Social Network Analysis -- Community Detection

Instructor: Jen-Wei Huang

Office: 92528 in the EE building
jwhuang@mail.ncku

Outline

- Minimum-cut method
  - Kernighan-Lin (KL) Algorithm
- Hierarchical clustering
  - Hierarchical clustering
- Edge-Removal
  - Girvan-Newman (GN) Algorithm
  - Bridge cut algorithm
- Density-based
  - SCAN Algorithm
- Ranking-based
  - C-Rank Algorithm
- Model-based
  - EM Algorithm

Minimum-cut method Kernighan-Lin (KL) Algorithm

B. Kernighan and S. Lin,
"An Efficient Heuristic Procedure for Partitioning of Electrical Circuits."
Bell System Technical Journal, pp291-307, 1970

Problem definition

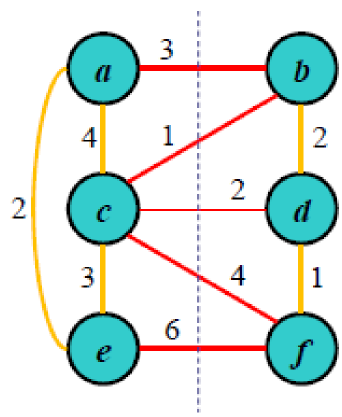
- Input: A weighted graph G = (V, E) with
  - Vertex set V (|V| = 2n)
  - Edge Set E. (|E| = e)
  - Cost c(A,B) for each edge {A, B} in E
- Output: 2 equal-size partitions X & Y such that
  - Minimizing total cost of edges "crossing" the partition
  - Maximizing the internal cost

Idea of KL algorithm

- Initial: Community separates into two partitions X and Y
- Steps: trying to reduce the external cost T by a series of interchanges of elements in X and Y
- Stop: when no further improvement can be obtained

Idea of 2-way Partitioning

- Let a∈A,
  - An external cost $E_a = \sum_{y \in \mathrm{B}} C_{ay}$
  - An internal cost $I_a = \sum_{x \in \mathrm{A}} C_{ax}$
- Let $D_z = E_z - I_z$ for a node z
  - The difference between its external and internal costs

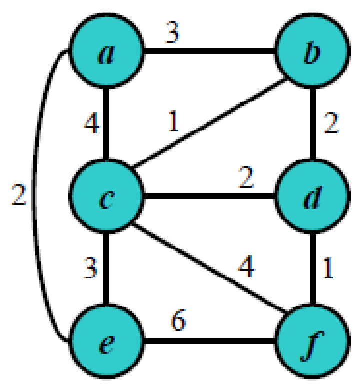


Idea of 2-way Partitioning

- Lemma 1: **consider any a $\in$ A, b$\in$ B. If a and b are interchanged, the gain (i.e., the reduction in cost) is precisely Da+Db-2Cab**
  - Let z be the total cost due to all connections between A and B that do not involve a or b
  - Then the cost T = z + Ea + Eb – Cab
  - Exchange a and b; let T' be the new cost
  - We obtain T' = z + Ia + Ib + Cab
  - Gain = old cost – new cost = T – T' = Da + Db – 2Cab

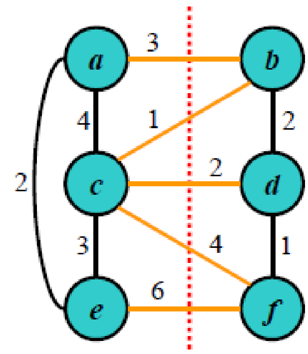
Kernighan-Lin Algorithm

- Given: Initial weighted graph G with V(G) = { a, b, c, d, e, f }
- Start with any partition of V(G) into A and B,
  - A={a, c, e}
  - B={b, d, f}



Kernighan-Lin Algorithm

- Compute the D-values
  - $D_a = E_a - I_a = 3 - 4 - 2 = -3$
  - $D_c = E_c - I_c = 1 + 2 + 4 - 4 - 3 = 0$
  - $D_e = E_e - I_e = 6 - 2 - 3 = +1$
  - $D_b = E_b - I_b = 3 + 1 - 2 = +2$
  - $D_d = E_d - I_d = 2 - 2 - 1 = -1$
  - $D_f = E_f - I_f = 4 + 6 - 1 = +9$

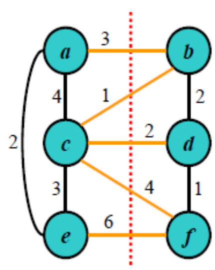


Cut-size =16
A={a, c, e}
B={b, d, f}

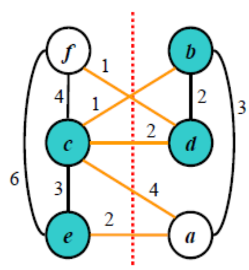
Kernighan-Lin Algorithm

- Compute the gains
  - $g_{ab} = D_a + D_b - 2w_{ab} = -7$
  - $g_{ad} = D_a + D_d - 2w_{ad} = -4$
  - $g_{af} = D_a + D_f - 2w_{af} = +6$
  - $g_{cb} = D_c + D_b - 2w_{cb} = 0$
  - $g_{cd} = D_c + D_d - 2w_{cd} = -5$
  - $g_{cf} = D_c + D_f - 2w_{cf} = +1$
  - $g_{eb} = D_e + D_b - 2w_{eb} = +1$
  - $g_{ed} = D_e + D_d - 2w_{ed} = 0$
  - $g_{ef} = D_e + D_f - 2w_{ef} = -2$

$D_a = -3$ $D_c = 0$
$D_e = +1$
$D_b = +2$ $D_d = -1$
$D_f = +9$



cut-size = 16

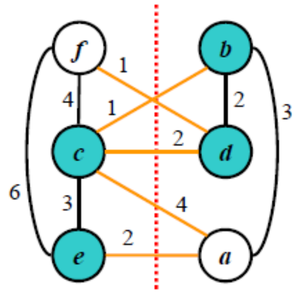


cut-size = 16 – 6 = 10

Exchange nodes a and f.
Then lock nodes a and f.

Kernighan-Lin Algorithm

- Update the D-values of unlocked nodes
  - $D'_c = D_c + 2w_{ca} - 2w_{cf} = 0$
  - $D'_e = D_e + 2w_{ea} - 2w_{ef} = -7$
  - $D'_b = D_b + 2w_{bf} - 2w_{ba} = -4$
  - $D'_d = D_d + 2w_{df} - 2w_{da} = 1$

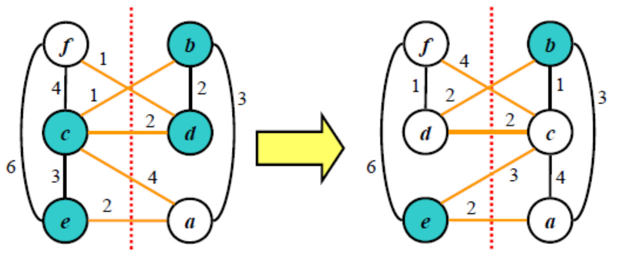


cut-size = 16 – 6 = 10

$A' = \{ c, e \}$
$B' = \{ b, d \}$

Kernighan-Lin Algorithm

- Compute the gains
  - $g_{cb} = D_c + D_b - 2w_{cb} = -6$
  - $g_{cd} = D_c + D_d - 2w_{cd} = -3$
  - $g_{eb} = D_e + D_b - 2w_{eb} = -11$
  - $g_{ed} = D_e + D_d - 2w_{ed} = -9$



cut-size = 10

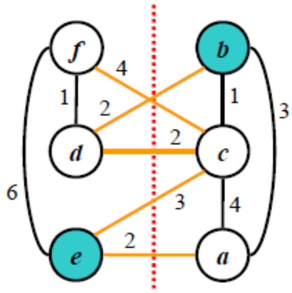
cut-size = 10 – (–3) = 13

$D'_c = 0$ $D'_e = -7$
$D'_b = -4$ $D'_d = 1$

Exchange nodes c and d.
Then lock nodes c and d.

Kernighan-Lin Algorithm

- Update the D-values of unlocked nodes
  - $D'_e = D_e + 2w_{ed} - 2w_{ec} = -1$
  - $D'_b = D_b + 2w_{bd} - 2w_{bc} = -2$
- Compute the gains
  - $g_{eb} = D_e + D_b - 2w_{eb} = -3$



cut-size = 10 – (–3) = 13

A' = { e }
B' = { b }

Kernighan-Lin Algorithm

- Summary of the Gains…
  - $G1 = +6$
  - $G1 + G2 = +6 - 3 = +3$
  - $G1 + G2 + G3 = +6 - 3 - 3 = 0$
- Maximum Gain = $G1 = +6$
- Exchange only nodes *a* and *f*
- End at the first pass.

- Repeat Kernighan-Lin Algorithm for each community

Time Complexity of KL

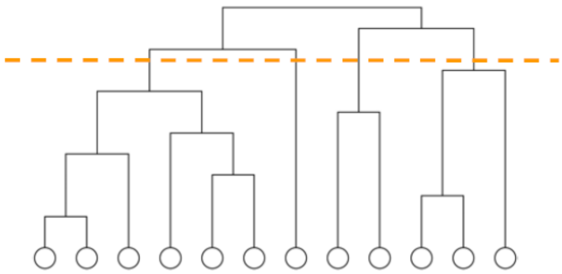
- For each pass,
  - O($\boldsymbol{n}^2$) time to find the best pair to exchange
  - *At most n* pairs exchanged
  - Total time is O(n^3) per pass

Network Hierarchical Clustering

E. Ravasz, et al.
"Hierarchical Organization of Modularity Metabolic Networks."
Science, pp.1551-1555, 2002.

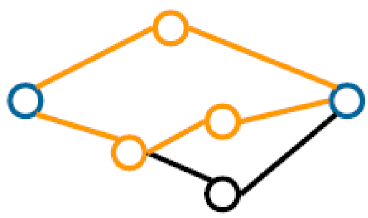
Hierarchical Clustering

- Calculate the distance matrix W for all pairs of vertices
- Apply existing **agglomerative** hierarchical clustering algorithms using W
- Result: nested components (dendogram), where one can take a slice at any level to produce communities



Distance Matrix

- Compute **weights** W_{ij} for each pair of vertices
  - [option 1] **# of node non-overlapped paths** between vertices

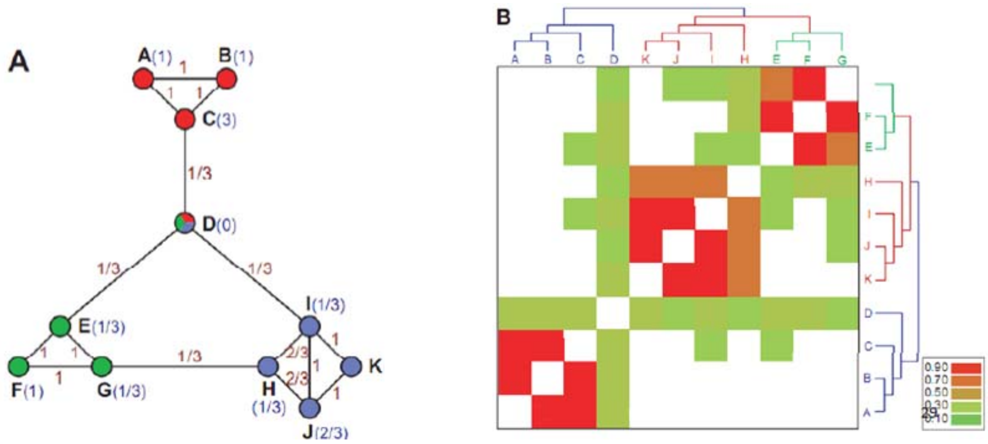


$W_{ij} = 2$

  - [option 2] **# all paths** between vertices (weighted by length of path)

Distance Matrix

- Wij = Jn(i,j)/[min(ki,kj)]
  - $J_n(i,j)$ = # of nodes that both i and j link to (+1 for linking to each other)
  - k_i is the degree of node i



Girvan-Newman (GN) Algorithm

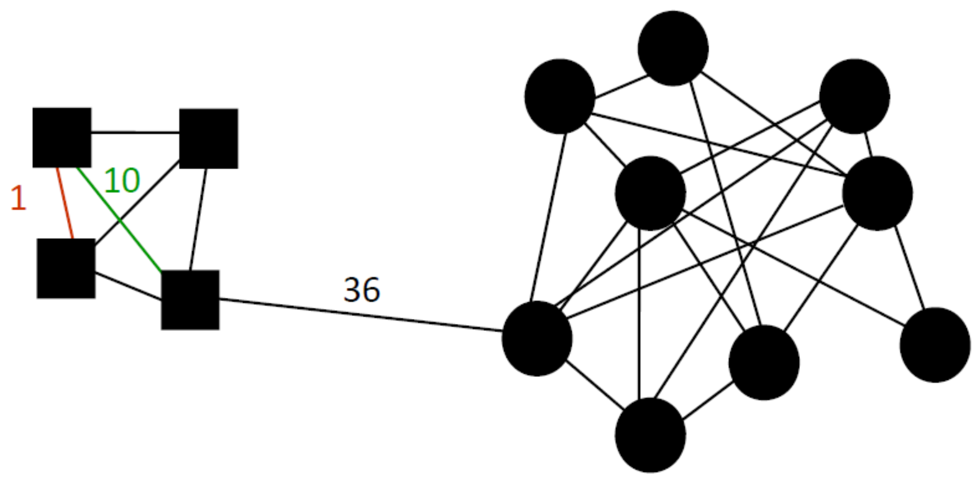
M. E. J. Newman,
"Fast algorithm for detecting community structure in networks".
Phys. Rev. , 2004.

Edge Betweenness

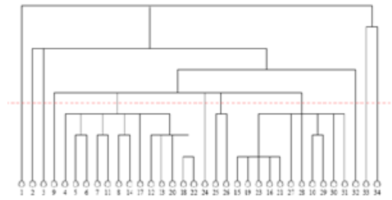
- Shortest-path
  - Number of shortest paths between vertex pairs through the edge
- Random-walk
  - Number of times a random walk between a particular pair of vertices come across an edge
- Current-flow
  - Value of the current along the edge (those with least resistance carrying the greatest fraction of the current)

Shortest-path Betweenness

- The number of shortest paths through each edge



GN Algorithm



1. Calculate betweenness for all existing edges
2. The edge with the highest betweenness is removed
3. Recalculate betweenness for edges after removal
   - If two communities are connected by more than two edges, there is generally no guarantee that all edges have high betweenness
4. Repeat step 2. and step 3. until no edges remain to create a dendogram
5. Cut down the dendrogram to identify communities
   - By removing these edges, isolated components gradually appear

Disadvantages

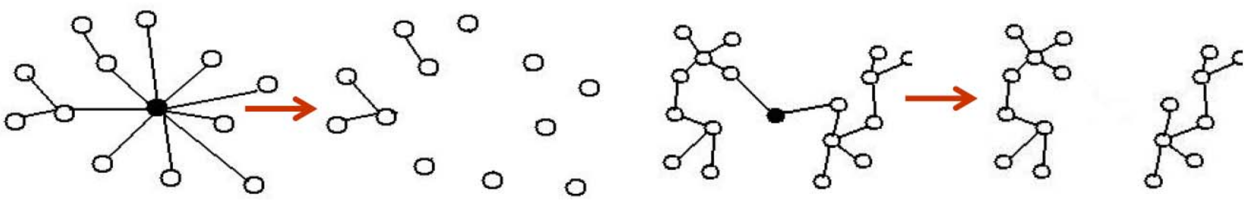
- Shortest-path calculation is **slow** and might not as proper
  - Time complexity: at least O(m^2n)
  - O(nm) for edge betweenness
- It provides no guide about when to stop (i.e. the number of communities is unknown)
  - Need to estimate how good a particular division is

Bridge-Cut Algorithm

Woochang Hwang, et. al.
"Bridging Centrality: Graph Mining from Element Level to Group Level."
KDD 2008.

Idea of Bridge cut algorithm

- Find important bottleneck between modules
- Try not to causing substantial disturbance of the network structure during removal



Terminology

- N(v): the set of directly connected nodes to v
- d(v): the degree of node v
- Density D(G)=2e/n(n-1), the density of a community shall be larger than a threshold

Bridging Centrality

- **Bridge**
  - A node or an edge
  - Connects different communities in a G
  - Can be measured through **bridging centrality**
- **Bridging Centrality**
  - For node v $C_{Br}(v) = R_{\Phi(v)} \cdot R_{\Psi(v)}$
    - $R_{\Phi(v)}$ is the rank of a node v in betweenness
    - $R_{\Psi(v)}$ is the rank of a node v in bridging coefficient
  - For edge e $C_{Br}(e) = R_{\Phi(e)} \cdot R_{\Psi(e)}$
    - $R_{\Phi(e)}$ is the rank of an edge e in betweenness
    - $R_{\Psi(e)}$ is the rank of an edge e in bridging coefficient

Betweenness Centrality

- For node v $\Phi(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$
  - σ_{st} is the number of shortest paths between node s and t
  - $\sigma_{st}(v)$ is the number of shortest paths passing through a node v out of σ_{st}
- For edge e $\Phi(e) = \sum_{s \neq t \in V} \frac{\sigma_{st}(e)}{\sigma_{st}}$
  - $\sigma_{st}(e)$ is the number of shortest paths passing through an edge e out of σ_{st}

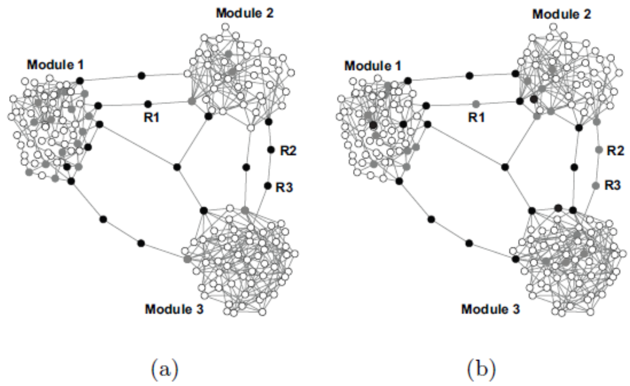
Bridging Coefficient

- For node v $\Psi(v) = \frac{1}{d(v)} \sum_{i \in \mathrm{N(v)}} \frac{\delta(i)}{d(i)-1}$
  - $d(v)$ is the degree of a node v
  - $\delta(i)$ is the number of edges leaving the direct neighbor subgraph of node v among the edges incident to each direct neighbor node i
- For edge e $\Psi(e) = \frac{d(i)\Psi(i)+d(j)\Psi(j)}{(d(i)+d(j))(|C(i,j)|+1)}$ $e(i,j) \in \mathrm{E}$
  - C (i , j) is the set of common direct neighbor nodes of nodes i and j.

Algorithm

While G != empty **do**
  Calculate bridging centrality for all edges in graph G
  topEdge = The edge with the highest bridging centrality
  remove topEdge
  if there is a new isolated module s then
    if density(s,G') > densityThreshold then
      ClusterList.add(s)
      G.remove(s)
    End if

  End if
End while



(a) Bridging centrality (b) Betweenness centrality

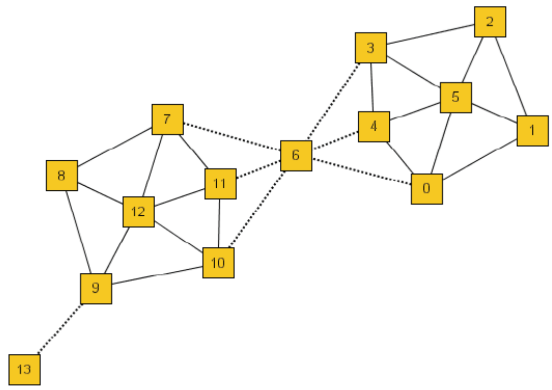
SCAN Algorithm

X. Xu, et al.
"SCAN: A Structural Clustering Algorithm for Networks."
KDD 2007

Idea of SCAN Algorithm

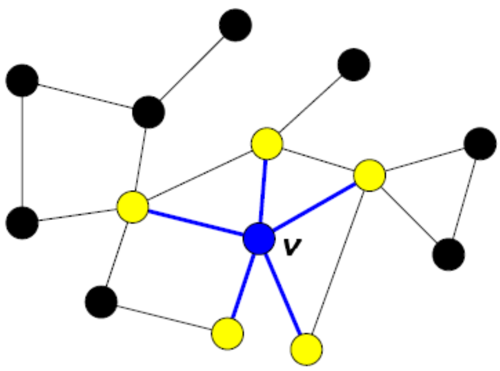
- Use the neighborhood of the vertices as clustering criteria
  - Instead of direct connections
  - **Vertices are grouped into clusters by how they share neighbors**

	Neighbors
0	{**0**,**1**,**4**,**5**,6}
5	{**0**,**1**,2,3,**4**,**5**}
9	{8,**9**,10,12,**13**}
13	{**9**,**13**}



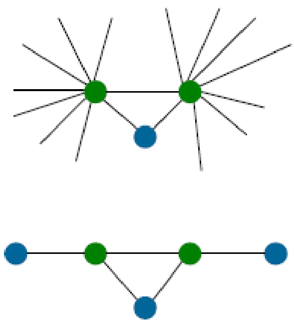
Vertex structure

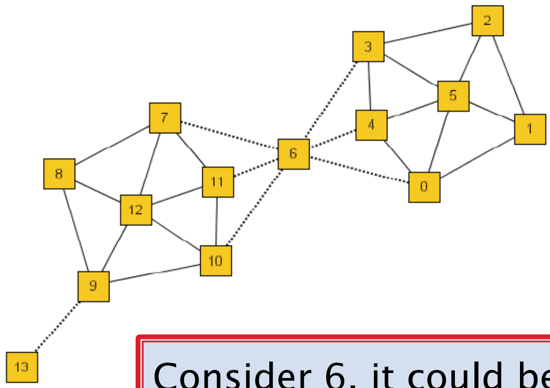
- Let $v \in V$, the structure of v is defined by its neighborhood, denoted by $\Gamma(v)$
- $\Gamma(v) = \{w \in V \mid (v,w) \in E\} \cup \{v\}$



Structural Similarity

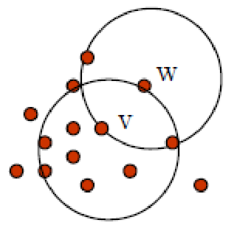
- Define the SS of two nodes as $\sigma(v,w) = \frac{|\Gamma(v)\cap\Gamma(w)|}{\sqrt{|\Gamma(v)||(\Gamma w)}}$





Consider 6, it could be clustered to either or cause the two clusters to merge **if use only shared neighbors**

Directly Structure Reachable



$\mu = 5$, $\varepsilon = 0.8$

- ϵ-Neighborhood

$$N_\varepsilon(v) = \{w \epsilon \Gamma(v) | \sigma(v,w) \geq \varepsilon\}$$

- $Core_{\varepsilon,\mu}$

$$CORE_{\varepsilon,\mu}(v) \Leftrightarrow |N_\varepsilon(v)| \geq \mu$$

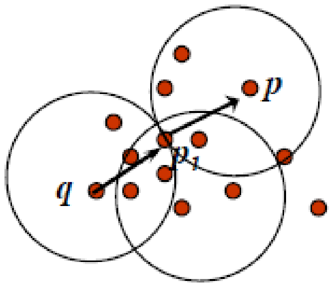
  - If a vertex is in ϵ-*neighborhood* of a core, it should be in the same cluster

- Directly Structure Reachable

$$DirREACH_{\varepsilon,\mu}(v,w) \Leftrightarrow CORE_{\varepsilon,\mu}(v) \ \wedge \ w \epsilon N_\varepsilon(v)$$

  - w is directly structure reachable from v

Structure-reachable

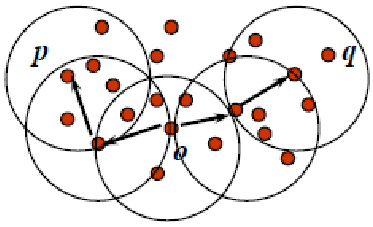


- Structure-reachable

  $$REACH_{\varepsilon,\mu}(v,w) \Leftrightarrow$$
  $$\exists v_1 \ldots v_n \in V: v_1 \in v \wedge v_n \in w \wedge$$
  $$\forall i \in \{1,\ldots,n-1\}: DirREACH_{\varepsilon,\mu}(v,w)(v_i,v_{i+1})$$

  - Transitive closure of direct structure reachability
- Structure-connected

  $$CONNECT_{\varepsilon,\mu}(v,w) \Leftrightarrow$$
  $$\exists u \in V: REACH_{\varepsilon,\mu}(u,v) \wedge REACH_{\varepsilon,\mu}(u,w)$$

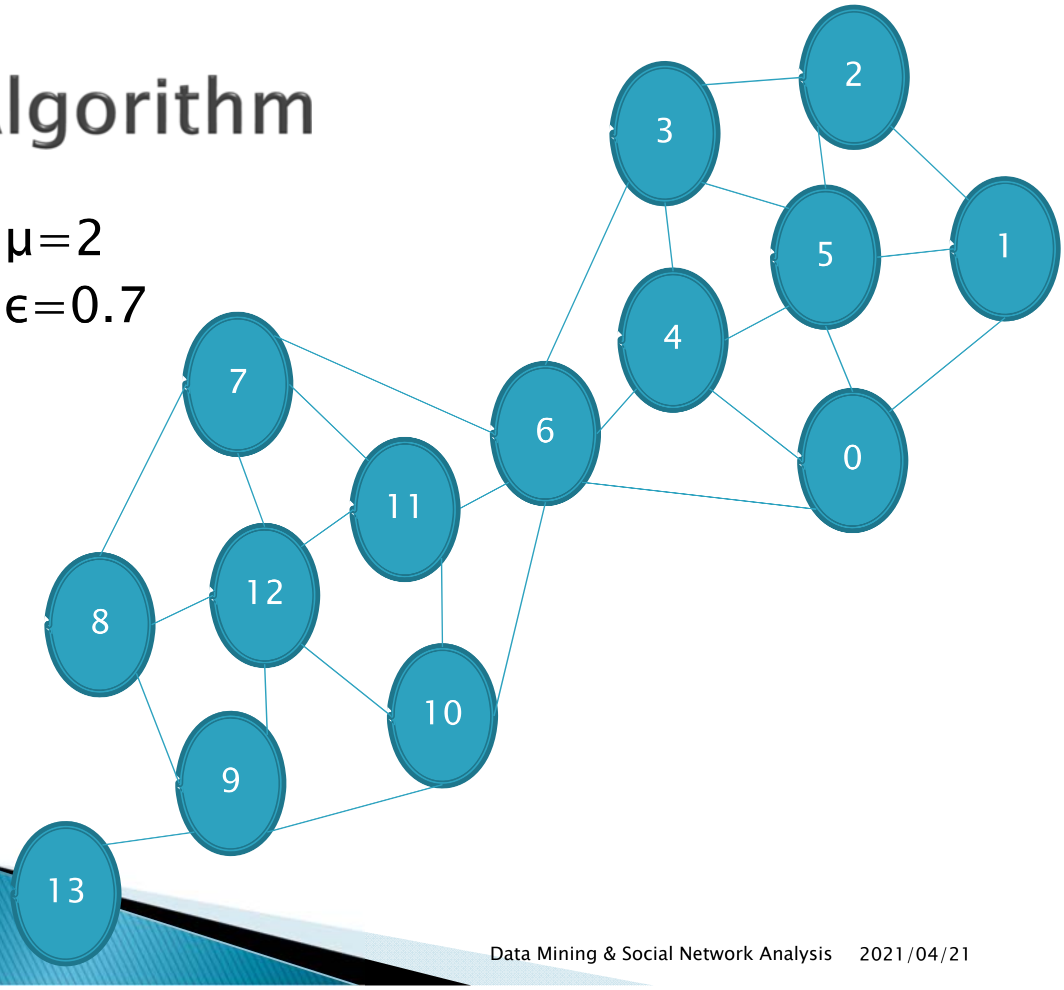


Structure-connected cluster

- Structure-connected cluster
- $CLUSTERING_{\varepsilon,\mu}(P) \Leftrightarrow P = \{C \subseteq V | CLUSTERING_{\varepsilon,\mu}(C)\}$
- Hub
  - Not belong to any cluster
  - Bridge to many clusters
- Outlier
  - Not belong to any cluster
  - Connect to less clusters

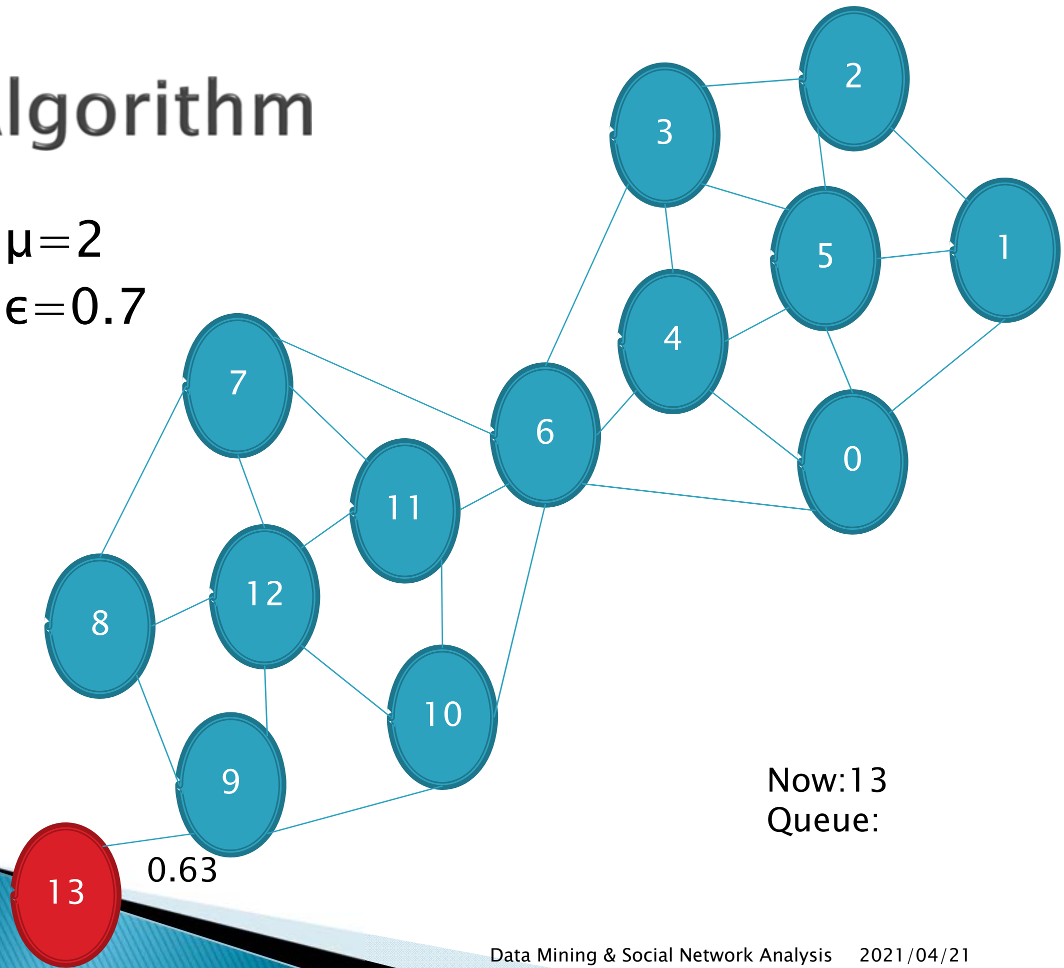
Algorithm

- μ=2
- ϵ=0.7



Algorithm

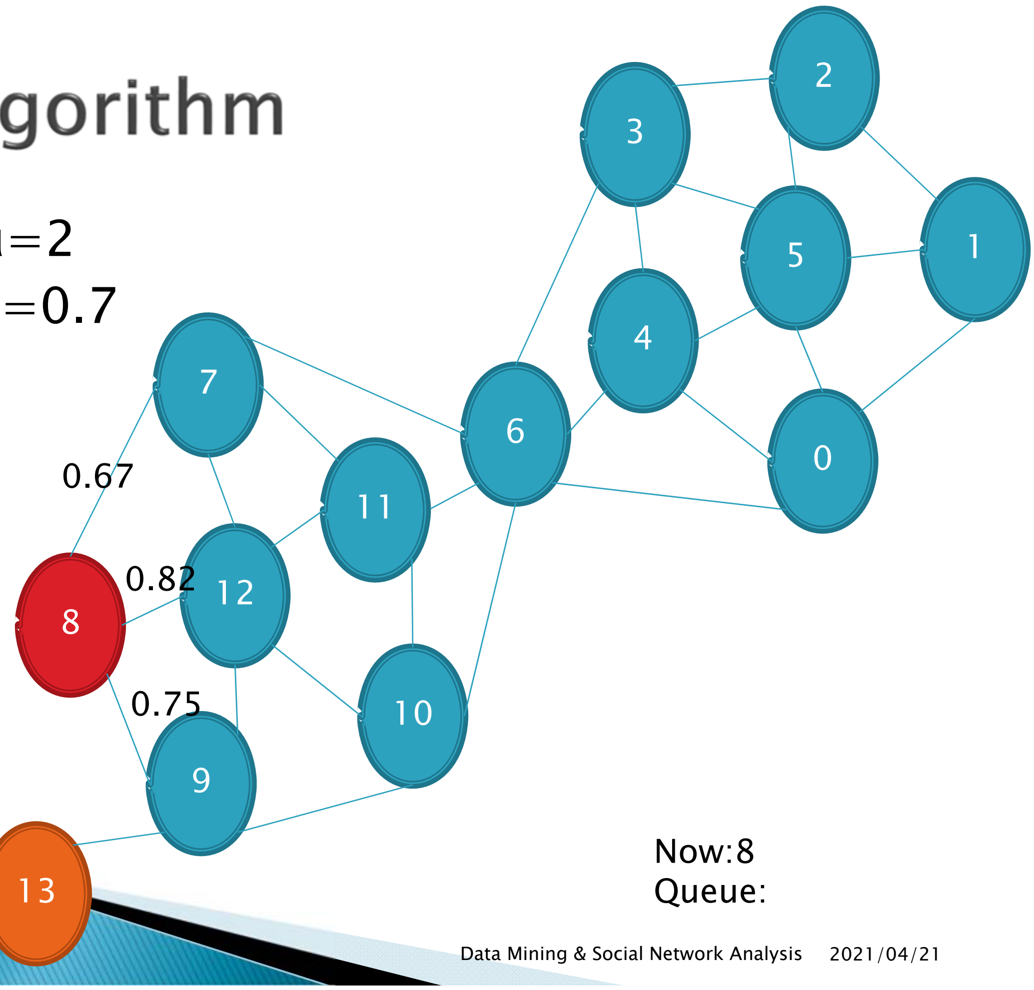
- μ=2
- ϵ=0.7



Now:13
Queue:

Algorithm

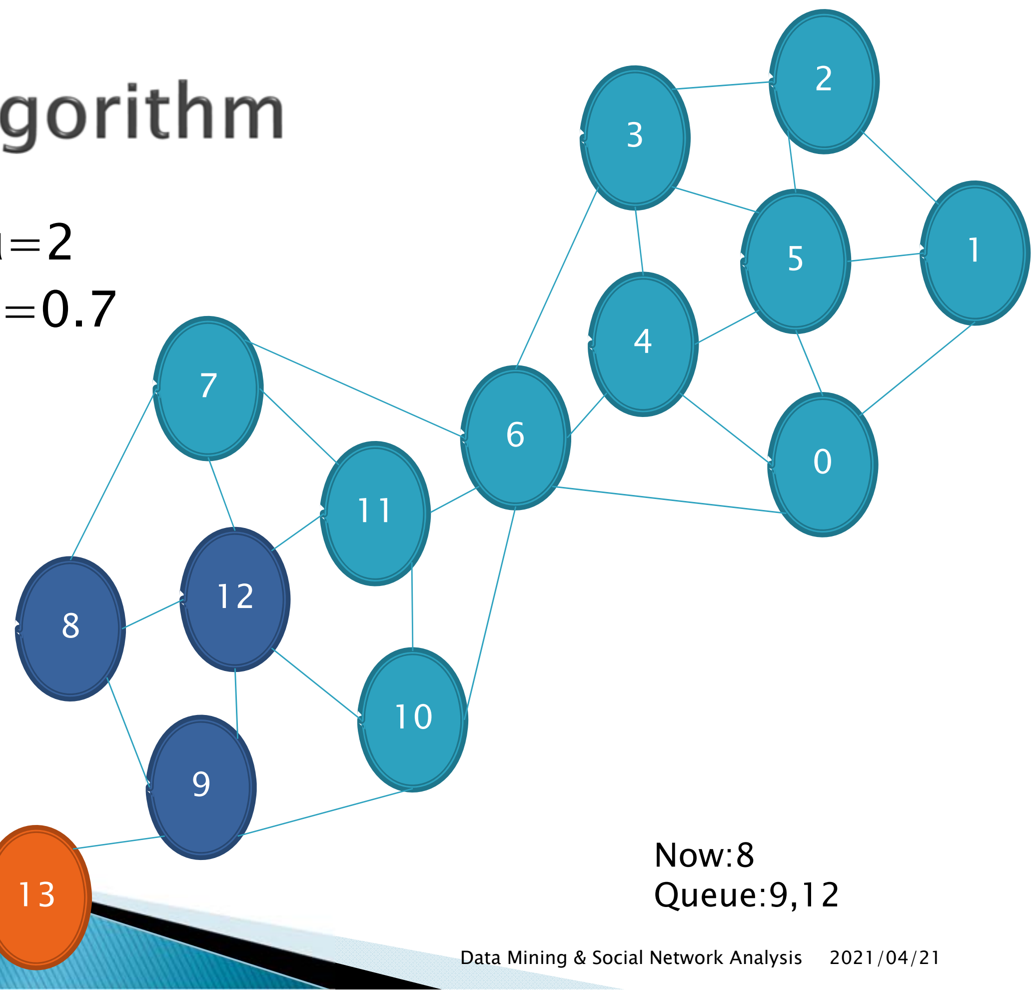
- μ=2
- ϵ=0.7



Now:8
Queue:

Algorithm

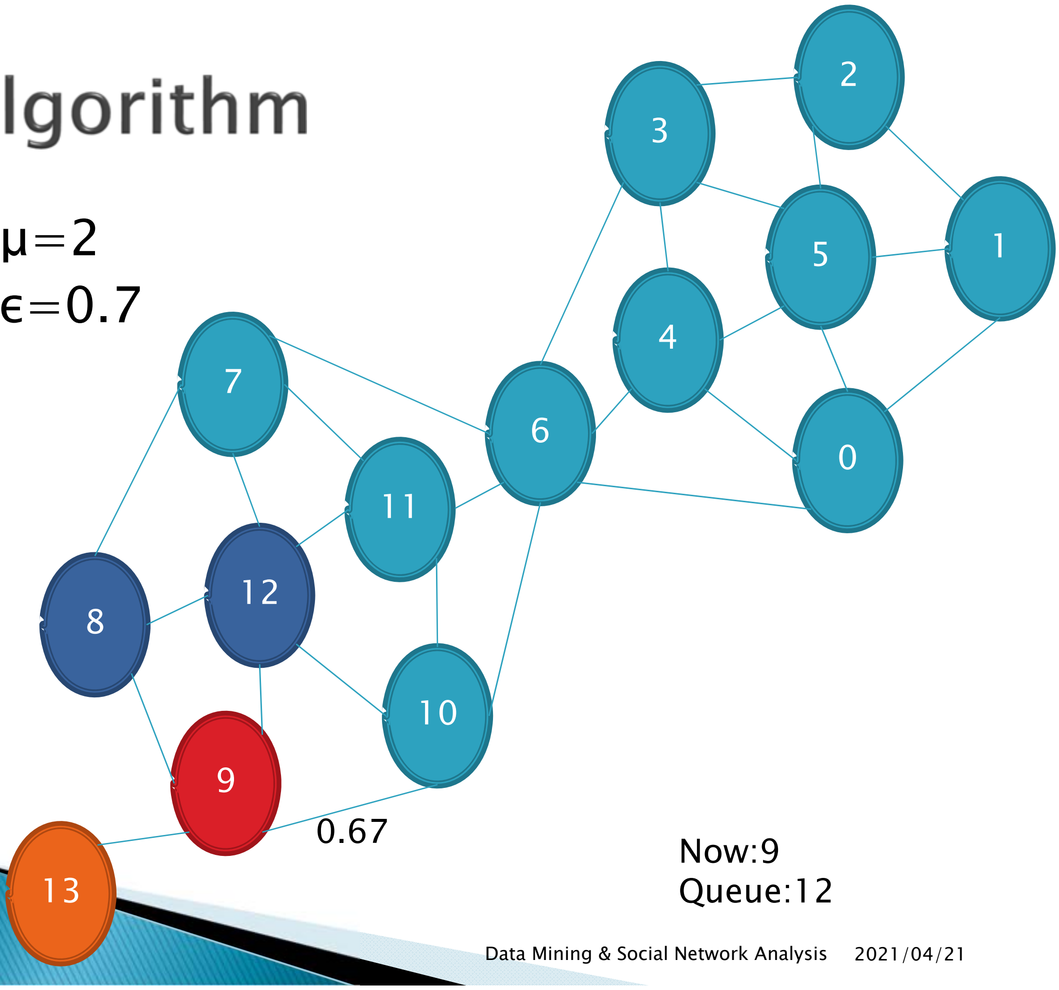
- μ=2
- ϵ=0.7



Now:8
Queue:9,12

Algorithm

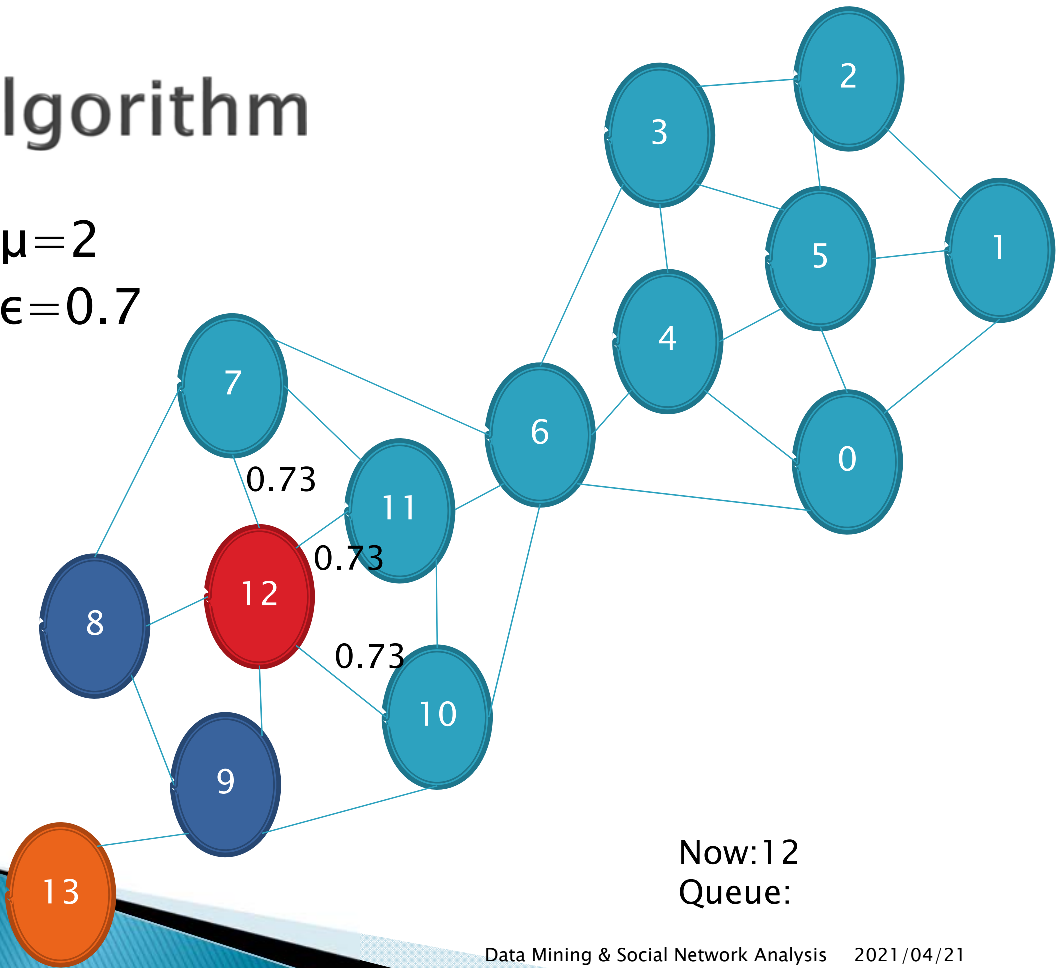
- μ=2
- ϵ=0.7



Now:9
Queue:12

Algorithm

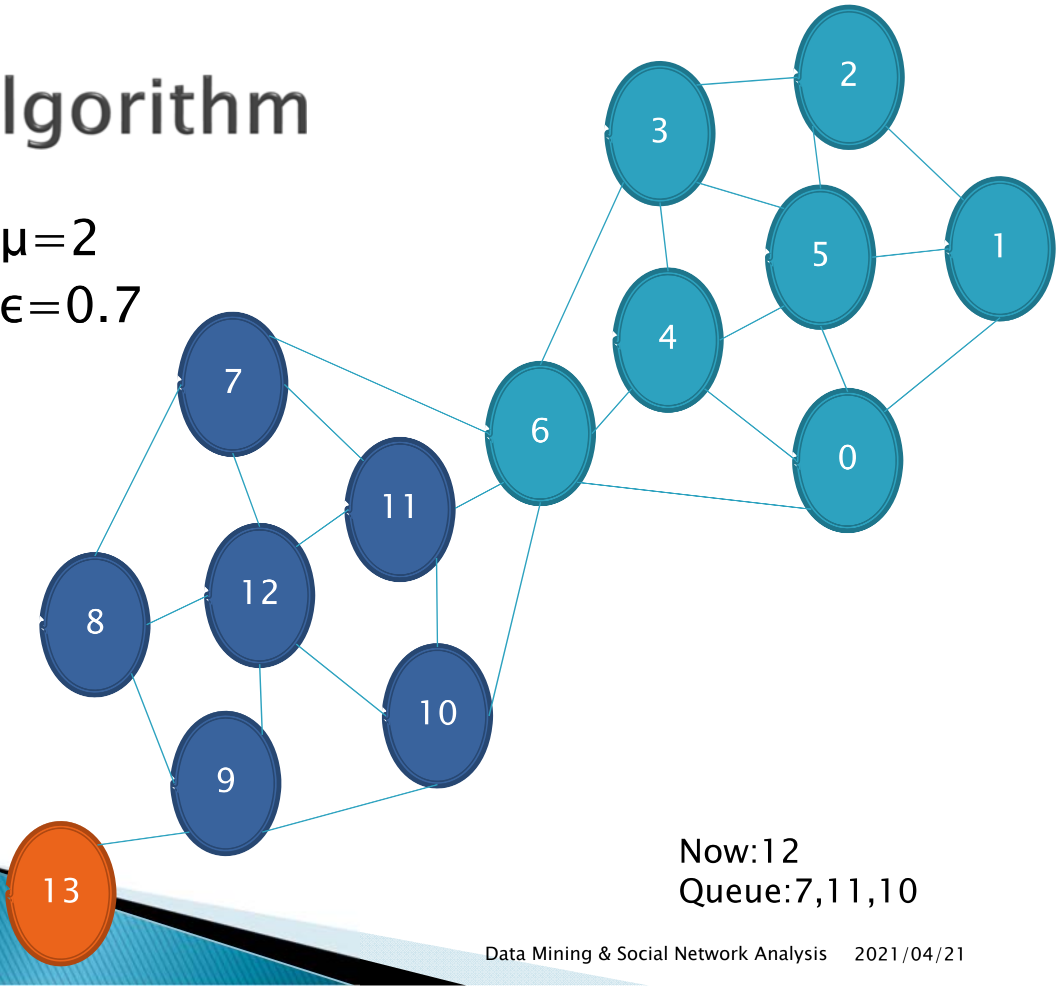
- μ=2
- ϵ=0.7



Now:12
Queue:

Algorithm

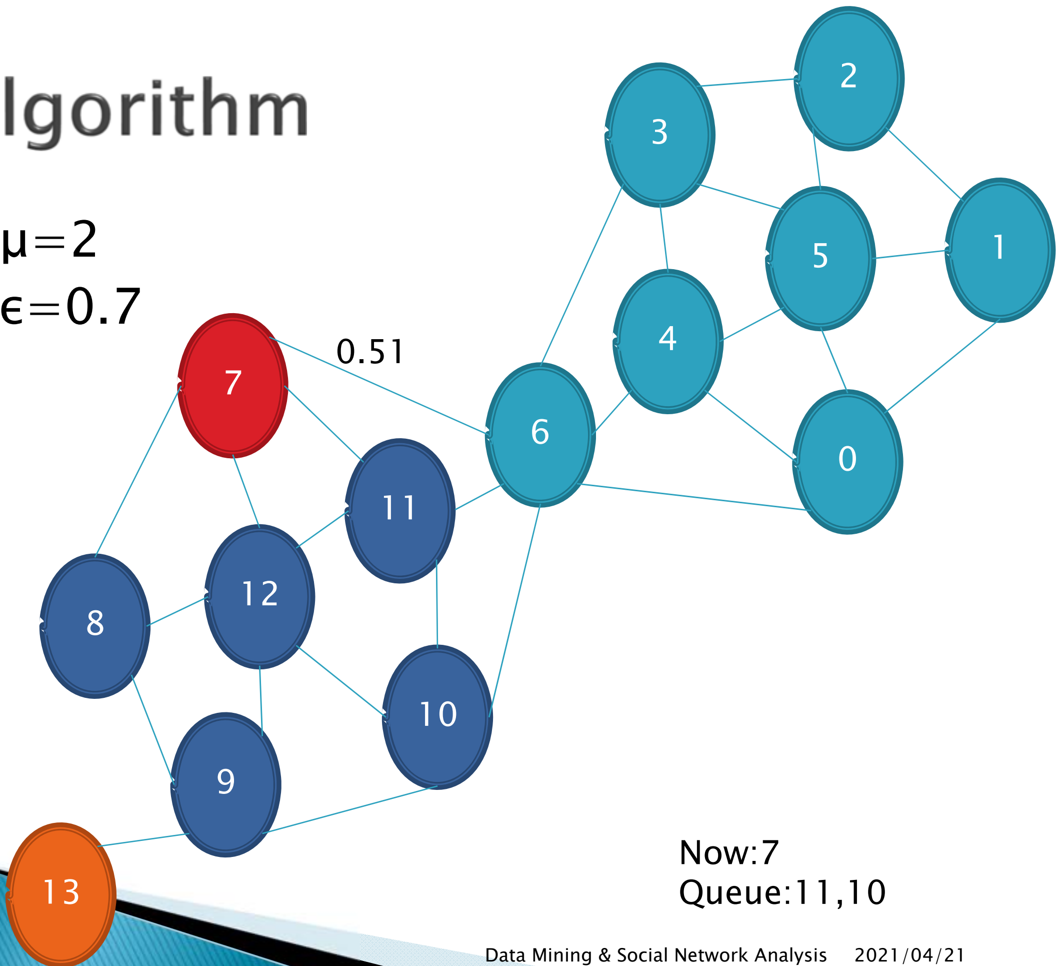
- μ=2
- ϵ=0.7



Now:12
Queue:7,11,10

Algorithm

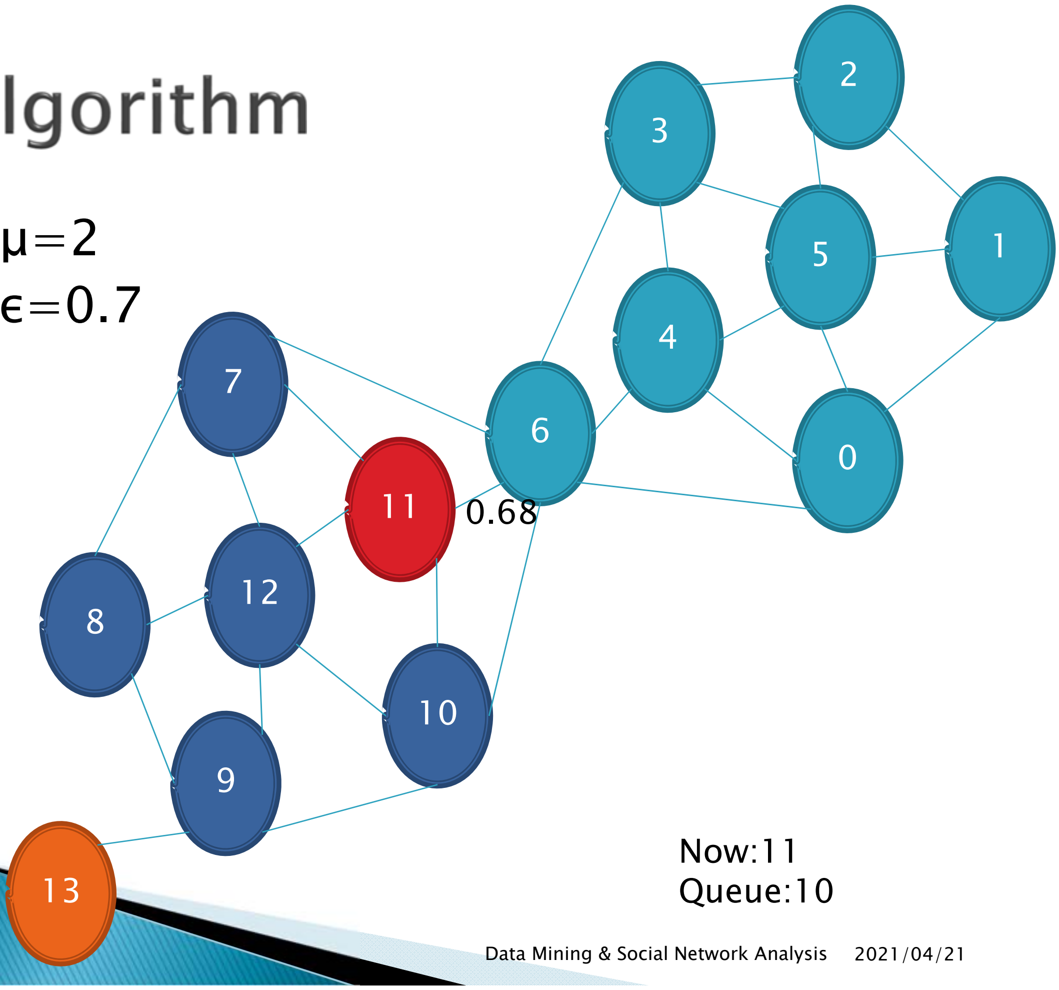
- μ=2
- ϵ=0.7



Now:7
Queue:11,10

Algorithm

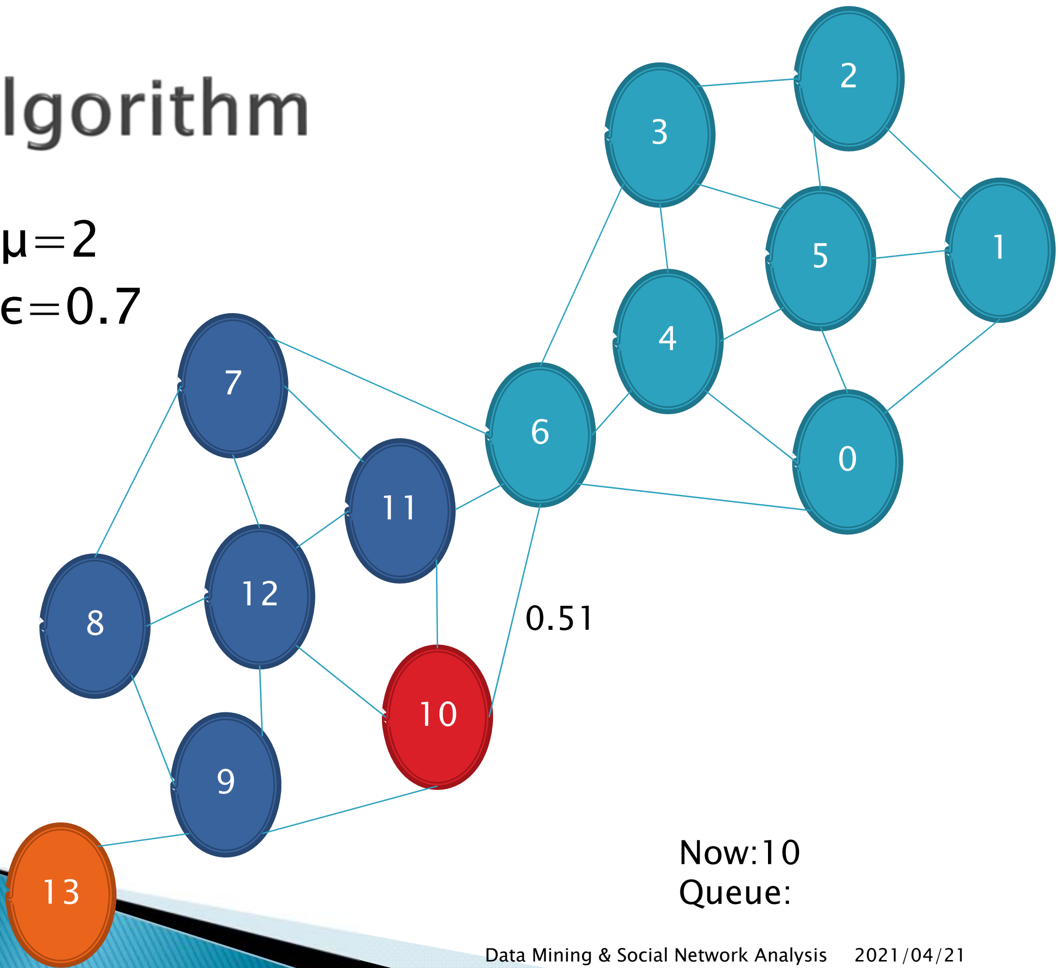
- μ=2
- ϵ=0.7



Now:11
Queue:10

Algorithm

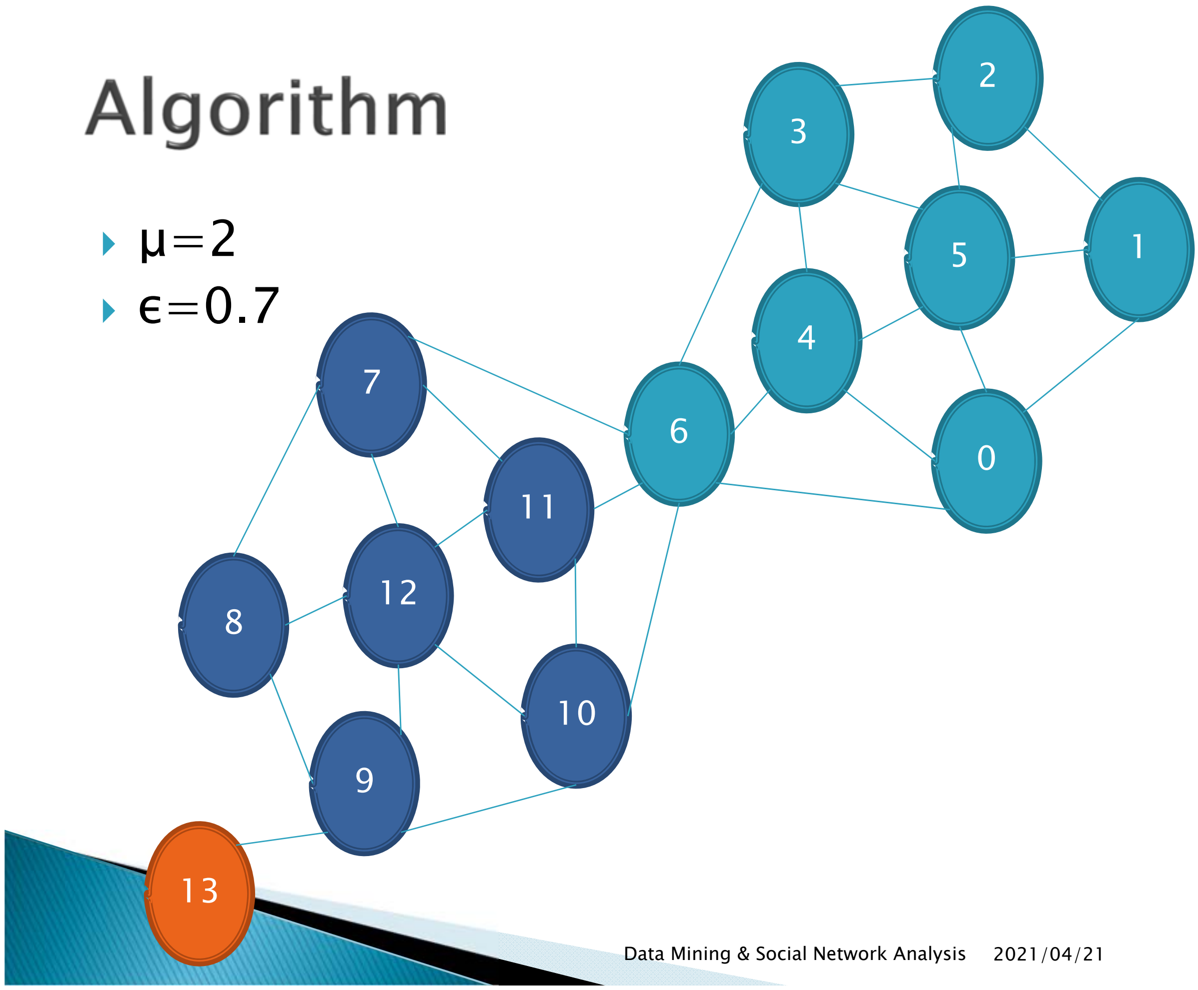
- μ=2
- ϵ=0.7



Now:10
Queue:

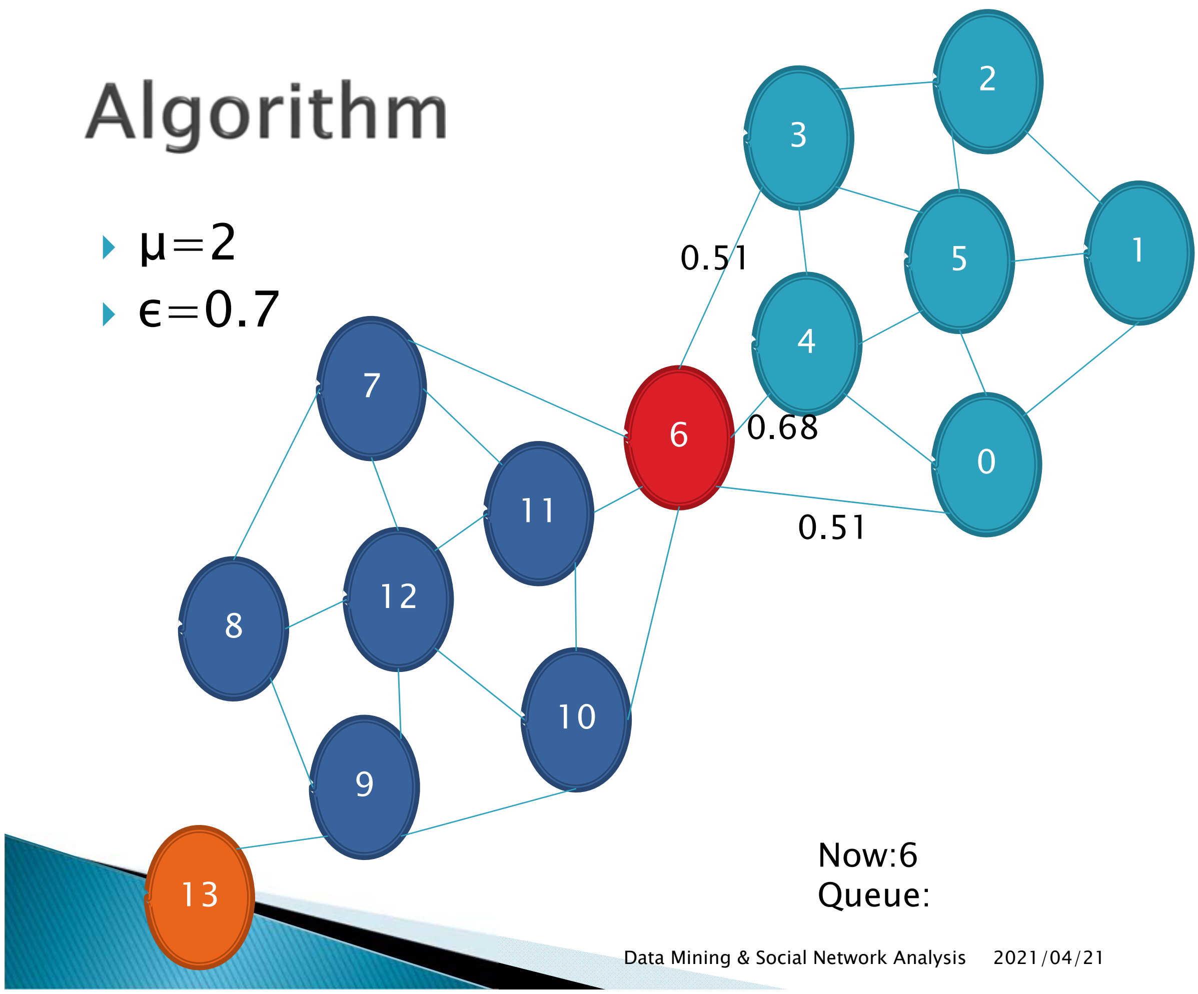
Algorithm

- μ=2
- ϵ=0.7



Algorithm

- μ=2
- ϵ=0.7



Algorithm

- $\mu=2$
- $\epsilon=0.7$