Graph Visualizer

Given a adjacency or a incidence matrix construct the corresponding graph

Aneesh Epari (IWM2014502) Kshitij Tripathi (ISM2014501) Rishabh Verma (ICM2014004)

IIIT Allahabad

***Abstract*— This report describes about how a directed/undirected graph can be constructed from two of its representations Adjacency matrix and Incidence matrix.**

***Keywords*— Edges, vertices, Adjacency matrix, Incidence matrix, Directed graph, Undirected graph**

I. Introduction

Graph consists of vertices and edges. These vertices and edges hold certain mutual relations. In order to describe a graph into any representation, relationships between these vertices and edges should be preserved. Hence, Incidence and Adjacency matrix are the two suitable representations to express the graph. This document discusses about various methods to visualize graph from its representations.

II. Motivation

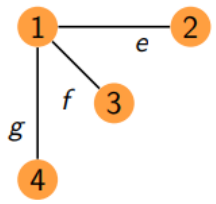
This is the era of internet in where everyone is connected to everybody no matter how big distance is between us. This is also called a Network. The graphical visualization of a network is called graph which could be otherwise viewed as relationship between nodes. In a people network each person plays the role of a node. If we increase the number of nodes the complexity of network increases, Graph visualizer helps graphically see the connectivity among the nodes in a complex network. More is the connectivity, more related are the nodes in the network.

III. Background And Methodology

Our input for the construction of graph may be an incidence or adjacency matrix. If it is an adjacency matrix we could directly proceed to construction of the matrix but if it's an incidence matrix we first need to convert it to its corresponding adjacency representation.

The incidence matrix is of 2 kinds oriented and un-oriented incidence matrix. For the directed graph we have oriented incidence matrix and un-oriented for the un-directed graph [5].
Definition of **Incidence Matrix**: Let G = (V, E) be a graph where V = {1, 2, 3. . . n} and E = {e_1, e_2. . .e_m}.

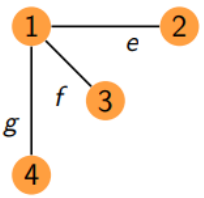
The incidence matrix of G is an n × m matrix B = (b_{ik}), where each row corresponds to a vertex and each column corresponds to an edge such that if e_k is an edge between i and j, then all elements of column k are 0 except b_{ik} = b_{jk} = 1.



$$B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 1 Incidence Matrix Representation

Definition of **Adjacency Matrix**: Let G = (V, E) be a graph with no multiple edges where V = {1, 2, 3. . . n}.
The adjacency matrix of G is the n × n matrix A = (a_{ij}), where a_{ij} = 1 if there is an edge between vertex i and vertex j and a_{ij} = 0 otherwise.



$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Figure 2 Adjacency Matrix Representation

Algorithm – 1

Input: Let Incidence [][] be the incidence matrix
Output: Return the corresponding the adjacency matrix

Procedure:
1. Let V be the number of Vertices
2. Let E be the number of Edges
3. Let Adj[V][V] be the Adjacency matrix with all entries initialized to 0.
4. For each edge e in Incidence
   4.1. If (Incidence[v1] [e] is 1 And Incidence[v2] [e] is 1)
      4.1.1. Adj[v1] [v2] = 1
5. Return Adj

After having the adjacency matrix, the task remains to simply layout the vertices and construct the graph according to the adjacency among the vertices.

For achieving sophisticated 3D results, we have used the existing framework for Java Graph Visualization and Python NetworkX Frameworks which use force fields to graphically layout the vertices. Along with former implementations, we have used a simple circular layout algorithm where we have positioned the nodes circularly.

Algorithm-2 is used to compute the positions of the vertices on the circle given the number of vertices.

Algorithm – 2

Input: V – Number of Vertices,
Output: circlePoints[V] – Coordinate Positions of Vertices

Procedure:
1. Let V be the number of Vertices
2. Create a circlePoints array to store the Coordinate position of vertices.
3. Let slice be (2 * PI) / V.
4. Let radius be V / 2.
5. Let center of circle be C (V / 2, V / 2).
6. For i from 1 to V
   6.1. Let angle be slice * i.
   6.2. circlePoints[i]. x = C. x + radius * cos(angle).
   6.3. circlePoints[i]. y = C. y + radius * sin(angle).
7. Return circlePoints

After positioning the nodes circularly, we simply draw lines between the vertices if the adjacency entry corresponding to v1 and v2 is 1.

Algorithm – 3

Input: Adj – Adjacency Matrix, circlePoints (assuming simple undirected graph)
Output: Constructed Circlarly Layed out graph

Procedure:
1. Let Adj be the adjacency matrix.
2. For v1 from 1 to V
   2.1. For v2 from 1 to V
      2.1.1. If (Adj[v1] [v2] is 1)
         2.1.1.1. drawLine(circlePoints[v1], circlePoints[v2])

IV. Implementation And Results

To implement the Graphics for the Graph we've used API's from Java, python which provide methods to graphically display the vertices, edges, directed graphs and undirected graphs. Along with it we've also implemented from scratch a basic JAVA SWING [4] User Interface complemented with JAVA AWT, JAVA GRAPHICS Libraries to display the Graph GUI.

A. Java Swing GUI and Graphics2D implementation

The implementation is pretty simple and straightforward. Prior to the construction of the matrix firstly we need to set up a Grid or a co-ordinate plane where we could draw points, lines and extract co-ordinates. To do this we will firstly divide the JPanel into a number of virtual cells. We divide JPanel into V * V cells.

Cell width = Panel width / V

Cell height = Panel height / V

Now that we have the cells, we need to extract the points from each cell, We'll be each Vertex approximately in the middle of each cell. We can calculate the x, y co-ordinates of each Center(C) corresponding to each Cell using the top-left point(TL), Cell Width and Cell Height.

C.x = (TL.x + Cell Width / 2)

C.y = (TL.y + Cell Height / 2)

After this preprocessing step, we can now construct the graph from its adjacency matrix. To achieve we would just need to override the paintComponent(Graphics Object) method of the Grid Class. At each instant there's repaint method call which in turn calls paintComponent(Graphics Object). This would reflect any changes in Grid with time.

The input would be the adjacency matrix. Firstly, we would place all the vertices (V) along the circumference of circle based on the relation explained in the methodology.

corresponding to each vertex we'll be hashing it co-ordinates. Now, based on whether the adjacency matrix entry, we draw a line between two vertices given the co-ordinates to indicate the edge between two. (Algorithm – 1, 2, 3)

Below figures – 3, 4, 5 show the graph corresponding to some random adjacency matrix.

B. JAVA Graph Stream API

For the other implementation we have used the GraphStream[1] API. To construct the graph, we just need to use the graph object of the GraphStream class and add edges according to the entries in the adjacency matrix. For laying out nodes, the API uses a force driven algorithm that makes all the nodes repel other nodes and a contradictory force that makes all nodes tied by an edge attract each other. The edges have a default length of one graph unit and try to fit this length. The result of this force algorithm is that the graph will tend to make nodes tied with each other close.

The API methods used are the addNode(String) and the addEdge(String, String, String). The parameters correspond to the unique identifier for vertices and edges respectively

C. NetworkX Python implementation

For visualizing the graph properly in Python its packages NetworkX and Matplotlib are used [2][3].

NetworkX is a python package for the creation, manipulation and study of the structure, dynamics and functions of complex networks. Matplotlib is a Python-2D plotting library which produces plots, histograms, bar charts and error charts etc.

1. For designing the matrix, in case of Adjacency matrix only number of vertices are taken as input and in case of Incidence matrix, both number of vertices and edges are taken as inputs.
2. After scanning whole matrix, in graph structure all edges are added up in graph in such a way that if there is a vertex between i and j then the tuple (i,j) is added to graph.
3. After creating the edge structure of graph, some of drawing parameters are edited such as node color, edge color, edge thickness, node labelling font type etc.
4. Finally, after defining all the parameters, Graph is constructed.

The implementation is being done in 2 stages: -

Input parameters consist of number of vertices, edges followed by Adjacency matrix or Incidence matrix. During the matrix input, in both types of matrices, edges are estimated by checking the entries of matrix and simultaneously added in the graph as tuple (i,j) if there is an edge between vertex i to j.

In 2nd step, all visual parameters related to graph like vertex color, edge color, edge thickness, font-type etc. are initialized and graph is plotted.

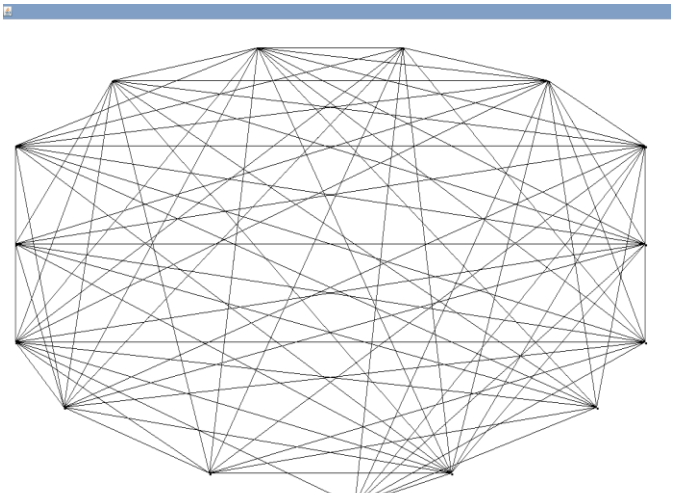


Figure 3 No. of Vertices is 15 (Implementation – A)

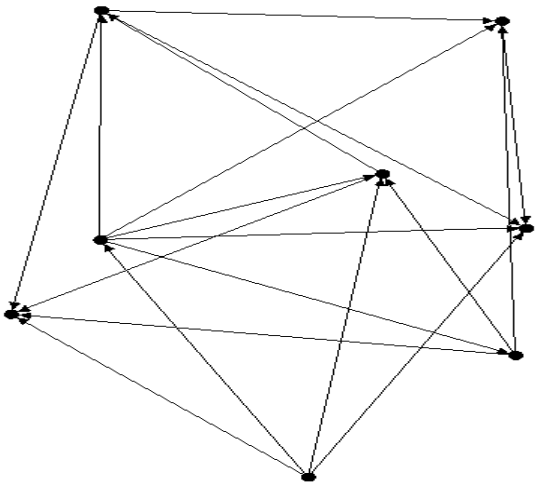


Figure 4 Directed Graph (Implementation – B)

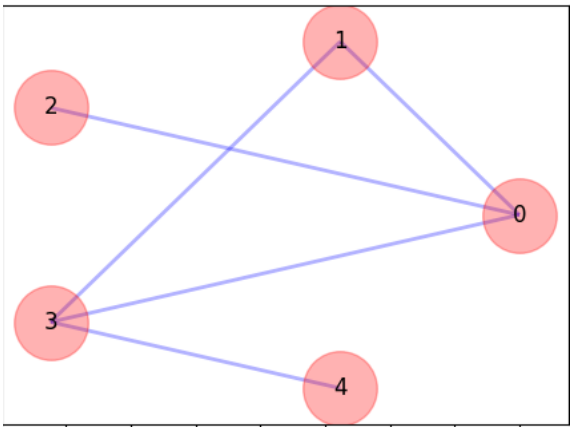


Figure 5 Undirected Graph (Implementation – C)

Conclusion

We have worked out different types of implementations in various languages such as python, JAVA to help visualize the graph from its Incidence and Adjacency matrix.

REFERENCES

[1] "GraphStream", GitHub, 2017. [Online]. Available: https://github.com/graphstream. [Accessed: 14- Aug- 2017].

[2] "networkx/networkx", GitHub, 2017. [Online]. Available: https://github.com/networkx/networkx. [Accessed: 14- Aug- 2017].

[3] "matplotlib/matplotlib", GitHub, 2017. [Online]. Available: https://github.com/matplotlib/matplotlib. [Accessed: 14- Aug- 2017].

[4] "Java Platform SE 7", Docs.oracle.com, 2017. [Online]. Available: https://docs.oracle.com/javase/7/docs/api/. [Accessed: 14- Aug- 2017].

[5] V. Balakrishnan, Introductory discrete mathematics. Englewood Cliffs, N.J.: Prentice Hall, 1991.