IMPROVED LCD-SN

|  |  |  |
| --- | --- | --- |
| NETWORKS | OLD LCD-SN | Improved LCD-SN |
| Karate | 0.3569197896120973 | 0.37146614069691 |
| Dolphins | 0.4646176970847672 | 0.48239784818638504 |
| Polbooks | 0.5064222211938442 | 0.508962315084764 |
| NetScience | 0.8693876362954299 | 0.8634974129707302 |
| GD01\_b | 0.2927295918367347 | 0.3335459183673469 |
| Lesmis | 0.4661719214753122 | 0.4661719214753122 |
| CAG\_mat72 | 0.16477882360652432 | 0.16477882360652432 |
| Trefethen\_150 | 0.8890834174653618 | 0.8916171560922972 |
| USAir97 | 0.015398659339839528 | 0.04218584758352555 |
| bcsstk22 | 0.9288186628447975 | 0.9641684794987614 |

Formulas:  
  
Dynamic Importance Calculation

**Where:**

* Updated importance of node .
* α: Weight for direct neighbors.
* β: Weight for second-degree neighbors.
* : Weight of the edge between nodes and .
* Current importance of neighbor .
* Total weight of edges connected to node for normalization

Importance Normalization Formula

where:

* : Set of all nodes in the graph.
* New importance value for node after updating.

Modified GLHN Similarity Calculation

**Where:**

* ): Similarity score between nodes and .
* Set of neighbors of node .
* Set of neighbors of node .
* Number of common neighbors between nodes and .
* Total number of neighbors of nodes and , respectively.

Node-to-Community Similarity

**Where:**

* Similarity of node to community .
* : Community to which node ’s similarity is measured.
* Number of nodes in community .
* GLHN similarity between nodes and , where is a member of community .

Community-to-Community Similarity

**Where:**

* Similarity between communities ​ and ​.
* ​ and : Sets of nodes in the two communities being compared.
* : GLHN similarity between nodes in ​ and in .

Modularity Calculation

**Where:**

* : Modularity score of the partition.
* Adjacency matrix entry (1 if nodes and are connected, 0 otherwise).
* Degree of node .
* : Total number of edges in the graph.
* Kronecker delta (1 if nodes and are in the same community, 0 otherwise).

**Algorithm: Improved LCD\_SN(G, α, β, max\_iterations, mc):**

**Input:**

G: Social Network Graph

α: Influence of first-degree neighbors

β: Influence of second-degree neighbors

max\_iterations: Maximum iterations for convergence

mc: Threshold for merging small or weak communities

**Output:**

Final\_Communities: List of node sets representing communities

Modularity: Quality score of community structure

**Step 1: Load Graph**

Load graph G from adjacency matrix or edge list.

If weights are available, add edges with weights.

**Step 2: Calculate Dynamic Node Importance**

For each node v in G:

importance[v] = 1.0

for iteration from 1 to max\_iterations do:

for each node v in G:

new\_importance[v] = 0.0

for each neighbor u of v do:

new\_importance[v] += α \* weight(v, u) \* importance[u]

for each second-degree neighbor w of v do:

new\_importance[v] += β \* weight(v, w) \* importance[w]

// Check for convergence

if max(abs(new\_importance[v] - importance[v]) for v in G) < convergence\_threshold then:

break

// Update importance scores

importance[v] = new\_importance[v] / sum(new\_importance)

**Step 3: Form Initial Communities**

Sort nodes by importance in descending order

For each unassigned node v in sorted order do:

community[v] = new community containing v and its neighbors

mark all nodes in community[v] as assigned

**Step 4: Resolve Community Overlaps**

For each node v belonging to multiple communities do:

similarities = []

for each community C do:

similarity = calculate\_similarity(v, C) // Based on shared neighbors

similarities.append((C, similarity))

// Move v to the community with the highest similarity

best\_community = max(similarities, key=lambda x: x[1])[0]

move v to best\_community

**Step 5: Merge Small and Weak Communities**

// Merge Small Communities

For each community C in communities do:

if size(C) < 3 then:

for each node v in C do:

most\_similar\_community = find\_most\_similar\_neighboring\_community(v)

merge C with most\_similar\_community

// Merge Weak Communities

For each community C\_i in communities do:

E\_in = calculate\_internal\_edges(C\_i)

E\_out = calculate\_external\_edges(C\_i)

if E\_in <= mc \* E\_out then:

most\_similar\_community = find\_most\_similar\_neighboring\_community(C\_i)

merge C\_i with most\_similar\_community

**Step 6: Calculate Modularity**

Assign community IDs to nodes

Modularity = calculate\_modularity(partition structure)

**Step 7: Output**

Return Final\_Communities and Modularity