Chapter 6

Graph Transformation:
Backboning & Sparsification

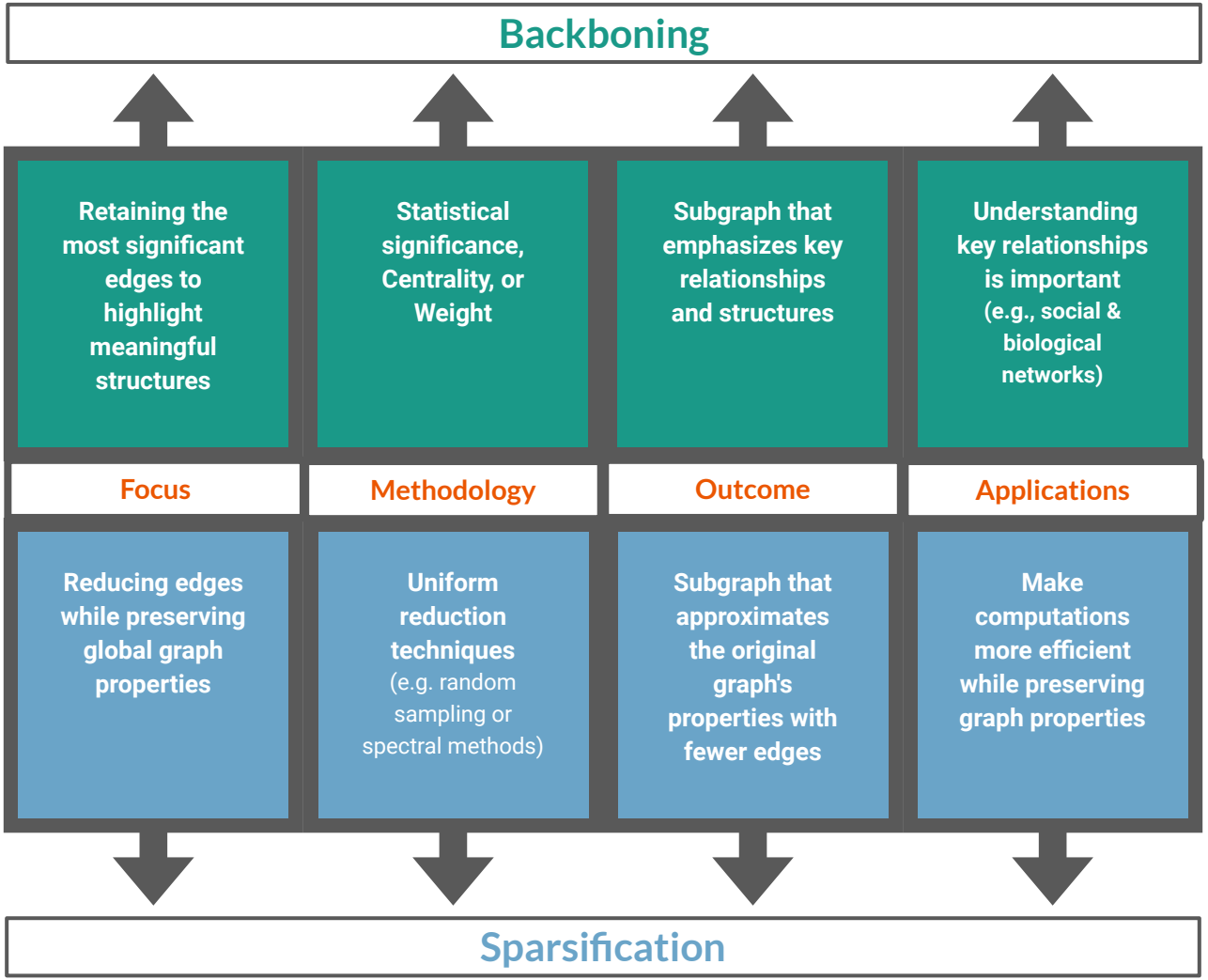
Backboning vs. Sparsification

Backboning

Goal:
Extract a subgraph that retains the most significant edges, preserving the core structure and most important relationships in the original graph.

Sparsification

Goal:
Reduce the number of edges in the graph while approximating the properties of the original graph, such as distances between nodes, graph cuts, or spectral properties.


Backboning
Retaining the most significant edges to highlight meaningful structures
Statistical significance, Centrality, or Weight
Subgraph that emphasizes key relationships and structures
Understanding key relationships is important (e.g., social & biological networks)
Focus
Methodology
Outcome
Applications
Reducing edges while preserving global graph properties
Uniform reduction techniques (e.g. random sampling or spectral methods)
Subgraph that approximates the original graph's properties with fewer edges
Make computations more efficient while preserving graph properties
Sparsification

Backboning

Backboning
Applications

Network analysis
e.g., identifying influential nodes in social networks

Data visualization
e.g., simplifying graphs to improve presentation

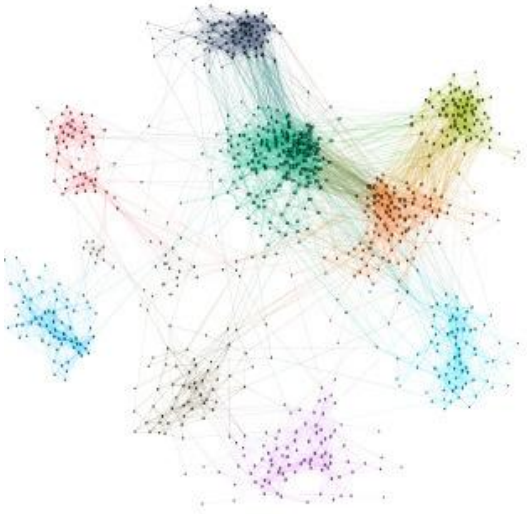
Bioinformatics
e.g., simplifying protein-protein interaction networks

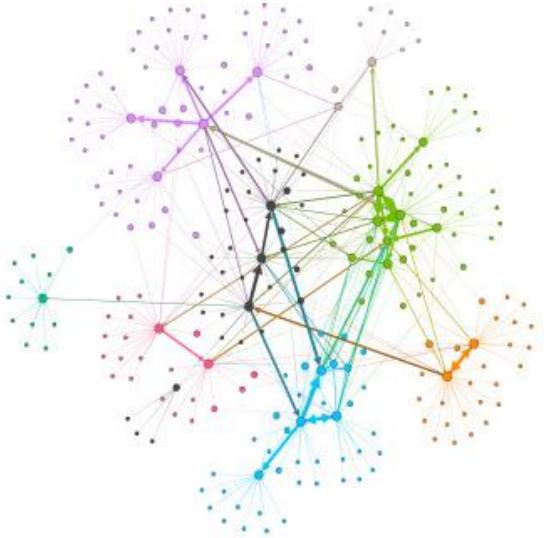
Infrastructure optimization
e.g., simplifying transportation or communication networks

Backboning
Methodologies

Structural approaches
- Naive thresholding,
- Maximum spanning tree,
- Doubly Stochastic,
- High Salience Skeleton,
- Convex network reduction

Statistical approaches
- Disparity Filter,
- Noise Corrected





Assumption:
We are working with weighted graphs

Structural Backboning

Structural
Maximum Spanning Tree

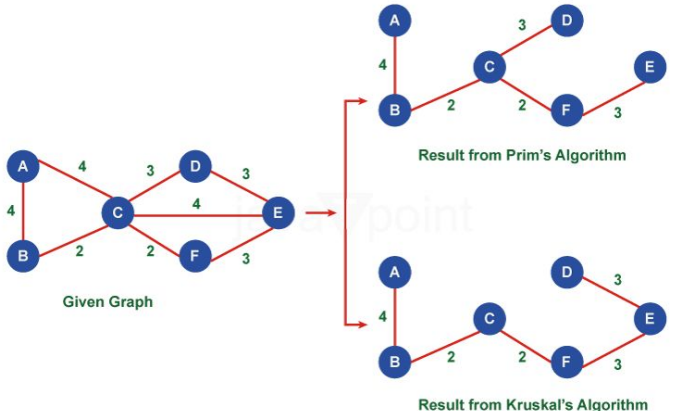
The MST is a subset of the edges that connects all the vertices together, without any cycles and with the maximum possible total edge weight.

Algorithms:
- Kruskal
- Prim

Advantages:
- Simplicity
- Efficiency [O(E log E) Kruskal, O(V^2) Prim]
- Connectivity guarantee

Prim's Algorithm:

1. Start with an arbitrary node and add it to the MST.
2. Find the edge with the maximum weight that connects a vertex in the MST to a vertex outside the MST.
3. Add the edge and the vertex to the MST.
4. Repeat until all vertices are included in the MST.



Shortcomings:
- Loss of redundant (but informative) edges
- Single-scale: focuses on edge weights not on contextual importance
- Sensitivity: if weights are not representative…
- No multi-criteria (e.g., centrality, attributes)

Structural
Naive Thresholding

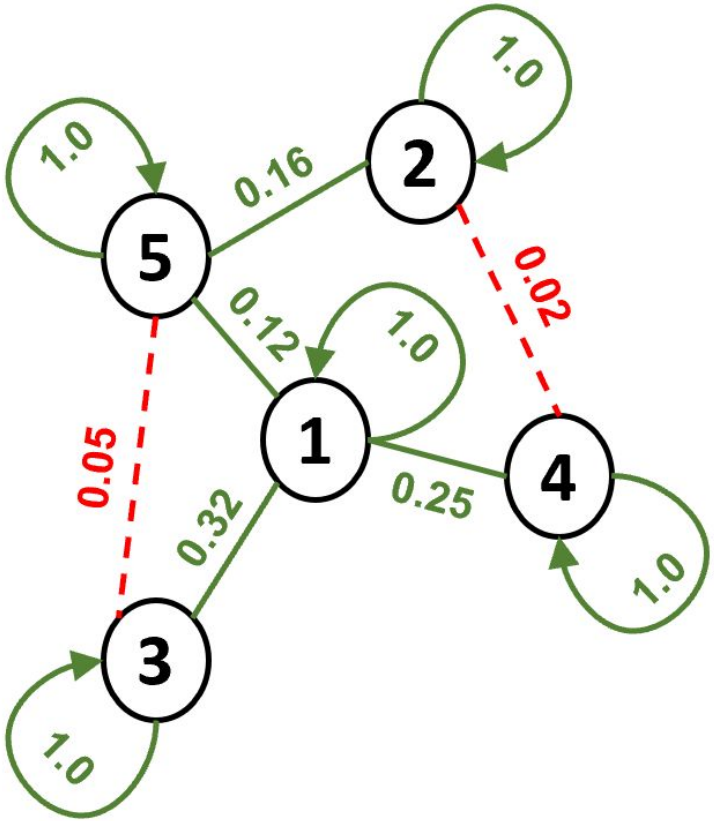
Fix a threshold value τ and remove edges with weights below it from the graph.

Advantages:

- Simplicity: straightforward and easy to implement
- Interpretability
- Flexibility: The threshold can be adjusted

Shortcomings

- Arbitrary Threshold Selection
- Loss of Context
- Sensitivity to Noise
- No Multi-Scale Consideration



Structural
Doubly Stochastic

A doubly stochastic matrix is a square matrix of nonnegative real numbers, where each row and each column sums to one.

- This matrix is used to normalize the adjacency matrix of the graph.
- The normalization rescales the edge-weights and breaks local correlations
- Threshold filtering can then be applied

$$\mathbf{X} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 1/3 & 1/3 & 0 & 1/3 \\ 1/3 & 1/3 & 0 & 1/3 \\ 0 & 0 & 1 & 0 \\ 1/3 & 1/3 & 0 & 1/3 \end{pmatrix} \end{array}$$

Advantages:

- Accounts for variation in node connectivity
- Global context is considered

Shortcomings:

- Computational Complexity
- Implementation Complexity

Structural
High Salience Skeleton

Extract edges that are crucial for the connectivity and flow of information within the graph.

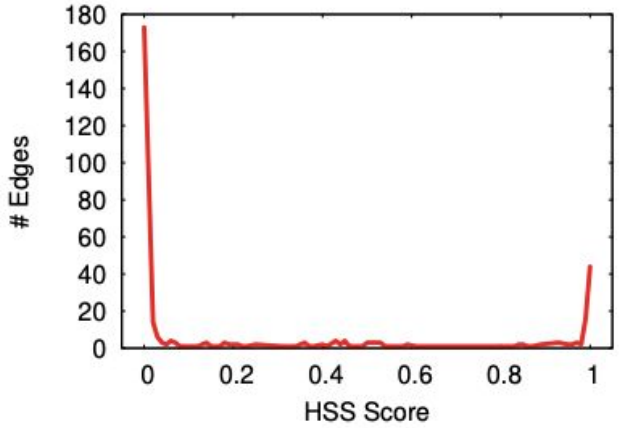
Salient edges are identified based on their contribution to the shortest paths, betweenness centrality, or other measures of structural importance.

Advantages:

- Preservation of Critical Structure (connectivity, flow)
- Flexibility (different significance metrics)
- (almost) parameter-free

Algorithmic Schema:

1. Compute Edge Significance
2. Rank edges based on their significance scores
3. Set a threshold to select edges
4. Construct the Backbone



Shortcomings:

- Computational Complexity

Structural
Convex Network Reduction

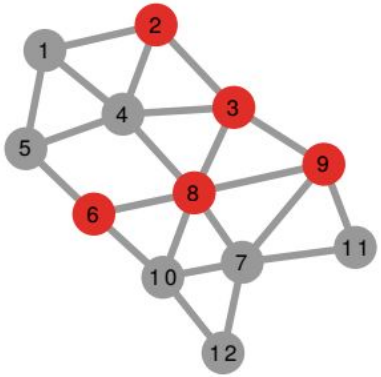
Extract a subgraph where the preserved edges ensure that the subgraph remains convex.

Convexity property:
For any pair of nodes in the backbone, the shortest path between them in the original graph is entirely within the backbone.

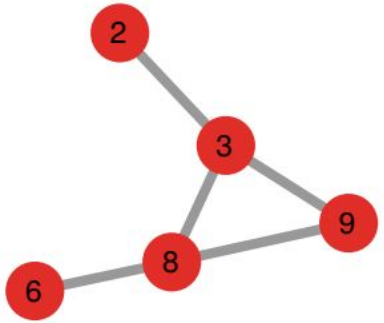
Advantages:
- Preservation of Essential Paths
- Convex Subgraph (shortest paths integrity)

Algorithmic Schema:

1. Compute Shortest Paths
2. Identify Critical Edges
   i.e., those that are part of the shortest paths for any pair of nodes
3. Construct the Backbone



(a)



(b)

Shortcomings:
- Computational Complexity
- Complexity in Implementation
- Potential Over-Retention:
  If many edges are part of shortest paths, the resulting backbone might still be relatively dense

Statistical Backboning

Structural
Disparity Filter

Identify edges whose weights are significantly different from what would be expected if they were distributed uniformly at random, given the node's degree and total weight.

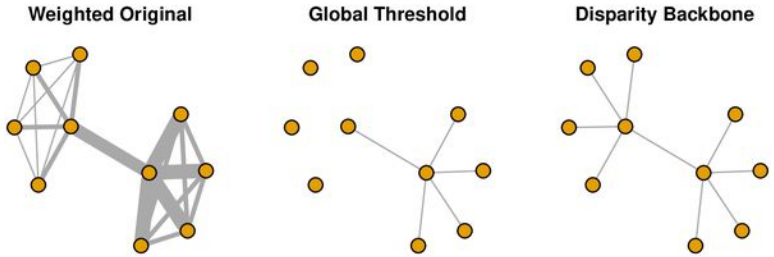
Advantages:

- Statistical Rigor:
  The Disparity Filter provides a rigorous method for identifying significant edges based on local properties.
- Noise Reduction:
  By filtering out edges that are not statistically significant, the method effectively reduces noise
- Adaptability:
  The approach can be adapted to different significance levels

Algorithmic Schema:

1. Calculate Local Weight Distributions
2. Compute p-values for Edge Weights p((u,v), u)
   For each edge (i,j) with weight wij, compute a p-value based on the null hypothesis that the weights are uniformly distributed
3. Significance Thresholding (e.g., α=0.01)
4. Construct the Backbone

$$p((u,v),u) = \left(1 - \frac{w_{u,v}}{\sum_{v' \in N_u} w_{u,v'}}\right)^{(|N_u|-1)},$$



Shortcomings:

- Computational Complexity
- Parameter Sensitivity
- Potential Over-Retention:
  If many edges are part of shortest paths, the resulting backbone might still be relatively dense

Structural
Noise Corrected

Assess the significance of each edge weight by comparing it to an expected distribution derived from the null model.

Edges that significantly deviate from this model are considered part of the backbone.

Advantages:

- Statistical Basis:
  The method relies on statistical principles to distinguish significant edges from noise.
- Noise Reduction
- Adaptability:
  The approach can be adapted to different significance levels and null models

Algorithmic Schema:

1. Compute Expected Weights
   - For each node i, calculate the total strength si (sum of edge weights connected to it) and the degree ki
   - Define the expected weight for an edge (i,j) as $E[w_{ij}]=s_i s_j / 2m$, where m is the total weight of all edges in the graph
2. Calculate Significance Scores:
   - For each edge (i,j), calculate the z-score: $z_{ij} = (w_{ij} - E[w_{ij}]) / \sigma_{ij}$
   - where σ_{ij} is the standard deviation of the edge weight as approx. by a null model
3. Significance Thresholding (e.g., $\alpha=0.01$)
4. Construct the Backbone

Shortcomings:

- Computational Complexity
- Parameter Sensitivity
- Assumptions:
  The method assumes the null model accurately represents the expected distribution of edge weights

Representativeness of Extracted Backbones

Structural Metrics Comparison

Compare structural metrics between the original graph and the backbone to assess how well the backbone represents the overall structure of the graph.

Functional Evaluation

How well the backbone preserves aspects/characteristics relevant to the application domain?
(e.g., node importance, connectivity)

Edge Preservation and Information Loss

How much information is retained/lost during the backbone extraction process?
(e.g., edge coverage, mutual information)

Graph Sparsification

Graph Sparsification
Applications

Spectral Properties:
Graph sparsification is often used to preserve the spectral properties (eigenvalues and eigenvectors) of the original graph. This is crucial for applications like spectral clustering, where maintaining the global structure is important.

Approximate Cut Values:
Sparsification methods, such as those based on spectral techniques, ensure that the weights of cuts in the sparsified graph approximate those in the original graph. This is essential for problems in network design and optimization.

Large-Scale Graphs:
Graph sparsification techniques can handle very large graphs efficiently. They reduce the number of edges while still preserving essential graph properties, making the analysis more scalable.

Algorithmic Speedup:
Many graph algorithms have a runtime that depends on the number of edges. Sparsification reduces the number of edges, thereby speeding up algorithms for tasks such as shortest path, clustering, and other graph-based computations.

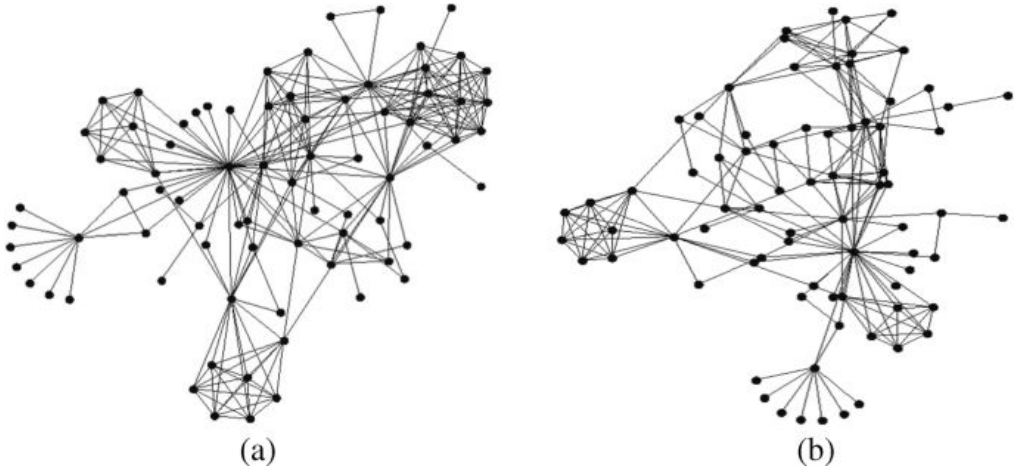
Graph Sparsification
Applications (cont'd)

Graph Compression:
In scenarios where the goal is to compress the graph while retaining its overall structure, sparsification is more appropriate as it focuses on global properties.

Network Simulation:
For simulating dynamics on networks (such as spreading processes or flow), a sparsified graph that preserves the overall connectivity and flow characteristics is often more suitable than a backbone that emphasizes only the most significant connections.

Graph Sparsification
Methodologies

- Random Edge Sampling
- Spectral Sparsification
- Spanner Construction
- Cut Sparsifier
- Local Sparsification Techniques
- Graph Coarsening



Assumption:
We are working with weighted graphs

Random Edge Sampling

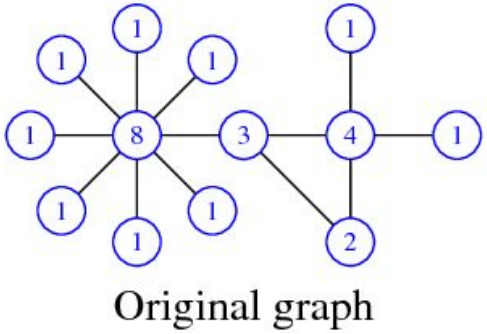
Selects a subset of edges randomly, often with probabilities proportional to their weights or uniformly.

Advantages:

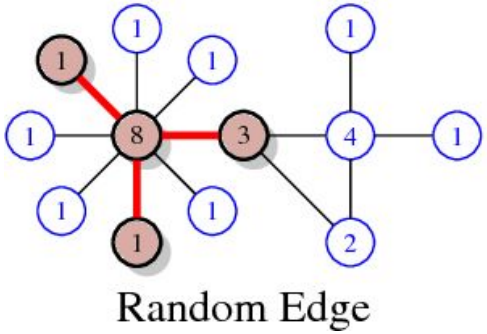
- Simple and easy to implement;
- Scalable to large graphs.

Shortcomings:

- May not preserve important structural properties accurately without careful sampling strategies



Original graph



Random Edge

Spectral Sparsification

Uses spectral properties (eigenvalues and eigenvectors) of the graph to retain edges that preserve the overall structure of the graph.

Algorithms:
- Spielman-Srivastava

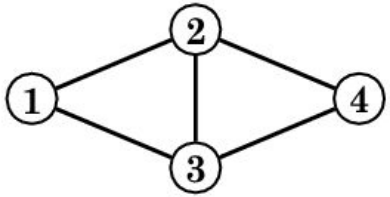
Advantages:
- Preserves global properties and provides theoretical guarantees.

Shortcomings:
- Computationally intensive;
- Requires knowledge of eigenvalues and eigenvectors.

Spielman-Srivastava's Algorithm:

1. For each edge (i,j) compute the resistance Rij (how important the edge is for the graph connectivity) using the Laplacian matrix;
2. Assign to each edge a sampling probability based on Rij;
3. Sample edges independently

$$D = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$



$$A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

$$L = \begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ 0 & -1 & -1 & 2 \end{bmatrix}$$

Graph Laplacian:
The Laplacian matrix L of a graph G is defined as L=D−A, were D is the degree matrix (a diagonal matrix with vertex degrees on the diagonal), and A is the adjacency matrix.

Spanner Construction

Constructs a subgraph (spanner) that approximates the distances between nodes within a certain factor (stretch factor).

Advantages:

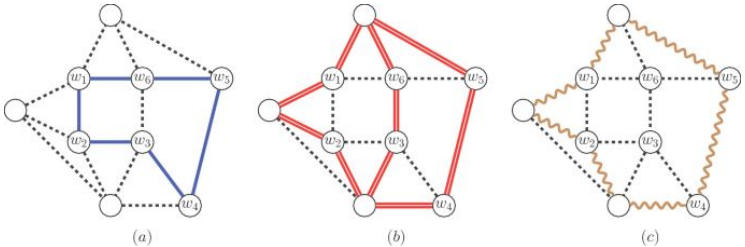
- Preserves approximate distances and shortest paths.

Shortcomings:

- May still include a large number of edges depending on the stretch factor.

Multiplicative Spanner Algorithm:

1. Sort edges in G by increasing weights
2. Edge addition
   - Iterate over the sorted (u, v):
   - check if adding (u, v) to H create a path violating the k factor
   - if d(u,v) in H is greater than k d(u,v) in G add the edge to H
3. Return H



3-spanner

Spanner:
A subgraph H of the original graph G is called a (k,ϵ)-spanner if for all pairs of nodes u and v, the distance d(u,v) in H is at most K times the distance IN G plus an additive term ϵ.

Cut Sparsifier

Preserve the values of all cuts in the graph within a certain approximation factor

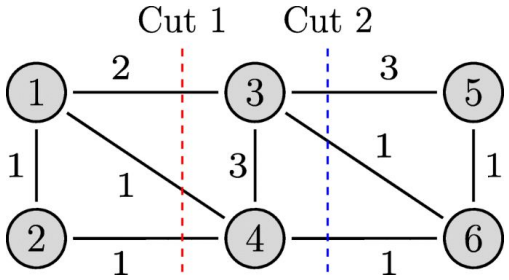
Algorithms:
- Benczúr-Karger

Advantages:
- Maintains cut properties
  (useful for network flow and partitioning problems)

Shortcomings:
- Computationally expensive for very large graphs.
- Parameter sensitivity

Benczúr-Karger's Algorithm:

1. Compute Edge Connectivity Values
   This value indicates how critical the edge is to maintaining the connectivity of the graph
2. Assign Sampling Probabilities based on the edge's connectivity value and the desired error parameter ϵ
3. Sample Edges
4. Return H



Cut:
A cut in a graph G is a partition of the vertices into two disjoint subsets S and S^-. The cut value is the sum of the weights of edges that have one endpoint in S and the other in S^-.

Cut Sparsifier:
H is a (1±ϵ)-cut sparsifier of G if, for every cut (S,S^-) in G, the total weight of edges in H crossing the cut is within a factor of (1±ϵ) of the total weight of edges crossing the cut in G.

Local Sparsification

Focuses on preserving local structures around each node or within subgraphs.

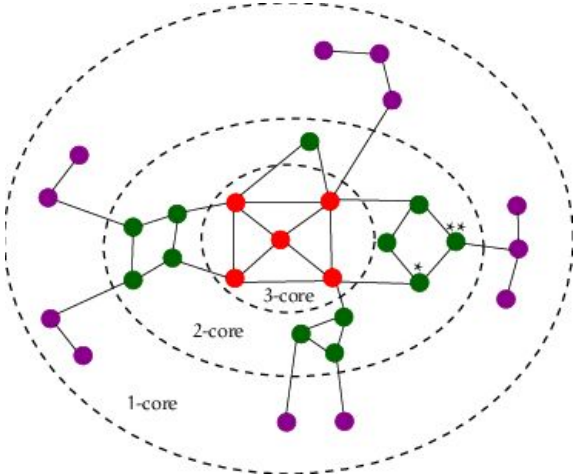
Algorithms:
- k-core decomposition
- k-plex

Advantages:
- Maintains local connectivity and clustering properties.
- Scalability

Shortcomings:
- May not preserve global properties well.
- Parameter sensitivity
- Disconnected components

K-core Algorithm:

1. Compute initial degrees
2. Peeling process:
   - Iteratively remove nodes with degree less than k and update their neighbors' degrees
   - Continue until no vertices with degree less than k remains
3. Return the k-core



k-core
A k-core of a graph is a maximal subgraph in which each vertex has at least degree k. In other words, every vertex in the k-core is connected to at least k other vertices within the k-core.

Graph Coarsening

Iteratively merges nodes and edges to form a coarser, smaller graph while preserving overall structure.

Advantages:

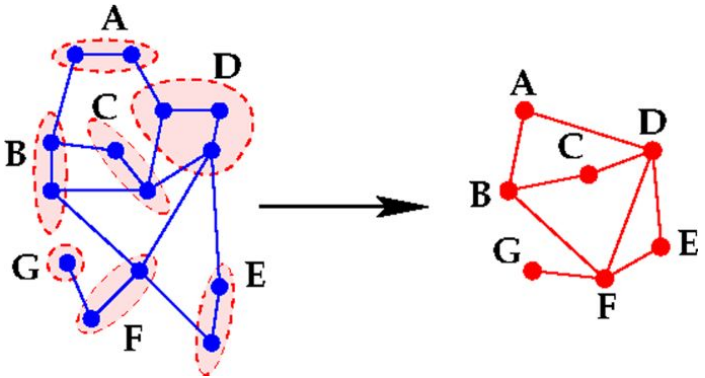
- Efficiency
- Preservation of Structure
- Reduces graph size significantly; useful for multiscale analysis.

Shortcomings:

- May lose fine-grained details
- Requires careful selection of merging criteria.

Coarsening Schema:

1. Matching:
   Pair nodes to form "supernodes" in the coarsened graph.
2. Contraction:
   - For each pair of matched nodes, create a single supernode in the coarsened graph;
   - Combine edges between the supernodes, summing the weights of edges that were originally between the matched nodes
3. Interpolation
   - Once the coarsened graph is processed the results are projected back to the original graph.



Discussion

Why should I care about it?

We have already seen how to sample a graph…
why should I also backbone/sparsify it?

Good question!

Short answer;

- Sampling imposes a constraint on the data to be collected;
- Backboning/Sparsification on the data we want to analyze

If sampling extract non representative structures backboning/sparsification will still be non representative

Not all topologies can be “sampled” directly
(i.e., bipartite graphs that whose projection we are interested in)

Sometimes you can access the whole network (lucky you!) and need to simplify it.

Chapter 6

Conclusion

Take Away Messages

1. Backboning and Sparsification are conceptually similar but move from different needs
2. Each transformation has an impact on topology characteristics

Suggested Readings

- The Atlas for the aspiring Network Scientist (Ch 24)

What's Next

Chapter 7: When Topology meets Semantics