Construction of a Graph from the given Incidence Matrix

For un-directed, directed and mixed graphs

Divija Palleti
IHM2016005[1]

Nidhish Kumar Reddy
ICM2016006[2]

Nanda Kishore
IWM2016002[3]

***Abstract*— The objective of this report is to describe how directed, un-directed and mixed graphs can be constructed from their Incidence matrix.**

***Index Terms*— Edges, Vertices, Un-directed graph, Directed graph, Mixed graph, Graph Representation.**

I. INTRODUCTION

Graph is a collection of finite sets of vertices or nodes **(V)** and edges **(E)**. In order to describe a graph into any representation, relationship between edges and vertices should be preserved. There are variety of ways to represent a graph. One of them is Incidence Matrix. Edges are represented as ordered pairs **(u, v)** where **(u, v)** indicates that there is an edge from vertex u to vertex v.

- Un-directed graph is a graph in which all the edges are bidirectional, essentially the edges dont point in a specific direction.
- Directed graph is a graph in which all the edges are unidirectional.
- Mixed graph is a graph in which both directed and un-directed edges may exist.

This document discusses about how to visualize a graph from its incidence matrix representation.

II. MOTIVATION

Graphs are one of the ancient and powerful tools that visually represent sets of information and therefore the relations among them. They have a really long history of serving to individuals communicate, perceive the important world, and solve scientific issues. Graph visualization helps users gain insight into data by turning the information components and their internal relationships into graphs. Graph visual image leverages the human visual system to support data discovery. it's been wide employed in several applications, like social networks and net communications.

III. ALGORITHM DESIGN

A. Background

A Incidence matrix of a graph gives the Binary-matrix which has a row for each vertex and column for each edge. In an incidence matrix :

- ***(v,e)* = 1** refers to edge e incident from a vertex v.
- ***(v,e)* = -1** refers to edge e incident to a vertex v.
- ***(v,e)* = 0** refers that the vertex v and edge e aren't incident at all.

B. Methodology

We have approached the problem of graph representation using simple circular layout algorithm when all the nodes or all the vertices are equally positioned circularly. We plot the*V* vertices on a imaginary circle of radius *r* and center *C (X, Y)*.

Initially we plot an axes of 100 units in both x and y direction. Let the radius *r* be 25 and the center *C (X, Y)* be (50,50). So,if we have *V* vertices we plot *V* points at a radius of 25 around the point (50,50) using :

$$V_{\mathrm{x}} = C_{\mathrm{y}} + r * \cos\theta \tag{1}$$

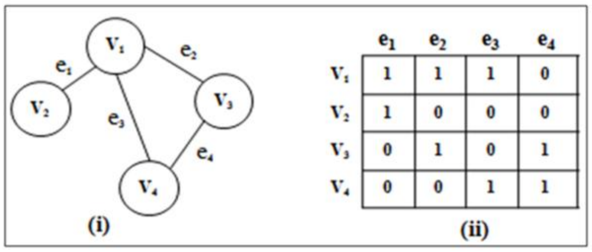
$$V_{\mathrm{y}} = C_{\mathrm{y}} + r * \cos\theta \tag{2}$$

where

$$\theta = 2 * \pi * i/V - (i^{\mathrm{th}} - vertex) \tag{3}$$

We make use of the function *drawgraph* to extract data from the given matrix :

- Number of vertices will be the number of rows of the given Incidence array/list.
- A start variable is utilized in order to store the value of the starting vertex for an edge.
- A end variable is utilized in order to store the value of the ending vertex for an edge.
- For an edge starting and ending at the same vertex both the start and end vertices are same.
- A 2D - list is formed with the above mentioned information.
- We also deal with the cases of self loop by measuring the length of the each element in the list.



	e1	e2	e3	e4
V1	1	1	1	0
V2	1	0	0	0
V3	0	1	0	1
V4	0	0	1	1

Fig. 1. Incidence matrix of an Un-directed Graph

$$\begin{array}{c|ccccccc} & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 \\ \hline v_1 & 1 & 0 & 0 & 0 & 0 & 1 & -1 \\ v_2 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ v_3 & 0 & -1 & 1 & 0 & 1 & 0 & 0 \\ v_4 & 0 & 0 & -1 & -1 & 0 & -1 & 0 \\ v_5 & 0 & 0 & 0 & 1 & -1 & 0 & 1 \end{array}$$

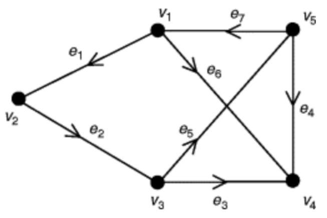


Fig. 2. Incidence matrix of a Directed Graph

C. Algorithm

Algorithm 1 Draw_Graph

```
procedure DRAW($no\_edge, no\_vert, x, y, verts$)
    $start \leftarrow 0$
    $end \leftarrow 0$
    for i=0 to (no_edge-1) do
        for j=0 to (no_vert-1) do
            if matrix[j][i]==1 then
                $start \leftarrow verts[j]$
                $ones.append[start]$
            if matrix[j][i]==-1 then
                $end \leftarrow verts[j]$
                $flag \leftarrow 1$
        if flag==1 then
            $ones.append(end)$
            if $ones.len$ == 1 then
                $ones.append(ones[0])$
            $ones.append(1)$
        if flag==0 then
            if $ones.len$ == 1 then
                $ones.append(ones[0])$
            $ones.append(0)$
        $edges.append(ones)$
    $return\ edges$
procedure MAIN                                  ▷ main function
    $midx \leftarrow 50$
    $midy \leftarrow 50$
    $radius \leftarrow 25$
    for i=0 to $(no\_vert)$ do
        $verts.append(i)$
    for i=0 to $(no\_vert)$ do
        $x.append(radius * cos(2 * PI * i/v) + midx)$
        $y.append(radius * sin(2 * PI * i/v) + midy)$
    $edges \leftarrow DRAW(no\_edge, no\_vert, x, y, verts)$
    $Plot(edges)$
```

Algorithm 2 Label_Graph

```
procedure WHICHQUA($x, y$)
    if x>=50 $and$ y>=50 then
        $return\ 0$
    if x>= 50 $and$ y<50 then
        $return\ 1$
    if x<=50 $and$ y<=50 then
        $return\ 2$
    if x<50 $and$ y>=50 then
        $return\ 3$
procedure MARK($x, y, val$)
    $qua \leftarrow whichqua(x, y)$
    $distance \leftarrow 1.5$
    if qua == 0 then
        $x \leftarrow x + distance$
        $y \leftarrow y + distance$
    if qua == 1 then
        $x \leftarrow x + distance$
        $y \leftarrow y - distance$
    if qua == 2 then
        $x \leftarrow x - distance - 1$
        $y \leftarrow y - distance - 1$
    if qua == 3 then
        $x \leftarrow x - distance$
        $y \leftarrow y + distance$
    $plt.text(x, y, val)$
procedure PLOT($edges$)
    for i=0 to $(no\_edge - 1)$ do
        $x0 \leftarrow x[edges[i][0]]$
        $x1 \leftarrow x[edges[i][1]]$
        $y0 \leftarrow y[edges[i][0]]$
        $y1 \leftarrow y[edges[i][1]]$
        $directed \leftarrow edges[i][2]$
        if directed == 0 then
            $drawundirectededge$
        if directed $!= 0$ then
            $drawdirectededge$
        $mark(x0, y0, edges[i][0])$
        $mark(x1, y1, edges[i][1])$
        if x1 == x0 $and$ y1 == y0 then
            $markpointingreencolortoindicateselfloop$
    $plt.show()$
```

D. Output Format

- A graph is constructed using *matplotlib* library of python.
- All the vertices are represented by red and all the edges by black.
- All the self looping vertices are represented by green.

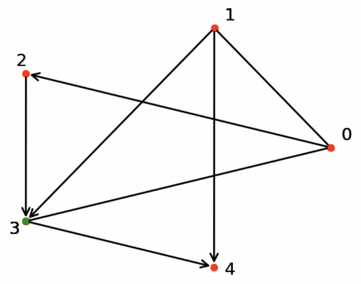


Fig. 3. Output of an Incidence matrix of a mixed graph

IV. Analysis

A. Analysis

1) Best Case: Best Case occurs when the graph has just one edge.So the time complexity will be directly proportional to constant time, i.e $O(1)$. But when we consider a connected graph the minimum number of edges is $V-1$ (V- number of vertices , E- number of edges). And we need to traverse the entire matrix to extract the edges. So the time complexity will be directly proportional to $O(V*E)$.

$$time_{best} \propto O(V*E)$$

But we know that E = $V-1$ (best case of a connected graph).

$$\implies time_{best} \propto O(V*(V-1))$$

$$\implies O(V^2) \text{ - A Quadratic Time Computation}$$

2) Worst Case Analysis : As discussed above in the worst case scenario we have each vertex connected all the other vertices. So, for a un - directed graph maximum number of edges is $V*V-1$ and for a directed graph the maximum number of edges is $V*V-1/2($ V- number of vertices , E- number of edges). And we need to traverse the entire matrix to extract the edges. So the time complexity will be directly proportional to $O(V*E)$.

$$time_{worst} \propto O(V*E)$$

But we know that $E \propto V^2($ worst case of a graph).

$$\implies time_{worst} \propto O(V*(V^2))$$

$$\implies O(V^3) \text{ - A Cubic Time Computation}$$

3) Average Case Analysis : We know that

$$time_{best} <= t_{average} <= t_{worst}$$

But from above

$$=> V^2 <= t_{avg} <= V^3$$

4) Space Complexity: The present algorithm has a space complexity of $O(V+E)$. An additional space of V is used in storing the vertices and an extra space of E is used in storing the list which contains the information regarding the incidence of these edges.

V. Experimental Study

The best way to study an algorithm is by graphs and profiling.

1) Graphs-Time-Complexity: We have seen that $t_{best} <= t_{worst}$. So,the graphs for best case lies below worst case.We can clearly see that best case is of order V^2 and worst case is of order V^3 for connected graphs.

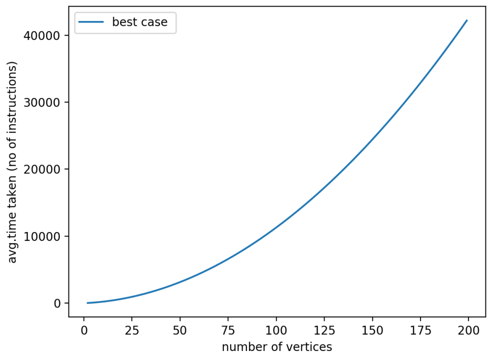


Fig. 4. Best case Complexity

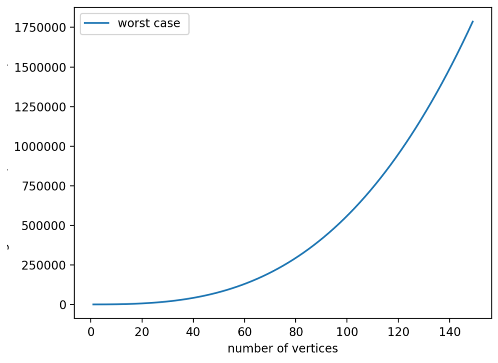


Fig. 5. Worst case complexity.

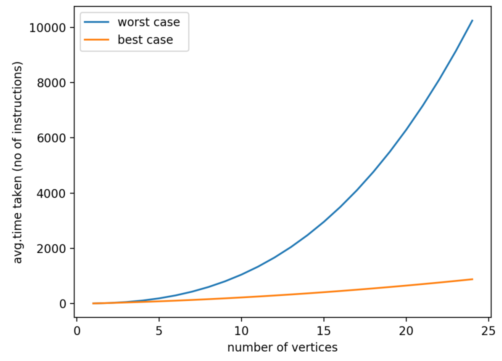


Fig. 6. Comparison between the best and the worst case.

2) Profiling:

$Table 1 : Number Of Computations$

V	t_{best}	t_{worst}
1	5	5
5	60	180
10	221	1045
15	411	2959
20	651	6294
30	1281	18739
40	2111	41384
50	3141	77229

- We can seen that $t_{best} <= t_{worst}$
- We can also see that time increases as the value of n increases and $time_{worst} \propto V^3$, $time_{best} \propto V^2$.

VI. Discussions

A. Adding offset to every vertex

In this context the $Label\_Graph$ Algorithm is used to add certain offset to vertices in order to label them.Thus we used the concept of quadrants, in order to add or subtract a buffer value from every vertex in the given incidence matrix.

VII. Conclusions

- We can see that the Complexity of Graph generation in above algorithm is $O(V^2)$ in best case,$O(V^3)$ in average and worst cases. Where V is the number of rows in the given incidence matrix.
- In this result we have restricted the size of the axes to 100 units in both X and Y direction, as these values are sufficient enough for us to understand and analyze the results.
- Simple circular layout algorithm and $matplotlib$ library were used in order to visualize the graph.

VIII. Applications

Practical implementations of the presented algorithm can be used to visualize graphs to gain insight into data by turning the information components and their internal relationships into graphs. It can be used to visualize the structure of various molecules and this helps us to study molecules in chemistry given their relation. It can also be used in visualizing the links between different places (E.g: Google Maps) where various locations are represented as vertices and the roads are represented as edges.

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

[2] Introduction to Algorithms English By Thomas H. Cormen , By Charles E. Leiserson , Ronald L. Rivest ,Clifford Stein(3rd edition)

[3] $https://github.com/matplotlib/matplotlib$

[4] $https://en.wikipedia.org/wiki/Incidence\_matrix$