990060
BFS	0.000026	0.000009	0.000154	0.000068
Fattest Path	0.001093	0.003078	0.001238	0.000154

Nossos resultados experimentais contradizem estas expectativas teóricas no caso dos algoritmo BFS e Fattest Path.

5 Ambiente de Teste

Os testes foram executados em um processador **Intel Core i5-10210U** de 1.60 GHz com 12 GB de RAM, em um sistema **Windows 11** utilizando **WSL**. O código foi desenvolvido em **C++** utilizando a IDE **Visual Studio Code**.

6 Conclusão

Este trabalho apresentou uma análise comparativa detalhada de três estratégias de busca para o algoritmo Ford-Fulkerson: DFS, BFS e Fattest Path. Os experimentos foram realizados em quatro tipos diferentes de grafos, permitindo uma avaliação ampla do comportamento destes algoritmos em diversos cenários.

Os principais resultados obtidos indicam que:

- O algoritmo Fattest Path demonstrou superioridade consistente em termos de tempo de execução em praticamente todos os cenários testados, chegando a ser centenas de vezes mais rápido que o DFS em grafos complexos.
- A estratégia BFS apresenta um meio-termo entre DFS e Fattest Path, com tempos de execução intermediários, mas visitando uma porcentagem significativamente maior de vértices por iteração.
- O DFS, apesar de sua simplicidade, apresenta desempenho severamente comprometido em grafos do tipo basic_line e square_mesh, tornando-o inviável para aplicações práticas com esses tipos de estruturas.

Este estudo demonstra a importância da escolha adequada do algoritmo de busca para problemas de fluxo máximo, evidenciando que o Fattest Path, apesar de sua maior complexidade de implementação devido ao uso de estruturas de dados adicionais (fila de prioridade), oferece benefícios significativos de desempenho que justificam sua adoção em aplicações práticas onde o tempo de processamento é crítico.

Referências

[1] Ford, L. R. and Fulkerson, D. R. (1956). Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404.

[2] Edmonds, J. and Karp, R. M. (1972). Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264.

[3] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms*. MIT Press, 3rd edition.