Graph Visualization

Introduction

Graph visualization is a crucial aspect of network analysis that helps researchers and practitioners understand complex network structures, identify patterns, and communicate findings effectively. This lecture covers various techniques and tools for visualizing graphs.

Visualization Principles

1. Layout Algorithms

```
import networkx as nx
G = nx.karate_club_graph()
pos = nx.spring_layout(G, seed=0)     # swap for any layout
nx.draw(G, pos, with_labels=False, node_size=80)
```

2. Visual Encoding

Node Attributes:

- **Size**: Represents degree, centrality, or other metrics
- **Color**: Indicates node type, community, or categorical attributes
- **Shape**: Distinguishes different node types
- **Opacity**: Shows importance or confidence

Edge Attributes:

- **Thickness**: Represents edge weight or strength
- **Color**: Indicates edge type or direction
- **Style**: Solid, dashed, or dotted lines
- **Curvature**: Shows edge direction in directed graphs

NetworkX Visualization

Basic Plotting

```
import networkx as nx
import matplotlib.pyplot as plt

# Create a simple graph
G = nx.Graph()
G.add_edges_from([(1, 2), (2, 3), (3, 1), (1, 4)])

# Basic visualization
nx.draw(G, with_labels=True)
plt.show()
```

Advanced Styling

```
# Custom styling
pos = nx.spring_layout(G, seed=42)
nx.draw(G, pos,
        node_color='lightblue',
        node_size=500,
        font_size=12,
        font_weight='bold',
        edge_color='gray',
        width=2)
plt.show()
```

Centrality Visualization

```
# Color nodes by centrality
centrality = nx.degree_centrality(G)
node_colors = [centrality[node] for node in G.nodes()]

nx.draw(G, pos,
        node_color=node_colors,
        cmap=plt.cm.Reds,
        node_size=500,
        with_labels=True)
plt.colorbar(plt.cm.ScalarMappable(cmap=plt.cm.Reds))
```

```
plt.show()
```

Gephi Visualization

Introduction to Gephi

Gephi is an open-source network analysis and visualization software that provides interactive exploration of networks.

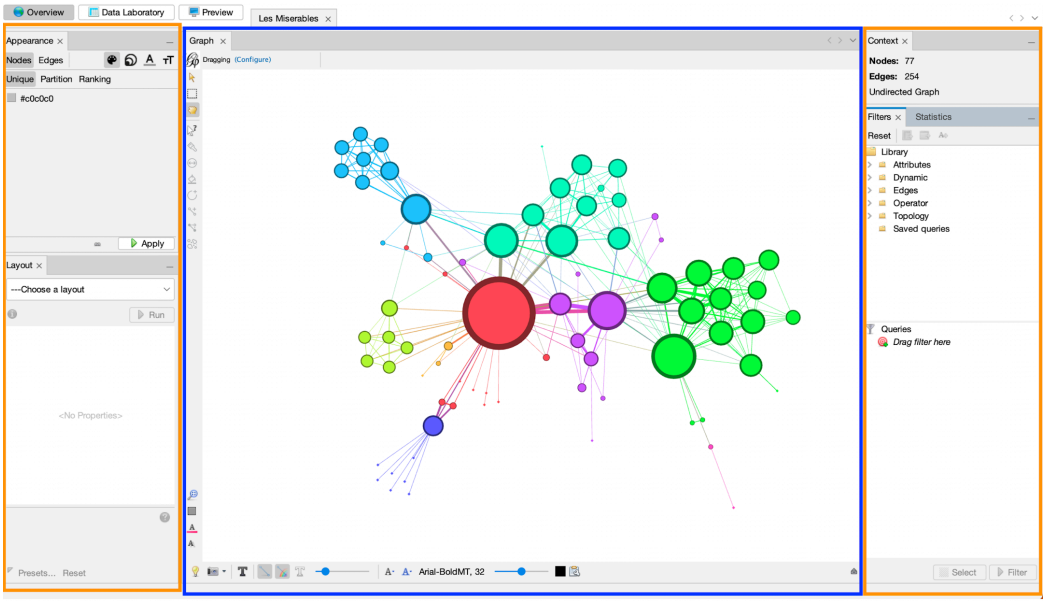
Key Features

1. **Interactive Layouts**:
2. ForceAtlas2: Scalable force-directed algorithm
3. OpenOrd: Fast layout for large networks
4. Yifan Hu: Multilevel layout algorithm
5. **Filtering**:
6. Range sliders for numerical attributes
7. Categorical filters
8. Topology-based filters
9. **Statistics**:
10. Network metrics calculation
11. Community detection
12. Centrality measures

Workflow

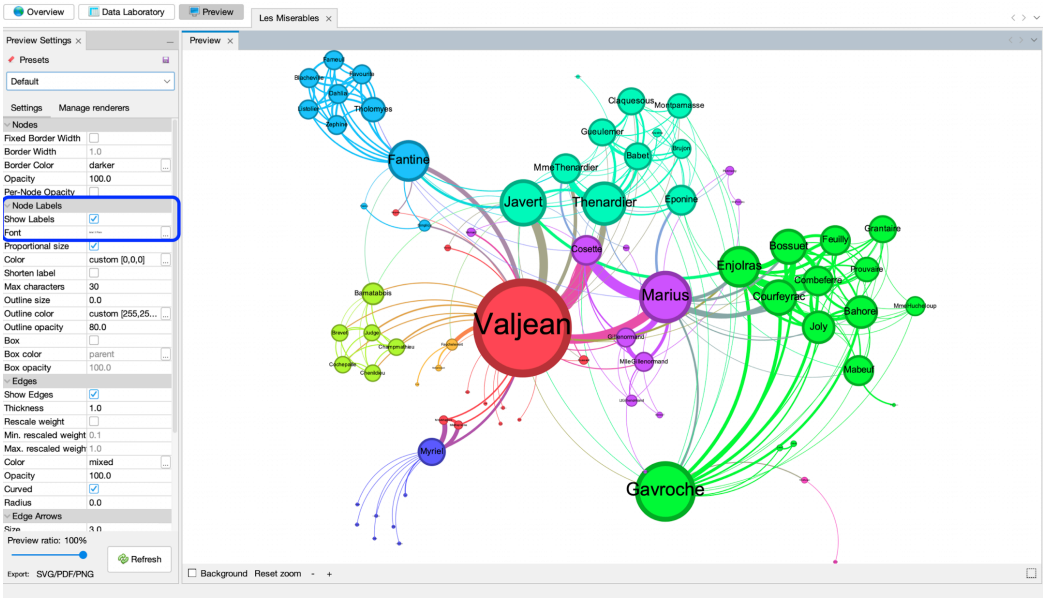
1. **Import Data**: Load network data (GEXF, CSV, etc.)
2. **Layout**: Apply layout algorithm
3. **Filter**: Remove unwanted nodes/edges
4. **Color**: Apply color schemes
5. **Size**: Adjust node/edge sizes
6. **Export**: Save visualization

Let's work on a sample graph in Gephi, les miserables with 77 nodes and 254 edges.



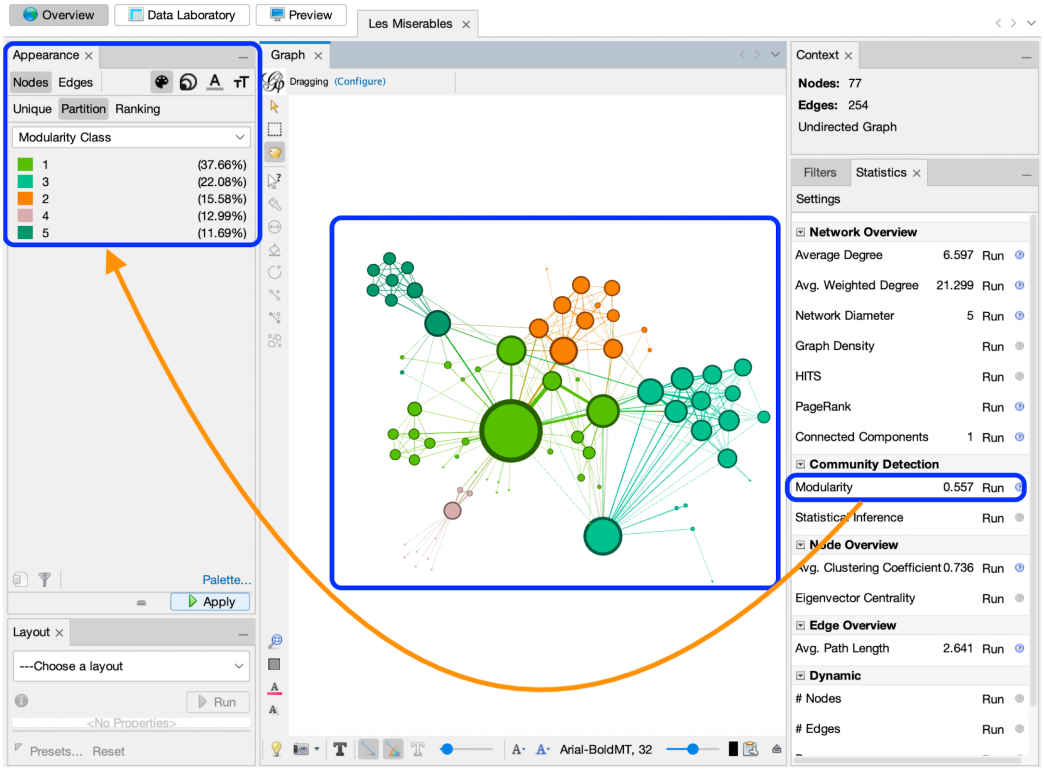
Gephi panes. Three of them are colored in rectangles.

See added node labels for the graph



Node labels added and font size is adjusted. See inside the blue rectangle. Notice the edges being curved.

You can color the nodes based on modularity classs (i.e. output of some clustering algorithms) that is computed under the statistics tab.



Coloring the nodes based on communities.

Interactive Visualizations

Plotly

```
import plotly.graph_objects as go
import networkx as nx

# Create network
G = nx.random_geometric_graph(20, 0.125)

# Get positions
pos = nx.spring_layout(G)

# Create edges
```

```
edge_x = []
edge_y = []
for edge in G.edges():
    x0, y0 = pos[edge[0]]
    x1, y1 = pos[edge[1]]
    edge_x.extend([x0, x1, None])
    edge_y.extend([y0, y1, None])

# Create nodes
node_x = []
node_y = []
for node in G.nodes():
    x, y = pos[node]
    node_x.append(x)
    node_y.append(y)

# Create figure
fig = go.Figure()

# Add edges
fig.add_trace(go.Scatter(
    x=edge_x, y=edge_y,
    line=dict(width=0.5, color='#888'),
    hoverinfo='none',
    mode='lines'))

# Add nodes
fig.add_trace(go.Scatter(
    x=node_x, y=node_y,
    mode='markers',
    hoverinfo='text',
    marker=dict(
        showscale=True,
        colorscale='YlGnBu',
        size=10,
        color=[],
        line_width=2)))

fig.show()
```

Large Network Visualization

Challenges

1. **Overplotting**: Too many nodes/edges overlap
2. **Performance**: Slow rendering for large networks
3. **Clarity**: Difficult to see patterns

Solutions

1. **Sampling**: Show subset of nodes/edges
2. **Clustering**: Group similar nodes
3. **Level-of-Detail**: Show different detail levels
4. **Filtering**: Remove less important elements

Community Visualization

Community Detection

```
# Detect communities
communities = nx.community.greedy_modularity_communities(G)

# Color nodes by community
colors = []
for node in G.nodes():
    for i, community in enumerate(communities):
        if node in community:
            colors.append(i)
            break

nx.draw(G, pos,
        node_color=colors,
        cmap=plt.cm.Set3,
        with_labels=True)
plt.show()
```

Hierarchical Communities

```
# Dendrogram visualization
from scipy.cluster.hierarchy import dendrogram, linkage
from scipy.spatial.distance import pdist

# Create linkage matrix
linkage_matrix = linkage(pdist(nx.adjacency_matrix(G).toarray()))

# Plot dendrogram
plt.figure(figsize=(10, 7))
dendrogram(linkage_matrix)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Node Index')
plt.ylabel('Distance')
plt.show()
```

Temporal Network Visualization

Time Slices

```
# Create temporal network
import pandas as pd

# Example temporal data
temporal_data = pd.DataFrame({
    'source': [1, 2, 3, 1, 2],
    'target': [2, 3, 1, 3, 1],
    'time': [1, 1, 2, 2, 3]
})

# Visualize each time slice
for time in temporal_data['time'].unique():
    edges = temporal_data[temporal_data['time'] == time]
    G_temp = nx.from_edgelist(edges[['source', 'target']].values)

    plt.figure()
    nx.draw(G_temp, with_labels=True)
    plt.title(f'Network at time {time}')
    plt.show()
```

Animation

```
import matplotlib.animation as animation

# Create animation
fig, ax = plt.subplots()

def animate(frame):
    ax.clear()
    edges = temporal_data[temporal_data['time'] == frame + 1]
    G_temp = nx.from_edgelist(edges[['source', 'target']].values)
    nx.draw(G_temp, ax=ax, with_labels=True)
    ax.set_title(f'Time {frame + 1}')

ani = animation.FuncAnimation(fig, animate, frames=3,
interval=1000)
plt.show()
```

Best Practices

Design Principles

1. **Clarity**: Make the visualization easy to understand
2. **Simplicity**: Avoid unnecessary visual elements
3. **Consistency**: Use consistent color schemes and styles
4. **Accessibility**: Consider colorblind-friendly palettes

Color Schemes

1. **Sequential**: For continuous data (e.g., centrality)
2. **Diverging**: For data with a meaningful center
3. **Categorical**: For discrete categories (e.g., communities)

Layout Guidelines

1. **Minimize Edge Crossings**: Use appropriate layout algorithms
2. **Uniform Edge Lengths**: Avoid very long or short edges
3. **Node Separation**: Ensure nodes don't overlap

4. **Symmetry**: Use symmetric layouts when appropriate