Measures of centrality

Last updated: 2018/03/23, 12:08:15

Complex Networks | @networksvox
CSYS/MATH 303, Spring, 2018

Prof. Peter Dodds | @peterdodds

Dept. of Mathematics & Statistics | Vermont Complex Systems Center
Vermont Advanced Computing Core | University of Vermont

PoCS
Principles of Complex Systems
@pocsvox
What's the Story?

These slides are brought to you by:

These slides are also brought to you by:

Special Guest Executive Producer

On Instagram at pratchett_the_cat

PoCS
Principles of Complex Systems
@pocsvox
What's the Story?

The University of Vermont

Outline

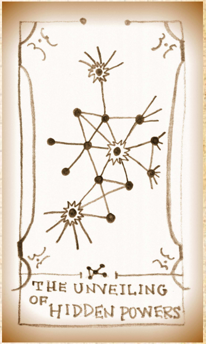
Background

Centrality measures
- Degree centrality
- Closeness centrality
- Betweenness
- Eigenvalue centrality
- Hubs and Authorities

Nutshell

References


THE SUN
THE GOLDEN AGE OF REDUCTIONISM
THE MANIFESTO
THE WELL OF SCALES
THE LAWS OF DECAYING POWER
THE UNEXPECTED TRANSFIGURATION
MINISTRY OF RANDOM WALKS
THE RICH GET RICHER
THE DREAM OF EFFICIENT LANGUAGE
THE LAW OF DIGITS FIRST
THE SURPRISE OF BEING ROBUST YET FRAGILE
THE MISREADING OF THE BOOK OF SAND
THE MANY TAILS OF MANY TAILS
THE MEASURING
THE DATA POOR
THE POURING DATA
THE CHARMS OF THE MANY WRONG PATHS
THE KNOWLEDGE OF DEAD MONKS
THE KNOWLEDGE OF DEAD MONKS
THE EMERGENCE OF STRUCTURE
THE EMERGENCE OF DESTRUCTION
THE EMERGENCE OF THINKING
THE EMERGENCE OF STORIES
THE MECHANICS STATISTICAL
THE THRESHOLDS OF PERCOLATION
THE NETWORKS COMPLEX
THE ELEMENTS OF WEBS
THE BIRDS OF A FEATHER
THE BIRDS OF NOT A FEATHER
THE ARK OF THE RANDOM NETWORK
THE STRANGENESS OF FRIENDS
THE SMALLNESS OF WORLDS
THE THEORY OF SIX DEGREES
THE NETWORKS FREE OF SCALE
THE CONTAGION
THE CONFUSION OF CONTAGIONS
THE DISEASE OF RANDOM MIXING
THE UNENDING PANDEMICS
THE THRESHOLDS OF THE MIND
THE SOCIAL WILD
THE SOCIAL CONTAGION
THE RATIO OF GAIN
THE MECHANICS OF FAME
THE TALK OF FAME
THE STORIES CONTAGIOUS
THE JOHN DORY
THE ADJACENT NARRATIVES
THE STORYTELLERS
THE PATHS OF UNIVERSALITY
THE DEPENDENT PATHS
THE TROUSERS OF BIFURCATING REALITIES
THE UNKNOWN MECHANISM
THE LANDSCAPES OF FORKING PATHS
THE NETWORKS OF BLOOD
THE TREES OF REALITY
THE ORDERS OF STREAMS
THE LAWS OF BRANCHING
THE SCALING OF BROKEN HIERARCHIES
THE BRAID OR THE BRANCH
THE LAW OF OPTIMAL FORKS
THE CHURCH OF QUARTEROLOGY
THE TRUTHICIDE
THE FAMILY OF PANCAKE COWS
THE FAMILY OF SPHERICAL COWS
THE IMPROBABLE PACHYDERM
THE ILL-SCALING OF ALLOMETRIC BODIES
THE SUPPLY MOST FAIR
THE SUPPLY TO THE MOST
THE DARK ART OF GENERATING FUNCTIONS
THE RANDOM SUM OF RANDOMNESS
THE TWO-SIDED WORLDS
THE STORIES AND THE TROPES
THE TRUTH OF DIFFUSION
THE SCRYING FOR THE SHAPES OF THINGS
THE LIMITS OF ORGANIZATIONS
THE OVERCOMING OF CATASTROPHE
THE TROUBLES ILL-FORMED
THE NETWORKS OF MANY ILKS
THE CHAOS OF FETTERED CONTAGION
THE END


THE UNVEILING OF HIDDEN POWERS

How big is my node?

- Basic question: how ‘important’ are specific nodes and edges in a network?
- An important node or edge might:
  1. handle a relatively large amount of the network’s traffic (e.g., cars, information);
  2. bridge two or more distinct groups (e.g., liason, interpreter);
  3. be a source of important ideas, knowledge, or judgments (e.g., supreme court decisions, an employee who ‘knows where everything is’).
- So how do we quantify such a slippery concept as importance?
- We generate ad hoc, reasonable measures, and examine their utility ...

How big is my node?

- Basic question: how ‘important’ are specific nodes and edges in a network?
- An important node or edge might:
  1. handle a relatively large amount of the network’s traffic (e.g., cars, information);
  2. bridge two or more distinct groups (e.g., liason, interpreter);
  3. be a source of important ideas, knowledge, or judgments (e.g., supreme court decisions, an employee who ‘knows where everything is’).
- So how do we quantify such a slippery concept as importance?
- We generate ad hoc, reasonable measures, and examine their utility ...

How big is my node?

- Basic question: how 'important' are specific nodes and edges in a network?
- An important node or edge might:
  1. handle a relatively large amount of the network's traffic (e.g., cars, information);
  2. bridge two or more distinct groups (e.g., liason, interpreter);
  3. be a source of important ideas, knowledge, or judgments (e.g., supreme court decisions, an employee who 'knows where everything is').
- So how do we quantify such a slippery concept as importance?
- We generate ad hoc, reasonable measures, and examine their utility ...

How big is my node?

- Basic question: how 'important' are specific nodes and edges in a network?
- An important node or edge might:
  1. handle a relatively large amount of the network's traffic (e.g., cars, information);
  2. bridge two or more distinct groups (e.g., liason, interpreter);
  3. be a source of important ideas, knowledge, or judgments (e.g., supreme court decisions, an employee who 'knows where everything is').
- So how do we quantify such a slippery concept as importance?
- We generate ad hoc, reasonable measures, and examine their utility ...

How big is my node?

- Basic question: how 'important' are specific nodes and edges in a network?
- An important node or edge might:
  1. handle a relatively large amount of the network's traffic (e.g., cars, information);
  2. bridge two or more distinct groups (e.g., liason, interpreter);
  3. be a source of important ideas, knowledge, or judgments (e.g., supreme court decisions, an employee who 'knows where everything is').
- So how do we quantify such a slippery concept as importance?
- We generate ad hoc, reasonable measures, and examine their utility ...

How big is my node?

- **Basic question:** how 'important' are specific nodes and edges in a network?
- An important node or edge might:
  1. handle a relatively large amount of the network's traffic (e.g., cars, information);
  2. bridge two or more distinct groups (e.g., liason, interpreter);
  3. be a source of important ideas, knowledge, or judgments (e.g., supreme court decisions, an employee who 'knows where everything is').
- So how do we quantify such a slippery concept as importance?
- We generate ad hoc, reasonable measures, and examine their utility ...

How big is my node?

- Basic question: how 'important' are specific nodes and edges in a network?
- An important node or edge might:
  1. handle a relatively large amount of the network's traffic (e.g., cars, information);
  2. bridge two or more distinct groups (e.g., liason, interpreter);
  3. be a source of important ideas, knowledge, or judgments (e.g., supreme court decisions, an employee who 'knows where everything is').
- So how do we quantify such a slippery concept as importance?
- We generate ad hoc, reasonable measures, and examine their utility ...

Centrality

- One possible reflection of importance is centrality.
- Presumption is that nodes or edges that are (in some sense) in the middle of a network are important for the network's function.
- Idea of centrality comes from social networks literature [?].
- Many flavors of centrality ...
  1. Many are topological and quasi-dynamical;
  2. Some are based on dynamics (e.g., traffic).
- We will define and examine a few ...
- (Later: see centrality useful in identifying communities in networks.)

Centrality

- One possible reflection of importance is centrality.
- Presumption is that nodes or edges that are (in some sense) in the middle of a network are important for the network's function.
- Idea of centrality comes from social networks literature[?].
- Many flavors of centrality ...
  1. Many are topological and quasi-dynamical;
  2. Some are based on dynamics (e.g., traffic).
- We will define and examine a few ...
- (Later: see centrality useful in identifying communities in networks.)

Centrality

- One possible reflection of importance is centrality.
- Presumption is that nodes or edges that are (in some sense) in the middle of a network are important for the network's function.
- Idea of centrality comes from social networks literature [7].
- Many flavors of centrality ...
  1. Many are topological and quasi-dynamical;
  2. Some are based on dynamics (e.g., traffic).
- We will define and examine a few ...
- (Later: see centrality useful in identifying communities in networks.)

Centrality

- One possible reflection of importance is centrality.
- Presumption is that nodes or edges that are (in some sense) in the middle of a network are important for the network's function.
- Idea of centrality comes from social networks literature [7].
- Many flavors of centrality ...
  1. Many are topological and quasi-dynamical;
  2. Some are based on dynamics (e.g., traffic).
- We will define and examine a few ...
- (Later: see centrality useful in identifying communities in networks.)

Centrality

- One possible reflection of importance is centrality.
- Presumption is that nodes or edges that are (in some sense) in the middle of a network are important for the network's function.
- Idea of centrality comes from social networks literature [7].
- Many flavors of centrality ...
  1. Many are topological and quasi-dynamical;
  2. Some are based on dynamics (e.g., traffic).
- We will define and examine a few ...
- (Later: see centrality useful in identifying communities in networks.)

Centrality

- One possible reflection of importance is centrality.
- Presumption is that nodes or edges that are (in some sense) in the middle of a network are important for the network's function.
- Idea of centrality comes from social networks literature [7].
- Many flavors of centrality ...
  1. Many are topological and quasi-dynamical;
  2. Some are based on dynamics (e.g., traffic).
- We will define and examine a few ...
- (Later: see centrality useful in identifying communities in networks.)

Outline

Centrality

Degree centrality

- Naively estimate importance by node degree. [7]
- Doh: assumes linearity
  (If node i has twice as many friends as node j, it's twice as important.)
- Doh: doesn't take in any non-local information.

Centrality

Degree centrality

- Naively estimate importance by node degree. [7]
- Doh: assumes linearity
  (If node i has twice as many friends as node j, it's twice as important.)
- Doh: doesn't take in any non-local information.

Centrality

Degree centrality

- Naively estimate importance by node degree. [7]
- Doh: assumes linearity
  (If node i has twice as many friends as node j, it's twice as important.)
- Doh: doesn't take in any non-local information.

Outline

Closeness centrality

- Idea: Nodes are more central if they can reach other nodes 'easily.'
- Measure average shortest path from a node to all other nodes.
- Define Closeness Centrality for node i as

$$\frac{N-1}{\sum_{j,j\neq i}(\text{shortest distance from } i \text{ to } j)}.$$

- Range is 0 (no friends) to 1 (single hub).
- Unclear what the exact values of this measure tells us because of its ad-hocness.
- General problem with simple centrality measures: what do they exactly mean?
- Perhaps, at least, we obtain an ordering of nodes in terms of 'importance.'

Closeness centrality

- Idea: Nodes are more central if they can reach other nodes 'easily.'
- Measure average shortest path from a node to all other nodes.
- Define Closeness Centrality for node i as

$$\frac{N-1}{\sum_{j,j\neq i}(\text{shortest distance from } i \text{ to } j).}$$

- Range is 0 (no friends) to 1 (single hub).
- Unclear what the exact values of this measure tells us because of its ad-hocness.
- General problem with simple centrality measures: what do they exactly mean?
- Perhaps, at least, we obtain an ordering of nodes in terms of 'importance.'

Closeness centrality

- Idea: Nodes are more central if they can reach other nodes 'easily.'
- Measure average shortest path from a node to all other nodes.
- Define Closeness Centrality for node i as

$$\frac{N-1}{\sum_{j,j\neq i}(\text{shortest distance from } i \text{ to } j)}.$$

- Range is 0 (no friends) to 1 (single hub).
- Unclear what the exact values of this measure tells us because of its ad-hocness.
- General problem with simple centrality measures: what do they exactly mean?
- Perhaps, at least, we obtain an ordering of nodes in terms of 'importance.'

Closeness centrality

- Idea: Nodes are more central if they can reach other nodes 'easily.'
- Measure average shortest path from a node to all other nodes.
- Define Closeness Centrality for node i as

$$\frac{N-1}{\sum_{j,j\neq i}(\text{shortest distance from } i \text{ to } j).}$$

- Range is 0 (no friends) to 1 (single hub).
- Unclear what the exact values of this measure tells us because of its ad-hocness.
- General problem with simple centrality measures: what do they exactly mean?
- Perhaps, at least, we obtain an ordering of nodes in terms of 'importance.'

Closeness centrality

- Idea: Nodes are more central if they can reach other nodes 'easily.'
- Measure average shortest path from a node to all other nodes.
- Define Closeness Centrality for node i as

$$\frac{N-1}{\sum_{j,j\neq i}(\text{shortest distance from } i \text{ to } j).}$$

- Range is 0 (no friends) to 1 (single hub).
- Unclear what the exact values of this measure tells us because of its ad-hocness.
- General problem with simple centrality measures: what do they exactly mean?
- Perhaps, at least, we obtain an ordering of nodes in terms of 'importance.'

Closeness centrality

- Idea: Nodes are more central if they can reach other nodes 'easily.'
- Measure average shortest path from a node to all other nodes.
- Define Closeness Centrality for node i as

$$\frac{N-1}{\sum_{j,j\neq i}(\text{shortest distance from } i \text{ to } j)}.$$

- Range is 0 (no friends) to 1 (single hub).
- Unclear what the exact values of this measure tells us because of its ad-hocness.
- General problem with simple centrality measures: what do they exactly mean?
- Perhaps, at least, we obtain an ordering of nodes in terms of 'importance.'

Closeness centrality

- Idea: Nodes are more central if they can reach other nodes 'easily.'
- Measure average shortest path from a node to all other nodes.
- Define Closeness Centrality for node i as

$$\frac{N-1}{\sum_{j,j\neq i}(\text{shortest distance from } i \text{ to } j).}$$

- Range is 0 (no friends) to 1 (single hub).
- Unclear what the exact values of this measure tells us because of its ad-hocness.
- General problem with simple centrality measures: what do they exactly mean?
- Perhaps, at least, we obtain an ordering of nodes in terms of 'importance.'

Outline

Betweenness centrality

- **Betweenness centrality** is based on coherence of shortest paths in a network.
- Idea: If the quickest way between any two nodes on a network disproportionately involves certain nodes, then they are 'important' in terms of global cohesion.
- For each node i, count how many shortest paths pass through i.
- In the case of ties, divide counts between paths.
- Call frequency of shortest paths passing through node i the betweenness of i, B_i.
- Note: Exclude shortest paths between i and other nodes.
- Note: works for weighted and unweighted networks.

Betweenness centrality

- Betweenness centrality is based on coherence of shortest paths in a network.
- Idea: If the quickest way between any two nodes on a network disproportionately involves certain nodes, then they are 'important' in terms of global cohesion.
- For each node i, count how many shortest paths pass through i.
- In the case of ties, divide counts between paths.
- Call frequency of shortest paths passing through node i the betweenness of i, B_i.
- Note: Exclude shortest paths between i and other nodes.
- Note: works for weighted and unweighted networks.

Betweenness centrality

- Betweenness centrality is based on coherence of shortest paths in a network.
- Idea: If the quickest way between any two nodes on a network disproportionately involves certain nodes, then they are 'important' in terms of global cohesion.
- For each node i, count how many shortest paths pass through i.
- In the case of ties, divide counts between paths.
- Call frequency of shortest paths passing through node i the betweenness of i, B_i.
- Note: Exclude shortest paths between i and other nodes.
- Note: works for weighted and unweighted networks.

Betweenness centrality

- Betweenness centrality is based on coherence of shortest paths in a network.
- Idea: If the quickest way between any two nodes on a network disproportionately involves certain nodes, then they are ‘important’ in terms of global cohesion.
- For each node i, count how many shortest paths pass through i.
- In the case of ties, divide counts between paths.
- Call frequency of shortest paths passing through node i the betweenness of i, B_i.
- Note: Exclude shortest paths between i and other nodes.
- Note: works for weighted and unweighted networks.

Betweenness centrality

- Betweenness centrality is based on coherence of shortest paths in a network.
- Idea: If the quickest way between any two nodes on a network disproportionately involves certain nodes, then they are 'important' in terms of global cohesion.
- For each node i, count how many shortest paths pass through i.
- In the case of ties, divide counts between paths.
- Call frequency of shortest paths passing through node i the betweenness of i, B_i.
- Note: Exclude shortest paths between i and other nodes.
- Note: works for weighted and unweighted networks.

Betweenness centrality

- Betweenness centrality is based on coherence of shortest paths in a network.
- Idea: If the quickest way between any two nodes on a network disproportionately involves certain nodes, then they are ‘important’ in terms of global cohesion.
- For each node i, count how many shortest paths pass through i.
- In the case of ties, divide counts between paths.
- Call frequency of shortest paths passing through node i the betweenness of i, B_i.
- Note: Exclude shortest paths between i and other nodes.
- Note: works for weighted and unweighted networks.

Betweenness centrality

- Betweenness centrality is based on coherence of shortest paths in a network.
- Idea: If the quickest way between any two nodes on a network disproportionately involves certain nodes, then they are 'important' in terms of global cohesion.
- For each node i, count how many shortest paths pass through i.
- In the case of ties, divide counts between paths.
- Call frequency of shortest paths passing through node i the betweenness of i, B_i.
- Note: Exclude shortest paths between i and other nodes.
- Note: works for weighted and unweighted networks.

- Consider a network with N nodes and m edges (possibly weighted).
- Computational goal: Find $\binom{N}{2}$ shortest paths between all pairs of nodes.
- Traditionally use Floyd-Warshall algorithm.
- Computation time grows as $O(N^3)$.
- See also:
  1. Dijkstra's algorithm for finding shortest path between two specific nodes,
  2. and Johnson's algorithm which outperforms Floyd-Warshall for sparse networks: $O(mN + N^2 \log N)$.
- Newman (2001) [4, 5] and Brandes (2001) [1] independently derive equally fast algorithms that also compute betweenness.
- Computation times grow as:
  1. $O(mN)$ for unweighted graphs;
  2. and $O(mN + N^2 \log N)$ for weighted graphs.

- Consider a network with N nodes and m edges (possibly weighted).
- Computational goal: Find $\binom{N}{2}$ shortest paths between all pairs of nodes.
- Traditionally use Floyd-Warshall algorithm.
- Computation time grows as $O(N^3)$.
- See also:
  1. Dijkstra's algorithm for finding shortest path between two specific nodes,
  2. and Johnson's algorithm which outperforms Floyd-Warshall for sparse networks: $O(mN + N^2 \log N)$.
- Newman (2001) [4, 5] and Brandes (2001) [1] independently derive equally fast algorithms that also compute betweenness.
- Computation times grow as:
  1. $O(mN)$ for unweighted graphs;
  2. and $O(mN + N^2 \log N)$ for weighted graphs.

- Consider a network with N nodes and m edges (possibly weighted).
- Computational goal: Find $\binom{N}{2}$ shortest paths between all pairs of nodes.
- Traditionally use Floyd-Warshall algorithm.
- Computation time grows as $O(N^3)$.
- See also:
  1. Dijkstra's algorithm for finding shortest path between two specific nodes,
  2. and Johnson's algorithm which outperforms Floyd-Warshall for sparse networks: $O(mN + N^2 \log N)$.
- Newman (2001) [4, 5] and Brandes (2001) [1] independently derive equally fast algorithms that also compute betweenness.
- Computation times grow as:
  1. $O(mN)$ for unweighted graphs;
  2. and $O(mN + N^2 \log N)$ for weighted graphs.

- Consider a network with N nodes and m edges (possibly weighted).
- Computational goal: Find $\binom{N}{2}$ shortest paths between all pairs of nodes.
- Traditionally use Floyd-Warshall algorithm.
- Computation time grows as $O(N^3)$.
- See also:
  1. Dijkstra's algorithm for finding shortest path between two specific nodes,
  2. and Johnson's algorithm which outperforms Floyd-Warshall for sparse networks: $O(mN + N^2 \log N)$.
- Newman (2001)[4, 5] and Brandes (2001)[1] independently derive equally fast algorithms that also compute betweenness.
- Computation times grow as:
  1. $O(mN)$ for unweighted graphs;
  2. and $O(mN + N^2 \log N)$ for weighted graphs.

- Consider a network with N nodes and m edges (possibly weighted).
- Computational goal: Find $\binom{N}{2}$ shortest paths between all pairs of nodes.
- Traditionally use Floyd-Warshall algorithm.
- Computation time grows as $O(N^3)$.
- See also:
  1. Dijkstra's algorithm for finding shortest path between two specific nodes,
  2. and Johnson's algorithm which outperforms Floyd-Warshall for sparse networks: $O(mN + N^2 \log N)$.
- Newman (2001) [4, 5] and Brandes (2001) [1] independently derive equally fast algorithms that also compute betweenness.
- Computation times grow as:
  1. $O(mN)$ for unweighted graphs;
  2. and $O(mN + N^2 \log N)$ for weighted graphs.

- Consider a network with N nodes and m edges (possibly weighted).
- Computational goal: Find $\binom{N}{2}$ shortest paths between all pairs of nodes.
- Traditionally use Floyd-Warshall algorithm.
- Computation time grows as $O(N^3)$.
- See also:
  1. Dijkstra's algorithm for finding shortest path between two specific nodes,
  2. and Johnson's algorithm which outperforms Floyd-Warshall for sparse networks: $O(mN + N^2 \log N)$.
- Newman (2001) [4, 5] and Brandes (2001) [1] independently derive equally fast algorithms that also compute betweenness.
- Computation times grow as:
  1. $O(mN)$ for unweighted graphs;
  2. and $O(mN + N^2 \log N)$ for weighted graphs.

- Consider a network with N nodes and m edges (possibly weighted).
- Computational goal: Find $\binom{N}{2}$ shortest paths between all pairs of nodes.
- Traditionally use Floyd-Warshall algorithm.
- Computation time grows as $O(N^3)$.
- See also:
  1. Dijkstra's algorithm for finding shortest path between two specific nodes,
  2. and Johnson's algorithm which outperforms Floyd-Warshall for sparse networks: $O(mN + N^2 \log N)$.
- Newman (2001) [4, 5] and Brandes (2001) [1] independently derive equally fast algorithms that also compute betweenness.
- Computation times grow as:
  1. $O(mN)$ for unweighted graphs;
  2. and $O(mN + N^2 \log N)$ for weighted graphs.

- Consider a network with N nodes and m edges (possibly weighted).
- Computational goal: Find $\binom{N}{2}$ shortest paths between all pairs of nodes.
- Traditionally use Floyd-Warshall algorithm.
- Computation time grows as $O(N^3)$.
- See also:
  1. Dijkstra's algorithm for finding shortest path between two specific nodes,
  2. and Johnson's algorithm which outperforms Floyd-Warshall for sparse networks: $O(mN + N^2 \log N)$.
- Newman (2001) [4, 5] and Brandes (2001) [1] independently derive equally fast algorithms that also compute betweenness.
- Computation times grow as:
  1. $O(mN)$ for unweighted graphs;
  2. and $O(mN + N^2 \log N)$ for weighted graphs.

- Consider a network with N nodes and m edges (possibly weighted).
- Computational goal: Find $\binom{N}{2}$ shortest paths between all pairs of nodes.
- Traditionally use Floyd-Warshall algorithm.
- Computation time grows as $O(N^3)$.
- See also:
    1. Dijkstra's algorithm for finding shortest path between two specific nodes,
    2. and Johnson's algorithm which outperforms Floyd-Warshall for sparse networks: $O(mN + N^2 \log N)$.
- Newman (2001) [4, 5] and Brandes (2001) [1] independently derive equally fast algorithms that also compute betweenness.
- Computation times grow as:
    1. $O(mN)$ for unweighted graphs;
    2. and $O(mN + N^2 \log N)$ for weighted graphs.

- Consider a network with N nodes and m edges (possibly weighted).
- Computational goal: Find $\binom{N}{2}$ shortest paths between all pairs of nodes.
- Traditionally use Floyd-Warshall algorithm.
- Computation time grows as $O(N^3)$.
- See also:
  1. Dijkstra's algorithm for finding shortest path between two specific nodes,
  2. and Johnson's algorithm which outperforms Floyd-Warshall for sparse networks: $O(mN + N^2 \log N)$.
- Newman (2001) [4, 5] and Brandes (2001) [1] independently derive equally fast algorithms that also compute betweenness.
- Computation times grow as:
  1. $O(mN)$ for unweighted graphs;
  2. and $O(mN + N^2 \log N)$ for weighted graphs.

Shortest path between node i and all others:

- Consider unweighted networks.
- Use breadth-first search:
  1. Start at node i, giving it a distance $d = 0$ from itself.
  2. Create a list of all of i's neighbors and label them as being at a distance $d = 1$.
  3. Go through list of most recently visited nodes and find all of their neighbors.
  4. Exclude any nodes already assigned a distance.
  5. Increment distance d by 1.
  6. Label newly reached nodes as being at distance d.
  7. Repeat steps 3 through 6 until all nodes are visited.
- Record which nodes link to which nodes moving out from i (former are 'predecessors' with respect to i's shortest path structure).
- Runs in $O(m)$ time and gives $N - 1$ shortest paths.
- Find all shortest paths in $O(mN)$ time

Shortest path between node i and all others:

- Consider unweighted networks.
- Use breadth-first search:
  1. Start at node i, giving i a distance $d = 0$ from itself.
  2. Create a list of all of i's neighbors and label them as being at a distance $d = 1$.
  3. Go through list of most recently visited nodes and find all of their neighbors.
  4. Exclude any nodes already assigned a distance.
  5. Increment distance d by 1.
  6. Label newly reached nodes as being at distance d.
  7. Repeat steps 3 through 6 until all nodes are visited.
- Record which nodes link to which nodes moving out from i (former are 'predecessors' with respect to i's shortest path structure).
- Runs in $O(m)$ time and gives $N - 1$ shortest paths.
- Find all shortest paths in $O(mN)$ time

Shortest path between node i and all others:

- Consider unweighted networks.
- Use breadth-first search:
  1. Start at node i, giving it a distance $d = 0$ from itself.
  2. Create a list of all of i's neighbors and label them being at a distance $d = 1$.
  3. Go through list of most recently visited nodes and find all of their neighbors.
  4. Exclude any nodes already assigned a distance.
  5. Increment distance d by 1.
  6. Label newly reached nodes as being at distance d.
  7. Repeat steps 3 through 6 until all nodes are visited.
- Record which nodes link to which nodes moving out from i (former are 'predecessors' with respect to i's shortest path structure).
- Runs in $O(m)$ time and gives $N-1$ shortest paths.
- Find all shortest paths in $O(mN)$ time

Shortest path between node i and all others:

- Consider unweighted networks.
- Use breadth-first search:
  1. Start at node i, giving it a distance $d = 0$ from itself.
  2. Create a list of all of i's neighbors and label them being at a distance $d = 1$.
  3. Go through list of most recently visited nodes and find all of their neighbors.
  4. Exclude any nodes already assigned a distance.
  5. Increment distance d by 1.
  6. Label newly reached nodes as being at distance d.
  7. Repeat steps 3 through 6 until all nodes are visited.
- Record which nodes link to which nodes moving out from i (former are 'predecessors' with respect to i's shortest path structure).
- Runs in $O(m)$ time and gives $N - 1$ shortest paths.
- Find all shortest paths in $O(mN)$ time

Shortest path between node i and all others:

- Consider unweighted networks.
- Use breadth-first search:
  1. Start at node i, giving it a distance $d = 0$ from itself.
  2. Create a list of all of i's neighbors and label them being at a distance $d = 1$.
  3. Go through list of most recently visited nodes and find all of their neighbors.
  4. Exclude any nodes already assigned a distance.
  5. Increment distance d by 1.
  6. Label newly reached nodes as being at distance d.
  7. Repeat steps 3 through 6 until all nodes are visited.
- Record which nodes link to which nodes moving out from i (former are 'predecessors' with respect to i's shortest path structure).
- Runs in $O(m)$ time and gives $N - 1$ shortest paths.
- Find all shortest paths in $O(mN)$ time

Shortest path between node i and all others:

- Consider unweighted networks.
- Use breadth-first search:
  1. Start at node i, giving it a distance $d = 0$ from itself.
  2. Create a list of all of i's neighbors and label them being at a distance $d = 1$.
  3. Go through list of most recently visited nodes and find all of their neighbors.
  4. Exclude any nodes already assigned a distance.
  5. Increment distance d by 1.
  6. Label newly reached nodes as being at distance d.
  7. Repeat steps 3 through 6 until all nodes are visited.
- Record which nodes link to which nodes moving out from i (former are 'predecessors' with respect to i's shortest path structure).
- Runs in $O(m)$ time and gives $N - 1$ shortest paths.
- Find all shortest paths in $O(mN)$ time

Shortest path between node i and all others:

- Consider unweighted networks.
- Use breadth-first search:
  1. Start at node i, giving it a distance $d = 0$ from itself.
  2. Create a list of all of i's neighbors and label them being at a distance $d = 1$.
  3. Go through list of most recently visited nodes and find all of their neighbors.
  4. Exclude any nodes already assigned a distance.
  5. Increment distance d by 1.
  6. Label newly reached nodes as being at distance d.
  7. Repeat steps 3 through 6 until all nodes are visited.
- Record which nodes link to which nodes moving out from i (former are 'predecessors' with respect to i's shortest path structure).
- Runs in $O(m)$ time and gives $N - 1$ shortest paths.
- Find all shortest paths in $O(mN)$ time

Shortest path between node i and all others:

- Consider unweighted networks.
- Use breadth-first search:
  1. Start at node i, giving it a distance $d = 0$ from itself.
  2. Create a list of all of i's neighbors and label them being at a distance $d = 1$.
  3. Go through list of most recently visited nodes and find all of their neighbors.
  4. Exclude any nodes already assigned a distance.
  5. Increment distance d by 1.
  6. Label newly reached nodes as being at distance d.
  7. Repeat steps 3 through 6 until all nodes are visited.
- Record which nodes link to which nodes moving out from i (former are 'predecessors' with respect to i's shortest path structure).
- Runs in $O(m)$ time and gives $N-1$ shortest paths.
- Find all shortest paths in $O(mN)$ time

Shortest path between node i and all others:

- Consider unweighted networks.
- Use breadth-first search:
  1. Start at node i, giving it a distance $d = 0$ from itself.
  2. Create a list of all of i's neighbors and label them being at a distance $d = 1$.
  3. Go through list of most recently visited nodes and find all of their neighbors.
  4. Exclude any nodes already assigned a distance.
  5. Increment distance d by 1.
  6. Label newly reached nodes as being at distance d.
  7. Repeat steps 3 through 6 until all nodes are visited.
- Record which nodes link to which nodes moving out from i (former are 'predecessors' with respect to i's shortest path structure).
- Runs in $O(m)$ time and gives $N - 1$ shortest paths.
- Find all shortest paths in $O(mN)$ time

Shortest path between node i and all others:

- Consider unweighted networks.
- Use breadth-first search:
  1. Start at node i, giving it a distance $d = 0$ from itself.
  2. Create a list of all of i's neighbors and label them being at a distance $d = 1$.
  3. Go through list of most recently visited nodes and find all of their neighbors.
  4. Exclude any nodes already assigned a distance.
  5. Increment distance d by 1.
  6. Label newly reached nodes as being at distance d.
  7. Repeat steps 3 through 6 until all nodes are visited.
- Record which nodes link to which nodes moving out from i (former are 'predecessors' with respect to i's shortest path structure).
- Runs in $O(m)$ time and gives $N - 1$ shortest paths.
- Find all shortest paths in $O(mN)$ time

Shortest path between node i and all others:

- Consider unweighted networks.
- Use breadth-first search:
  1. Start at node i, giving it a distance $d = 0$ from itself.
  2. Create a list of all of i's neighbors and label them being at a distance $d = 1$.
  3. Go through list of most recently visited nodes and find all of their neighbors.
  4. Exclude any nodes already assigned a distance.
  5. Increment distance d by 1.
  6. Label newly reached nodes as being at distance d.
  7. Repeat steps 3 through 6 until all nodes are visited.
- Record which nodes link to which nodes moving out from i (former are 'predecessors' with respect to i's shortest path structure).
- Runs in $O(m)$ time and gives $N - 1$ shortest paths.
- Find all shortest paths in $O(mN)$ time

Shortest path between node i and all others:

- Consider unweighted networks.
- Use breadth-first search:
  1. Start at node i, giving it a distance $d = 0$ from itself.
  2. Create a list of all of i's neighbors and label them being at a distance $d = 1$.
  3. Go through list of most recently visited nodes and find all of their neighbors.
  4. Exclude any nodes already assigned a distance.
  5. Increment distance d by 1.
  6. Label newly reached nodes as being at distance d.
  7. Repeat steps 3 through 6 until all nodes are visited.
- Record which nodes link to which nodes moving out from i (former are 'predecessors' with respect to i's shortest path structure).
- Runs in $O(m)$ time and gives $N - 1$ shortest paths.
- Find all shortest paths in $O(mN)$ time

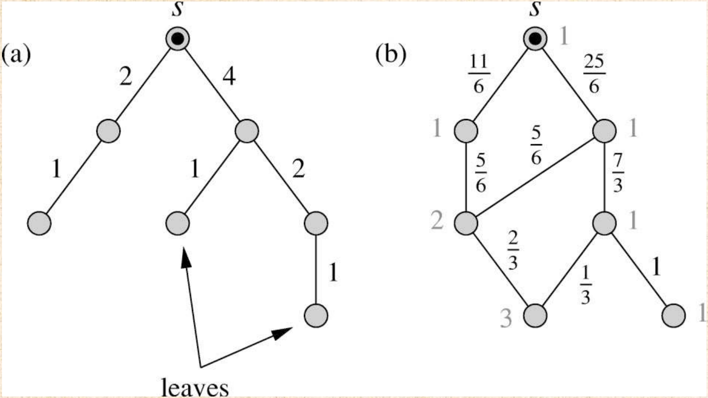
Shortest path between node i and all others:

- Consider unweighted networks.
- Use breadth-first search:
  1. Start at node i, giving it a distance $d = 0$ from itself.
  2. Create a list of all of i's neighbors and label them being at a distance $d = 1$.
  3. Go through list of most recently visited nodes and find all of their neighbors.
  4. Exclude any nodes already assigned a distance.
  5. Increment distance d by 1.
  6. Label newly reached nodes as being at distance d.
  7. Repeat steps 3 through 6 until all nodes are visited.
- Record which nodes link to which nodes moving out from i (former are 'predecessors' with respect to i's shortest path structure).
- Runs in $O(m)$ time and gives $N - 1$ shortest paths.
- Find all shortest paths in $O(mN)$ time

Shortest path between node i and all others:

- Consider unweighted networks.
- Use breadth-first search:
  1. Start at node i, giving it a distance $d = 0$ from itself.
  2. Create a list of all of i's neighbors and label them being at a distance $d = 1$.
  3. Go through list of most recently visited nodes and find all of their neighbors.
  4. Exclude any nodes already assigned a distance.
  5. Increment distance d by 1.
  6. Label newly reached nodes as being at distance d.
  7. Repeat steps 3 through 6 until all nodes are visited.
- Record which nodes link to which nodes moving out from i (former are 'predecessors' with respect to i's shortest path structure).
- Runs in $O(m)$ time and gives $N - 1$ shortest paths.
- Find all shortest paths in $O(mN)$ time

Newman's Betweenness algorithm: [4]



Newman's Betweenness algorithm: [4]

1. Set all nodes to have a value $c_{ij} = 0$, $j = 1, ...$ (c for count).
2. Select one node i and find shortest paths to all other $N - 1$ nodes using breadth-first search.
3. Record # equal shortest paths reaching each node.
4. Move through nodes according to their distance from i, starting with the furthest.
5. Travel back towards i from each starting node j, along shortest path(s), adding 1 to every value of $c_{i\ell}$ at each node ℓ along the way.
6. Whenever more than one possibility exists, apportion according to total number of short paths coming through predecessors.
7. Exclude starting node j and i from increment.
8. Repeat steps 2–8 for every node i and obtain betweenness as $B_j = \sum_{i=1}^{N} c_{ij}$.

Newman's Betweenness algorithm: [4]

1. Set all nodes to have a value $c_{ij} = 0$, $j = 1, \dots$ (c for count).
2. Select one node i and find shortest paths to all other $N - 1$ nodes using breadth-first search.
3. Record # equal shortest paths reaching each node.
4. Move through nodes according to their distance from i, starting with the furthest.
5. Travel back towards i from each starting node j, along shortest path(s), adding 1 to every value of $c_{i\ell}$ at each node ℓ along the way.
6. Whenever more than one possibility exists, apportion according to total number of short paths coming through predecessors.
7. Exclude starting node j and i from increment.
8. Repeat steps 2–8 for every node i and obtain betweenness as $B_j = \sum_{i=1}^{N} c_{ij}$.

Newman's Betweenness algorithm: [4]

1. Set all nodes to have a value $c_{ij} = 0$, $j = 1, \ldots$ (c for count).
2. Select one node i and find shortest paths to all other $N - 1$ nodes using breadth-first search.
3. Record # equal shortest paths reaching each node.
4. Move through nodes according to their distance from i, starting with the furthest.
5. Travel back towards i from each starting node j, along shortest path(s), adding 1 to every value of $c_{i\ell}$ at each node ℓ along the way.
6. Whenever more than one possibility exists, apportion according to total number of short paths coming through predecessors.
7. Exclude starting node j and i from increment.
8. Repeat steps 2–8 for every node i and obtain betweenness as $B_j = \sum_{i=1}^{N} c_{ij}$.

Newman's Betweenness algorithm: [4]

1. Set all nodes to have a value $c_{ij} = 0$, $j = 1, \ldots$ (c for count).
2. Select one node i and find shortest paths to all other $N - 1$ nodes using breadth-first search.
3. Record # equal shortest paths reaching each node.
4. Move through nodes according to their distance from i, starting with the furthest.
5. Travel back towards i from each starting node j, along shortest path(s), adding 1 to every value of $c_{i\ell}$ at each node ℓ along the way.
6. Whenever more than one possibility exists, apportion according to total number of short paths coming through predecessors.
7. Exclude starting node j and i from increment.
8. Repeat steps 2–8 for every node i and obtain betweenness as $B_j = \sum_{i=1}^{N} c_{ij}$.

Newman's Betweenness algorithm: [4]

1. Set all nodes to have a value $c_{ij} = 0$, $j = 1, \ldots$ (c for count).
2. Select one node i and find shortest paths to all other $N - 1$ nodes using breadth-first search.
3. Record # equal shortest paths reaching each node.
4. Move through nodes according to their distance from i, starting with the furthest.
5. Travel back towards i from each starting node j, along shortest path(s), adding 1 to every value of $c_{i\ell}$ at each node ℓ along the way.
6. Whenever more than one possibility exists, apportion according to total number of short paths coming through predecessors.
7. Exclude starting node j and i from increment.
8. Repeat steps 2–8 for every node i and obtain betweenness as $B_j = \sum_{i=1}^{N} c_{ij}$.

Newman's Betweenness algorithm: [4]

1. Set all nodes to have a value $c_{ij} = 0$, $j = 1, ...$ (c for count).
2. Select one node i and find shortest paths to all other $N - 1$ nodes using breadth-first search.
3. Record # equal shortest paths reaching each node.
4. Move through nodes according to their distance from i, starting with the furthest.
5. Travel back towards i from each starting node j, along shortest path(s), adding 1 to every value of $c_{i\ell}$ at each node ℓ along the way.
6. Whenever more than one possibility exists, apportion according to total number of short paths coming through predecessors.
7. Exclude starting node j and i from increment.
8. Repeat steps 2–8 for every node i and obtain betweenness as $B_j = \sum_{i=1}^{N} c_{ij}$.

Newman's Betweenness algorithm: [4]

1. Set all nodes to have a value $c_{ij} = 0$, $j = 1, \ldots$ (c for count).
2. Select one node i and find shortest paths to all other $N - 1$ nodes using breadth-first search.
3. Record # equal shortest paths reaching each node.
4. Move through nodes according to their distance from i, starting with the furthest.
5. Travel back towards i from each starting node j, along shortest path(s), adding 1 to every value of $c_{i\ell}$ at each node ℓ along the way.
6. Whenever more than one possibility exists, apportion according to total number of short paths coming through predecessors.
7. Exclude starting node j and i from increment.
8. Repeat steps 2–8 for every node i and obtain betweenness as $B_j = \sum_{i=1}^{N} c_{ij}$.

Newman's Betweenness algorithm: [4]

1. Set all nodes to have a value $c_{ij} = 0$, $j = 1, ...$ (c for count).
2. Select one node i and find shortest paths to all other $N-1$ nodes using breadth-first search.
3. Record # equal shortest paths reaching each node.
4. Move through nodes according to their distance from i, starting with the furthest.
5. Travel back towards i from each starting node j, along shortest path(s), adding 1 to every value of $c_{i\ell}$ at each node ℓ along the way.
6. Whenever more than one possibility exists, apportion according to total number of short paths coming through predecessors.
7. Exclude starting node j and i from increment.
8. Repeat steps 2–8 for every node i and obtain betweenness as $B_j = \sum_{i=1}^{N} c_{ij}$.

Newman's Betweenness algorithm: [4]

- For a pure tree network, c_{ij} is the number of nodes beyond j from i's vantage point.
- Same algorithm for computing drainage area in river networks (with 1 added across the board).
- For edge betweenness, use exact same algorithm but now
  1. j indexes edges,
  2. and we add one to each edge as we traverse it.
- For both algorithms, computation time grows as

$$O(mN).$$

Newman's Betweenness algorithm: [4]

- For a pure tree network, c_{ij} is the number of nodes beyond j from i's vantage point.
- Same algorithm for computing drainage area in river networks (with 1 added across the board).
- For edge betweenness, use exact same algorithm but now
  1. j indexes edges,
  2. and we add one to each edge as we traverse it.
- For both algorithms, computation time grows as

$$O(mN).$$

Newman's Betweenness algorithm: [4]

- For a pure tree network, c_{ij} is the number of nodes beyond j from i's vantage point.
- Same algorithm for computing drainage area in river networks (with 1 added across the board).
- For edge betweenness, use exact same algorithm but now
  1. j indexes edges,
  2. and we add one to each edge as we traverse it.
- For both algorithms, computation time grows as

$$O(mN).$$

Newman's Betweenness algorithm: [4]

- For a pure tree network, c_{ij} is the number of nodes beyond j from i's vantage point.
- Same algorithm for computing drainage area in river networks (with 1 added across the board).
- For edge betweenness, use exact same algorithm but now
  1. j indexes edges,
  2. and we add one to each edge as we traverse it.
- For both algorithms, computation time grows as

$$O(mN).$$

Newman's Betweenness algorithm: [4]

- For a pure tree network, c_{ij} is the number of nodes beyond j from i's vantage point.
- Same algorithm for computing drainage area in river networks (with 1 added across the board).
- For edge betweenness, use exact same algorithm but now
  1. j indexes edges,
  2. and we add one to each edge as we traverse it.
- For both algorithms, computation time grows as

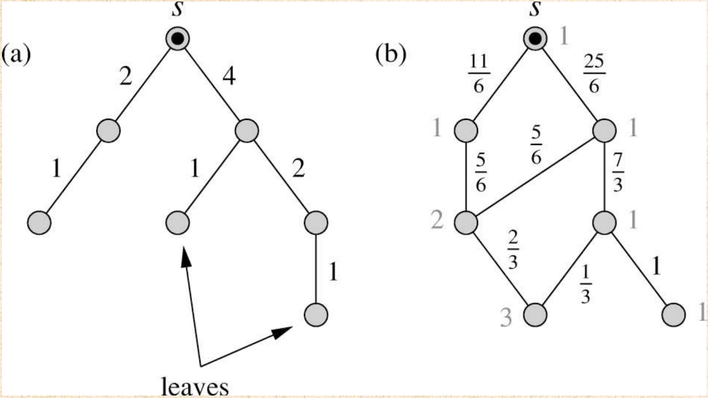
$$O(mN).$$

Newman's Betweenness algorithm: [4]

- For a pure tree network, c_{ij} is the number of nodes beyond j from i's vantage point.
- Same algorithm for computing drainage area in river networks (with 1 added across the board).
- For edge betweenness, use exact same algorithm but now
  1. j indexes edges,
  2. and we add one to each edge as we traverse it.
- For both algorithms, computation time grows as

$$O(mN).$$

Newman's Betweenness algorithm:[4]



Outline

Important nodes have important friends:

- Define x_i as the 'importance' of node i.
- Idea: x_i depends (somehow) on x_j if j is a neighbor of i.
- Recursive: importance is transmitted through a network.
- Simplest possibility is a linear combination:

$$x_i \propto \sum_j a_{ji} x_j$$

- Assume further that constant of proportionality, c, is independent of i.
- Above gives $\vec{x} = c\mathbf{A}^{\mathsf{T}}\vec{x}$ or $\boxed{\mathbf{A}^{\mathsf{T}}\vec{x} = c^{-1}\vec{x} = \lambda\vec{x}}$.
- Eigenvalue equation based on adjacency matrix ...
- Note: Lots of despair over size of the largest eigenvalue.[7] Lose sight of original assumption's non-physicality.

PoCS
Principles of Complex Systems
@pocsvox
What's the Story?

The University of Vermont

Important nodes have important friends:

- Define x_i as the 'importance' of node i.
- Idea: x_i depends (somehow) on x_j
  if j is a neighbor of i.
- Recursive: importance is transmitted through a network.
- Simplest possibility is a linear combination:

$$x_i \propto \sum_j a_{ji} x_j$$

- Assume further that constant of proportionality, c, is independent of i.
- Above gives $\vec{x} = c\mathbf{A}^{\mathsf{T}}\vec{x}$ or $\boxed{\mathbf{A}^{\mathsf{T}}\vec{x} = c^{-1}\vec{x} = \lambda\vec{x}}$.
- Eigenvalue equation based on adjacency matrix ...
- Note: Lots of despair over size of the largest eigenvalue.[7] Lose sight of original assumption's non-physicality.

Important nodes have important friends:

- Define x_i as the 'importance' of node i.
- Idea: x_i depends (somehow) on x_j
  if j is a neighbor of i.
- Recursive: importance is transmitted through a network.
- Simplest possibility is a linear combination:

$$x_i \propto \sum_j a_{ji} x_j$$

- Assume further that constant of proportionality, c, is independent of i.
- Above gives $\vec{x} = c\mathbf{A}^{\mathsf{T}}\vec{x}$ or $\boxed{\mathbf{A}^{\mathsf{T}}\vec{x} = c^{-1}\vec{x} = \lambda\vec{x}}$.
- Eigenvalue equation based on adjacency matrix ...
- Note: Lots of despair over size of the largest eigenvalue.[7] Lose sight of original assumption's non-physicality.

Important nodes have important friends:

- Define x_i as the 'importance' of node i.
- Idea: x_i depends (somehow) on x_j
  if j is a neighbor of i.
- Recursive: importance is transmitted through a network.
- Simplest possibility is a linear combination:

$$x_i \propto \sum_j a_{ji} x_j$$

- Assume further that constant of proportionality, c, is independent of i.
- Above gives $\vec{x} = c\mathbf{A}^{\mathsf{T}}\vec{x}$ or $\boxed{\mathbf{A}^{\mathsf{T}}\vec{x} = c^{-1}\vec{x} = \lambda\vec{x}}$.
- Eigenvalue equation based on adjacency matrix ...
- Note: Lots of despair over size of the largest eigenvalue.[7] Lose sight of original assumption's non-physicality.

Important nodes have important friends:

- Define x_i as the 'importance' of node i.
- Idea: x_i depends (somehow) on x_j
  if j is a neighbor of i.
- Recursive: importance is transmitted through a network.
- Simplest possibility is a linear combination:

$$x_i \propto \sum_j a_{ji} x_j$$

- Assume further that constant of proportionality, c, is independent of i.
- Above gives $\vec{x} = c\mathbf{A}^{\mathsf{T}}\vec{x}$ or $\boxed{\mathbf{A}^{\mathsf{T}}\vec{x} = c^{-1}\vec{x} = \lambda\vec{x}}$.
- Eigenvalue equation based on adjacency matrix ...
- Note: Lots of despair over size of the largest eigenvalue. [7] Lose sight of original assumption's non-physicality.

Important nodes have important friends:

- Define x_i as the 'importance' of node i.
- Idea: x_i depends (somehow) on x_j
  if j is a neighbor of i.
- Recursive: importance is transmitted through a network.
- Simplest possibility is a linear combination:

$$x_i \propto \sum_j a_{ji} x_j$$

- Assume further that constant of proportionality, c, is independent of i.
- Above gives $\vec{x} = c\mathbf{A}^{\mathsf{T}}\vec{x}$ or $\mathbf{A}^{\mathsf{T}}\vec{x} = c^{-1}\vec{x} = \lambda\vec{x}$.
- Eigenvalue equation based on adjacency matrix ...
- Note: Lots of despair over size of the largest eigenvalue. [⊞] Lose sight of original assumption's non-physicality.

Important nodes have important friends:

- Define x_i as the 'importance' of node i.
- Idea: x_i depends (somehow) on x_j
  if j is a neighbor of i.
- Recursive: importance is transmitted through a network.
- Simplest possibility is a linear combination:

$$x_i \propto \sum_j a_{ji} x_j$$

- Assume further that constant of proportionality, c, is independent of i.
- Above gives $\vec{x} = c\mathbf{A}^{\mathsf{T}}\vec{x}$ or $\mathbf{A}^{\mathsf{T}}\vec{x} = c^{-1}\vec{x} = \lambda\vec{x}$
- Eigenvalue equation based on adjacency matrix ...
- Note: Lots of despair over size of the largest eigenvalue.[7] Lose sight of original assumption's non-physicality.

Important nodes have important friends:

- Define x_i as the 'importance' of node i.
- Idea: x_i depends (somehow) on x_j
  if j is a neighbor of i.
- Recursive: importance is transmitted through a network.
- Simplest possibility is a linear combination:

$$x_i \propto \sum_j a_{ji} x_j$$

- Assume further that constant of proportionality, c, is independent of i.
- Above gives $\vec{x} = c\mathbf{A}^{\mathsf{T}}\vec{x}$ or $\boxed{\mathbf{A}^{\mathsf{T}}\vec{x} = c^{-1}\vec{x} = \lambda\vec{x}}$.
- Eigenvalue equation based on adjacency matrix ...
- Note: Lots of despair over size of the largest eigenvalue.[7] Lose sight of original assumption's non-physicality.

Important nodes have important friends:

- Define x_i as the 'importance' of node i.
- Idea: x_i depends (somehow) on x_j if j is a neighbor of i.
- Recursive: importance is transmitted through a network.
- Simplest possibility is a linear combination:

$$x_i \propto \sum_j a_{ji} x_j$$

- Assume further that constant of proportionality, c, is independent of i.
- Above gives $\vec{x} = c\mathbf{A}^{\mathsf{T}}\vec{x}$ or $\boxed{\mathbf{A}^{\mathsf{T}}\vec{x} = c^{-1}\vec{x} = \lambda\vec{x}}$.
- Eigenvalue equation based on adjacency matrix ...
- Note: Lots of despair over size of the largest eigenvalue. [7] Lose sight of original assumption's non-physicality.

Important nodes have important friends:

- Define x_i as the 'importance' of node i.
- Idea: x_i depends (somehow) on x_j if j is a neighbor of i.
- Recursive: importance is transmitted through a network.
- Simplest possibility is a linear combination:

$$x_i \propto \sum_j a_{ji} x_j$$

- Assume further that constant of proportionality, c, is independent of i.
- Above gives $\vec{x} = c\mathbf{A}^{\mathsf{T}}\vec{x}$ or $\boxed{\mathbf{A}^{\mathsf{T}}\vec{x} = c^{-1}\vec{x} = \lambda\vec{x}}$.
- Eigenvalue equation based on adjacency matrix ...
- Note: Lots of despair over size of the largest eigenvalue. [7] Lose sight of original assumption's non-physicality.

Important nodes have important friends:

- Define x_i as the 'importance' of node i.
- Idea: x_i depends (somehow) on x_j
  if j is a neighbor of i.
- Recursive: importance is transmitted through a network.
- Simplest possibility is a linear combination:

$$x_i \propto \sum_j a_{ji} x_j$$

- Assume further that constant of proportionality, c, is independent of i.
- Above gives $\vec{x} = c\mathbf{A}^{\mathsf{T}}\vec{x}$ or $\boxed{\mathbf{A}^{\mathsf{T}}\vec{x} = c^{-1}\vec{x} = \lambda\vec{x}}$.
- Eigenvalue equation based on adjacency matrix ...
- Note: Lots of despair over size of the largest eigenvalue. [7] Lose sight of original assumption's non-physicality.

Important nodes have important friends:

- So: solve $\mathbf{A}^{\mathsf{T}}\vec{x} = \lambda\vec{x}$.
- But which eigenvalue and eigenvector?
- We, the people, would like:
  1. A unique solution.
  2. λ to be real.
  3. Entries of $\vec{x}$ to be real.
  4. Entries of $\vec{x}$ to be non-negative.
  5. λ to actually mean something ...
  6. Values of x_i to mean something (what does an observation that $x_3 = 5x_7$ mean?) (maybe only ordering is informative ...)
  7. λ to equal 1 would be nice ...
  8. Ordering of $\vec{x}$ entries to be robust to reasonable modifications of linear assumption

Important nodes have important friends:

- So: solve $\mathbf{A}^{\mathsf{T}}\vec{x} = \lambda\vec{x}$.
- But which eigenvalue and eigenvector?
- We, the people, would like:
  1. A unique solution.
  2. λ to be real.
  3. Entries of $\vec{x}$ to be real.
  4. Entries of $\vec{x}$ to be non-negative.
  5. λ to actually mean something ...
  6. Values of x_i to mean something (what does an observation that $x_3 = 5x_7$ mean?) (maybe only ordering is informative ...)
  7. λ to equal 1 would be nice ...
  8. Ordering of $\vec{x}$ entries to be robust to reasonable modifications of linear assumption

Important nodes have important friends:

- So: solve $\mathbf{A}^{\mathsf{T}}\vec{x} = \lambda\vec{x}$.
- But which eigenvalue and eigenvector?
- We, the people, would like:
  1. A unique solution.
  2. λ to be real.
  3. Entries of $\vec{x}$ to be real.
  4. Entries of $\vec{x}$ to be non-negative.
  5. λ to actually mean something ...
  6. Values of x_i to mean something (what does an observation that $x_3 = 5x_7$ mean?) (maybe only ordering is informative ...)
  7. λ to equal 1 would be nice ...
  8. Ordering of $\vec{x}$ entries to be robust to reasonable modifications of linear assumption

Important nodes have important friends:

- So: solve $\mathbf{A}^{\mathsf{T}}\vec{x} = \lambda\vec{x}$.
- But which eigenvalue and eigenvector?
- We, the people, would like:
  1. A unique solution.
  2. λ to be real.
  3. Entries of $\vec{x}$ to be real.
  4. Entries of $\vec{x}$ to be non-negative.
  5. λ to actually mean something ...
  6. Values of x_i to mean something
     (what does an observation that $x_3 = 5x_7$ mean?)
     (maybe only ordering is informative ...)
  7. λ to equal 1 would be nice ...
  8. Ordering of $\vec{x}$ entries to be robust to reasonable modifications of linear assumption

Important nodes have important friends:

- So: solve $\mathbf{A}^{\mathsf{T}}\vec{x} = \lambda\vec{x}$.
- But which eigenvalue and eigenvector?
- We, the people, would like:
  1. A unique solution.
  2. λ to be real.
  3. Entries of $\vec{x}$ to be real.
  4. Entries of $\vec{x}$ to be non-negative.
  5. λ to actually mean something ...
  6. Values of x_i to mean something (what does an observation that $x_3 = 5x_7$ mean?) (maybe only ordering is informative ...)
  7. λ to equal 1 would be nice ...
  8. Ordering of $\vec{x}$ entries to be robust to reasonable modifications of linear assumption

Important nodes have important friends:

- So: solve $\mathbf{A}^{\mathsf{T}}\vec{x} = \lambda\vec{x}$.
- But which eigenvalue and eigenvector?
- We, the people, would like:
  1. A unique solution.
  2. λ to be real.
  3. Entries of $\vec{x}$ to be real.
  4. Entries of $\vec{x}$ to be non-negative.
  5. λ to actually mean something ...
  6. Values of x_i to mean something
     (what does an observation that $x_3 = 5x_7$ mean?)
     (maybe only ordering is informative ...)
  7. λ to equal 1 would be nice ...
  8. Ordering of $\vec{x}$ entries to be robust to reasonable modifications of linear assumption

Important nodes have important friends:

- So: solve $\mathbf{A}^{\mathsf{T}}\vec{x} = \lambda\vec{x}$.
- But which eigenvalue and eigenvector?
- We, the people, would like:
  1. A unique solution.
  2. λ to be real.
  3. Entries of $\vec{x}$ to be real.
  4. Entries of $\vec{x}$ to be non-negative.
  5. λ to actually mean something ...
  6. Values of x_i to mean something (what does an observation that $x_3 = 5x_7$ mean?) (maybe only ordering is informative ...)
  7. λ to equal 1 would be nice ...
  8. Ordering of $\vec{x}$ entries to be robust to reasonable modifications of linear assumption

Important nodes have important friends:

- So: solve $\mathbf{A}^{\mathsf{T}}\vec{x} = \lambda\vec{x}$.
- But which eigenvalue and eigenvector?
- We, the people, would like:
  1. A unique solution.
  2. λ to be real.
  3. Entries of $\vec{x}$ to be real.
  4. Entries of $\vec{x}$ to be non-negative.
  5. λ to actually mean something ...
  6. Values of x_i to mean something
     (what does an observation that $x_3 = 5x_7$ mean?)
     (maybe only ordering is informative ...)
  7. λ to equal 1 would be nice ...
  8. Ordering of $\vec{x}$ entries to be robust to reasonable modifications of linear assumption

Important nodes have important friends:

- So: solve $\mathbf{A}^{\mathsf{T}}\vec{x} = \lambda\vec{x}$.
- But which eigenvalue and eigenvector?
- We, the people, would like:
  1. A unique solution.
  2. λ to be real.
  3. Entries of $\vec{x}$ to be real.
  4. Entries of $\vec{x}$ to be non-negative.
  5. λ to actually mean something ...
  6. Values of x_i to mean something
     (what does an observation that $x_3 = 5x_7$ mean?)
     (maybe only ordering is informative ...)
  7. λ to equal 1 would be nice ...
  8. Ordering of $\vec{x}$ entries to be robust to reasonable modifications of linear assumption

Important nodes have important friends:

- So: solve $\mathbf{A}^{\mathsf{T}}\vec{x} = \lambda\vec{x}$.
- But which eigenvalue and eigenvector?
- We, the people, would like:
  1. A unique solution.
  2. λ to be real.
  3. Entries of $\vec{x}$ to be real.
  4. Entries of $\vec{x}$ to be non-negative.
  5. λ to actually mean something ...
  6. Values of x_i to mean something
     (what does an observation that $x_3 = 5x_7$ mean?)
     (maybe only ordering is informative ...)

  7. λ to equal 1 would be nice ...
  8. Ordering of $\vec{x}$ entries to be robust to reasonable modifications of linear assumption

Important nodes have important friends:

- So: solve $\mathbf{A}^{\mathsf{T}}\vec{x} = \lambda\vec{x}$.
- But which eigenvalue and eigenvector?
- We, the people, would like:
  1. A unique solution.
  2. λ to be real.
  3. Entries of $\vec{x}$ to be real.
  4. Entries of $\vec{x}$ to be non-negative.
  5. λ to actually mean something ...
  6. Values of x_i to mean something
     (what does an observation that $x_3 = 5x_7$ mean?)
     (maybe only ordering is informative ...)

  7. λ to equal 1 would be nice ...
  8. Ordering of $\vec{x}$ entries to be robust to reasonable modifications of linear assumption

Important nodes have important friends:

- So: solve $\mathbf{A}^{\mathsf{T}}\vec{x} = \lambda\vec{x}$.
- But which eigenvalue and eigenvector?
- We, the people, would like:
  1. A unique solution.
  2. λ to be real.
  3. Entries of $\vec{x}$ to be real.
  4. Entries of $\vec{x}$ to be non-negative.
  5. λ to actually mean something ... (maybe too much)
  6. Values of x_i to mean something (what does an observation that $x_3 = 5x_7$ mean?) (maybe only ordering is informative ...) (maybe too much)
  7. λ to equal 1 would be nice ... (maybe too much)
  8. Ordering of $\vec{x}$ entries to be robust to reasonable modifications of linear assumption (maybe too much)

Important nodes have important friends:

- So: solve $\mathbf{A}^{\mathsf{T}}\vec{x} = \lambda\vec{x}$.
- But which eigenvalue and eigenvector?
- We, the people, would like:
  1. A unique solution.
  2. λ to be real.
  3. Entries of $\vec{x}$ to be real.
  4. Entries of $\vec{x}$ to be non-negative.
  5. λ to actually mean something ... (maybe too much)
  6. Values of x_i to mean something
     (what does an observation that $x_3 = 5x_7$ mean?)
     (maybe only ordering is informative ...)
     (maybe too much)
  7. λ to equal 1 would be nice ... (maybe too much)
  8. Ordering of $\vec{x}$ entries to be robust to reasonable modifications of linear assumption (maybe too much)
- We rummage around in bag of tricks and pull out the Perron-Frobenius theorem ...

Important nodes have important friends:

- So: solve $\mathbf{A}^{\mathsf{T}}\vec{x} = \lambda\vec{x}$.
- But which eigenvalue and eigenvector?
- We, the people, would like:
  1. A unique solution. ✔
  2. λ to be real. ✔
  3. Entries of $\vec{x}$ to be real. ✔
  4. Entries of $\vec{x}$ to be non-negative. ✔
  5. λ to actually mean something ... (maybe too much)
  6. Values of x_i to mean something
     (what does an observation that $x_3 = 5x_7$ mean?)
     (maybe only ordering is informative ...)
     (maybe too much)
  7. λ to equal 1 would be nice ... (maybe too much)
  8. Ordering of $\vec{x}$ entries to be robust to reasonable modifications of linear assumption (maybe too much)
- We rummage around in bag of tricks and pull out the Perron-Frobenius theorem ...

Perron-Frobenius theorem: If an $N \times N$ matrix A has non-negative entries then:

1. A has a real eigenvalue $\lambda_1 \geq |\lambda_i|$ for $i = 2, \ldots, N$.
2. λ_1 corresponds to left and right 1-d eigenspaces for which we can choose a basis vector that has non-negative entries.
3. The dominant real eigenvalue λ_1 is bounded by the minimum and maximum row sums of A:

$$\min_i \sum_{j=1}^{N} a_{ij} \leq \lambda_1 \leq \max_i \sum_{j=1}^{N} a_{ij}$$

4. All other eigenvectors have one or more negative entries.

6. Note: Proof is relatively short for symmetric matrices that are strictly positive [6] and just non-negative [3].

Perron-Frobenius theorem: If an $N \times N$ matrix A has non-negative entries then:

1. A has a real eigenvalue $\lambda_1 \geq |\lambda_i|$ for $i = 2, \ldots, N$.
2. λ_1 corresponds to left and right 1-d eigenspaces for which we can choose a basis vector that has non-negative entries.
3. The dominant real eigenvalue λ_1 is bounded by the minimum and maximum row sums of A:

$$\min_i \sum_{j=1}^{N} a_{ij} \leq \lambda_1 \leq \max_i \sum_{j=1}^{N} a_{ij}$$

4. All other eigenvectors have one or more negative entries.

6. Note: Proof is relatively short for symmetric matrices that are strictly positive [6] and just non-negative [3].

Perron-Frobenius theorem: If an $N \times N$ matrix A has non-negative entries then:

1. A has a real eigenvalue $\lambda_1 \geq |\lambda_i|$ for $i = 2, \ldots, N$.
2. λ_1 corresponds to left and right 1-d eigenspaces for which we can choose a basis vector that has non-negative entries.
3. The dominant real eigenvalue λ_1 is bounded by the minimum and maximum row sums of A:

$$\min_i \sum_{j=1}^{N} a_{ij} \leq \lambda_1 \leq \max_i \sum_{j=1}^{N} a_{ij}$$

4. All other eigenvectors have one or more negative entries.

6. Note: Proof is relatively short for symmetric matrices that are strictly positive [6] and just non-negative [8].

Perron-Frobenius theorem: If an $N \times N$ matrix A has non-negative entries then:

1. A has a real eigenvalue $\lambda_1 \geq |\lambda_i|$ for $i = 2, \ldots, N$.
2. λ_1 corresponds to left and right 1-d eigenspaces for which we can choose a basis vector that has non-negative entries.
3. The dominant real eigenvalue λ_1 is bounded by the minimum and maximum row sums of A:

$$\min_i \sum_{j=1}^{N} a_{ij} \leq \lambda_1 \leq \max_i \sum_{j=1}^{N} a_{ij}$$

4. All other eigenvectors have one or more negative entries.

6. Note: Proof is relatively short for symmetric matrices that are strictly positive [6] and just non-negative [8].

Perron-Frobenius theorem: If an $N \times N$ matrix A has non-negative entries then:

1. A has a real eigenvalue $\lambda_1 \geq |\lambda_i|$ for $i = 2, \dots, N$.
2. λ_1 corresponds to left and right 1-d eigenspaces for which we can choose a basis vector that has non-negative entries.
3. The dominant real eigenvalue λ_1 is bounded by the minimum and maximum row sums of A:

$$\min_i \sum_{j=1}^{N} a_{ij} \leq \lambda_1 \leq \max_i \sum_{j=1}^{N} a_{ij}$$

4. All other eigenvectors have one or more negative entries.

6. Note: Proof is relatively short for symmetric matrices that are strictly positive [6] and just non-negative [8].

Perron-Frobenius theorem: If an $N \times N$ matrix A has non-negative entries then:

1. A has a real eigenvalue $\lambda_1 \geq |\lambda_i|$ for $i = 2, \ldots, N$.
2. λ_1 corresponds to left and right 1-d eigenspaces for which we can choose a basis vector that has non-negative entries.
3. The dominant real eigenvalue λ_1 is bounded by the minimum and maximum row sums of A:

$$\min_i \sum_{j=1}^{N} a_{ij} \leq \lambda_1 \leq \max_i \sum_{j=1}^{N} a_{ij}$$

4. All other eigenvectors have one or more negative entries.
5. The matrix A can make toast.
6. Note: Proof is relatively short for symmetric matrices that are strictly positive [6] and just non-negative [3].

Perron-Frobenius theorem: If an $N \times N$ matrix A has non-negative entries then:

1. A has a real eigenvalue $\lambda_1 \geq |\lambda_i|$ for $i = 2, \ldots, N$.
2. λ_1 corresponds to left and right 1-d eigenspaces for which we can choose a basis vector that has non-negative entries.
3. The dominant real eigenvalue λ_1 is bounded by the minimum and maximum row sums of A:

$$\min_i \sum_{j=1}^{N} a_{ij} \leq \lambda_1 \leq \max_i \sum_{j=1}^{N} a_{ij}$$

4. All other eigenvectors have one or more negative entries.
5. The matrix A can make toast.
6. Note: Proof is relatively short for symmetric matrices that are strictly positive [6] and just non-negative [3].

Other Perron-Frobenius aspects:

- Assuming our network is irreducible, meaning there is only one component, is reasonable: just consider one component at a time if more than one exists.
- Irreducibility means largest eigenvalue's eigenvector has strictly non-negative entries.
- Analogous to notion of ergodicity: every state is reachable.
- (Another term: Primitive graphs and matrices.)

Other Perron-Frobenius aspects:

- Assuming our network is irreducible, meaning there is only one component, is reasonable: just consider one component at a time if more than one exists.
- Irreducibility means largest eigenvalue's eigenvector has strictly non-negative entries.
- Analogous to notion of ergodicity: every state is reachable.
- (Another term: Primitive graphs and matrices.)

Other Perron-Frobenius aspects:

- Assuming our network is irreducible, meaning there is only one component, is reasonable: just consider one component at a time if more than one exists.
- Irreducibility means largest eigenvalue's eigenvector has strictly non-negative entries.
- Analogous to notion of ergodicity: every state is reachable.
- (Another term: Primitive graphs and matrices.)

Other Perron-Frobenius aspects:

- Assuming our network is irreducible, meaning there is only one component, is reasonable: just consider one component at a time if more than one exists.
- Irreducibility means largest eigenvalue's eigenvector has strictly non-negative entries.
- Analogous to notion of ergodicity: every state is reachable.
- (Another term: Primitive graphs and matrices.)

Other Perron-Frobenius aspects:

- Assuming our network is irreducible, meaning there is only one component, is reasonable: just consider one component at a time if more than one exists.
- Irreducibility means largest eigenvalue's eigenvector has strictly non-negative entries.
- Analogous to notion of ergodicity: every state is reachable.
- (Another term: Primitive graphs and matrices.)

Outline

Hubs and Authorities

- Generalize eigenvalue centrality to allow nodes to have two attributes:
  1. Authority: how much knowledge, information, etc., held by a node on a topic.
  2. Hubness (or Hubosity or Hubbishness or Hubtasticness): how well a node 'knows' where to find information on a given topic.
- Original work due to the legendary Jon Kleinberg. [4]
- Best hubs point to best authorities.
- Recursive: Hubs authoritatively link to hubs, authorities hubbishly link to other authorities.
- More: look for dense links between sets of 'good' hubs pointing to sets of 'good' authorities.
- Known as the HITS algorithm (Hyperlink-Induced Topics Search).

Hubs and Authorities

- Generalize eigenvalue centrality to allow nodes to have two attributes:
    1. Authority: how much knowledge, information, etc., held by a node on a topic.
    2. Hubness (or Hubosity or Hubbishness or Hubtasticness): how well a node 'knows' where to find information on a given topic.
- Original work due to the legendary Jon Kleinberg.[1]
- Best hubs point to best authorities.
- Recursive: Hubs authoritatively link to hubs, authorities hubbishly link to other authorities.
- More: look for dense links between sets of 'good' hubs pointing to sets of 'good' authorities.
- Known as the HITS algorithm (Hyperlink-Induced Topics Search).

Hubs and Authorities

- Generalize eigenvalue centrality to allow nodes to have two attributes:
  1. Authority: how much knowledge, information, etc., held by a node on a topic.
  2. Hubness (or Hubosity or Hubbishness or Hubtasticness): how well a node 'knows' where to find information on a given topic.
- Original work due to the legendary Jon Kleinberg.[4]
- Best hubs point to best authorities.
- Recursive: Hubs authoritatively link to hubs, authorities hubbishly link to other authorities.
- More: look for dense links between sets of 'good' hubs pointing to sets of 'good' authorities.
- Known as the HITS algorithm (Hyperlink-Induced Topics Search).

Hubs and Authorities

- Generalize eigenvalue centrality to allow nodes to have two attributes:
  1. Authority: how much knowledge, information, etc., held by a node on a topic.
  2. Hubness (or Hubosity or Hubbishness or Hubtasticness): how well a node 'knows' where to find information on a given topic.
- Original work due to the legendary Jon Kleinberg. [2]
- Best hubs point to best authorities.
- Recursive: Hubs authoritatively link to hubs, authorities hubbishly link to other authorities.
- More: look for dense links between sets of 'good' hubs pointing to sets of 'good' authorities.
- Known as the HITS algorithm (Hyperlink-Induced Topics Search).

Hubs and Authorities

- Generalize eigenvalue centrality to allow nodes to have two attributes:
  1. Authority: how much knowledge, information, etc., held by a node on a topic.
  2. Hubness (or Hubosity or Hubbishness or Hubtasticness): how well a node 'knows' where to find information on a given topic.
- Original work due to the legendary Jon Kleinberg. [2]
- Best hubs point to best authorities.
- Recursive: Hubs authoritatively link to hubs, authorities hubbishly link to other authorities.
- More: look for dense links between sets of 'good' hubs pointing to sets of 'good' authorities.
- Known as the HITS algorithm (Hyperlink-Induced Topics Search).

Hubs and Authorities

- Generalize eigenvalue centrality to allow nodes to have two attributes:
  1. Authority: how much knowledge, information, etc., held by a node on a topic.
  2. Hubness (or Hubosity or Hubbishness or Hubtasticness): how well a node 'knows' where to find information on a given topic.
- Original work due to the legendary Jon Kleinberg. [2]
- Best hubs point to best authorities.
- Recursive: Hubs authoritatively link to hubs, authorities hubbishly link to other authorities.
- More: look for dense links between sets of 'good' hubs pointing to sets of 'good' authorities.
- Known as the HITS algorithm (Hyperlink-Induced Topics Search).

Hubs and Authorities

- Generalize eigenvalue centrality to allow nodes to have two attributes:
  1. Authority: how much knowledge, information, etc., held by a node on a topic.
  2. Hubness (or Hubosity or Hubbishness or Hubtasticness): how well a node 'knows' where to find information on a given topic.
- Original work due to the legendary Jon Kleinberg. [2]
- Best hubs point to best authorities.
- Recursive: Hubs authoritatively link to hubs, authorities hubbishly link to other authorities.
- More: look for dense links between sets of 'good' hubs pointing to sets of 'good' authorities.
- Known as the HITS algorithm (Hyperlink-Induced Topics Search).

Hubs and Authorities

- Generalize eigenvalue centrality to allow nodes to have two attributes:
  1. Authority: how much knowledge, information, etc., held by a node on a topic.
  2. Hubness (or Hubosity or Hubbishness or Hubtasticness): how well a node 'knows' where to find information on a given topic.
- Original work due to the legendary Jon Kleinberg. [2]
- Best hubs point to best authorities.
- Recursive: Hubs authoritatively link to hubs, authorities hubbishly link to other authorities.
- More: look for dense links between sets of 'good' hubs pointing to sets of 'good' authorities.
- Known as the HITS algorithm (Hyperlink-Induced Topics Search).

Hubs and Authorities

- Give each node two scores:
  1. x_i = authority score for node i
  2. y_i = hubtasticness score for node i
- As for eigenvector centrality, we connect the scores of neighboring nodes.
- New story I: a good authority is linked to by good hubs.
- Means x_i should increase as $\sum_{j=1}^{N} a_{ji} y_j$ increases.
- Note: indices are ji meaning j has a directed link to i.
- New story II: good hubs point to good authorities.
- Means y_i should increase as $\sum_{j=1}^{N} a_{ij} x_j$ increases.
- Linearity assumption:

$$\vec{x} \propto A^{T}\vec{y} \text{ and } \vec{y} \propto A\vec{x}$$

Hubs and Authorities

- Give each node two scores:
  1. x_i = authority score for node i
  2. y_i = hubtasticness score for node i
- As for eigenvector centrality, we connect the scores of neighboring nodes.
- New story I: a good authority is linked to by good hubs.
- Means x_i should increase as $\sum_{j=1}^{N} a_{ji} y_j$ increases.
- Note: indices are ji meaning j has a directed link to i.
- New story II: good hubs point to good authorities.
- Means y_i should increase as $\sum_{j=1}^{N} a_{ij} x_j$ increases.
- Linearity assumption:

$$\vec{x} \propto A^T \vec{y} \text{ and } \vec{y} \propto A\vec{x}$$

Hubs and Authorities

- Give each node two scores:
  1. x_i = authority score for node i
  2. y_i = hubtasticness score for node i
- As for eigenvector centrality, we connect the scores of neighboring nodes.
- New story I: a good authority is linked to by good hubs.
- Means x_i should increase as $\sum_{j=1}^{N} a_{ji} y_j$ increases.
- Note: indices are ji meaning j has a directed link to i.
- New story II: good hubs point to good authorities.
- Means y_i should increase as $\sum_{j=1}^{N} a_{ij} x_j$ increases.
- Linearity assumption:

$$\vec{x} \propto A^T \vec{y} \text{ and } \vec{y} \propto A \vec{x}$$

Hubs and Authorities

- Give each node two scores:
  1. x_i = authority score for node i
  2. y_i = hubtasticness score for node i
- As for eigenvector centrality, we connect the scores of neighboring nodes.
- New story I: a good authority is linked to by good hubs.
- Means x_i should increase as $\sum_{j=1}^{N} a_{ji} y_j$ increases.
- Note: indices are ji meaning j has a directed link to i.
- New story II: good hubs point to good authorities.
- Means y_i should increase as $\sum_{j=1}^{N} a_{ij} x_j$ increases.
- Linearity assumption:

$$\vec{x} \propto A^T \vec{y} \text{ and } \vec{y} \propto A\vec{x}$$

Hubs and Authorities

- Give each node two scores:
  1. x_i = authority score for node i
  2. y_i = hubtasticness score for node i
- As for eigenvector centrality, we connect the scores of neighboring nodes.
- New story I: a good authority is linked to by good hubs.
- Means x_i should increase as $\sum_{j=1}^{N} a_{ji} y_j$ increases.
- Note: indices are ji meaning j has a directed link to i.
- New story II: good hubs point to good authorities.
- Means y_i should increase as $\sum_{j=1}^{N} a_{ij} x_j$ increases.
- Linearity assumption:

$$\vec{x} \propto A^T \vec{y} \text{ and } \vec{y} \propto A \vec{x}$$

Hubs and Authorities

- Give each node two scores:
  1. x_i = authority score for node i
  2. y_i = hubtasticness score for node i
- As for eigenvector centrality, we connect the scores of neighboring nodes.
- New story I: a good authority is linked to by good hubs.
- Means x_i should increase as $\sum_{j=1}^{N} a_{ji} y_j$ increases.
- Note: indices are ji meaning j has a directed link to i.
- New story II: good hubs point to good authorities.
- Means y_i should increase as $\sum_{j=1}^{N} a_{ij} x_j$ increases.
- Linearity assumption:

$$\vec{x} \propto A^T \vec{y} \text{ and } \vec{y} \propto A \vec{x}$$

Hubs and Authorities

- Give each node two scores:
  1. x_i = authority score for node i
  2. y_i = hubtasticness score for node i
- As for eigenvector centrality, we connect the scores of neighboring nodes.
- New story I: a good authority is linked to by good hubs.
- Means x_i should increase as $\sum_{j=1}^{N} a_{ji} y_j$ increases.
- Note: indices are ji meaning j has a directed link to i.
- New story II: good hubs point to good authorities.
- Means y_i should increase as $\sum_{j=1}^{N} a_{ij} x_j$ increases.
- Linearity assumption:

$$\vec{x} \propto A^T \vec{y} \text{ and } \vec{y} \propto A\vec{x}$$

Hubs and Authorities

- Give each node two scores:
  1. x_i = authority score for node i
  2. y_i = hubtasticness score for node i
- As for eigenvector centrality, we connect the scores of neighboring nodes.
- New story I: a good authority is linked to by good hubs.
- Means x_i should increase as $\sum_{j=1}^{N} a_{ji} y_j$ increases.
- Note: indices are ji meaning j has a directed link to i.
- New story II: good hubs point to good authorities.
- Means y_i should increase as $\sum_{j=1}^{N} a_{ij} x_j$ increases.
- Linearity assumption:

$$\vec{x} \propto A^T \vec{y} \text{ and } \vec{y} \propto A \vec{x}$$

Hubs and Authorities

- Give each node two scores:
  1. x_i = authority score for node i
  2. y_i = hubtasticness score for node i
- As for eigenvector centrality, we connect the scores of neighboring nodes.
- New story I: a good authority is linked to by good hubs.
- Means x_i should increase as $\sum_{j=1}^{N} a_{ji} y_j$ increases.
- Note: indices are ji meaning j has a directed link to i.
- New story II: good hubs point to good authorities.
- Means y_i should increase as $\sum_{j=1}^{N} a_{ij} x_j$ increases.
- Linearity assumption:

$$\vec{x} \propto A^T \vec{y} \text{ and } \vec{y} \propto A \vec{x}$$

Hubs and Authorities

- Give each node two scores:
  1. x_i = authority score for node i
  2. y_i = hubtasticness score for node i
- As for eigenvector centrality, we connect the scores of neighboring nodes.
- New story I: a good authority is linked to by good hubs.
- Means x_i should increase as $\sum_{j=1}^{N} a_{ji} y_j$ increases.
- Note: indices are ji meaning j has a directed link to i.
- New story II: good hubs point to good authorities.
- Means y_i should increase as $\sum_{j=1}^{N} a_{ij} x_j$ increases.
- Linearity assumption:

$$\vec{x} \propto A^T \vec{y} \text{ and } \vec{y} \propto A\vec{x}$$

Hubs and Authorities

- So let's say we have

$$\vec{x} = c_1 A^T \vec{y} \text{ and } \vec{y} = c_2 A \vec{x}$$

where c_1 and c_2 must be positive.

- Above equations combine to give

$$\vec{x} = c_1 A^T c_2 A \vec{x} = \lambda A^T A \vec{x}.$$

where $\lambda = c_1 c_2 > 0$.

- It's all good: we have the heart of singular value decomposition before us ...

Hubs and Authorities

- So let's say we have

$$\vec{x} = c_1 A^T \vec{y} \text{ and } \vec{y} = c_2 A \vec{x}$$

where c_1 and c_2 must be positive.

- Above equations combine to give

$$\vec{x} = c_1 A^T c_2 A \vec{x} = \lambda A^T A \vec{x},$$

where $\lambda = c_1 c_2 > 0$.

- It's all good: we have the heart of singular value decomposition before us ...

Hubs and Authorities

- So let's say we have

$$\vec{x} = c_1 A^T \vec{y} \text{ and } \vec{y} = c_2 A \vec{x}$$

where c_1 and c_2 must be positive.

- Above equations combine to give

$$\vec{x} = c_1 A^T c_2 A \vec{x} = \lambda A^T A \vec{x}.$$

where $\lambda = c_1 c_2 > 0$.

- It's all good: we have the heart of singular value decomposition before us ...

We can do this:

- $A^T A$ is symmetric.
- $A^T A$ is semi-positive definite so its eigenvalues are all ≥ 0.
- $A^T A$'s eigenvalues are the square of A's singular values.
- $A^T A$'s eigenvectors form a joyful orthogonal basis.
- Perron-Frobenius tells us that only the dominant eigenvalue's eigenvector can be chosen to have non-negative entries.
- So: linear assumption leads to a solvable system.
- What would be very good: find networks where we have independent measures of node 'importance' and see how importance is actually distributed.

We can do this:

- $A^T A$ is symmetric.
- $A^T A$ is semi-positive definite so its eigenvalues are all ≥ 0.
- $A^T A$'s eigenvalues are the square of A's singular values.
- $A^T A$'s eigenvectors form a joyful orthogonal basis.
- Perron-Frobenius tells us that only the dominant eigenvalue's eigenvector can be chosen to have non-negative entries.
- So: linear assumption leads to a solvable system.
- What would be very good: find networks where we have independent measures of node 'importance' and see how importance is actually distributed.

We can do this:

- $A^T A$ is symmetric.
- $A^T A$ is semi-positive definite so its eigenvalues are all ≥ 0.
- $A^T A$'s eigenvalues are the square of A's singular values.
- $A^T A$'s eigenvectors form a joyful orthogonal basis.
- Perron-Frobenius tells us that only the dominant eigenvalue's eigenvector can be chosen to have non-negative entries.
- So: linear assumption leads to a solvable system.
- What would be very good: find networks where we have independent measures of node 'importance' and see how importance is actually distributed.

We can do this:

- $A^T A$ is symmetric.
- $A^T A$ is semi-positive definite so its eigenvalues are all ≥ 0.
- $A^T A$'s eigenvalues are the square of A's singular values.
- $A^T A$'s eigenvectors form a joyful orthogonal basis.
- Perron-Frobenius tells us that only the dominant eigenvalue's eigenvector can be chosen to have non-negative entries.
- So: linear assumption leads to a solvable system.
- What would be very good: find networks where we have independent measures of node 'importance' and see how importance is actually distributed.

We can do this:

- $A^T A$ is symmetric.
- $A^T A$ is semi-positive definite so its eigenvalues are all ≥ 0.
- $A^T A$'s eigenvalues are the square of A's singular values.
- $A^T A$'s eigenvectors form a joyful orthogonal basis.
- Perron-Frobenius tells us that only the dominant eigenvalue's eigenvector can be chosen to have non-negative entries.
- So: linear assumption leads to a solvable system.
- What would be very good: find networks where we have independent measures of node 'importance' and see how importance is actually distributed.

We can do this:

- $A^T A$ is symmetric.
- $A^T A$ is semi-positive definite so its eigenvalues are all ≥ 0.
- $A^T A$'s eigenvalues are the square of A's singular values.
- $A^T A$'s eigenvectors form a joyful orthogonal basis.
- Perron-Frobenius tells us that only the dominant eigenvalue's eigenvector can be chosen to have non-negative entries.
- So: linear assumption leads to a solvable system.
- What would be very good: find networks where we have independent measures of node 'importance' and see how importance is actually distributed.

We can do this:

- $A^T A$ is symmetric.
- $A^T A$ is semi-positive definite so its eigenvalues are all ≥ 0.
- $A^T A$'s eigenvalues are the square of A's singular values.
- $A^T A$'s eigenvectors form a joyful orthogonal basis.
- Perron-Frobenius tells us that only the dominant eigenvalue's eigenvector can be chosen to have non-negative entries.
- So: linear assumption leads to a solvable system.
- What would be very good: find networks where we have independent measures of node 'importance' and see how importance is actually distributed.

Nutshell:

- Measuring centrality is well motivated if hard to carry out well.
- We've only looked at a few major ones.
- Methods are often taken to be more sophisticated than they really are.
- Centrality can be used pragmatically to perform diagnostics on networks (see structure detection).
- Focus on nodes rather than groups or modules is a homo narrativus constraint.
- Possible that better approaches will be developed.

Nutshell:

- Measuring centrality is well motivated if hard to carry out well.
- We've only looked at a few major ones.
- Methods are often taken to be more sophisticated than they really are.
- Centrality can be used pragmatically to perform diagnostics on networks (see structure detection).
- Focus on nodes rather than groups or modules is a homo narrativus constraint.
- Possible that better approaches will be developed.

Nutshell:

- Measuring centrality is well motivated if hard to carry out well.
- We've only looked at a few major ones.
- Methods are often taken to be more sophisticated than they really are.
- Centrality can be used pragmatically to perform diagnostics on networks (see structure detection).
- Focus on nodes rather than groups or modules is a homo narrativus constraint.
- Possible that better approaches will be developed.

Nutshell:

- Measuring centrality is well motivated if hard to carry out well.
- We've only looked at a few major ones.
- Methods are often taken to be more sophisticated than they really are.
- Centrality can be used pragmatically to perform diagnostics on networks (see structure detection).
- Focus on nodes rather than groups or modules is a homo narrativus constraint.
- Possible that better approaches will be developed.

Nutshell:

- Measuring centrality is well motivated if hard to carry out well.
- We've only looked at a few major ones.
- Methods are often taken to be more sophisticated than they really are.
- Centrality can be used pragmatically to perform diagnostics on networks (see structure detection).
- Focus on nodes rather than groups or modules is a homo narrativus constraint.
- Possible that better approaches will be developed.

Nutshell:

- Measuring centrality is well motivated if hard to carry out well.
- We've only looked at a few major ones.
- Methods are often taken to be more sophisticated than they really are.
- Centrality can be used pragmatically to perform diagnostics on networks (see structure detection).
- Focus on nodes rather than groups or modules is a homo narrativus constraint.
- Possible that better approaches will be developed.

References I

[1] U. Brandes.
A faster algorithm for betweenness centrality.
J. Math. Sociol., 25:163–177, 2001. pdf

[2] J. M. Kleinberg.
Authoritative sources in a hyperlinked environment.
Proc. 9th ACM-SIAM Symposium on Discrete Algorithms, 1998. pdf

[3] K. Y. Lin.
An elementary proof of the perron-frobenius theorem for non-negative symmetric matrices.
Chinese Journal of Physics, 15:283–285, 1977. pdf

References II

[4] M. E. J. Newman.
Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality.
Phys. Rev. E, 64(1):016132, 2001. pdf

[5] M. E. J. Newman and M. Girvan.
Finding and evaluating community structure in networks.
Phys. Rev. E, 69(2):026113, 2004. pdf

[6] F. Ninio.
A simple proof of the Perron-Frobenius theorem for positive symmetric matrices.
J. Phys. A.: Math. Gen., 9:1281–1282, 1976. pdf

References III

[7] S. Wasserman and K. Faust.
Social Network Analysis: Methods and Applications.
Cambridge University Press, Cambridge, UK, 1994.