README for Graph Reduction MATLAB Code

1 Overview

This MATLAB code is designed to reduce a large graph into smaller subgraphs using various graph analysis techniques. The code utilizes the `SCCAnanyzerClass` to perform several steps in analyzing and reducing the graph. This document will explain the functions and provide an example of how to use them.

2 File Description

- `SCCAnanyzerClass.m`: Contains the class definition for `SCCAnanyzerClass`, which includes methods for graph analysis and reduction.

3 Functions

The `SCCAnanyzerClass` provides several methods to analyze and reduce graphs. Here is a description of the key functions:

3.1 `NeighborStructure`

```
[omega, disc, zeta] = g.NeighborStructure(G, n);
```

- **Description**: This function analyzes the neighbor structure of the graph, determining connectivity and identifying strongly connected components (SCCs).
- **Parameters**:
  - `G`: The directed graph (`digraph`) object.
  - `n`: Number of nodes in the graph.
- **Returns**:
  - `omega`: Matrix representing the neighbor structure.
  - `disc`: Indicates if the graph is strongly connected.
  - `zeta`: Matrix representing another aspect of the neighbor structure.

- **Formula:**

$$
\begin{aligned}
\xi_{i,j}(k+1) &= \max_{l \in \mathcal{N}_i \cup i} \xi_{l,j}(k), \quad j \in \mathcal{N}, \quad k = 0, \cdots, (n-1) \\
\omega_{i,j}(k+1) &= \begin{cases} \omega_{i,j}(k) & \text{if } \xi_{i,j}(k+1) = \xi_{i,j}(k), \\ \min_{l \in \mathcal{N}_i}(\omega_{l,j}(k)+1) & \text{if } \xi_{i,j}(k+1) > \xi_{i,j}(k) \end{cases} \\
& \quad k = 0, \cdots, (n-1) \\
\rho_i(k+1) &= \max_{j \in \mathcal{N}_i \cup \{i\}} \rho_j(k), \quad k = n \cdots, 2n
\end{aligned}
$$

where $\mathcal{N}_i$ is the in-neighbor set of the node i and the initial conditions are given as

$$
\begin{aligned}
\xi_{i,j}(0) &= \begin{cases} 1, & \text{if } j = i \\ 0, & \text{otherwise} \end{cases} \\
\omega_{i,j}(0) &= \begin{cases} 0, & \text{if } j = i \\ \infty, & \text{otherwise} \end{cases} \\
\rho_i(n) &= \begin{cases} 0 & \text{if } \xi_i(n) = \mathbf{1}_n \\ 1, & \text{if } \xi_i(n) \neq \mathbf{1}_n. \end{cases}
\end{aligned}
$$

3.2 `MatrixOutput`

```
output = g.MatrixOutput(matrix);
```

- **Description**: This function outputs matrices in a readable format, which helps in visualizing the results of different graph analysis steps.
- **Parameters**:
  - `matrix`: The matrix to output.
- **Returns**:
  - `output`: Formatted matrix output.

3.3 `InformationNumber`

```
[ci] = g.InformationNumber(G, n);
```

- **Description**: This function calculates the information number of the graph, which is used to identify SCCs and analyze their structure.
- **Parameters**:
  - `G`: The directed graph (`digraph`) object.
  - `n`: Number of nodes in the graph.

- **Returns**:
  - `ci`: Matrix representing the information number.
- **Formula:**

$$\eta_i(k+1) = \begin{bmatrix}\eta_{i,1}(k+1) & \cdots & \eta_{i,n}(k+1)\end{bmatrix}.$$

The distributed protocol for updating these information numbers is defined as follows:

$$\eta_{i,j}(k+1) = \max_{l \in \mathcal{N}_i^c \cup \{i\}} \eta_{l,j}(k),$$

where $\mathcal{N}_i^c$ represents the in-neighbor set of node i, including node i itself. The initial condition for this recursive update is:

$$\eta_{i,j}(n) = \begin{cases} {}^T_n\xi_i(n), & \text{if } j = i \\ 0, & \text{otherwise} \end{cases},$$

3.4 `findingSCC`

```
[SC, SP] = g.findingSCC(G, n);
```

- **Description**: Finds strongly connected components (SCCs) of the graph.
- **Parameters**:
  - `G`: The directed graph (`digraph`) object.
  - `n`: Number of nodes in the graph.
- **Returns**:
  - `SC`: Matrix representing SCCs.
  - `SP`: Matrix representing the shortest paths within SCCs.
- **Formula:** $SCC = \begin{bmatrix}SCC_1 & \cdots & SCC_n\end{bmatrix}$. Each SCC is composed of nodes j such that:

$$SCC_l = \{j \in \mathcal{N} : \zeta_{l,j}(n) = 1 \text{ and } \eta_{l,j}(2n) = \eta_{l,l}(2n)\}.$$

3.5 `indexSCC`

```
[Vsour, Vsink, Visol, Vmid] = g.indexSCC(G, n);
```

- **Description**: Indexes SCCs into different categories, sourc
- **Parameters**:
  - `G`: The directed graph (`digraph`) object.

- `n`: Number of nodes in the graph.

- **Returns**:
  - `Vsour`: Source SCCs.
  - `Vsink`: Sink SCCs.
  - `Visol`: Isolated SCCs.
  - `Vmid`: Middle SCCs.

3.6 `SCCStructure`

```
[theta, ~] = g.SCCStructure(G, n);
```

- **Description**: Analyzes the structure of SCCs.
- **Parameters**:
  - `G`: The directed graph (`digraph`) object.
  - `n`: Number of nodes in the graph.
- **Returns**:
  - `theta`: Matrix representing the SCC structure.
- **Formula:**

$$\begin{aligned}\theta_i(k+1) &= \begin{bmatrix}\theta_{i,1}(k+1) & \cdots & \theta_{i,n}(k+1)\end{bmatrix},\\ \theta_i^*(k+1) &= \text{a row vector of size equal to \# of SCCs},\\ \nu_i^*(k+1) &= \text{a row vector of size equal to \# of SCCs}.\end{aligned}$$

  to map the connectivity to other SCCs, with θ_i^* focusing specifically on interactions between SCCs.

  To analyze inter-SCC connectivity, θ_i and θ_i^* are updated using:

$$\theta_{i,j}(k+1) = \max_{l \in \mathcal{N}_i^c \cup \{i\}} \theta_{l,j}(k), \quad \theta_{i,j}(2n) = \begin{cases} 1, & \text{if } j \in SCC_i \\ 0, & \text{otherwise} \end{cases},$$

3.7 `SCCReducedStructure`

```
[indexset, thetaRdd, nuRdd] = g.SCCReducedStructure(G, n);
```

- **Description**: Reduces the graph based on SCCs.
- **Parameters**:
  - `G`: The directed graph (`digraph`) object.
  - `n`: Number of nodes in the graph.
- **Returns**:

- `indexset`: Indices of the reduced graph.
- `thetaRdd`: Matrix representing the reduced SCC structure.
- `nuRdd`: Another matrix representing the reduced structure.

- **Formula:**

$$\theta^*_{V_i^*,V_j^*}(k+1) = \max_{\substack{l \in [(\mathcal{N}^c_{i^*} \cup \{i^*\}) \cap (\mathcal{N} - SCC^*_{i^*})] \\ m \in [(\mathcal{N}^c_{j^*} \cup \{j^*\}) \cap (\mathcal{N} - SCC^*_{j^*})]}} \theta_{l,m}(k)$$

to establish whether virtual leaders of different SCCs can reach each other through their respective members.

$$\nu^*_{V_i^*,V_j^*}(k+1) = \begin{cases} \nu^*_{V_p^*,V_j^*}(k)+1, & \text{if } \theta^*_{V_i^*,V_p^*} = 1 \text{ for some } p^* \neq i^* \text{ and} \\ & \quad \theta_{l,m}(k+1) - \theta_{q,m}(k) = 1 \\ & \quad \text{for some } l \in SCC^*_{V_i^*}, q \in SCC^*_{V_p^*}, \text{ and} \\ & \quad m \in SCC^*_{V_j^*}, \\ \nu^*_{V_i^*,V_j^*}(k), & \text{otherwise} \end{cases}$$

4 Running the Code

1. **Initialize the Class**: Create an instance of the `SCCAnanyzerClass`.

```
g = SCCAnanyzerClass;
```

2. **Define the Graph**: Specify the vectors of starting (`s`) and terminating (`t`) nodes.

```
% Define the graph using the vectors of starting nodes and terminating nodes
s = [1 2 2 3 3 4 4 4 4 5 5 6 6 6 7 8 8 9];
t = [2 3 4 1 4 1 2 3 5 4 6 5 7 8 5 6 9 8];
G = digraph(s, t);
A1a = adjacency(G);
```

3. **Analyze and Reduce the Graph**: Use the provided methods to analyze and reduce the graph.

   - Neighbor Structure

   ```
   [omega, disc, zeta] = g.NeighborStructure(G, n);
   ```

   - Information Number

```
[ci] = g.InformationNumber(G, n);
```

- Finding SCC and it's representation

```
[SC, SP] = g.findingSCC(G, n);
[Vsour, Vsink, Visol, Vmid] = g.indexSCC(G, n);
```

- SCC Structure and it's reduced form

```
[theta, ~] = g.SCCStructure(G, n);
[indexset, thetaRdd, nuRdd] = g.SCCReducedStructure(G, n);
```

4. **Plot the Results**: Visualize the original and reduced graphs.

By following these steps, you can effectively reduce large graphs into smaller subgraphs using the provided MATLAB code and class functions.