Identification of all disconnected components of a graph and realignment of the adjacency and incidence matrix to be close to a diagonal matrix.

Graph Theory: Assignment 2

Akshat Aggarwal
IIM2014005
7th Semester Dual Degree (IT)

Yogesh Gupta
IRM2014004
7th Semester Dual Degree (IT)

Vishesh Middha
ISM2014007
7th Semester Dual Degree (IT)

***Abstract*—In this assignment we strive to identify all the disconnected components of a graph and realign its adjacency and incidence matrix to be close to a diagonal matrix. We have worked on two different algorithms to achieve the same.**

***Index Terms*—graph, adjacency, incidence, diagonal, matrix.**

I. Introduction

A. Basic Definitions

1) Graph: Formally, graph is a pair G = (V, E) where,

- V is a set of vertices (or nodes), and
- E $\subseteq$ (V x V) is a set of edges.

A graph may be directed or undirected. If the graph is undirected, then the adjacency relation defined by edges is symmetric.

2) Incidence Matrix: Incidence matrix of an undirected graph is a matrix, B of dimension N x M, where N is the number of vertices and M is the number of edges, such that $B_{i,j}$= 1, if the vertex v_i and edge e_j are incident and 0 otherwise.

3) Adjacency Matrix: Adjacency matrix of an undirected graph is a matrix, B of dimension N x N, where N is the number of vertices, such that $B_{i,j}$= 1, if there is an edge between vertices *i* and *j*, and 0 otherwise.

4) Connected Components in an undirected graph: A connected component in an undirected graph is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the supergraph. A vertex with no incident edges is itself a connected component. A graph that is itself connected has exactly one connected component, consisting of the whole graph.

5) Diagonal Matrix: For a disconnected graph with two components G_1 and G_2, the incidence or adjacency matrix of graph G, can be written as:

$$A(G) = \begin{bmatrix} A(G_1) & 0 \\ 0 & A(G_2) \end{bmatrix}$$

where, $A(G_1)$ and $A(G_2)$ are the incidence/adjacency matrices of the two components G_1 and G_2.

B. Objective

Here, we propose algorithms to identify the disconnected components in a graph and realign the adjacency and incidence matrices to as close to diagonal matrix.

II. Motivation

Graphs are a very important data structure. They are used in network related algorithms, routing, finding relations among objects, shortest paths and many more real life related applications. On e-commerce websites, relationship graphs are used to show recommendations. Connecting with friends on social media, where each user is a vertex and the connection between two users is represented by an edge, is also an important application of graphs.
The need of finding the disconnected components in a graph arises due to the fact that the sub-graphs which have some common attributes are connected and those which have nothing in common are disconnected. Furthermore the necessity of realignment is that it helps segregating the disconnected components and provides an easy access to all the components by removing a haphazard arrangement while still keeping all the disconnected graph in a single matrix.

III. Methods and Algorithms

A. Identifying disconnected components in undirected graph given its adjacency matrix and realigning the adjacency matrix to diagonal matrix.

Their are two main steps that are required to be performed:

1) Find the Disconnected Components
2) Realign the adjacency matrix into a diagonal matrix

Phase I: Finding Disconnected Components
Following is the algorithm used -

Algorithm I: Breadth first search algorithm to find Disconnected Components
Input: Adjacency Matrix
Output: Connected Component Matrix
The algorithm to find Disconnected Components -

```
components = []
bool visited[n+1] = {false}
Queue q
for vertex ← 1 to n+1
    if !visited[vertex]:
        l = []
        l.append(vertex)
        q.push(vertex)
        visited[vertex] = 1
        while !q.empty():
            u = q.front()
            for i ← 1 to n+1:
                if graph[u][j] != 0 and !visited[j]:
                    q.push(j);
                    visited[j] = true;
         components.append(l);
```

Phase II: Realigning to Diagonal Matrix
Following is the algorithm used -

Algorithm II: Realign the adjacency matrix into a diagonal matrix
Input: Connected Component Matrix
Output: Diagonal Matrix
The algorithm to realign the adjacency matrix into a diagonal matrix -

```
k = 1
indexing = []
reverseIndexing = []

for list in components:
    for vertex in list:
        Indexing[k] = vertex
        reverseIndexing[i] = k
     k += 1
 for i = 1 to n do
     u = indexing[i]
     for j = 1 to n do
        if graph[u][j] != 0 then
           v = reverseIndexing[j]
           res[i][v] = 1
```

B. Identifying disconnected components in undirected graph given its incidence matrix and realigning the incidence matrix to diagonal matrix.

Following is the algorithm used-

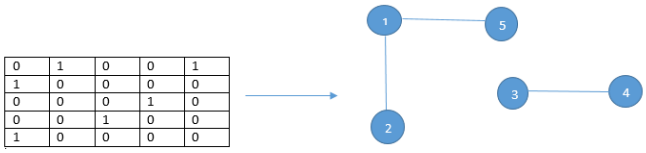
Algorithm III: Extract disconnected components from incidence matrix and realign it to diagonal matrix.
Input: Incidence Matrix - graph, n - number of vertices, m - number of edges
Output: Diagonal Matrix with components - result

```
// Extract the edge list from the input incidence matrix
edgelist = []
for i = 1 to m do
      from = -1,to = -1
      for j = 1 to n do
            if graph[j][i] == 1 then
                 if from == -1 then
                      from = j
                 else
                      to  = j
                 break
            endif
       endif
      endfor
      edgeList.append({from, to})
endfor

// representative of set to which i'th vertex belongs
representative = []

for i = 1 to n do
    representative[i] = i
endfor

// representative of set to which i'th edge belongs
edgeRepresentative = []

for i = 1 to m do
    u = edgeList[i].first, v = edgeList[i].second
    representativeU = findRepresentative(representative, u)
    representativeV = findRepresentative(representative, v)
    if representativeU != representativeV then
       // if not already connected.
       doUnion(representativeU, representativeV, representative);
    endif

    // Also add i'th edge to representativeU component.
    edgeRepresentative[j] = representativeU;
endfor
```

```
41.//Associate each vertex, edge with the set representative
42.for i = 1 to n do
43.    representative[i] = findRepresentative(representative, i)
44.endfor
45.
46.for i = 1 to m do
47.    edgeRepresentative[i] = findRepresentative(representative,
                               edgeRepresentative[i])
48.endfor
49.
50.vertexComps = []
51.hash = []
52.numberofComponents = 0
53.for i = 1 to n do
54.    int rep = representative[i]
55.    if hash[rep] == 0 then
56.       // a new component
57.       numberofComponents = numberofComponents + 1
58.       hash[rep] = numberOfComponents
59.       tmpplist = []
60.       tmplist.append(i)
61.       vertexComps.append(tmplist)
62.    else
63.       // Add vertex i to the corresponding component
64.       vertexComps[hash[rep]].append(i)
65.    endif
66.endfor
67.
68.
69.edgeComps = []
70.for i = 1 to m do
71.    rep = edgeRepresentative[i]
72.    edgeComps[hash[rep] - 1].append(i)
73.endfor
74.
75.result[n][m] = {0}
76.
77.// provide new labelings for the rows of result matrix
78.vertexLabelings = []
79.for component in vertexComps do
80.    for val in component do
81.       vertexLabelings.append(val)
82.    endfor
83.endfor
84.
85.// fill the result matrix column by column,
   // with value 1 if there is an edge between vertex and edge
86.
87.c = 1 // c is used for moving to the next column
88.for component in edgeComps do
89.    for val in component do
90.       // Get new labelings for vertices
91.       newfrom = vertexLabelings[edgeList[val].first];
92.       newto = vertexLabelings[edgeList[val].second];
93.
94.       result[newfrom][c] = 1;
95.       result[newto][c] = 1;
96.       c = c  + 1
97.    endfor
98.endfor
99.return result
```

IV. Results and Conclusion

We successfully got a segregation of the disconnected components in the graph and the diagonal representation of adjacency and incidence matrix representation of all the disconnected components in the form of both adjacency and incidence matrix representation. For example, the following graph of two components,

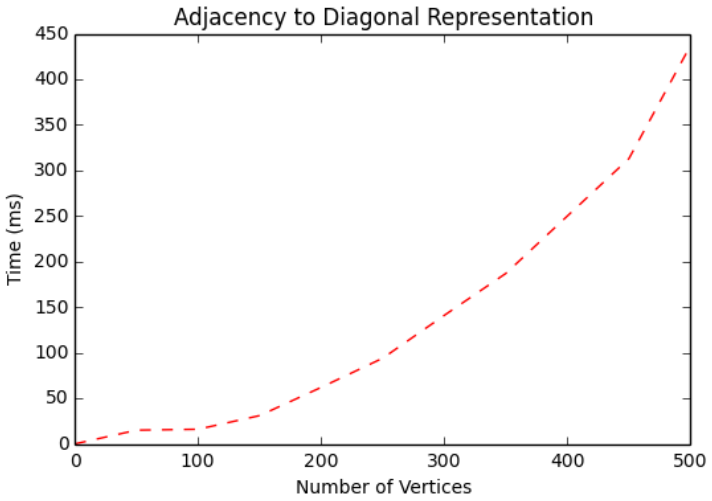


is converted to diagonal matrix of 2 disconnected components as follows,

Adjacency matrix converted to diagonal matrix of components
(Green marked areas represent the subgraph matrix of each component)

For each of the above algorithm proposed, we list below their time and space complexities.

A. Adjacency matrix to diagonal matrix of disconnected components



However, in the case where 0's or 1's dont occur contiguously may result in an increase in the space required. For example,
Space complexity -

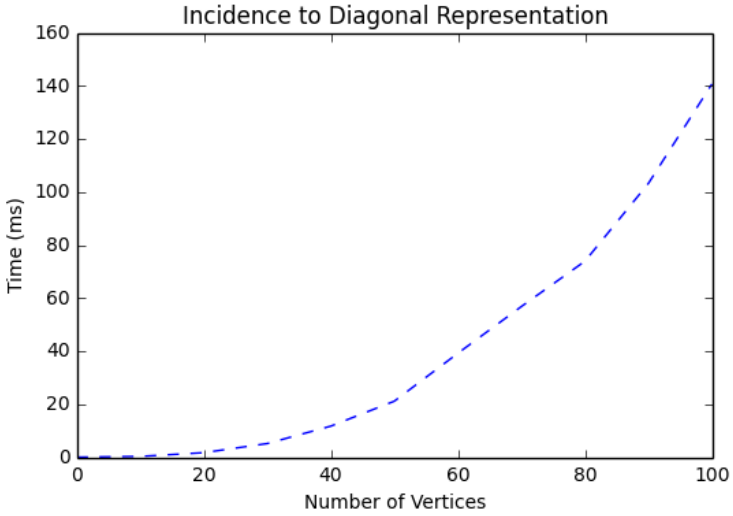
- O(n*n) for adjacency matrix.

Time complexity -

- O(n*n) for adjacency matrix.

where n - number of vertices, m - number of edges

B. Incidence matrix to diagonal matrix of disconnected components



Space complexity -

- O(n*m) for incidence matrix.

Time complexity -

- O(n*m) for incidence matrix.

where n - number of vertices, m - number of edges

REFERENCES

[1] $https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)$
[2] $https://en.wikipedia.org/wiki/Connected_component_(graph_theory)$
[3] $https://en.wikipedia.org/wiki/Breadth-first_search$