Network Visualization

Miha Lazić[a,1], **Luka Gulič**[a], **and Matevž Crček**[a]

[a]University of Ljubljana, Faculty of Computer and Information Science, Večna pot 113, SI-1000 Ljubljana, Slovenia

The manuscript was compiled on May 30, 2025

Network visualization serves as an essential tool for understanding the structural and functional properties of complex networks. As networks increase in size and complexity, their raw adjacency representations become increasingly difficult to interpret, necessitating effective visual techniques. This handout provides an overview of core concepts in network visualization, focusing on the transition from raw graph data to meaningful spatial layouts. We discuss the principles guiding layout algorithms—such as symmetry, minimization of edge crossings, and uniform edge lengths—and analyze the strengths and limitations of major layout computation techniques, including force-directed models (Eades, Fruchterman-Reingold), energy-based approaches (Kamada-Kawai), and blockmodeling method.

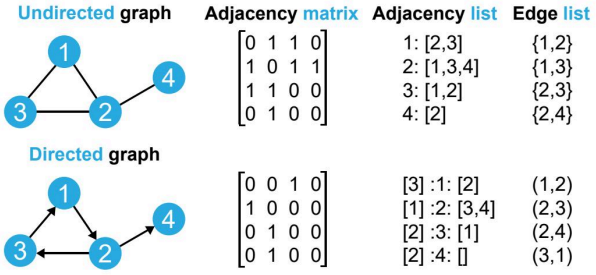

Fig. 1. Various tabular graph representations.

Network visualization plays a fundamental role in enabling researchers and analysts to explore, interpret, and communicate complex relational data. The core problem is how to represent the topological and functional properties of a network in a spatial layout that is both informative and visually comprehensible. The motivation for studying network visualization arises from its wide applicability across disciplines, from analyzing protein interactions in biology, to studying social ties in sociology, to mapping internet infrastructure in computer science. Effective visualizations reveal patterns, clusters, and anomalies that may remain hidden in purely statistical analyses.

1. From graph representation to network visualization

A graph G can be defined as a set nodes V and a set of *undirected* or *directed* edges E such that an edge describes the existence of a relationship between two nodes i and j. Drawing a graph can help make better sense of the structure of these relationships than simply looking at the raw data in tabular representations, presented in Figure 1:

- **Adjacency matrix**: A square matrix where each entry (i, j) indicates the presence of an edge from node i to node j. This representation is particularly useful for elegant analytical derivations and is the basis for many matrix-based algorithms.
- **Adjacency list**: Each node stores a list of its neighbors. This structure is memory-efficient for sparse graphs and well-suited for algorithmic implementations, since most graph algorithms require quick access to neighbors.
- **Edge list**: A simple list of all edges, where each edge is represented by a pair (or tuple) of nodes. This format is ideal for storage and manipulation, as each edge is recorded only once and can be easily edited or processed.

The standard visualization process of constructing the ***wiring diagram*** typically follows a three-step procedure: *layout computation*, *link representation*, and *node representation*. The overall algorithm is outlined below.

Definition (Wiring Diagram)

A wiring diagram *is a spatial representation of a network in which nodes are placed at specific coordinates, and edges are drawn as curves or lines connecting the nodes.*

Algorithm 1 Wiring Diagram Construction

Require: Graph $G = (V, E)$
Ensure: Visual representation of G
Step 1: Layout Computation
Compute node coordinates in the Euclidean plane using a layout-computation algorithm (e.g., force-directed)
Step 2: Representation of Links
for each edge $(i, j) \in E$ **do**
    Set visual properties: line thickness (strength), pattern (type), color (weight/class), etc.
Step 3: Representation of Nodes
for each node $i \in V$ **do**
    Set visual properties: size (e.g., degree), shape (e.g., type), color (e.g., community), label
return Rendered visualization of G

2. The layout-computation algorithm is all you need

Simply drawing a graph is not enough—its layout and visual appearance significantly shape how users interpret relationships. Due to the ***Gestalt principle of proximity***, users tend to perceive spatially close nodes as being related, even if no actual connection exists. Therefore, finding a layout-computation algorithm that emphasizes real relationships without misleading the user is crucial (1).

All authors contributed equally to this work.

[1]To whom correspondence should be addressed. E-mail: ml6163@student.uni-lj.si.

Definition (Gestalt Principle of Proximity)

The Gestalt principle of proximity *states that visual elements positioned close to one another are perceived as belonging together.*

In addition, graph layout should respect a set of aesthetic principles to improve readability and comprehension:

- **Minimise edge crossings**: Improves readability and helps users follow paths through the graph. Excessive crossings can obscure important information and make the graph feel less approachable (2).
- **Symmetry**: Helps users understand the overall structure of the graph more easily (1).
- **Uniform edge lengths**: Promotes a regular appearance and prevents distortion in the graph layout (1).
- **Uniform node distribution**: Enhances visual appeal and prevents the graph from appearing cluttered (3).
- **Separate non-adjacent nodes**: Since proximity suggests a relationship, non-adjacent nodes should not appear too close together (1).
- **Avoid node–edge overlap**: Prevents ambiguity about where edges begin and end and keeps visual elements from appearing too clustered (3).

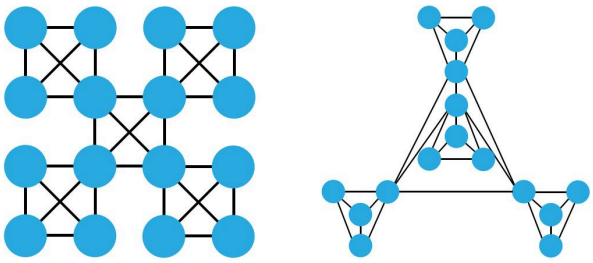

Fig. 2. Two layouts of the same graph. Although both are symmetrical, users consistently find the layout on the left easier to interpret, even though it violates the principle of minimizing edge crossings. This highlights that visual clarity and symmetry can sometimes outweigh strict adherence to aesthetic rules. (Adapted from Kamada and Kawai (4).)

Based on (1), algorithms for computing the node's coordinates can be broadly categorized into three families: **Force-Directed Methods**, Dimension Reduction Techniques, and Computational Improvements.

Force-Directed Methods. Force-directed algorithms model the graph as a physical system, where nodes repel each other like charged particles and edges act as springs pulling connected nodes together. The layout is computed by iteratively adjusting node positions to minimize the overall energy of the system, ideally reaching a state of equilibrium. A classic example is Eades' spring-embedder algorithm (5), while Fruchterman-Reingold (6) and Kamada-Kawai (4) are among the most influential variants. All three algorithms are described in details in upcoming sections.

A. Eades' Spring-Embedded Layout. One of the foundational approaches to graph layout is the spring-embedded algorithm proposed by Eades. In this model, we imagine each node as a steel ring and each edge as a spring connecting two rings. The layout is formed by simulating the motion of this system until it settles into a balanced configuration, where the forces on all nodes are close to zero. Two types of forces act in this system:

- **Attractive forces** pull connected nodes closer together, like a stretched spring trying to return to its rest length.
- **Repulsive forces** push all nodes away from each other, mimicking electrical repulsion to prevent overlapping and crowding.

These forces are defined mathematically as:

$$\textbf{Attractive force:} \quad f_a(i,j) = c_1 \cdot \log\left(\frac{l_{ij}}{c_2}\right)$$

$$\textbf{Repulsive force:} \quad f_r(i,j) = -\frac{c_3}{l_{ij}^2}$$

where:

- l_{ij} is the Euclidean distance between nodes i and j,
- c_1, c_2, c_3 are constants that control the strength of the forces.

Notice that the attractive force grows logarithmically rather than linearly with distance. This choice was intentional: linear springs pull too strongly when stretched, which can distort the layout. A logarithmic spring avoids this by pulling gently at long distances but still keeping nodes together.

Algorithm 2 Eades' Spring-Embedded Layout

Require: Undirected graph $G = (V, E)$
Ensure: Node positions that visually balance attractive and repulsive forces

Initialization
for each node $i \in V$ **do**
    Place node i at a random position in the plane
Force Definitions
for each pair of nodes (i, j) **do**
    Compute Euclidean distance l_{ij} between nodes i and j
    if $(i, j) \in E$ **then**
        Attractive force: $f_a(i,j) = c_1 \cdot \log\left(\frac{l_{ij}}{c_2}\right)$
    else
        Repulsive force: $f_r(i,j) = -\frac{c_3}{l_{ij}^2}$
Iterative Layout Adjustment
while not converged **do**
    for each node $i \in V$ **do**
        Compute net force $\vec{F}_i$ from all other nodes
        Move node i slightly in the direction of $\vec{F}_i$
return Final node positions as the layout of G

B. Fruchterman–Reingold Force-Directed Layout. Fruchterman and Reingold proposed a simplified and faster adaptation of Eades' spring-embedder algorithm. Their method was designed to produce aesthetically pleasing layouts while being computationally efficient and requiring minimal parameter

tuning. The layout aims to ensure that connected nodes are placed close together (but not too close), edge lengths are uniform, edge crossings are minimized, nodes are symmetrically distributed, and the entire graph fits well within the drawing area. The forces are defined as follows:

$$\textbf{Attractive force:} \quad f_a(i,j) = \frac{l_{ij}^2}{k}$$

$$\textbf{Repulsive force:} \quad f_r(i,j) = -\frac{k^2}{l_{ij}}$$

where:

- l_{ij} is the Euclidean distance between nodes i and j,
- k is the optimal inter-node distance, computed as $k = C \cdot \sqrt{\frac{\text{area}}{n}}$, where *area* is the available drawing space and n is the number of nodes,
- C is a user-defined constant, typically determined through experimentation.

Although the algorithm is not guaranteed to converge, Fruchterman and Reingold found that approximately 50 iterations were sufficient to produce high-quality layouts for most graphs with up to 100 nodes. To further improve runtime, they introduced the Grid Variant Algorithm (GVA), which computes repulsive forces only between nodes located in the same or neighboring grid cells, effectively reducing computational complexity. This variant also prevents nodes from spreading too far apart and, in some cases, may enhance layout quality according to graph drawing aesthetics (1).

C. Kamada–Kawai Graph Theoretic Layout. Kamada and Kawai introduced an energy-based graph drawing algorithm built upon the principle that the Euclidean distance between nodes in the layout should approximate their graph-theoretic distance—i.e., the length of the shortest path between them. Their layout is derived from classical mechanics, where edges are modeled as springs, and the cost function represents the system's total energy. The resulting configuration aims to minimize this energy to produce a visually balanced, symmetric layout with few edge crossings.

The energy function they propose is:

$$E = \frac{1}{2} \sum_{i<j} k_{ij} (\|x_i - x_j\| - l_{ij})^2$$

where:

- x_i, x_j are the positions of nodes i and j in the layout,
- $l_{ij} = L \cdot d_{ij}$ is the ideal Euclidean distance between nodes i and j,
- d_{ij} is the graph-theoretic (geodesic) distance between nodes i and j,
- $k_{ij} = \frac{1}{d_{ij}^2}$ is the spring stiffness constant,
- L is a scaling factor used to fit the drawing within a given space.

This layout algorithm iteratively moves one node at a time to minimize the total energy, using the Newton–Raphson method. Unlike force-directed models that update all nodes simultaneously, Kamada–Kawai's approach selects the node with the highest gradient (i.e., the node most "out of balance") in each iteration, refining its position until equilibrium is achieved.

D. Evaluation - Which visualization is better?. The quality of a computed layout is typically assessed based on adherence to classical graph drawing principles, described in section 2, as well as computational efficiency. However, in many studies, visual inspection alone and direct comparing between visualizations is used to judge layout quality (1).

In Figure 3, the Eades' algorithm performs well for small and sparse graphs up to 30 nodes (1), achieving similar link lengths and symmtery. The Fruchterman-Reingold algorithm builds on the same physical metaphor but incorporates a global repulsion that pushes all nodes away from each other, balancing it against edge attractions. This produces more evenly spaced layouts in shorter time without the need of parameter optimization (1). The Kamada-Kawai algorithm approaches the layout problem by minimizing an energy function based on graph-theoretic distances, striving to place nodes such that their Euclidean distance in the layout reflects their shortest-path distance in the graph. This results in more symmetric and geometrically balanced layouts, as seen in both the *Karate Club* (7) and *Dolphins* (8) networks, where the clusters appear evenly distributed and well-separated. However, Kamada-Kawai is computationally intensive and may struggle with larger networks, much bigger than *American Football* (9).

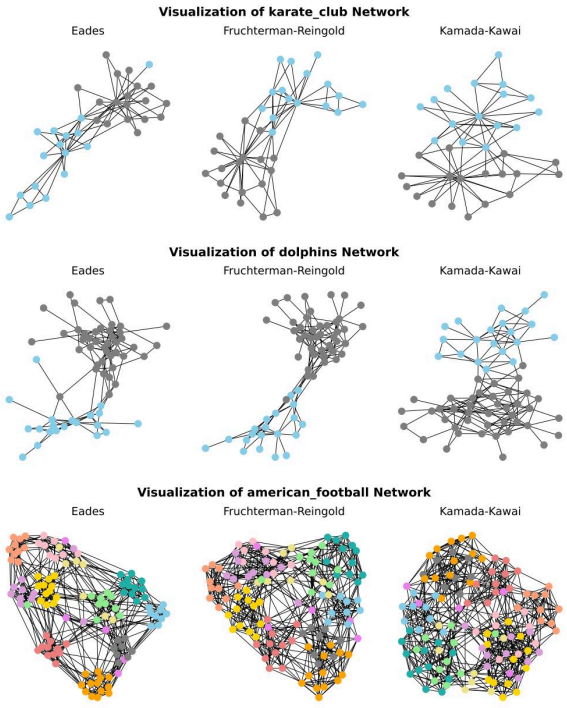

Fig. 3. Comparison of layout-computation algorithms on different networks.

Overall, these comparisons highlight that Eades is best suited for small, sparse graphs, Fruchterman-Reingold offers a balance between speed and clarity for medium-sized graphs, and Kamada-Kawai produces the most visually appealing layouts for small networks, though it becomes less practical for larger, complex graphs.

In contrast, Figure 4 illustrates the visualizations produced by a selection of layout algorithms from `networkx` library (10) that fail to provide meaningful insights into the graph's structure. The *Random Layout* arranges nodes without any structural logic. The *Spectral Layout*, based on the graph Laplacian, produces elongated and skewed layouts that do not reveal any community structures or centralities. The *Spiral Layout* arranges nodes in a spiral shape, which is visually distinct but often artificial and disconnected from the graph's topology. The *Shell Layout* places nodes on concentric circles, creating an arbitrary ring structure that lacks relevance to node properties or connectivity. Lastly, the *Circular Layout* places nodes evenly around a circle, which is at least visually symmetrical.

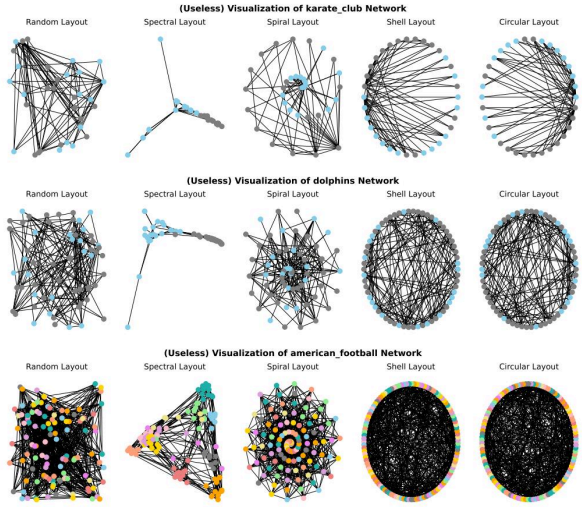

Fig. 4. Comparison of non-informative (useless) layout-computation algorithms on different networks.

3. Blocks, blocks, blocks, ...

While layout computation algorithms aim to produce visually appealing representations of networks, they often lack interpretability in terms of node roles or group structure. ***Blockmodeling*** offers an alternative approach by abstracting the network into blocks of equivalent nodes, revealing the relational patterns and structural roles that shape the network's organization.

Definition (Blockmodeling)

Blockmodeling *is a method for simplifying and analyzing the structure of a network by grouping nodes into clusters (blocks) based on a predefined* ***measure of equivalence****, such as structural or regular equivalence. These blocks represent roles or groups in the network.*

The standard procedure of creating the block model consists of the following steps:

1. Perform blockmodeling via (hierarchical) clustering with computational complexity $\mathcal{O}(n^2)$.
2. Return the block model at the desired clustering resolution.

A. Structural Equivalence Measures. The classical approach is known as *structural equivalence*, which captures the idea that two nodes are similar if they share many common neighbors. This notion emphasizes direct positional similarity in the network. Several measures have been developed to quantify this similarity, each with different assumptions about how to interpret shared neighbors.

Standard Structural Equivalence (11) This measure defines the similarity between two nodes i and j as the number of common neighbors they share:

$$\sigma_{ij} = \sum_x A_{ix} A_{xj} = |\Gamma_i \cap \Gamma_j|.$$

Intuitively, if two nodes are connected to the same other nodes, they are considered structurally similar. This is a strict form of equivalence that only considers exact neighbor overlap and ignores the broader network context.

Salton Structural Equivalence (12) Salton's measure (also known as the cosine similarity) extends the standard measure by normalizing the number of shared neighbors by the degrees of the nodes:

$$\sigma_{ij} = \cos\theta_{ij} = \frac{\sum_x A_{ix} A_{xj}}{\sqrt{\sum_x A_{ix}^2}\sqrt{\sum_x A_{jx}^2}} = \frac{|\Gamma_i \cap \Gamma_j|}{\sqrt{k_i k_j}},$$

where k_i and k_j are the degrees of nodes i and j. This normalization accounts for the fact that nodes with many connections are more likely to share neighbors by chance. As such, it emphasizes the proportion of shared neighbors rather than their raw count.

Leicht Structural Equivalence (13) Leicht et al. proposed a further normalization that adjusts for the expected number of shared neighbors under a random graph model:

$$\sigma_{ij} = \frac{|\Gamma_i \cap \Gamma_j|}{k_i k_j / n},$$

where n is the total number of nodes in the network. This formulation compares the observed overlap of neighborhoods to what would be expected if links were formed randomly, thereby highlighting statistically significant similarities rather than just raw or proportionate overlap.

B. Regular Equivalence Measures. *Regular equivalence* generalizes *structural equivalence* by the requirement that similar nodes have equivalent neighbors.

Standard Regular Equivalence (14) The standard regular equivalence measures how similar the *roles* of two nodes are, based on the similarity of their neighbors' roles. Specifically, two nodes i and j are considered similar if their neighbors $x \in \Gamma_i$ and $y \in \Gamma_j$ are themselves similar to each other. The similarity σ_{ij} is thus computed recursively by averaging the similarities

of all neighbor pairs (x, y) between i and j. This results in a global, self-consistent measure of role equivalence: nodes are equivalent if their neighbors are equivalent, and so on. Over iterations, the similarity matrix σ converges to reflect higher-order structural roles within the network, independent of exact neighbor overlap.

$$\sigma_{ij} = \alpha \sum_{xy} A_{ix} A_{jy} \sigma_{xy} + \delta_{ij} = \alpha \sum_{x \in \Gamma_i} \sum_{y \in \Gamma_j} \sigma_{xy} + \delta_{ij},$$

$$\sigma = \alpha A \sigma A + I,$$

$$\sigma^{(0)} = 0, \quad \sigma^{(1)} = I, \quad \sigma^{(2)} = \alpha A^2 + I, \quad \sigma^{(3)} = \alpha^2 A^4 + \alpha A^2 + I, \ldots$$

Katz Regular Equivalence (15) Katz regular equivalence simplifies this idea by focusing on paths of influence. Here, node i is considered similar to node j if i is connected to other nodes x that are themselves similar to j. Unlike the standard version, this measure only aggregates over i's neighbors, comparing them directly to j. This creates an asymmetric, path-based notion of similarity where influence flows from i's neighborhood toward j, and longer paths are down-weighted by the factor α. As a result, the Katz measure captures how easily node j can be reached from node i through sequences of structurally similar nodes, emphasizing the accessibility of roles rather than mutual equivalence.

$$\sigma_{ij} = \alpha \sum_{x} A_{ix} \sigma_{xj} + \delta_{ij} = \alpha \sum_{x \in \Gamma_i} \sigma_{xj} + \delta_{ij},$$

$$\sigma = \alpha A \sigma + I,$$

$$\sigma^{(0)} = 0, \quad \sigma^{(1)} = I, \quad \sigma^{(2)} = \alpha A + I, \quad \sigma^{(3)} = \alpha^2 A^2 + \alpha A + I, \ldots$$

Definition (Meso-scale Network Structure)

Meso-scale network structure *refers to the intermediate-level organization of a network, capturing patterns of connectivity that are not visible at the level of individual nodes (micro-scale) or the entire network (macro-scale). Typical examples of meso-scale structures include communities, which reveal groups of nodes with dense internal connections and specific intergroup linking patterns.*

Blockmodeling examples with Infomap. To investigate the ***meso-scale structure*** of real-world networks, we applied the *Infomap* community detection algorithm to three well-known datasets: *karate_club*, *dolphins*, and *american_football*. *Infomap* is based on the idea of compressing the description length of a random walk on the network and has proven to be highly effective in detecting meaningful communities.

To visualize the block structure induced by the detected communities, we constructed adjacency matrices where nodes are reordered based on community membership. The resulting blockmodel representations are shown in Figure 5. Each block along the diagonal corresponds to a densely connected community, while sparse off-diagonal regions indicate fewer inter-community connections.

The visualization reveals strong internal structure in all three networks, particularly evident in the *american_football* network, where clear modularity reflects the team-based organization. The *karate_club* network displays a few large, cohesive groups, while the *dolphins* network exhibits a more fragmented community pattern.

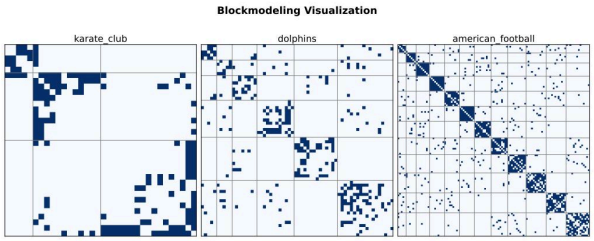

Fig. 5. Blockmodel visualization of the *karate_club*, *dolphins*, and *american_football* networks based on Infomap community detection.

4. Conclusion

In this handout, we have explored the theoretical foundations and practical considerations of network visualization, emphasizing its role in transforming complex graph data into interpretable visual structures. We examined key layout algorithms—Eades' spring-embedded model, Fruchterman-Reingold's force-directed layout, and Kamada-Kawai's energy-based approach—and discussed their respective advantages, limitations, and suitability for different network sizes and types. Additionally, we highlighted the importance of visual principles such as symmetry, minimal edge crossings, and node distribution in producing clear and meaningful visualizations. While layout algorithms provide a powerful means of graph representation, they can sometimes obscure underlying structures such as roles and communities; here, blockmodeling offers an alternative by focusing on the relational patterns within the network. Ultimately, effective network visualization requires balancing aesthetic principles, computational efficiency, and the interpretability of the resulting layout to uncover insights that might remain hidden in raw data, supporting deeper analysis across a wide range of disciplines.

1. Helen Gibson, Joe Faith, and Paul Vickers. A survey of two-dimensional graph layout techniques for information visualisation. *Information visualization*, 12(3-4):324–357, 2013.
2. Weidong Huang and Maolin Huang. Exploring the relative importance of crossing number and crossing angle. In *Proceedings of the 3rd international symposium on visual information communication*, pages 1–8, 2010.
3. Chris Bennett, Jody Ryall, Leo Spalteholz, and Amy Gooch. The aesthetics of graph visualization. In *CAe*, pages 57–64, 2007.
4. Tomihisa Kamada, Satoru Kawai, et al. An algorithm for drawing general undirected graphs. *Information processing letters*, 31(1):7–15, 1989.
5. Peter Eades. A heuristic for graph drawing. *Congressus numerantium*, 42(11):149–160, 1984.
6. Thomas MJ Fruchterman and Edward M Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.
7. Wayne W Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473, 1977.
8. David Lusseau, Karsten Schneider, Oliver J Boisseau, Patti Haase, Elisabeth Slooten, and Steve M Dawson. The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations: can geographic isolation explain this unique trait? *Behavioral ecology and sociobiology*, 54:396–405, 2003.
9. Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.
10. Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.
11. Francois Lorrain and Harrison C White. Structural equivalence of individuals in social networks. *The Journal of mathematical sociology*, 1(1):49–80, 1971.
12. Gerard Salton. Modern information retrieval. *(No Title)*, 1983.
13. Elizabeth A Leicht, Petter Holme, and Mark EJ Newman. Vertex similarity in networks. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics*, 73(2):026120, 2006.
14. Douglas R White and Karl P Reitz. Graph and semigroup homomorphisms on networks of relations. *Social Networks*, 5(2):193–234, 1983.
15. Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.