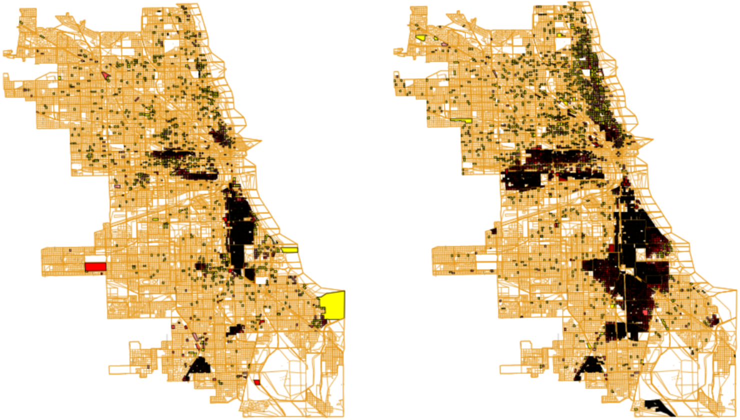
Detecting Community Structures in Social Networks by Graph Sparsification

Partha Basuchowdhuri, **Satyaki Sikdar**, Sonu Shreshtha, Subhasis Majumder

Department of Computer Science and Engineering,
Heritage Institute of Technology, Kolkata, India



(a) *Chicago, 1940* (b) *Chicago, 1960*

Figure: The tendency of people to live in racially homogeneous neighborhoods[1]. In yellow and orange blocks % of Afro-Americans ≤ 25, in brown and black boxes % ≥ 75.

Definition of a Community

For a given graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, find a cover $\mathbb{C} = \{C_1, C_2, ..., C_k\}$ such that $\bigcup_i C_i = \mathcal{V}$.

- For disjoint communities, $\forall i, j$ we have $C_i \bigcap C_j = \emptyset$
- For overlapping communities, $\exists i, j$ where $C_i \bigcap C_j \neq \emptyset$

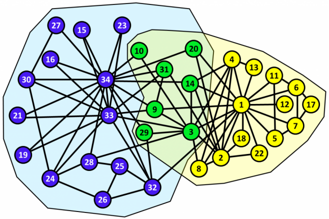


Figure: Zachary's Karate Club Network

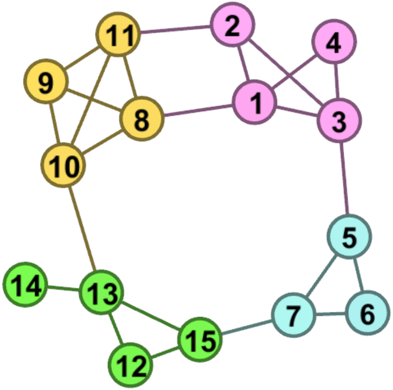
$\mathbb{C} = \{C_1, C_2, C_3\}$, C_1 = yellow nodes, C_2 = green, C_3 = blue is a disjoint cover

However, $\bar{\mathbb{C}} = \{\bar{C_1}, \bar{C_2}\}$, $\bar{C_1}$ = yellow & green nodes and $\bar{C_2}$ = blue & green nodes is an overlapping cover

For our problem, we concentrate on *disjoint* community detection

A Little Background: Edge Betweenness Centrality

$$c_B(e) = \sum_{\substack{s,t \in \mathcal{V} \\ s \neq t}} \frac{\sigma(s,t \mid e)}{\sigma(s,t)}$$



Top 6 edges

Edge	$c_B(e)$	Type
(10, 13)	0.3	inter
(3, 5)	0.23333	inter
(7, 15)	0.2079	inter
(1, 8)	0.1873	inter
(13, 15)	0.1746	intra
(5, 7)	0.1476	intra

Bottom 6 edges

Edge	$c_B(e)$	Type
(8, 11)	0.022	intra
(1, 2)	0.0269	intra
(9, 11)	0.031	intra
(8, 9)	0.0412	intra
(12, 15)	0.052	intra
(3, 4)	0.060	intra

The Girvan-Newman Algorithm

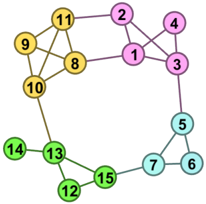
Proposed by Michelle Girvan and Mark Newman[2] in 2002

The Key Ideas

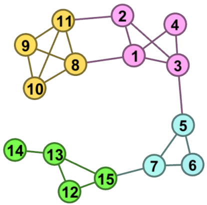
- Based on reachability of nodes - shortest paths
- Edges are selected on the basis of the edge betweenness centrality

The algorithm

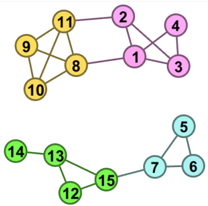
1. Compute centrality for all edges
2. Remove edge with largest centrality; ties can be broken randomly
3. Recalculate the centralities on the running graph
4. Iterate from step 2, stop when you get clusters of desirable quality



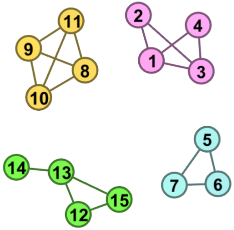
(a) Best edge: (10, 13)



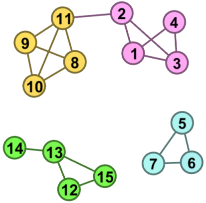
(b) Best edge: (3, 5)



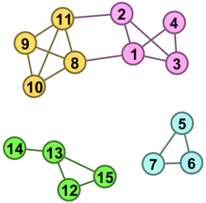
(c) Best edge: (7, 15)



(f) Final graph



(e) Best edge: (2, 11)

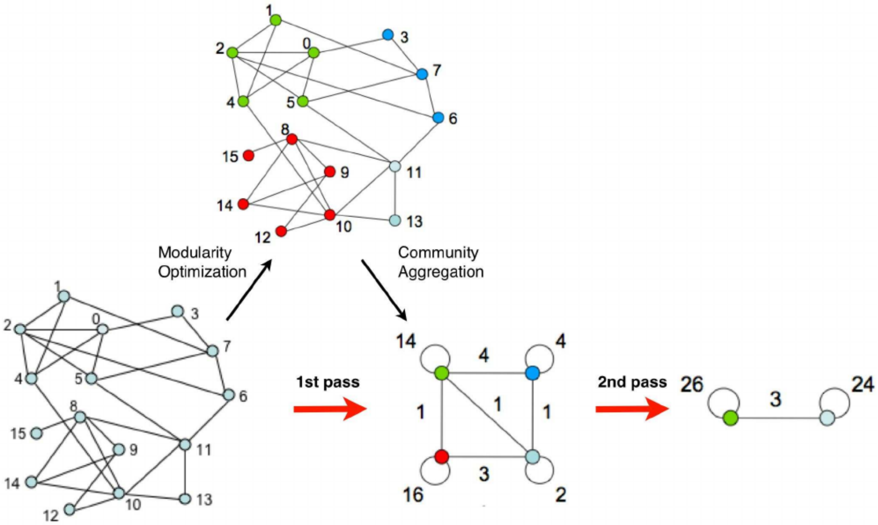


(d) Best edge: (1, 8)

Louvain Method: A Greedy Approach

- Proposed by Blondel et al[3] in 2008
- Takes the greedy maximization approach
- Very fast in practice, it's the current state-of-the-art in disjoint community detection
- Performs hierarchical partitioning, stopping when there cannot be any further improvement in modularity
- Contracts the graph in each iteration thereby speeding up the process

Louvain Method in Action



Our Method

Input: An unweighted network $\mathcal{G}(\mathcal{V}, \mathcal{E})$

Output: A disjoint cover $\mathbb{C}$

1. Use Jaccard coefficient to turn $\mathcal{G}$ into a weighted network $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{W})$
2. Construct an t-spanner of $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{W})$. Take the complement of $\mathcal{G}_S$, call it $\mathcal{G}_{comm}$
3. Use LINCOM to break $\mathcal{G}_{comm}$ into small but pure fragments
4. Use the second phase of Louvain Method to piece all the small bits and pieces together to get $\mathbb{C}$

Jaccard Intro

Definition

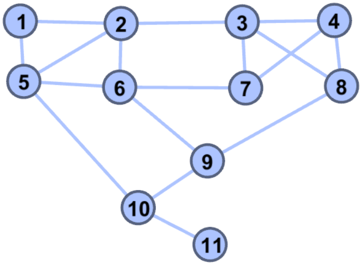
$$w_J(e(v_i, v_j)) = \frac{|\Gamma(v_i) \cap \Gamma(v_j)|}{|\Gamma(v_i) \cup \Gamma(v_j)|}$$

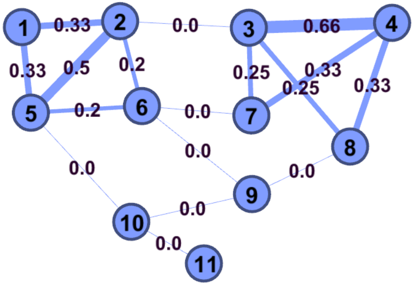
where $\Gamma(v_i)$ is the neighborhood of the node v_i

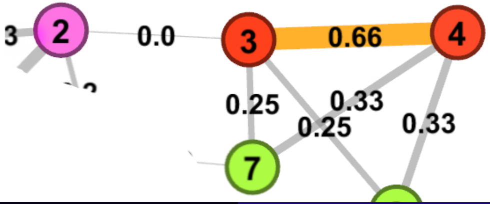
$$\therefore w_J \in [0, 1]$$

- Jaccard works well in domains where local influence is important[4][5][6]
- The computation takes $O(m)$ time

Jaccard Example







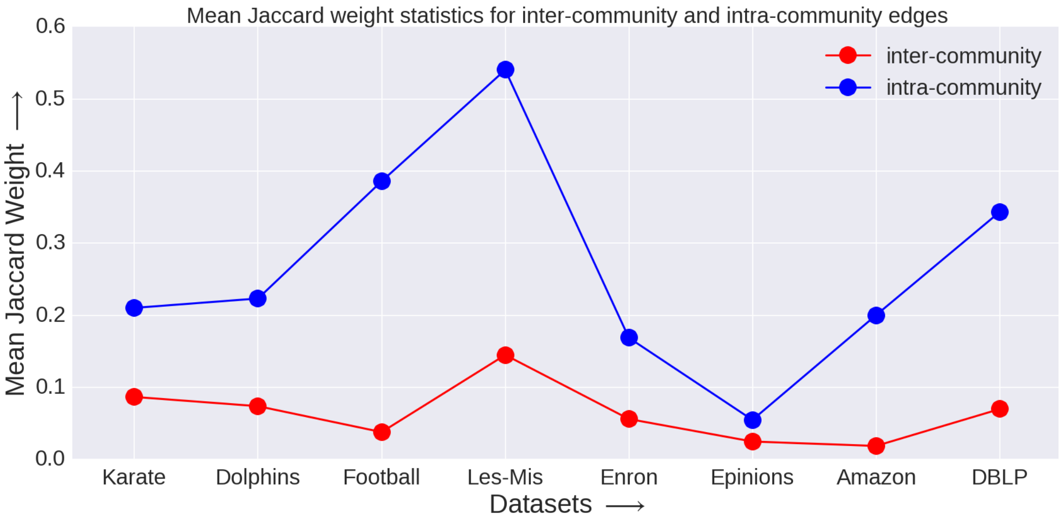

Mean Jaccard weight statistics for inter-community and intra-community edges
inter-community
intra-community
Mean Jaccard Weight ⟶
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Karate
Dolphins
Football
Les-Mis
Enron
Epinions
Amazon
DBLP
Datasets ⟶

Table: Jaccard weight statistics for top 10% edges in terms of w_J.

Network	$\|E\|$	intra-cluster edge count	top 10% edges in terms of w_J		
			Total edges	Intra-edge	Fraction
Karate	78	21	7	7	1
Dolphin	159	39	15	15	1
Football	613	179	61	61	1
Les-Mis	254	56	25	25	1
Enron	180,811	48,498	18,383	18,220	0.99113
Epinions	405,739	146,417	40,573	36,589	0.90180
Amazon	925,872	54,403	92,587	92,584	0.99996
DBLP	1,049,866	164,268	104,986	104,986	1

Spanner

- A (α, β)-spanner of a graph $\mathcal{G} = (\mathcal{V}, \ \mathcal{E}, \ \mathcal{W})$ is a subgraph $\mathcal{G}_\mathcal{S} = (\mathcal{V}, \mathcal{E}_\mathcal{S}, \mathcal{W}_\mathcal{S})$, such that,

$$\delta_S(u, v) \ \leq \ \alpha \ . \ \delta(u, v) + \beta \quad \forall \ \ u, v \in V$$

- A t-spanner is a special case of (α, β) spanner where $\alpha = t$ and $\beta = 0$

Authors	Size	Running Time
Althöfer et al. [1993] [7]	$O(n^{1+\frac{1}{k}})$	$O(m(n^{1+\frac{1}{k}} \ + \ nlogn))$
Althöfer et al. [1993] [7]	$\frac{1}{2}n^{1+\frac{1}{k}}$	$O(mn^{1+\frac{1}{k}})$
Roddity et al. [2004] [8]	$\frac{1}{2}n^{1+\frac{1}{k}}$	$O(kn^{2+\frac{1}{k}})$
Roddity et al. [2005] [9]	$O(kn^{1+\frac{1}{k}})$	$O(km)$ (det.)
Baswana and Sen [2007] [10]	$O(kn^{1+\frac{1}{k}})$	$O(km)$ (rand.)

Algorithm 1: Greedy Construction of t-spanner (GREEDY(G, t))

Input : Undirected, weighted graph $G(V, E, W)$, (t)
Output: t-spanner edge set E_S

```
begin
    E_S ← ∅
    foreach e(s, t) ∈ E, in non-decreasing order of weight do
        if d_(V,E_S)(s, t) > (2k-1)w(e(s, t)) then
            E_S ← E_S ∪ {e(s, t)}
    return E_S
```

Algorithm 2: Spanner based Community Detection (SPAN(G, k))

Input : Undirected, weighted graph $G(V, E, W)$, (k)
Output: $G_{comm} = (V, E_{comm})$

begin
  $E_S \leftarrow \emptyset$, $C_0 \leftarrow \{\{v\} | v \in V\}$, $E' \leftarrow E_{comm} \leftarrow E$
  $\mathcal{E}'_i \leftarrow \mathcal{E}_i \leftarrow \emptyset$, $\mathcal{R}_0 \leftarrow V$
  for $i \leftarrow 1$ **to** k-1 **do**
    $\mathcal{R}_i \leftarrow$ randomly pick $n^{1-\frac{i}{k}}$ clusters from C_{i-1}
    $\mathcal{E}_i \leftarrow \emptyset$, $C_i \leftarrow \mathcal{R}_i$
    forall the $e(u,v) \in \mathcal{E}_{i-1}$ **do**
      if $u, v \in c, c \in \mathcal{R}_i$ **then**
        $\mathcal{E}_i \leftarrow \mathcal{E}_i \bigcup \{e(u,v)\}$
    foreach $v \in V'$ *AND* $\in c \notin \mathcal{R}_i$ **do**
      foreach $e(v, v')$, $v' \in c \mid c \in C_{i-1}$ **do**
        if $c \in \mathcal{R}_i$ **then**
          $v_{min} \leftarrow \arg\min_{\forall v' \in \Gamma(v), c}\ d(v, v')$
        else
          $v_{min}(c) \leftarrow \arg\min_{\forall v' \in \Gamma(v), c}\ d(v, v')$
      forall the $v_{min}, v_{min}(c) \in \Gamma(v), c \mid c \in C_{i-1}$ **do**
        $\triangleright v_{min}$ chosen from sampled nodes only
        if v_{min} *exists for* v **then**
          $E_S \leftarrow E_S \cup \{e(v, v_{min})\}$
          $E_{comm} \leftarrow E_{comm} \setminus \{e(v, v_{min})\}$
          $\mathcal{E}_i \leftarrow \mathcal{E}_i \cup \{e(v, v_{min})\}$
          $E' \leftarrow E' \setminus \{E'(v, c_{min})\} \mid v_{min} \in c_{min}$
          if $v_{min}(c) < v_{min}$ **then**
            $E_S \leftarrow E_S \cup \{e(v, v_{min}(c))\}$
            $E_{comm} \leftarrow E_{comm} \setminus \{e(v, v_{min}(c))\}$
            $E' \leftarrow E' \setminus \{E'(v, c)\} \mid v_{min}(c) \in c$
        else
          $E_S \leftarrow E_S \cup \{e(v, v_{min}(c))\}$
          $E_{comm} \leftarrow E_{comm} \setminus \{e(v, v_{min}(c))\}$
          $E' \leftarrow E' \setminus \{E'(v, c)\} \mid v_{min}(c) \in c$
    forall the $e(v, u) \mid u, v \in c', c' \in E'$ **do**
      $E' \leftarrow E' \setminus \{e(v, u)\}$
  forall the $v \in V'$ **do**
    forall the $c \in C_{k-1}$ **do**
      $E_S \leftarrow E_S \cup \{e(v, v_{min}(c))\}$
      $E_{comm} \leftarrow E_{comm} \setminus \{e(v, v_{min}(c))\}$
      $E' \leftarrow E' \setminus \{E'(v, c)\} \mid v_{min}(c) \in c$
  return G_{comm}

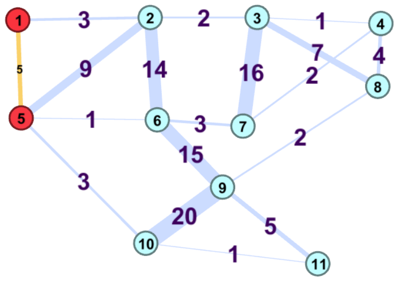


Figure: Original network $n = 11, m = 18$
$\delta(1,5) = 5$

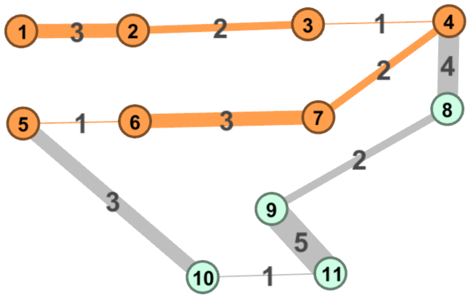


Figure: A 3-spanner of the network
$n = 11, m = 11$ $\delta_s(1,5) = 12$

Since $\delta_s(1,5) < t\ .\ \delta(1,5)$, the edge $(1,5)$ is discarded
The other edges are discarded in a similar fashion.

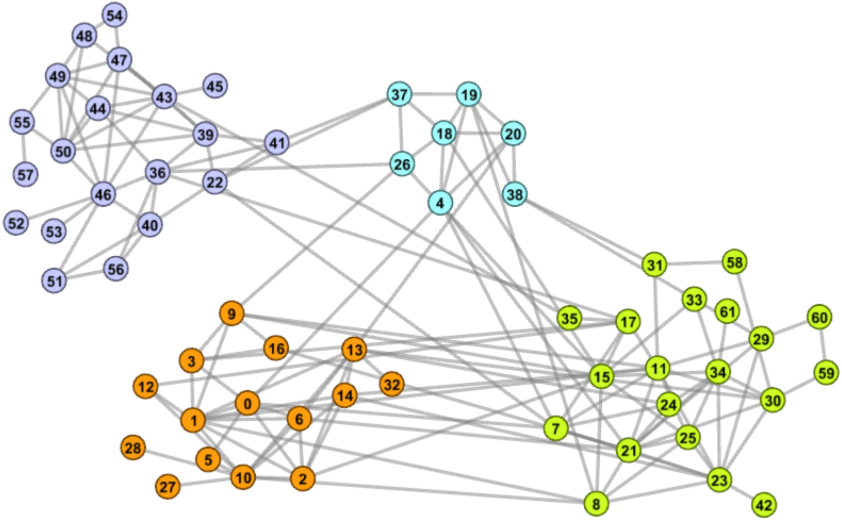


Figure: Dolphin network. n = 62, m = 159

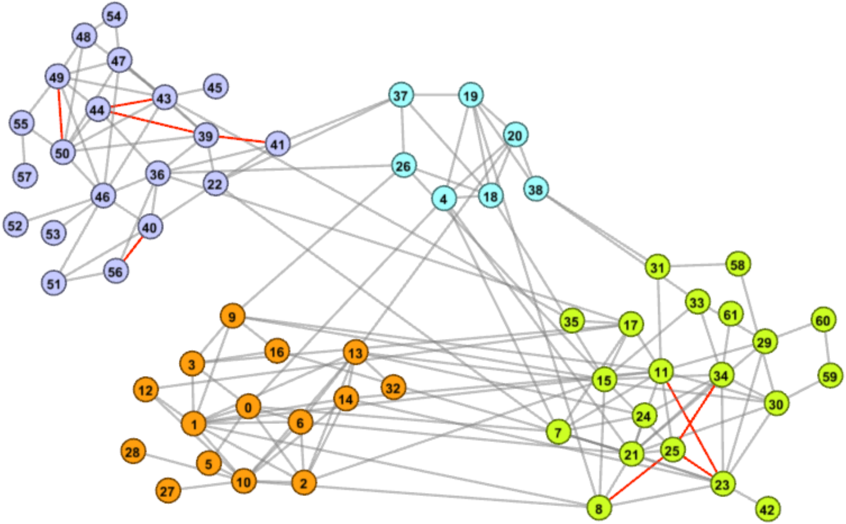


Figure: 3-spanner. n = 62, m = 150

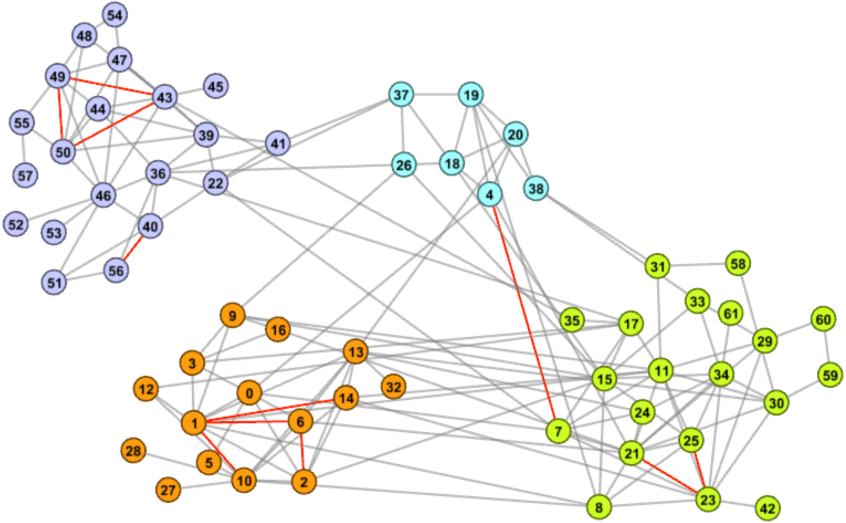


Figure: 5-spanner. n = 62, m = 148

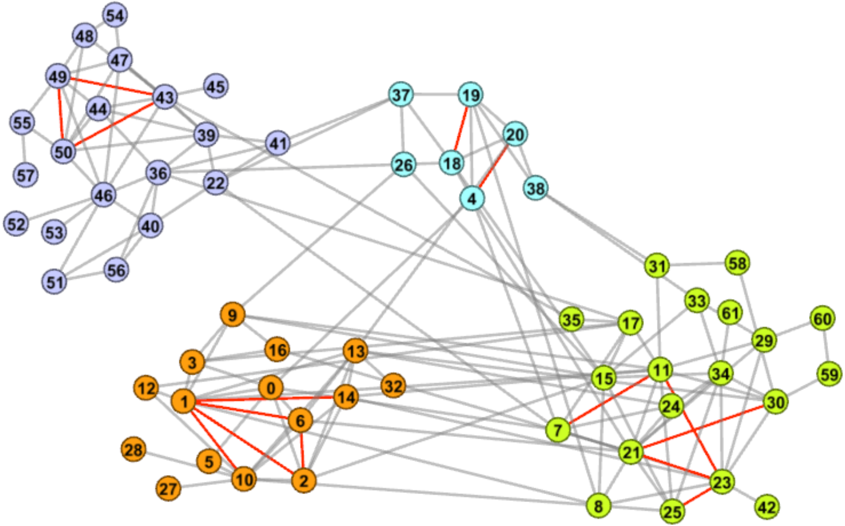


Figure: 7-spanner. n = 62, m = 144

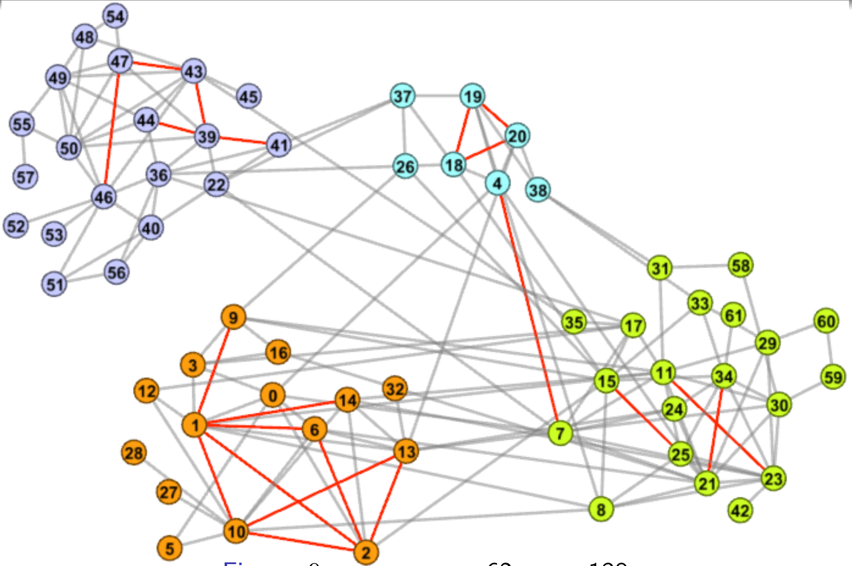
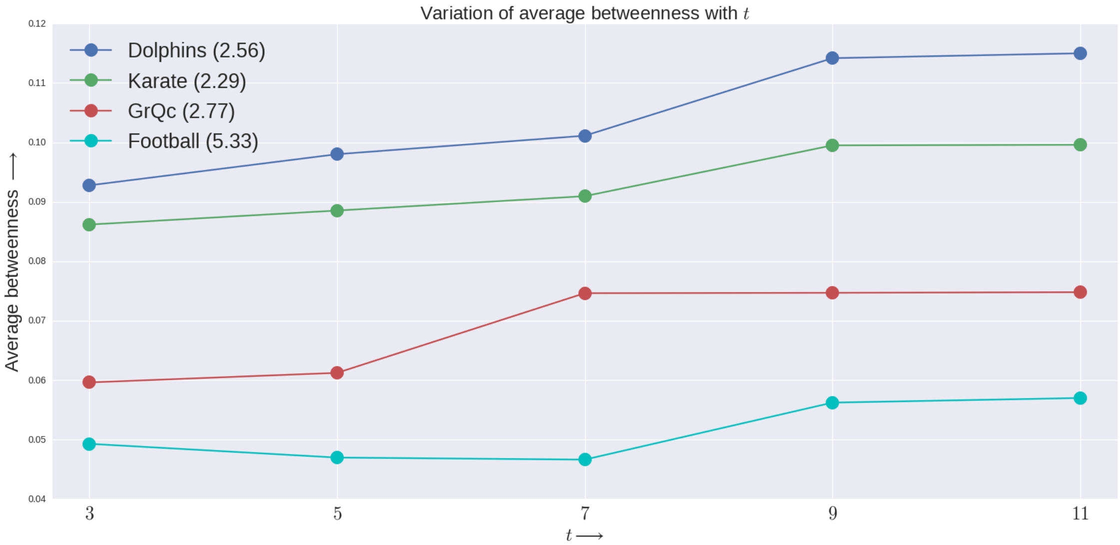


Figure: 9-spanner. n = 62, m = 138


Variation of average betweenness with t
Dolphins (2.56)
Karate (2.29)
GrQc (2.77)
Football (5.33)
Average betweenness ⟶
t ⟶
0.12
0.11
0.10
0.09
0.08
0.07
0.06
0.05
0.04
3
5
7
9
11

Name	n	Spanner	#intra-community	#inter-community
Karate	34	Original	59	19
		3	57	19
		5	53	19
		7	51	18
		9	48	19
Dolphin	59	Original	120	39
		3	117	38
		5	102	38
		7	100	38
		9	90	38
Football	115	Original	447	163
		3	385	166
		5	376	166
		7	293	166
		9	286	165

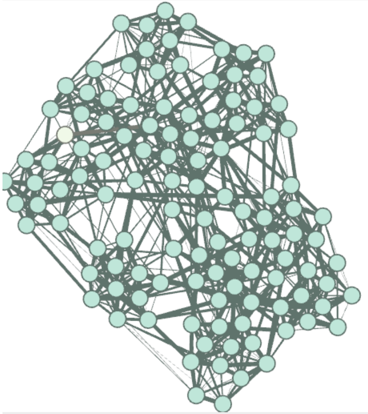


Figure: Original US Football network

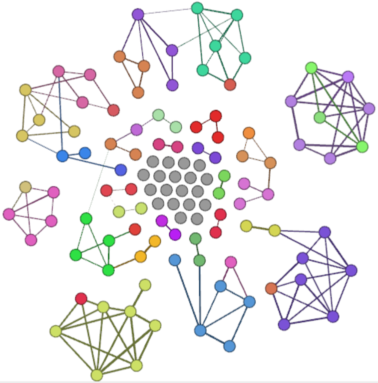


Figure: Sparsified network $\mathcal{G}_{comm}$

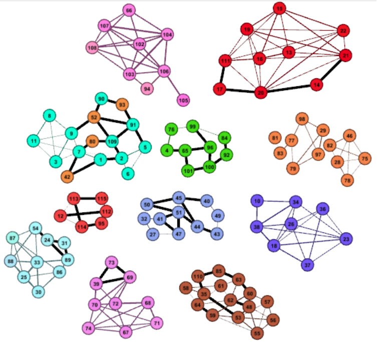
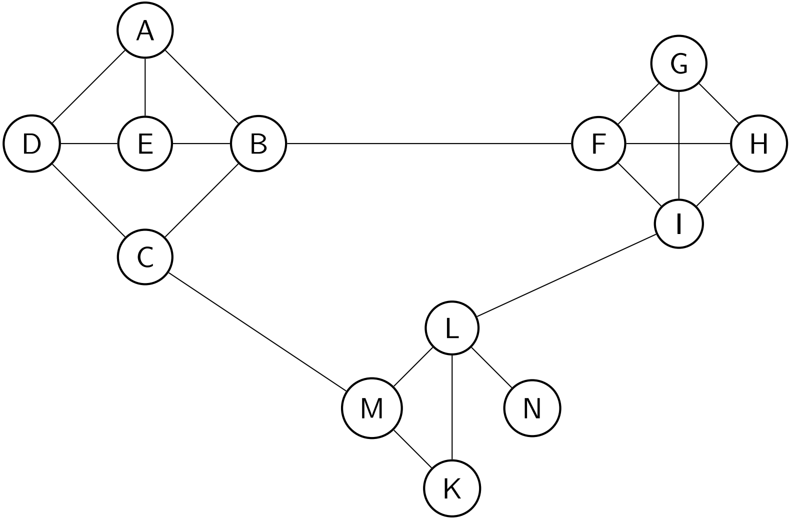


Figure: Final network with communities marked as separate components

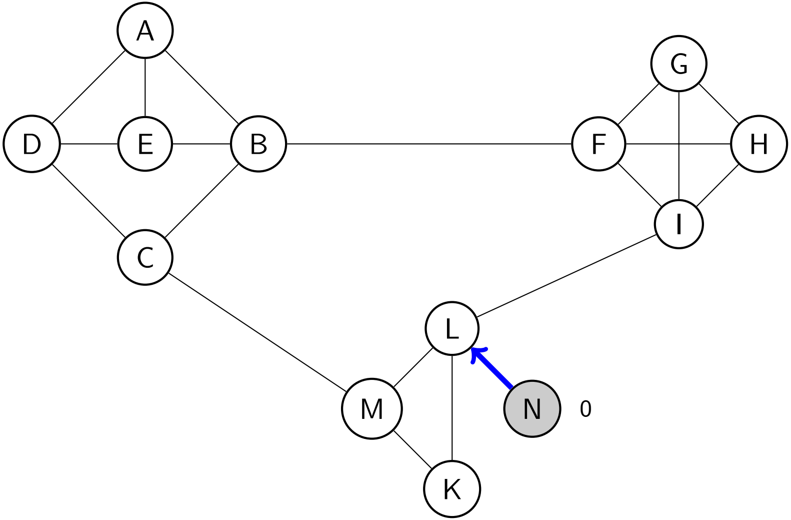
- At every timestamp, a node spreads influence to all of it's neighbors.
- Nodes are assigned a *Influenced Neighbors Score*(INS) value at every timestamp

$$INS_t(v) = \frac{\#\ neighbors\ influenced\ at\ time\ t}{Degree(v)}$$

- A threshold(THR) is set, and the nodes are classified accordingly. If $INS(v) \leq THR$, v is a **broker node**, otherwise v is a **community node**
- Broker nodes are placed in a **stack**, and community nodes in a **queue**. This results in a mix of breadth first and depth first traversals.
- Running time $O(m+n)$

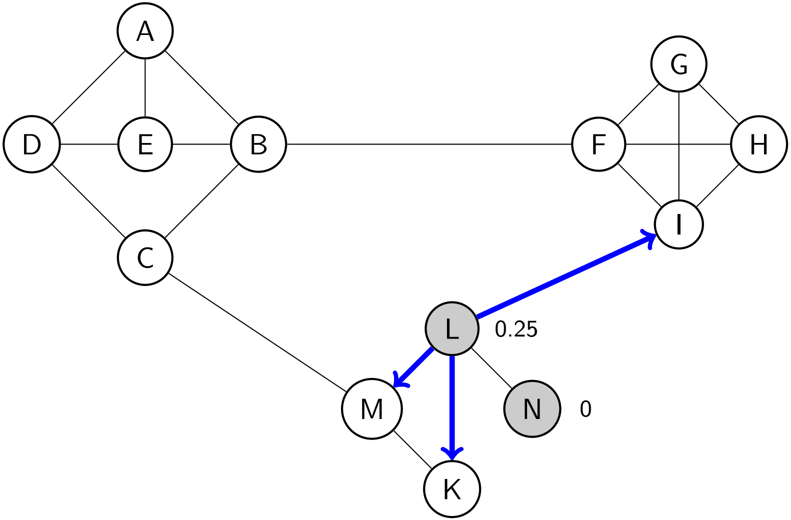


An Example - Threshold is taken to be 0.66.



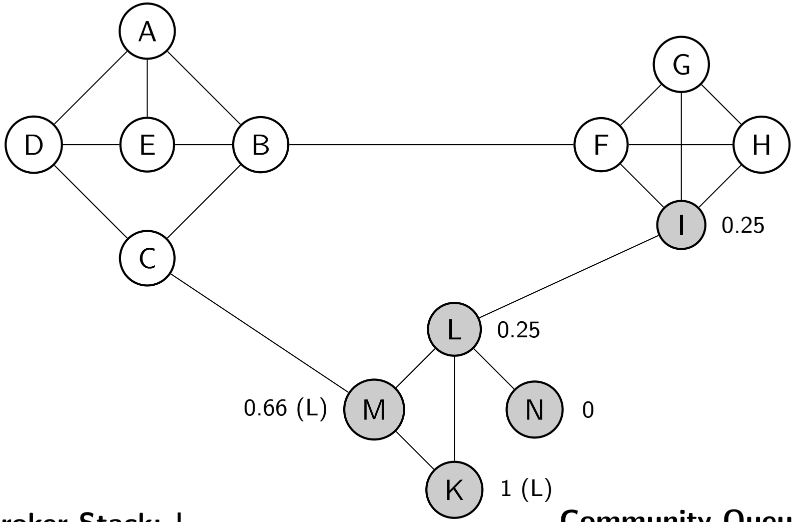
Broker Stack: N

Community Queue: $\emptyset$



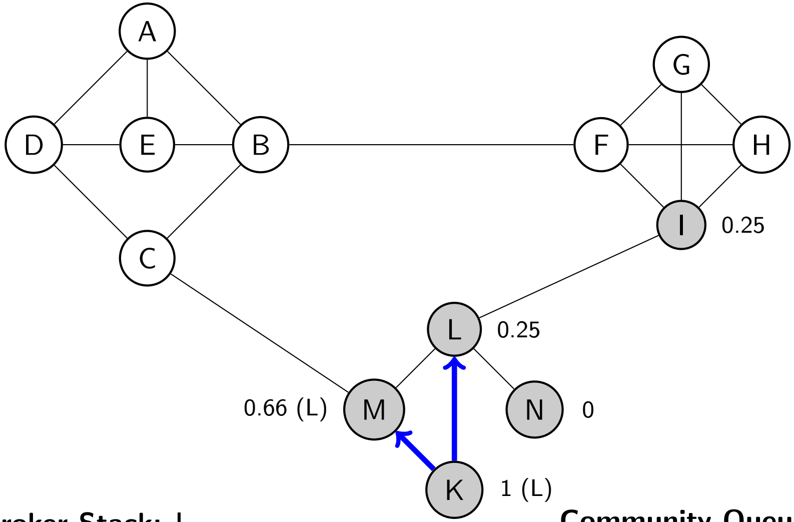
Broker Stack: L

Community Queue: $\emptyset$



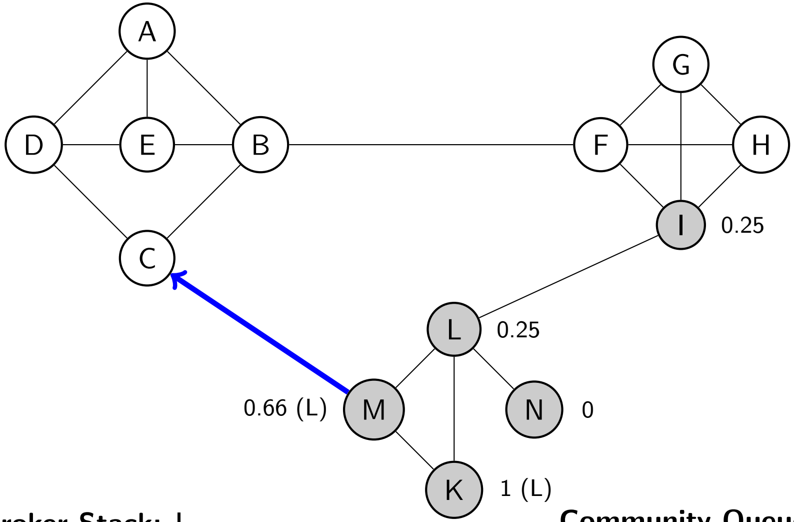
Broker Stack: I

Community Queue: K M



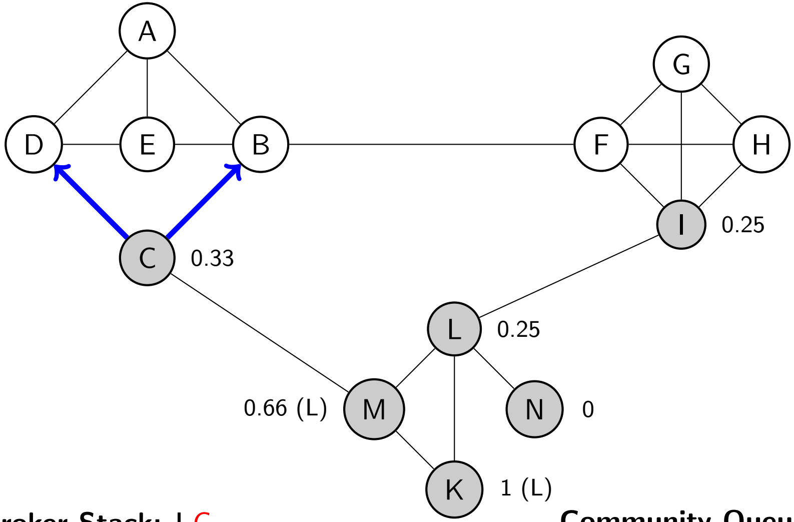
Broker Stack: I

Community Queue: K M



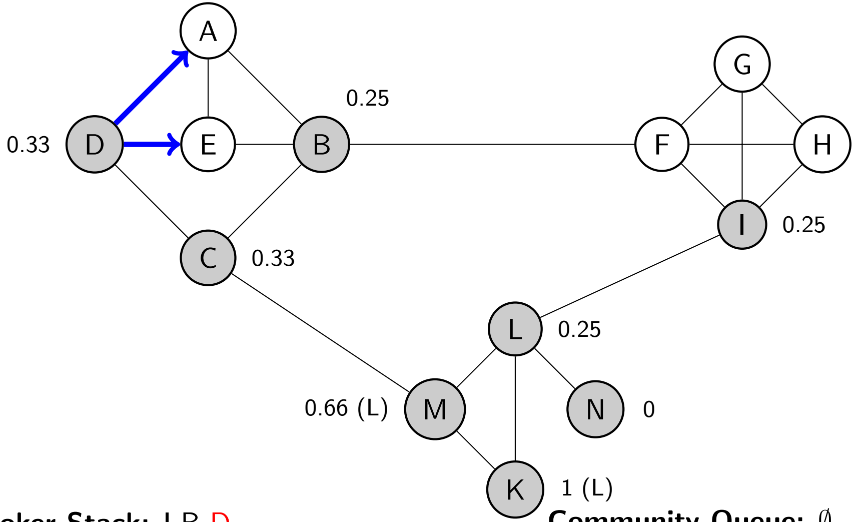
Broker Stack: I

Community Queue: M



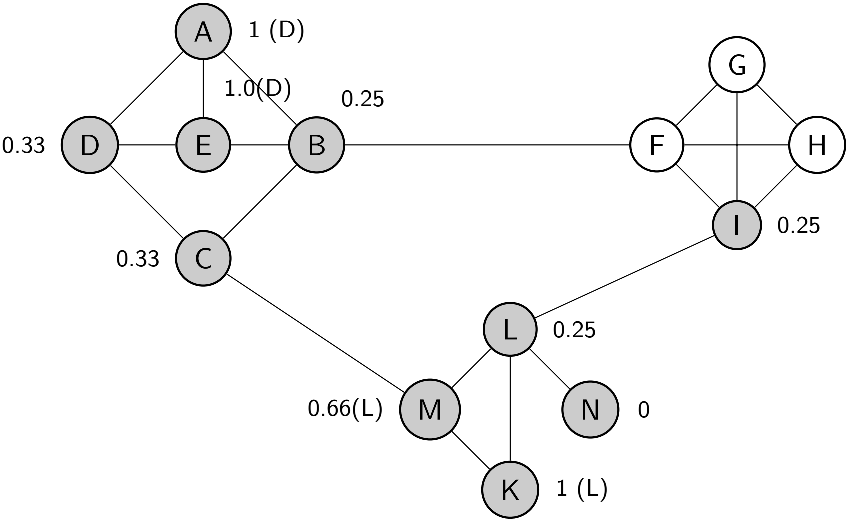
Broker Stack: | C

Community Queue: $\emptyset$



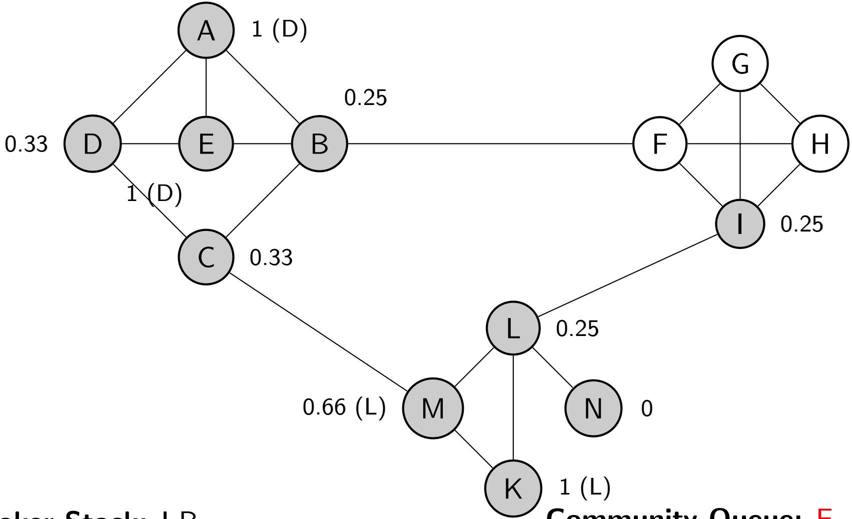
Broker Stack: I B D

Community Queue: $\emptyset$



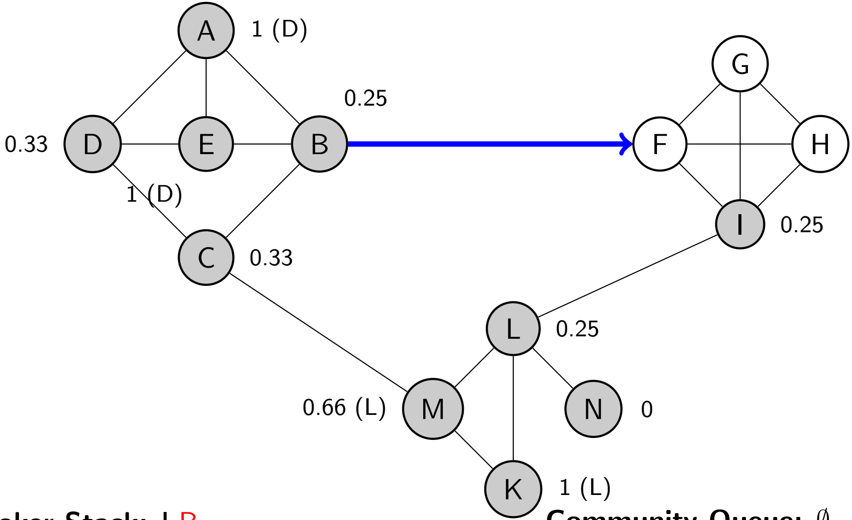
Broker Stack: I B

Community Queue: A E



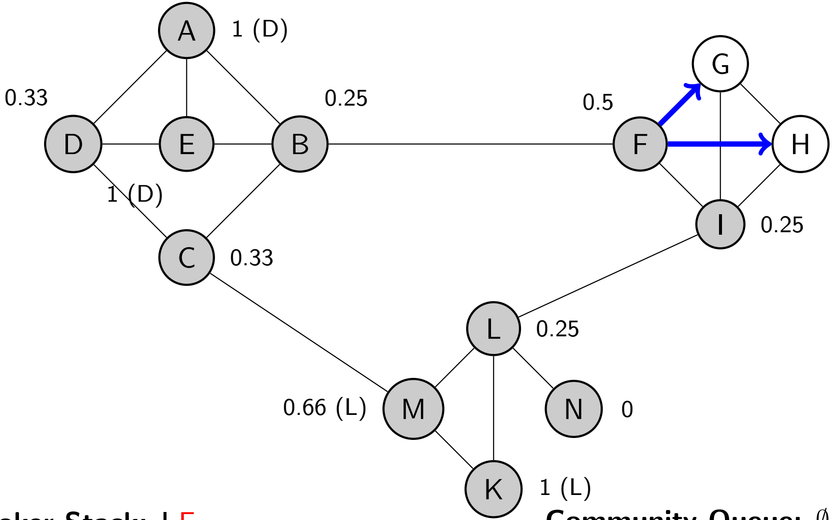
Broker Stack: I B

Community Queue: E



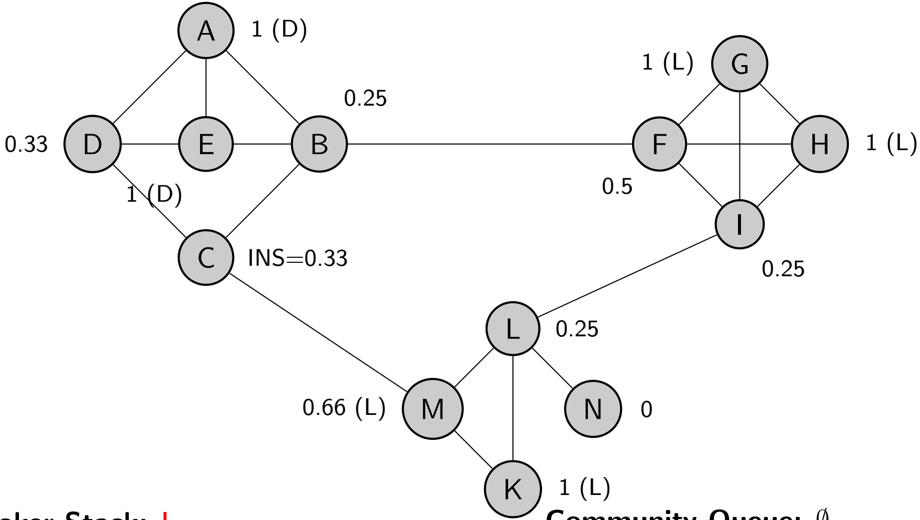
Broker Stack: I B

Community Queue: $\emptyset$



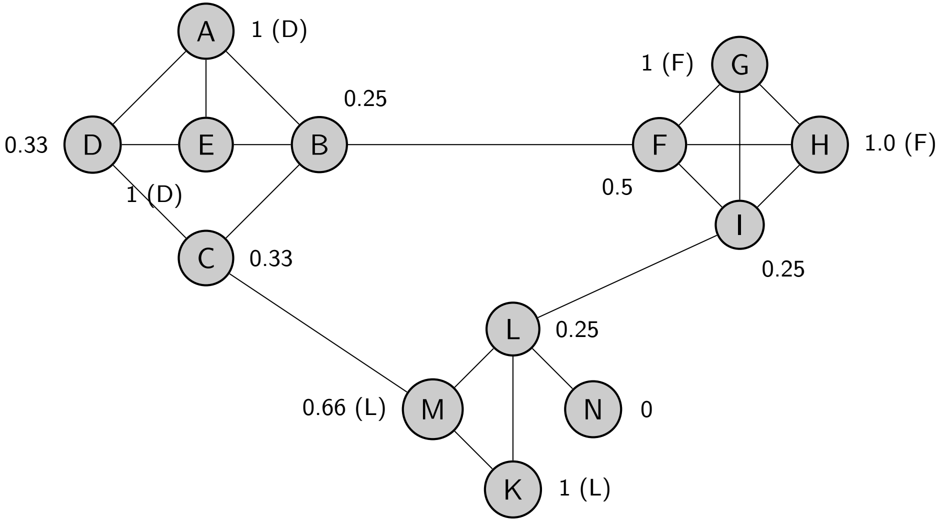
Broker Stack: I F

Community Queue: $\emptyset$

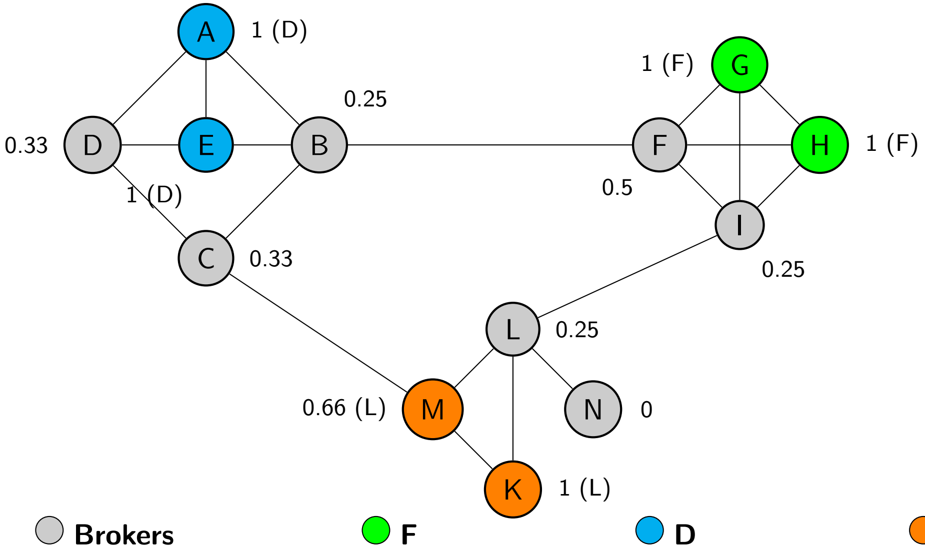


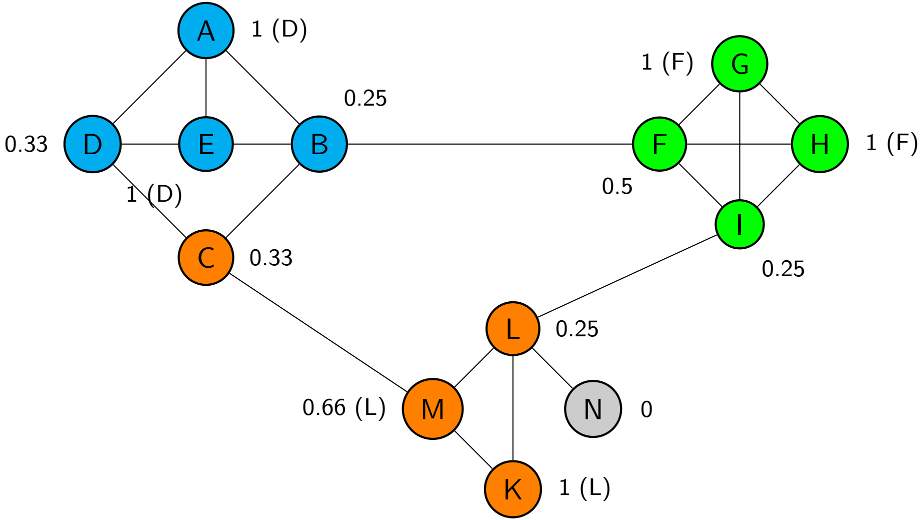
Broker Stack: I

Community Queue: $\emptyset$

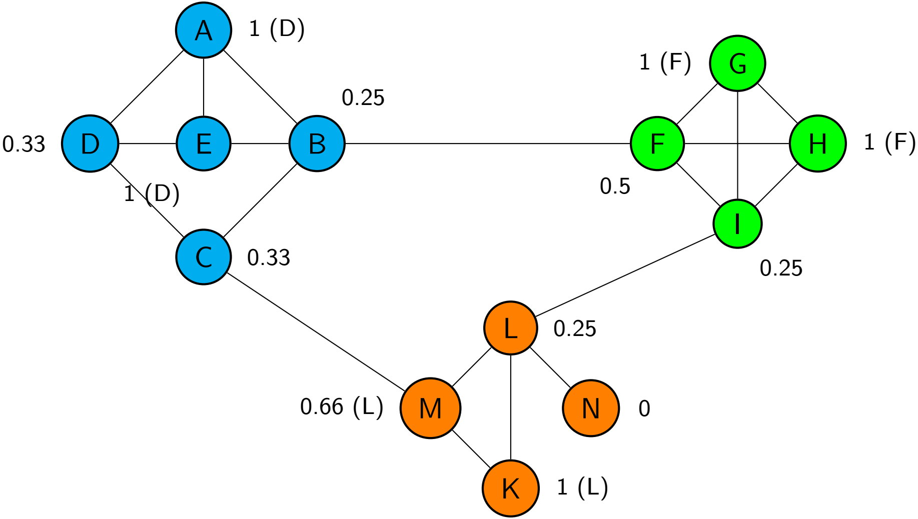


Both the **Broker Stack** and **Community Queue** are empty, so, the process stops.


A
1 (D)
0.25
D
E
B
0.33
1 (D)
C
0.33
1 (F)
G
F
H
1 (F)
0.5
I
0.25
L
0.25
0.66 (L)
M
N
0
K
1 (L)
Brokers
F
D
L



Post-processing - Brokers are assigned to communities (if possible).


A
1 (D)
0.25
0.33
D
E
B
1 (D)
C
0.33
1 (F)
G
F
H
1 (F)
0.5
I
0.25
L
0.25
0.66 (L)
M
N
0
K
1 (L)

Algorithm 3: Traversal-based Linear Time Community Detection (LINCOM(G, r))

Input : Undirected, unwtd. graph $G(V, E)$, threshold (r)
Output: Cover of k communities, $G_s = \{G_{s_1}, G_{s_2}, ..., G_{s_k}\}$

begin
  find $v_{start} \in V$, $\ni v_{start}$ is the node with lowest degree
  forall the $v \in V$ **do**
    nodeType(v) $\leftarrow$ community(v) $\leftarrow$ covered(v) $\leftarrow$ 0
  $S \leftarrow Q \leftarrow \emptyset$ ▷brokerStack(S), communityQueue(Q)
  coverCount $\leftarrow$ 1
  $v \leftarrow v_{start}$
  NODE-CAT(G, v, Q, S)
  while *coverCount* $< n$ **do**
    if Q *is non-empty* **then**
      $v \leftarrow$ dequeue(Q)
    else
      $v \leftarrow$ pop(S)
    NODE-CAT(G, v, Q, S)
  forall the $v \in V$ **do**
    check community(v) to form G_s
  POST-PROCESS(G_s)
  CONVERGE(G,G_s,t)
  return G_s

Algorithm 4: Categorizing uncovered nodes in $Nghb(v)$ (NODE-CAT(G, v, Q, S))

Input : Undirected, unwtd. graph $G(V, E)$, threshold (r), brokerStack (S), communityQueue (Q)
Output: Update Q, S, nodeType list, community list

begin
  SPREAD(v)
  for $i = 1$ *to* $deg(v)$ **do**
    $u \leftarrow adjList(v, i)$
    calculate INS(u)
    if *nodeType*(u) *is NON-ZERO* **then** continue

    if INS(u) $< r$ **then**
      nodeType(u) $\leftarrow$ 1 ▷marking the broker nodes
      push(S,u)
      community(u) $\leftarrow u$
    else
      nodeType(u) $\leftarrow$ 2 ▷marking the community nodes
      enqueue(Q,u)
      community(u) $\leftarrow$ community(v)
  return

Algorithm 5: Spreading Influence to the Neighbors (SPREAD(v))

Input : Undirected, unwtd. graph $G(V, E)$, root node (v)
Output: Update coverCount, covered list

begin
  for $i = 1$ *to* $deg(v)$ **do**
    if $covered(adjList(v, i))$ *EQUALS TO 0* **then**
      covered($adjList(v, i)$) $\leftarrow 1$
      coverCount $\leftarrow$ coverCount + 1
  return

Algorithm 6: Influenced Neighbors Score (INS(v))

Input : Undirected, unwtd. graph $G(V, E)$, root node (v)
Output: INS(v)

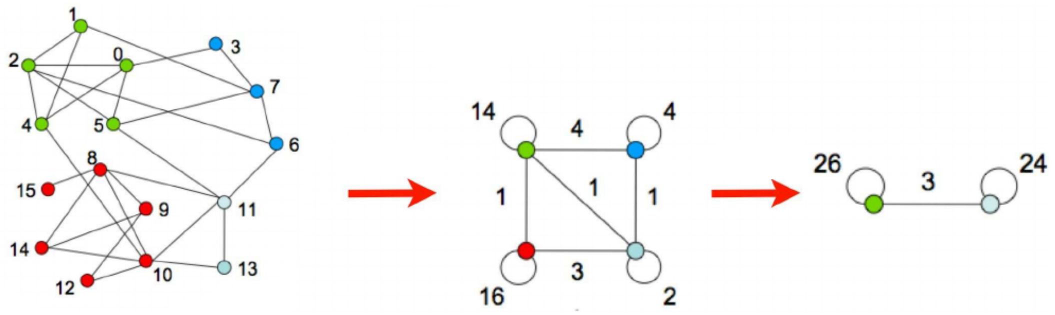
begin
  insTotal(v) $\leftarrow 0$
  for $i = 1$ *to* $deg(v)$ **do**
    if $covered(adjList(v, i))$ *EQUALS TO 1* **then**
      insTotal(v) $\leftarrow$ insTotal(v) + 1
  INS(v) $\leftarrow$ insTotal(v)/deg(v)
  return INS(v)

Algorithm 7: Post-processing of the Broker Nodes (POST-PROCESS(G_s))

Input : Undirected, unwtd. graph $G(V, E)$, root node (v)
Output: Update community(v) value for broker nodes

begin
  forall the $v \in V$ **do**
    $max(v) \leftarrow 0$
    $cover(v) \leftarrow 0$
  forall the $v \in V$ **do**
    if $nodeType(v)$ *EQUALS TO* 1 **then**
      forall the $G_{s_i} \in G_s$ **do**
        if $\frac{|Neighbors\ of\ v\ in\ G_{s_i}|}{|G_{s_i}|} > max(v)$ **then**
          $max(v) \leftarrow \frac{|Neighbors\ of\ v\ in\ G_{s_i}|}{|G_{s_i}|}$
      if $max(v)$ *is NON-ZERO* **then**
        community(v) $\leftarrow$ cover that leads to max(v)
  return

Modularity Maximization[3]



Modularity

We define modularity [2] as

$$Q = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j) \qquad\qquad Q \in [-1, 1]$$

- The higher the modularity, the better is the community structure*
- Actual social networks have modularity values between 0.30 - 0.60
- Suffers from resolution limit problems, but usually works very well in practice

Performance Comparison

Dataset Information [11]			Louvain Method		Our Algorithm (3-spanner	
Name	n	m	Modularity	Time(sec)	Modularity	Time(sec)
Karate	34	78	0.415	0	0.589	0.51
Dolphins	62	159	0.518	0	0.676	0.53
Football	115	613	0.604	0	0.8615	0.65
Enron	33,696	180,811	0.596	0.38	0.855	13.13
Epinions	75,877	405,739	0.45	0.97	0.695	27.03
Amazon	334,863	925,872	0.926	6	0.995	78.47
DBLP	317,080	1,049,866	0.819	11	0.961	76.8607

Conclusions and Future Scope

- Jaccard coefficient based edge weight preserves the community structure
- Spanners can be used to identify inter-community edges (to a large extent)
- LINCOM is effective in breaking the graph into pure fragments
- Works very fast in practice - produces good quality clusters

- The stretch-factor t and THR is predetermined, may turn out to be network dependent
- The algorithm can be modified to do overlapping community detection

References I

[1] Markus M Möbius and Tanya S Rosenblat.
The process of ghetto formation: Evidence from chicago.
Unpublished manuscript. http://trosenblat. nber. org/papers/Files/Chicago/chicago-dec7. pdf, 2001.

[2] M. Girvan and M. E. J. Newman.
Community structure in social and biological networks.
PNAS, 99(12):7821–7826, June 2002.

[3] V.D. Blondel, J.L. Guillaume, R. Lambiotte, and E.L.J.S. Mech.
Fast unfolding of communities in large networks.
J. Stat. Mech, page P10008, 2008.

[4] Venu Satuluri, Srinivasan Parthasarathy, and Yiye Ruan.
Local graph sparsification for scalable clustering.
In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 721–732. ACM, 2011.

References II

[5] David Liben-Nowell and Jon Kleinberg.
The link-prediction problem for social networks.
Journal of the American society for information science and technology, 58(7):1019–1031, 2007.

[6] Gerd Lindner, Christian L Staudt, Michael Hamann, Henning Meyerhenke, and Dorothea Wagner.
Structure-preserving sparsification of social networks.
In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, pages 448–454. ACM, 2015.

[7] Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares.
On sparse spanners of weighted graphs.
Discrete & Computational Geometry, 9(1):81–100, 1993.

[8] Liam Roditty and Uri Zwick.
On dynamic shortest paths problems.
In *Algorithms–ESA 2004*, pages 580–591. Springer, 2004.

References III

[9] Liam Roditty, Mikkel Thorup, and Uri Zwick.
Deterministic constructions of approximate distance oracles and spanners.
In *Automata, languages and programming*, pages 261–272. Springer, 2005.

[10] Surender Baswana and Sandeep Sen.
A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs.
Random Structures & Algorithms, 30(4):532–563, 2007.

[11] Jure Leskovec and Andrej Krevl.
SNAP Datasets: Stanford large network dataset collection.
http://snap.stanford.edu/data, June 2014.

"No one ever complains about a speech being too short!" - Ira Hayes

Thank you for your attention. Any questions?