

✓ Análise Comparativa de Grafos - Graph6

Este notebook realiza análise completa de grafos em formato Graph6 (.g6), calculando métricas de centralidade e conectividade, além de gerar visualizações e uma análise comparativa em DataFrame.

Estrutura do Notebook

1. Imports e configuração
2. Funções de carregamento
3. Funções de visualização
4. Funções de cálculo de métricas
5. Função principal de análise
6. Execução e resultados

✓ 1. Imports e Configuração

Importação das bibliotecas necessárias para análise de grafos, visualização e manipulação de dados.

```
import os
import networkx as nx
import matplotlib.pyplot as plt
import statistics
import pandas as pd
```

✓ 2. Função de Carregamento de Grafos

Função para ler arquivos (.g6) que podem conter um ou mais grafos (um por linha). Suporta formatos Graph6 e Sparse6.

```
def load_graphs_from_graph6_file(path):
    """
    Reads a .g6 file that can contain one or more graphs (one per line).
    Returns a list of NetworkX graphs.
    """
    graphs = []
    with open(path, "rb") as f:
        for raw in f:
            line = raw.strip()
            if not line:
                continue
            # Ignores optional ">>graph6<<" header if it appears
            if line == b">>graph6<<":
                continue
            # Graph6 typically starts without ":", Sparse6 starts with ":".
            if line.startswith(b":"):
                # Sparse6 line
                G = nx.from_sparse6_bytes(line)
            else:
                # Graph6 line
                G = nx.from_graph6_bytes(line)
            graphs.append(G)
    return graphs
```

✓ 3. Funções de Visualização

Duas funções para visualizar grafos:

- **visualize_graph**: Desenha o grafo com nós e arestas
- **plot_adjacency_matrix**: Exibe a matriz de adjacência como heatmap

```
def visualize_graph(G, title="Graph Visualization", file_name="graph.png"):
    """Visualiza o grafo e salva como imagem."""
    plt.figure(figsize=(6, 6))
    pos = nx.spring_layout(G, seed=42)

    nx.draw(
        G, pos,
        with_labels=True,
        node_color="lightblue",
```

```

        node_size=800,
        font_size=10,
        font_weight="bold",
        edge_color="gray"
    )

    plt.title(title, fontsize=14)
    plt.savefig(file_name)    # save instead of show
    plt.close()              # close the figure so nothing opens

```

```

def plot_adjacency_matrix(G: nx.Graph, title: str = "Adjacency Matrix", file_name="adjacency_matrix.png"):
    """
    Exibe a matriz de adjacência do grafo como um heatmap.
    - Mostra rótulos de nós quando o grafo é pequeno (<= 20 nós).
    - Salva a imagem em arquivo.
    """
    # Define ordem estável de nós (tenta ordenar caso comparáveis)
    nodes = list(G.nodes())
    try:
        nodes = sorted(nodes)
    except Exception:
        pass

    A = nx.to_numpy_array(G, nodelist=nodes, dtype=float)

    plt.figure(figsize=(6, 6))
    im = plt.imshow(A, cmap="Blues", interpolation="nearest")
    plt.title(title, fontsize=14)
    plt.xlabel("Nodes")
    plt.ylabel("Nodes")

    n = len(nodes)
    if n <= 20:
        plt.xticks(range(n), nodes, rotation=90)
        plt.yticks(range(n), nodes)
    else:
        plt.xticks([])
        plt.yticks([])

    plt.colorbar(im, fraction=0.046, pad=0.04)
    plt.tight_layout()
    plt.savefig(file_name)    # save instead of show
    plt.close()              # close the figure so nothing opens

```

✓ 4. Funções de Cálculo de Métricas

4.1 Cálculo de Centralidades

Calcula diversas medidas de centralidade e conectividade, retornando estatísticas resumidas (média, mínimo, máximo, desvio padrão) para métricas por-nó.

```

def calculate_centralities(G: nx.Graph, measures: dict):
    """
    Apply a set of measures (centralities/connectivities) to a graph.
    measures: dict {label: function}
    Returns: dict with summary statistics
    """
    results = {}
    for label, func in measures.items():
        try:
            if label == "Algebraic Connectivity":
                result = func(G, method="lanczos")
            elif label == "Katz Centrality":
                result = func(G, alpha=0.005, beta=1.0, max_iter=2000)
            elif label == "PageRank":
                result = func(G, alpha=0.85)
            else:
                result = func(G)
            print(f"\n>>> {label}:")

            # If result is a dict (per-node values)
            if isinstance(result, dict):
                # Calculate summary statistics
                values = list(result.values())
                average_value = sum(values) / len(values)
                min_value = min(values)
                max_value = max(values)
                std_dev = statistics.pstdev(values)

```

```

        # Store results for DataFrame
        results[label] = {
            "Average": average_value,
            "Minimum": min_value,
            "Maximum": max_value,
            "Standard Deviation": std_dev
        }
        print(f"Average_{label}: {average_value:.4f}")
        print(f"Minimum_{label}: {min_value:.4f}")
        print(f"Maximum_{label}: {max_value:.4f}")
        print(f"Standard Deviation_{label}: {std_dev:.4f}")
    else:
        # Single numeric value
        results[label] = {"Value": result}
        print(f"Value: {result}")

    except Exception as e:
        print(f"Error computing {label}: {e}")
        results[label] = {"Error": str(e)}

    return results

```

✓ 4.2 Avaliação de Conectividade Básica

Calcula métricas básicas de conectividade:

- Número de componentes conectados
- Tamanho do maior componente
- Diâmetro do grafo (do maior componente)

```

def evaluate_connectivity(G):
    """
    Evaluate basic connectivity measures of a graph.

    Returns:
    - dictionary with:
        * number of connected components
        * size of the largest connected component
        * diameter of the graph (largest component)
    """
    # Number of connected components
    num_components = nx.number_connected_components(G)

    # Largest connected component
    components = list(nx.connected_components(G))
    largest_component = max(components, key=len)
    largest_size = len(largest_component)

    # Subgraph induced by the largest component
    largest_subgraph = G.subgraph(largest_component)

    # Diameter of the largest component
    diameter = nx.diameter(largest_subgraph)

    result = {
        "Number of connected components": num_components,
        "Size of largest component": largest_size,
        "Graph diameter": diameter
    }
    print("\n>>> Connectivity Measures:")
    for key, value in result.items():
        print(f"{key}: {value}")
    return result

```

✓ 5. Função Principal de Análise

A função `main` processa todos os arquivos `.g6` de uma pasta e:

1. Carrega os grafos
2. Gera visualizações (grafo e matriz de adjacência)
3. Calcula centralidades
4. Avalia conectividade
5. **Cria um DataFrame comparativo** com todas as métricas
6. Exibe resumo final

```

def main(folder: str):
    """Função principal para processar todos os grafos e gerar análise comparativa."""
    # Dictionary to store the loaded graphs
    graphs = {}

    # Dicionários de medidas de centralidade
    dict_centralities = {
        # Standard centrality measures
        "Degree": nx.degree_centrality,
        "Closeness": nx.closeness_centrality,
        "Betweenness": nx.betweenness_centrality,
        "Eigenvector": nx.eigenvector_centrality,

        # Additional centrality measures
        "Katz Centrality": nx.katz_centrality,
        "PageRank": nx.pagerank,
        "Harmonic Centrality": nx.harmonic_centrality,
        "Current-flow Betweenness": nx.current_flow_betweenness_centrality
    }

    # Dicionários de medidas de conectividade
    dict_connectivity = {
        # Standard connectivity measures
        "Node Connectivity": nx.node_connectivity,
        "Edge Connectivity": nx.edge_connectivity,
        "Algebraic Connectivity": nx.algebraic_connectivity,

        # Additional connectivity measures
        "Average Node Connectivity": nx.average_node_connectivity,
        "Graph Density": nx.density,
        "Average Shortest Path Length": nx.average_shortest_path_length,
        "Global Clustering Coefficient": nx.transitivity,
        "Minimum Node Cut": nx.minimum_node_cut,
        "Minimum Edge Cut": nx.minimum_edge_cut
    }

    # DataFrame para armazenar resultados comparativos
    results_df = pd.DataFrame()

    # Iterates through all files in the folder
    for file in sorted(os.listdir(folder)):
        if file.endswith(".g6"):
            file_path = os.path.join(folder, file)
            print(f"\nProcessing file: {file}")
            try:
                graph_list = load_graphs_from_graph6_file(file_path)
                graphs[file] = graph_list

                for i, G in enumerate(graph_list):
                    print(f"\n{' '*80}")
                    print(f"GRAPH {i} from {file}")
                    print(f"Nodes: {G.number_of_nodes()}, Edges: {G.number_of_edges()}")
                    print(f"{' '*80}")

                    #! A) Graph visualization
                    visualize_graph(G, title=f"{file} - graph {i}",
                                    file_name=f"graph_{file}_graph_{i}.png")

                    #! A2) Adjacency matrix
                    plot_adjacency_matrix(G, title=f"Adjacency Matrix - {file} - graph {i}",
                                           file_name=f"adjacency_{file}_graph_{i}.png")

                    #! B) Centrality calculations
                    results_centralities = calculate_centralities(G, dict_centralities)

                    #! C) Connectivity evaluation
                    dict_evaluate = evaluate_connectivity(G)

                    #! C2) Additional connectivity measures
                    results_connectivity = calculate_centralities(G, dict_connectivity)

                    # Construir linha do DataFrame
                    new_row = {
                        "File": file,
                        "Graph_Index": i,
                        "Num_Nodes": G.number_of_nodes(),
                        "Num_Edges": G.number_of_edges(),
                        **{f"Centrality_{k}_{stat}": v
                           for k, stats in results_centralities.items()
                           for stat, v in stats.items()},
                        **{f"Connectivity_{k}_{stat}": v
                           for k, stats in results_connectivity.items()}
                    }

```

```
        for stat, v in stats.items()),
        **{k: v for k, v in dict_evaluate.items()}
    }
    results_df = pd.concat([results_df, pd.DataFrame([new_row])], ignore_index=True)

except Exception as e:
    print(f"Failed to read {file}: {e}")

print(f"\n>>> Finished processing {file}.")

# Display the loaded graphs information
print(f"\n{' '*80}")
print("SUMMARY OF ALL GRAPHS")
print(f"{' '*80}")
for name, graph_list in graphs.items():
    total_nodes = sum(G.number_of_nodes() for G in graph_list)
    total_edges = sum(G.number_of_edges() for G in graph_list)
    print(f"{name}: {len(graph_list)} graph(s), {total_nodes} nodes, {total_edges} edges")

return results_df
```

6. Execução da Análise

Execute a célula abaixo para processar todos os grafos e gerar o DataFrame comparativo.

```
# Executar análise
df_results = main(folder=os.path.join("final_work", "data_base"))
```

Maximum Classrooms: 0/2552

7. Visualização do DataFrame Comparativo

Visualize as primeiras linhas do DataFrame com todas as métricas calculadas.

```
# Exibir informações do DataFrame
print(f"Total de grafos analisados: {len(df_results)}")
print(f"Total de colunas (métricas): {len(df_results.columns)}")
print("\nPrimeiras linhas do DataFrame:")
df_results.head()
```

Total de grafos analisados: 4
Total de colunas (métricas): 48

Primeiras linhas do DataFrame:

	File	Graph_Index	Num_Nodes	Num_Edges	Centrality_Degree_Average	Centrality_Degree_Minimum	Centrality_Degree_Maximum
0	graph_1098.g6	0	112	560	0.090090	0.090090	0.090090
1	graph_1210.g6	0	112	168	0.027027	0.027027	0.027027
2	graph_1312.g6	0	120	720	0.100840	0.100840	0.100840
3	graph_660_petersen_graph.g6	0	10	15	0.333333	0.333333	0.333333

4 rows x 48 columns

8. Análise Exploratória das Métricas

8.1 Estatísticas Descritivas

```
# Estatísticas descritivas de métricas numéricas
df_results.describe().T
```


	count	mean	std	min	25%	50%	75%	
Graph_Index	4.0	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.0
Num_Nodes	4.0	8.850000e+01	5.246904e+01	1.000000e+01	8.650000e+01	1.120000e+02	1.140000e+02	1.2
Num_Edges	4.0	3.657500e+02	3.293209e+02	1.500000e+01	1.297500e+02	3.640000e+02	6.000000e+02	7.2
Centrality_Degree_Average	4.0	1.378227e-01	1.343456e-01	2.702703e-02	7.432432e-02	9.546521e-02	1.589636e-01	3.3
Centrality_Degree_Minimum	4.0	1.378227e-01	1.343456e-01	2.702703e-02	7.432432e-02	9.546521e-02	1.589636e-01	3.3
Centrality_Degree_Maximum	4.0	1.378227e-01	1.343456e-01	2.702703e-02	7.432432e-02	9.546521e-02	1.589636e-01	3.3
Centrality_Degree_Standard Deviation	4.0	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.0
Centrality_Closeness_Average	4.0	3.915884e-01	1.592381e-01	2.147010e-01	3.200932e-01	3.758262e-01	4.473214e-01	6.0

8.2 Comparação de Densidade entre Grafos

Centrality_Closeness_Minimum	4.0	3.915884e-01	1.592381e-01	2.147010e-01	3.200932e-01	3.758262e-01	4.473214e-01	6.0
Centrality_Closeness_Maximum	4.0	3.915884e-01	1.592381e-01	2.147010e-01	3.200932e-01	3.758262e-01	4.473214e-01	6.0
<pre># Comparar densidade dos grafos if 'Connectivity_Graph Density_Value' in df_results.columns: density_comparison = df_results[['File', 'Graph_Index', 'Num_Nodes', 'Num_Edges', 'Connectivity_Graph Density_Value']].sort_values('Connectivity_Graph Density_Value', ascending=False) print("Ranking de Densidade dos Grafos:") density_comparison else: print("Coluna de densidade não encontrada.")</pre>								
Ranking de Densidade dos Grafos:								
Centrality_Eigenvector_Average	4.0	1.491243e-01	1.114126e-01	9.128709e-02	9.369011e-02	9.449112e-02	1.499253e-01	3.7

8.3 Exportar Resultados para CSV

Centrality_Eigenvector_Minimum	4.0	1.491243e-01	1.114126e-01	9.128709e-02	9.369011e-02	9.449112e-02	1.499253e-01	3.7
Centrality_Eigenvector_Maximum	4.0	1.491243e-01	1.114126e-01	9.128709e-02	9.369011e-02	9.449112e-02	1.499253e-01	3.7
<pre># Salvar DataFrame em arquivo CSV output_path = os.path.join("final_work", "results", "comparative_metrics.csv") os.makedirs(os.path.dirname(output_path), exist_ok=True) df_results.to_csv(output_path, index=False) print(f"Resultados salvos em: {output_path}")</pre>								
Resultados salvos em: final_work\results\comparative_metrics.csv								
Centrality_Katz								
Centrality_Maximum	4.0	1.491243e-01	1.114126e-01	9.128709e-02	9.369011e-02	9.449112e-02	1.499253e-01	3.7

8.4 Métricas Adicionais: Definições e Comparações

Centrality_Katz	4.0	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.0
Centrality_Standard Deviation	4.0	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.0
Esta seção apresenta as métricas adicionais implementadas, suas definições formais, representações matemáticas e relações com as medidas padrão.								
Centrality_PageRank_Average	4.0	3.154762e-02	4.563578e-02	8.333333e-03	8.779762e-03	8.928571e-03	3.169643e-02	1.0
Centrality_PageRank_Minimum	4.0	3.154762e-02	4.563578e-02	8.333333e-03	8.779762e-03	8.928571e-03	3.169643e-02	1.0
Centrality_PageRank_Maximum	4.0	3.154762e-02	4.563578e-02	8.333333e-03	8.779762e-03	8.928571e-03	3.169643e-02	1.0
Centrality_PageRank_Standard Deviation	4.0	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.0

1. Katz Centrality (Centralidade de Katz)

Definição: Mede a importância de um nó considerando não apenas suas conexões diretas, mas também conexões indiretas através de caminhos mais longos, com peso decrescente baseado na distância.

Representação matemática:

$$C_{Katz}(i) = \sum_{k=1}^{\infty} \sum_{j=1}^n \alpha \left(\frac{1}{\lambda_{max}} \right)_{ji}^k$$

onde: λ_{max} é o maior autovalor de A

- A é a matriz de adjacência
- α é o fator de atenuação (tipicamente $0 < \alpha < \frac{1}{\lambda_{max}}$)
- λ_{max} é o maior autovalor de A

Relação com medidas padrão:

- Aproxima-se de: Eigenvector Centrality e Degree Centrality
- Diferença: Enquanto a Eigenvector só considera vizinhos imediatos ponderados recursivamente, Katz adiciona um termo constante (β) que garante que mesmo nós desconectados tenham valor positivo
- Vantagem: Funciona em grafos dirigidos e não direcionados

2. PageRank

Definição: Originalmente desenvolvido pelo Google, mede a importância de um nó baseado na qualidade e quantidade de links que apontam para ele. Simula um "caminhante aleatório" que ocasionalmente salta aleatoriamente para qualquer nó.

onde:	Representação matemática:	Connectivity_Average Node Connectivity_Value	4.0	7.000000e+00	4.690416e+00	3.000000e+00	3.000000e+00	6.500000e+00	1.050000e+01	1.2	
		Connectivity_Graph Density_Value	4.0	$PR(i) = \frac{1-d}{n} + d \sum_{j \in M(i)} \frac{PR(j)}{L(j)}$	1.378227e-01	1.343456e-01	2.702703e-02	7.432432e-02	9.546521e-02	1.589636e-01	3.3
		Connectivity_Average Shortest Path Length_Value	4.0		2.915493e+00	1.259524e+00	1.666667e+00	2.308559e+00	2.668824e+00	3.275759e+00	4.6
		Connectivity_Global Clustering Coefficient_Value	4.0		1.136364e-01	2.272727e-01	0.000000e+00	0.000000e+00	0.000000e+00	1.136364e-01	4.3
		$M(i)$ são os nós que apontam para i									
		$L(j)$ é o número de links saindo de j	4.0		1.000000e+00	0.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.0
		n é o tamanho total do conjunto	4.0		8.850000e+01	5.246904e+01	1.000000e+01	8.650000e+01	1.120000e+02	1.140000e+02	1.2
Relação com medidas padrão:			4.0	5.000000e+00	2.449490e+00	2.000000e+00	4.250000e+00	5.000000e+00	5.750000e+00	8.0	

- **Aproxima-se de:** Eigenvector Centrality
- **Diferença:** PageRank adiciona um fator de "teleportação" aleatória, tornando-o mais robusto para grafos com estruturas complexas
- **Vantagem:** Resolve o problema de "sinks" (nós sem saída) e "dangling nodes"

3. Harmonic Centrality (Centralidade Harmônica)

Definição: Variação da Closeness Centrality que usa a soma dos inversos das distâncias ao invés do inverso da soma. Funciona bem em grafos desconectados.

Representação matemática:

$$C_{Harmonic}(i) = \sum_{j \neq i} \frac{1}{d(i, j)}$$

onde $d(i, j)$ é a distância do caminho mais curto entre i e j (se não existe caminho, contribui 0)

Relação com medidas padrão:

- **Aproxima-se de:** Closeness Centrality
- **Diferença:** Ao invés de $\frac{n-1}{\sum d(i, j)}$, usa $\sum \frac{1}{d(i, j)}$
- **Vantagem:** Não requer que o grafo seja conectado (nós inacessíveis contribuem 0 ao invés de infinito)

4. Current-flow Betweenness Centrality

Definição: Baseado em teoria de circuitos elétricos, mede quanto "fluxo" passa por um nó quando se aplica corrente entre todos os pares de nós. Considera todos os caminhos possíveis, não apenas os mais curtos.

Representação matemática:

$$C_{CF}(v) = \sum_{s \neq t \neq v} \frac{|i_v(s, t)|}{|i_{st}|}$$

onde:

- $i_v(s, t)$ é a corrente que passa por v quando uma unidade de corrente é injetada em s e extraída de t
- Baseado na Lei de Kirchhoff e resistências unitárias

Relação com medidas padrão:

- **Aproxima-se de:** Betweenness Centrality
- **Diferença:** Betweenness conta apenas caminhos geodésicos (mais curtos), enquanto Current-flow considera todos os caminhos proporcionalmente à sua "condutância"
- **Vantagem:** Mais robusto a mudanças estruturais locais; captura redundância de caminhos

8.4.2 Métricas Adicionais de Conectividade

1. Average Node Connectivity (Conectividade Média de Nós)

Definição: Média aritmética da conectividade de nós entre todos os pares de nós do grafo. Representa o número médio de nós que precisam ser removidos para desconectar pares de nós.

Representação matemática:

$$\bar{\kappa}(G) = \frac{1}{\binom{n}{2}} \sum_{s \neq t} \kappa(s, t)$$

onde $\kappa(s, t)$ é o número mínimo de nós cuja remoção desconecta s de t

Relação com medidas padrão:

- **Aproxima-se de:** Node Connectivity (mínima)
- **Diferença:** Node Connectivity é o **mínimo** global, enquanto esta é a **média** de todos os pares
- **Vantagem:** Captura melhor a conectividade "típica" do grafo, não apenas o pior caso

2. Graph Density (Densidade do Grafo)

Definição: Razão entre o número de arestas existentes e o número máximo possível de arestas.

Representação matemática:

$$\delta(G) = \frac{2|E|}{|V|(|V| - 1)}$$

(para grafos não-dirigidos)

onde $|E|$ é o número de arestas e $|V|$ é o número de vértices

Relação com medidas padrão:

- **Aproxima-se de:** Degree Centrality (média)
- **Diferença:** Densidade é uma métrica global do grafo; Degree Centrality é local por nó
- **Interpretação:** $\delta = 0$ (grafo vazio), $\delta = 1$ (grafo completo), valores intermediários indicam esparsidade

3. Average Shortest Path Length (Comprimento Médio do Caminho Mais Curto)

Definição: Média das distâncias dos caminhos mais curtos entre todos os pares de nós. Mede a "eficiência" de comunicação no grafo.

Representação matemática:

$$L(G) = \frac{1}{\binom{n}{2}} \sum_{i \neq j} d(i, j)$$

onde $d(i, j)$ é a distância do caminho mais curto entre i e j

Relação com medidas padrão:

- **Aproxima-se de:** Closeness Centrality (inverso)
- **Diferença:** Closeness é métrica local (por nó), enquanto esta é global
- **Vantagem:** Indica quão "compacto" é o grafo; valores baixos indicam alta eficiência de comunicação
- **Observação:** Só definido para grafos conectados

4. Global Clustering Coefficient (Coeficiente de Agrupamento Global)

Definição: Também conhecido como "transitividade", mede a tendência de nós formarem triângulos fechados. Razão entre triângulos fechados e triplas conectadas.

Representação matemática:

$$C_{\Delta}(G) = \frac{3 \times \text{número de triângulos}}{\text{número de triplas conectadas}} = \frac{3 \times \Delta}{\tau}$$

onde:

- Δ é o número de triângulos fechados
- τ é o número de triplas conectadas (3 nós onde pelo menos 2 arestas estão presentes)

Relação com medidas padrão:

- **Aproxima-se de:** Não tem equivalente direto nas medidas padrão
- **Diferença:** Captura estrutura local de agrupamento em escala global
- **Interpretação:** Valores altos indicam estrutura de comunidades; $C = 0$ em árvores, $C = 1$ em grafos completos

5. Minimum Node Cut (Corte Mínimo de Nós)

Definição: Conjunto mínimo de nós cuja remoção desconecta o grafo. Retorna o conjunto específico, não apenas o tamanho.

Representação matemática:

$$S_{min} = \arg \min_{S \subset V} |S| \text{ tal que } G \setminus S \text{ é desconectado}$$

Relação com medidas padrão:

- **Equivalência a:** Node Connectivity (que retorna apenas o tamanho $|S_{min}|$)
- **Diferença:** Esta métrica retorna o conjunto específico de nós, permitindo identificar vértices críticos
- **Aplicação:** Útil para identificar vulnerabilidades em redes

6. Minimum Edge Cut (Corte Mínimo de Arestas)

Definição: Conjunto mínimo de arestas cuja remoção desconecta o grafo. Retorna o conjunto específico, não apenas o tamanho.

Representação matemática:

$$E_{min} = \arg \min_{E' \subset E} |E'| \text{ tal que } G \setminus E' \text{ é desconectado}$$

Relação com medidas padrão:

- **Equivalência a:** Edge Connectivity (que retorna apenas o tamanho $|E_{min}|$)
- **Diferença:** Esta métrica retorna o conjunto específico de arestas críticas

- **Aplicação:** Identifica "bottlenecks" e links vulneráveis em redes

7. Algebraic Connectivity (Conectividade Algébrica)

Definição: Segundo menor autovalor da matriz Laplaciana do grafo. Mede quão bem conectado é o grafo algebricamente. Também conhecida como **Fiedler value**.

Representação matemática:

$$\lambda_2(L) = \lambda_2(D - A)$$

onde:

- L é a matriz Laplaciana
- D é a matriz diagonal de graus
- A é a matriz de adjacência
- λ_2 é o segundo menor autovalor

Relação com medidas padrão:

- **Aproxima-se de:** Node/Edge Connectivity
- **Diferença:** É uma medida contínua (não discreta) e captura propriedades espectrais
- **Interpretação:** $\lambda_2 = 0$ se e somente se o grafo é desconectado; valores maiores indicam maior robustez
- **Vantagem:** Relacionada com taxa de mistura de caminhadas aleatórias e particionamento espectral

8.4.3 Resumo Comparativo

Métrica Adicional	Métrica Padrão Relacionada	Principal Diferença	Quando Usar
Katz Centrality	Eigenvector + Degree	Considera caminhos de todos os comprimentos com atenuação	Grafos dirigidos ou quando queremos capturar influência
PageRank	Eigenvector	Adiciona "teleportação" aleatória	Redes com estrutura de links (web, citações)
Harmonic Centrality	Closeness	Usa soma de inversos; funciona em grafos desconectados	Quando o grafo pode não ser totalmente conectado
Current-flow Betweenness	Betweenness	Considera todos os caminhos, não só os mais curtos	Quando redundância de rotas é importante
Average Node Connectivity	Node Connectivity	Média vs. mínimo	Para entender conectividade típica, não pior caso
Graph Density	Degree (média)	Métrica global vs. local	Caracterização global de esparsidade
Average Shortest Path	Closeness (inverso)	Métrica global vs. local	Medir eficiência geral de comunicação
Global Conductance	Edge Connectivity	Com equivalência direta	Detectar formação de comunidades (separação)