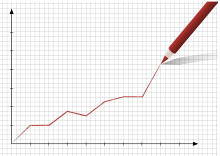
Graphs

Schwartz

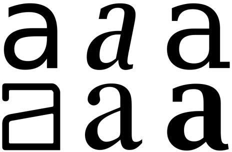
June 21, 2017

Six Degrees of Separation



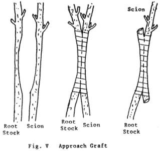
Graph

A plot;
to draw a plot



Graph
(graphemes, glyphs)

A written symbol for
an idea, a sound or a
linguistic expression



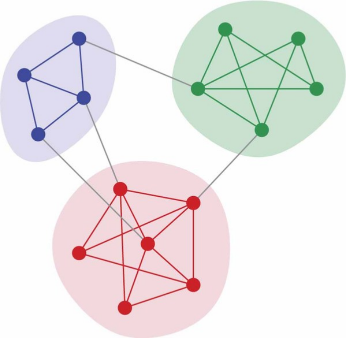
Graft

A piece of
living that is tissue
transplanted surgically

Giraffe

An african animal

1929	Frigyes Karinthy introduces the idea
1967	Small World Project estimates ‘3’ in US
1990	John Guare popularizes ‘6’ in a play
1994	Six Degrees of Kevin Bacon happens
1998	$\frac{\ln N}{\ln K}$ is ‘5.7’ for US and ‘6.6’ for World
2001	Small World Project estimates ‘6’ in IM
2011	Twitter degree of separation ‘3.43‘
2008	Facebook degree of separation ‘5.28‘
2011	Facebook degree of separation ‘4.74‘
2016	Facebook degree of separation ‘3.57‘



Objectives

- Graph things and stuff
- Graph data structures
- Graph search algs

- Centrality
- Community Detection
  - Modularity
  - Girvan-Newman Algorithm

Graph Things Quiz

- node (vertex)

Graph Things Quiz

- node (vertex)
- edge

Graph Things Quiz

- node (vertex)
- edge
  - (un)directed

Graph Things Quiz

- node (vertex)
- edge
  - (un)directed
  - (un)weighted

Graph Things Quiz

- node (vertex)
- edge
  - (un)directed
  - (un)weighted
- neighbors

Graph Things Quiz

- node (vertex)
- edge
  - (un)directed
  - (un)weighted
- neighbors
  - (in/out)degree

Graph Things Quiz

- node (vertex)
- edge
  - (un)directed
  - (un)weighted
- neighbors
  - (in/out)degree
  - clique

Graph Things Quiz

- node (vertex)
- edge
  - (un)directed
  - (un)weighted
- neighbors
  - (in/out)degree
  - clique
- path

Graph Things Quiz

- node (vertex)
- edge
  - (un)directed
  - (un)weighted
- neighbors
  - (in/out)degree
  - clique
- path
- graphs

Graph Things Quiz

- node (vertex)
- edge
  - (un)directed
  - (un)weighted
- neighbors
  - (in/out)degree
  - clique
- path
- graphs
  - connected

Graph Things Quiz

- node (vertex)
- edge
  - (un)directed
  - (un)weighted
- neighbors
  - (in/out)degree
  - clique
- path
- graphs
  - connected
  - complete

Graph Things Quiz

- node (vertex)
- edge
  - (un)directed
  - (un)weighted
- neighbors
  - (in/out)degree
  - clique
- path
- graphs
  - connected
  - complete
  - subgraph

Graph Data Structures

- Adjacency Lists

```
1: {2, 5}
2: {1, 3, 5}
3: {2, 4}
4: {5, 6}
5: {1, 2, 4}
6: {4}
```

- Adjacency Matrix

```
    1 2 3 4 5 6
1 [[0 1 0 0 1 0]
2  [1 0 1 0 1 1]
3  [0 1 0 1 0 1]
4  [0 0 1 0 1 1]
5  [1 1 0 1 0 1]
6  [0 0 0 1 0 0]]
```

Graph Data Structures

- Adjacency Lists

```
1: {2, 5}
2: {1, 3, 5}
3: {2, 4}
4: {5, 6}
5: {1, 2, 4}
6: {4}
```

- Adjacency Matrix

```
     1 2 3 4 5 6
1  [[0 1 0 0 1 0]
2   [1 0 1 0 1 1]
3   [0 1 0 1 0 1]
4   [0 0 1 0 1 1]
5   [1 1 0 1 0 1]
6   [0 0 0 1 0 0]]
```

Which implementation is better for:

- Getting a list of a nodes neighbors?

Graph Data Structures

- Adjacency Lists

```
1: {2, 5}
2: {1, 3, 5}
3: {2, 4}
4: {5, 6}
5: {1, 2, 4}
6: {4}
```

- Adjacency Matrix

```
    1 2 3 4 5 6
1 [[0 1 0 0 1 0]
2  [1 0 1 0 1 1]
3  [0 1 0 1 0 1]
4  [0 0 1 0 1 1]
5  [1 1 0 1 0 1]
6  [0 0 0 1 0 0]]
```

Which implementation is better for:

- Getting a list of a nodes neighbors?
- Calculating the degree of a node?

Graph Data Structures

- Adjacency Lists

```
1: {2, 5}
2: {1, 3, 5}
3: {2, 4}
4: {5, 6}
5: {1, 2, 4}
6: {4}
```

- Adjacency Matrix

```
     1 2 3 4 5 6
1  [[0 1 0 0 1 0]
2   [1 0 1 0 1 1]
3   [0 1 0 1 0 1]
4   [0 0 1 0 1 1]
5   [1 1 0 1 0 1]
6   [0 0 0 1 0 0]]
```

Which implementation is better for:

- Getting a list of a nodes neighbors?
- Calculating the degree of a node?
- Determining if two nodes are connected?

Graph Search Algorithms

- https://www.youtube.com/watch?v=YM6Swr6kcBw

Graph Search Algorithms

Breadth First Search (BFS)

1. Start Node
2. Neighbors
3. Neighbors of Neighbors
4. Neighbors of Neighbors of...

⋮ Etc.

Don't follow nodes already seen

Stop when neighbors exhausted

- https://www.youtube.com/watch?v=YM6Swr6kcBw

Graph Search Algorithms

Breadth First Search (BFS)

1. Start Node
2. Neighbors
3. Neighbors of Neighbors
4. Neighbors of Neighbors of...

⋮ Etc.

Don't follow nodes already seen

Stop when neighbors exhausted

- https://www.youtube.com/watch?v=YM6Swr6kcBw
- https://www.youtube.com/watch?v=dN1cPlfTcg8

Graph Search Algorithms

Breadth First Search (BFS)

1. Start Node
2. Neighbors
3. Neighbors of Neighbors
4. Neighbors of Neighbors of...

⋮ Etc.

Don't follow nodes already seen

Stop when neighbors exhausted

Depth First Search (DFS)

1. Start Node
2. Neighbor
3. Neighbor of Neighbor
4. Neighbor of Neighbor of...

⋮ Etc.

Don't follow nodes already seen

Go back if neighbors exhausted

- https://www.youtube.com/watch?v=YM6Swr6kcBw
- https://www.youtube.com/watch?v=dN1cPlfTcg8

Graph Search Algorithms

Breadth First Search (BFS)

1. Start Node
2. Neighbors
3. Neighbors of Neighbors
4. Neighbors of Neighbors of...

⋮ Etc.

Don't follow nodes already seen

Stop when neighbors exhausted

Depth First Search (DFS)

1. Start Node
2. Neighbor
3. Neighbor of Neighbor
4. Neighbor of Neighbor of...

⋮ Etc.

Don't follow nodes already seen

Go back if neighbors exhausted

- https://www.youtube.com/watch?v=YM6Swr6kcBw
- https://www.youtube.com/watch?v=dN1cPlfTcg8
- https://www.cs.usfca.edu/~galles/visualization

Implementation

Breadth First Search (BFS)

Depth First Search (DFS)

- How do you implement these algorithms (hint: queue/stack)?

Implementation

Breadth First Search (BFS)

- $V = \emptyset$
- $Q = \emptyset_{queue}$
- $Q \Leftarrow s$

Depth First Search (DFS)

- How do you implement these algorithms (hint: queue/stack)?

Implementation

Breadth First Search (BFS)

- $V = \emptyset$
- $Q = \emptyset_{queue}$
- $Q \Leftarrow s$
- while $Q \neq \emptyset_{queue}$

Depth First Search (DFS)

- How do you implement these algorithms (hint: queue/stack)?

Implementation

Breadth First Search (BFS)

- $V = \emptyset$
- $Q = \emptyset_{queue}$
- $Q \Leftarrow s$
- while $Q \neq \emptyset_{queue}$
  - $n \Leftarrow Q$
  - if $n \notin V$

Depth First Search (DFS)

- How do you implement these algorithms (hint: queue/stack)?

Implementation

Breadth First Search (BFS) **Depth First Search (DFS)**

- $V = \emptyset$
- $Q = \emptyset_{queue}$
- $Q \Leftarrow s$
- while $Q \neq \emptyset_{queue}$
  - $n \Leftarrow Q$
  - if $n \notin V$
    - check n for objective

- How do you implement these algorithms (hint: queue/stack)?

Implementation

Breadth First Search (BFS)

- $V = \emptyset$
- $Q = \emptyset_{queue}$
- $Q \Leftarrow s$
- while $Q \neq \emptyset_{queue}$
  - $n \Leftarrow Q$
  - if $n \notin V$
    - check n for objective
    - $n \Rightarrow V$

Depth First Search (DFS)

- How do you implement these algorithms (hint: queue/stack)?

Implementation

Breadth First Search (BFS)

- $V = \emptyset$
- $Q = \emptyset_{queue}$
- $Q \Leftarrow s$
- while $Q \neq \emptyset_{queue}$
  - $n \Leftarrow Q$
  - if $n \notin V$
    - check n for objective
    - $n \Rightarrow V$
    - $Q \Leftarrow \text{neighbors}(n)$

Depth First Search (DFS)

- How do you implement these algorithms (hint: queue/stack)?

Implementation

Breadth First Search (BFS)

- $V = \emptyset$
- $Q = \emptyset_{queue}$
- $Q \Leftarrow s$
- while $Q \neq \emptyset_{queue}$
  - $n \Leftarrow Q$
  - if $n \notin V$
    - check n for objective
    - $n \Rightarrow V$
    - $Q \Leftarrow$ neighbors(n)

Depth First Search (DFS)

- $V = \emptyset$
- $Q = \emptyset_{stack}$
- $s \Rightarrow Q$
- while $Q \neq \emptyset_{stack}$
  - $n \Leftarrow Q$
  - if $n \notin V$
    - check n for objective
    - $n \Rightarrow V$
    - neighbors(n) $\Rightarrow Q$

- How do you implement these algorithms (hint: queue/stack)?

Implementation

Breadth First Search (BFS)

- $V = \emptyset$
- $Q = \emptyset_{queue}$
- $Q \Leftarrow s$
- while $Q \neq \emptyset_{queue}$
  - $n \Leftarrow Q$
  - if $n \notin V$
    - check n for objective
    - $n \Rightarrow V$
    - $Q \Leftarrow \text{neighbors}(n)$

Depth First Search (DFS)

- $V = \emptyset$
- $Q = \emptyset_{stack}$
- $s \Rightarrow Q$
- while $Q \neq \emptyset_{stack}$
  - $n \Leftarrow Q$
  - if $n \notin V$
    - check n for objective
    - $n \Rightarrow V$
    - $\text{neighbors}(n) \Rightarrow Q$

- How do you implement these algorithms (hint: queue/stack)?
- How do BFS/DFS find the *shortest path* between two nodes?

Implementation

Breadth First Search (BFS)

- $V = \emptyset$
- $Q = \emptyset_{queue}$
- $Q \Leftarrow s$
- while $Q \neq \emptyset_{queue}$
  - $n \Leftarrow Q$
  - if $n \notin V$
    - check n for objective
    - $n \Rightarrow V$
    - $Q \Leftarrow \text{neighbors}(n)$

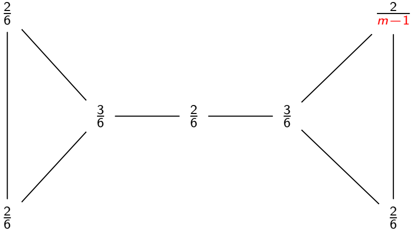
Depth First Search (DFS)

- $V = \emptyset$
- $Q = \emptyset_{stack}$
- $s \Rightarrow Q$
- while $Q \neq \emptyset_{stack}$
  - $n \Leftarrow Q$
  - if $n \notin V$
    - check n for objective
    - $n \Rightarrow V$
    - $\text{neighbors}(n) \Rightarrow Q$

- How do you implement these algorithms (hint: queue/stack)?
- How do BFS/DFS find the *shortest path* between two nodes?
- How do BFS/DFS find the *extended network* of a node?

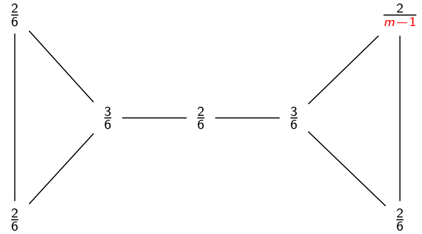
Centrality

Normalized Degree:

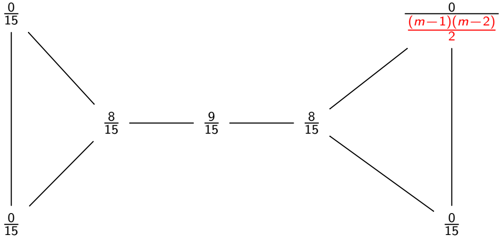


Centrality

Normalized Degree:



Normalized Betweenness:



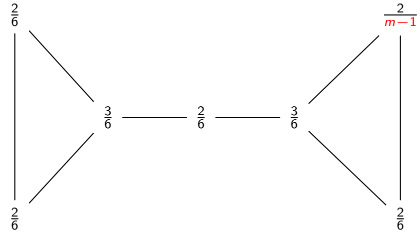
Shorted Paths from s to t passing trough n are denoted as $\sigma_{st}(n)$

Betweenness of n is

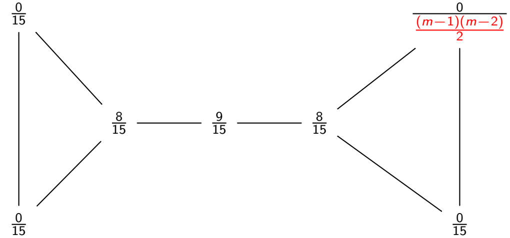
$$\sum_{s \neq n \neq t} \frac{|\sigma_{st}(n)|}{|\sigma_{st}(s)|}$$

Centrality

Normalized Degree:



Normalized Betweenness:

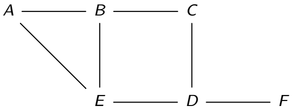


Shorted Paths from s to t passing trough n are denoted as $\sigma_{st}(n)$

Betweenness of n is

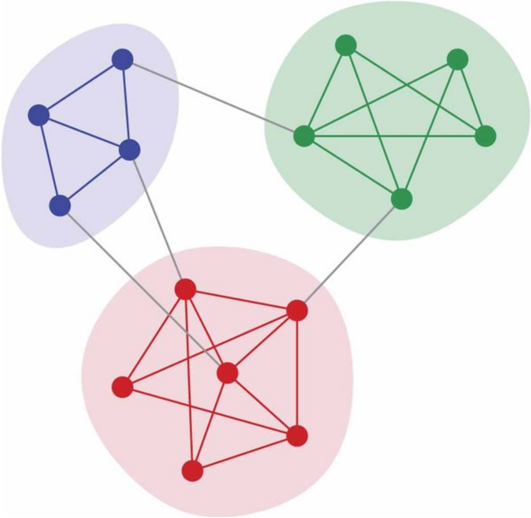
$$\sum_{s \neq n \neq t} \frac{|\sigma_{st}(n)|}{|\sigma_{st}(s)|}$$

What's the *normalized betweenness* of node D in this graph?



Community Detection

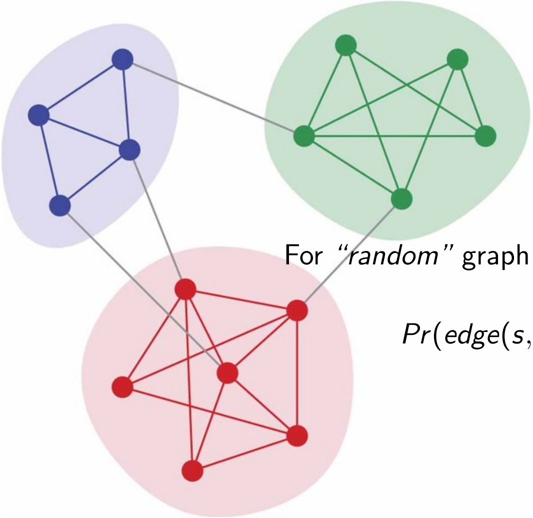
Modularity:
Measures edges *within* versus *between* groups



So a proposed *community partitioning* of a graph should have a high *modularity* score

Community Detection

Modularity:
Measures edges *within* versus *between* groups



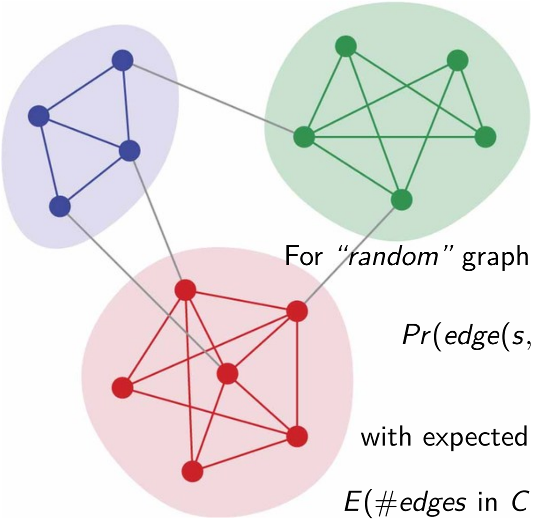
So a proposed *community partitioning* of a graph should have a high *modularity* score

For *"random"* graph G_R the connection chance is

$$Pr(edge(s,t)|G_R) = \frac{D(s)D(t)}{2m-1}$$

Community Detection

Modularity:
Measures edges *within* versus *between* groups



So a proposed *community partitioning* of a graph should have a high *modularity* score

For *"random"* graph G_R the connection chance is

$$Pr(edge(s,t)|G_R) = \frac{D(s)D(t)}{2m-1}$$

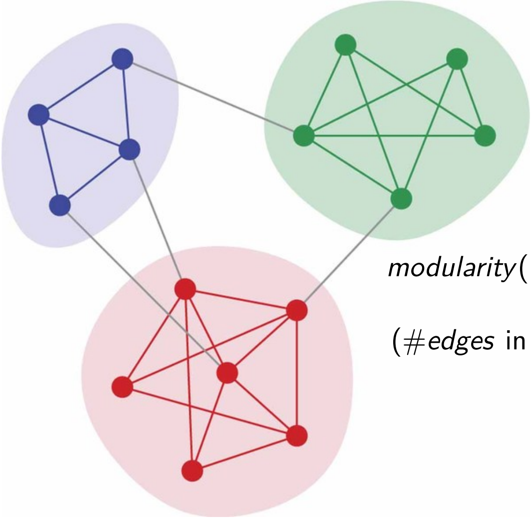
with expected number of community edges

$$E(\#edges \text{ in } C|G_R) = \sum_{s,t \in C} Pr(edge(s,t)|G_R)$$

Community Detection

Modularity:

Measures edges *within* versus *between* groups



So a proposed *community partitioning* of a graph should have a high *modularity* score

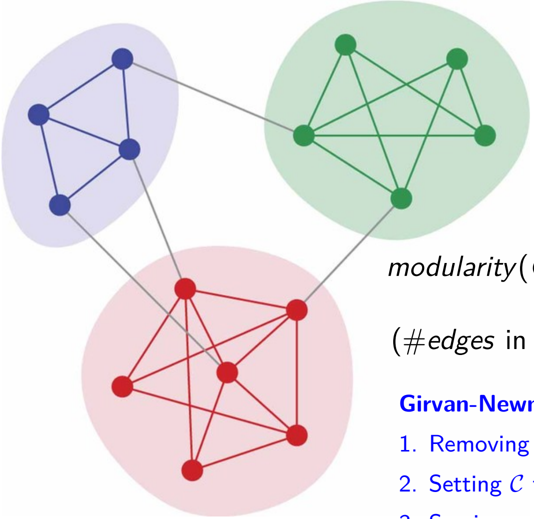
$$modularity(G, \mathcal{C}) = \frac{1}{m} \sum_{C \in \mathcal{C}}$$

$$(\#edges \text{ in } C|G) - E(\#edges \text{ in } C|G_R)$$

Community Detection

Modularity:

Measures edges *within* versus *between* groups



So a proposed *community partitioning* of a graph should have a high *modularity* score

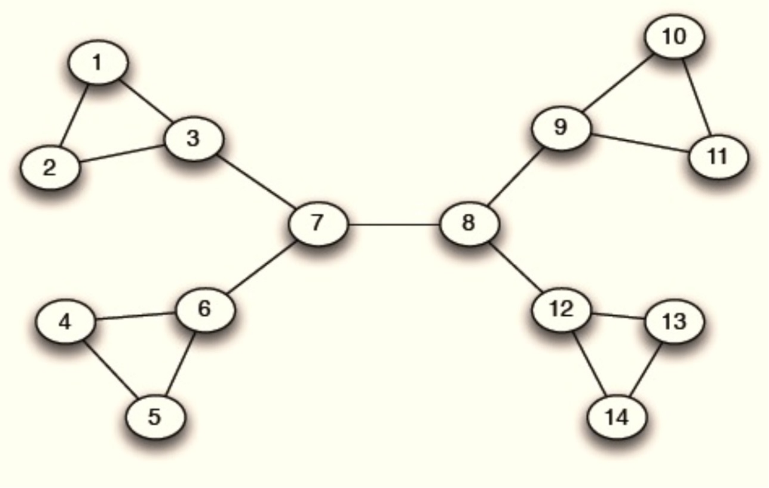
$$modularity(G, \mathcal{C}) = \frac{1}{m} \sum_{C \in \mathcal{C}}$$

$$(\#edges \text{ in } C|G) - E(\#edges \text{ in } C|G_R)$$

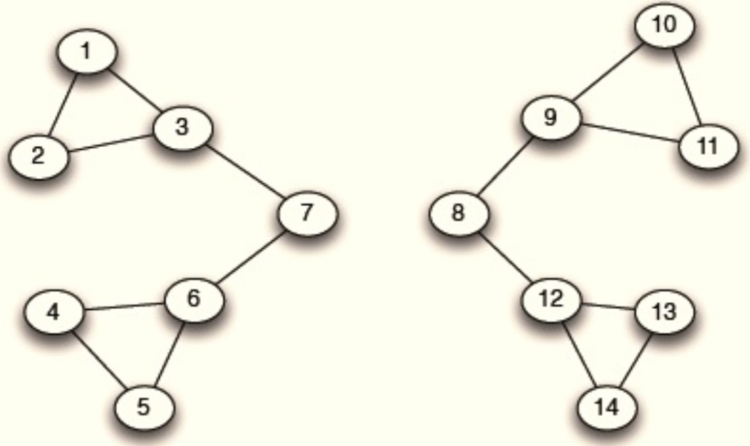
Girvan-Newman: exhaust all edges by

1. Removing highest edge-betweenness edge
2. Setting $\mathcal{C}$ to be the *unconnected subgraphs*
3. Scoring $modularity(G, \mathcal{C})$; Return to step 1

Girvan-Newman Algorithm

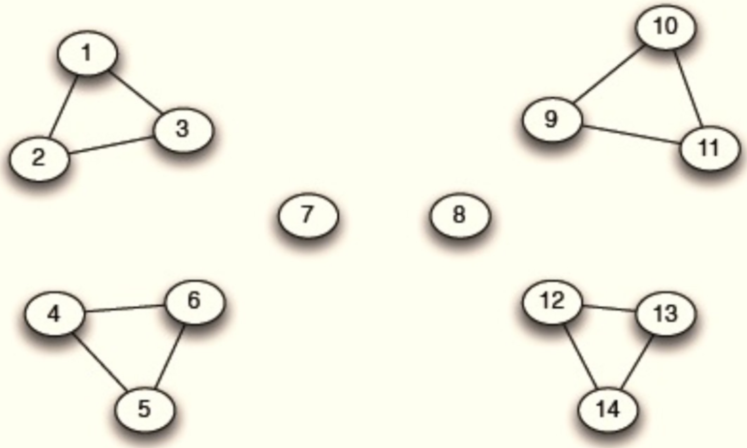


Girvan-Newman Algorithm



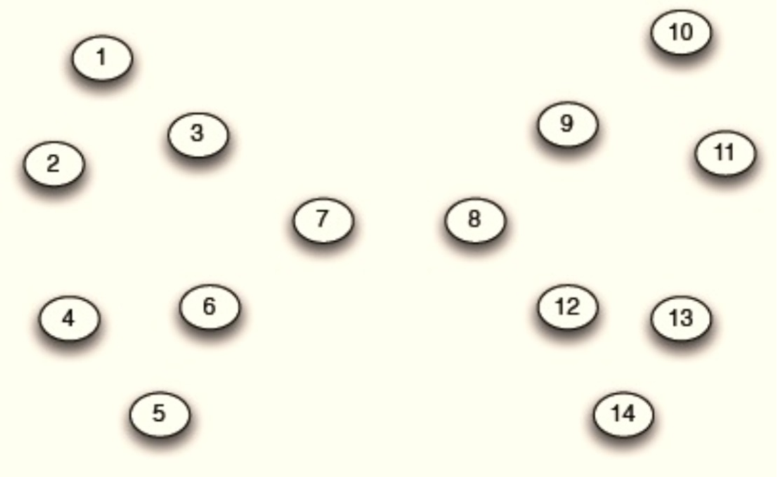
(a) *Step 1*

Girvan-Newman Algorithm

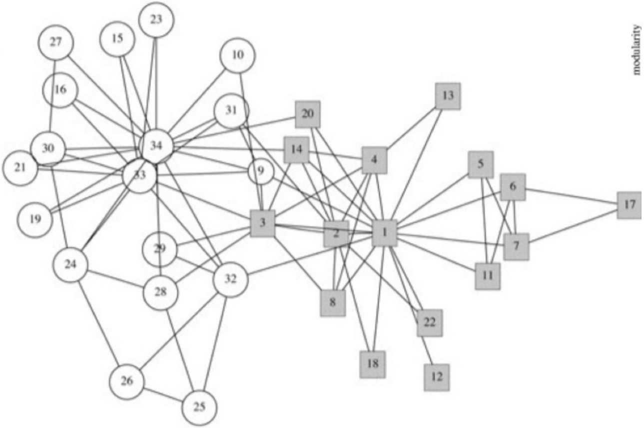


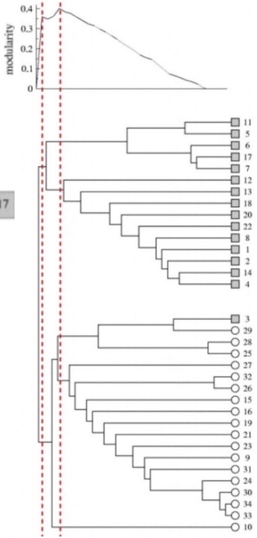
(b) *Step 2*

Girvan-Newman Algorithm



Girvan-Newman Algorithm





Modularity without recalculation