Representation of Generic Graph using incidence matrix and an efficient data structure

Swapnesh Narayan,Gaganjeet Reen,Vinay Surya Prakash , Leteesh Meena

***Abstract*— The problem of representing the generic graph using efficient data structure is a well defined problem in the field of graph theory.Given a generic graph(V, E) where V are the vertices and E are the edges in the graph as entered by the user we have to represent it in the most efficient way possible. In this paper we are first representing it by the incidence matrix and then by our efficient data structure.**

I. INTRODUCTION

A graph **G(V, E)**, where **V** represent set of vertices and **E** represent set of edges, can be represented using a matrix called Incidence matrix ($\mathbf{I_{N*M}}$) where N is number of vertices and M is number of edges. In an incidence matrix(Fig 2), each column stands for an edge and each row for a vertex and hence matrix entries marks the vertices which are connected by the correspoding edges. Basically, an incidence matrix for a graph $\mathbf{G(V=v_1,v_2,....,v_N,\ E=e_1,e_2,...,e_M)}$ as an N*M matrix I, where :

1. I_{ij} = 1 if edge e_j is coming out of the node v_i
2. I_{ij} = -1 if edge e_j is going into the node v_i
3. I_{ij} = 0 otherwise

For example, for the graph given in fig.1, we have four vertices (v_1, v_2, v_3, v_4) and five edges (e_1, e_2, e_3, e_4, e_5), the incidence matrix will be represented as given below.

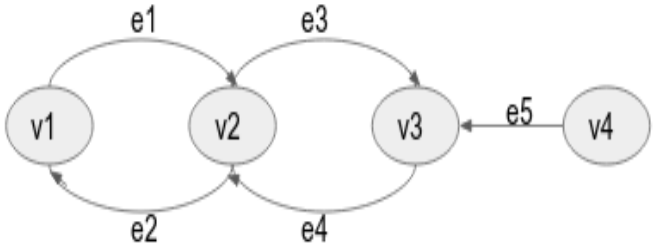

Fig. 1. An example graph

The incidence matrix in Figure 2 is created as follows : edge e_1 is directed from v_1 to v_2. So, the entry corresponding to $[v_1][e_1]$ is 1 as e_1 is coming out of v_1 and that of $I[v_2][e_1]$ is -1 as e_1 is going into the vertex v_2.

In a graph **G**, a pair of edges e_i and e_j are called parallel if and only if they are incident on the same pair of vertices i.e. if the two end points of the two edges are same then the two edges are parallel. In an undirected graph, there is only one type of parallel edges because no direction is involved. But in a directed graph, parallel edges are of two types: parallel and anti-parallel . If two edges are starting from same vertex and ending at the same vertex then the two edges are simply parallel. If one edge starts at a vertex where the other ends and also if this edge ends where the second edge starts, then the two edges involved are said to be anti-parallel.

vertices/edges	e1	e2	e3	e4	e5
v1	1	-1	0	0	0
v2	-1	1	1	1	0
v3	0	0	-1	-1	-1
v4	0	0	0	0	1

Fig. 2. Corresponding incidence Matrix

II. MOTIVATION

The ordinary incidence matrix is capable of representing a simple graph, however it fails in case the graph contains weighted self loops and weighted parallel edges.The representations suggested in this paper take care of the above cases.In case of the incidence matrix, instead of using +1 and -1 to represent the edges entering and leaving a vertex we have used +w and -w to indicate the weighted edges and the self loops. The efficient structure also allows for storing the weight of each link as well as any number of edges from one vertex to another without a great increase in the storage space(advantage over the incidence matrix).

III. METHODS AND DESCRIPTION

In this paper we are introducing new data structure(described below) to efficiently represent the generic graph conaining self loops, directed and undirected edges, parallel and anti-parallel edges , weighted and unweighted edges. The structure used is :-

```
struct Node
{
        struct OutgoingLink
        {
                int weight;
                Node *to;
        }
        struct IncomingLink{
                int weight;
```

```
        Node *from
    }
    int value;
    vector <OutgoingLink >outgoinglink;
    vector <IncomingLink >incominglink;
}
```

Space complexity comparison between the two methods:- Given the number of edges in a graph to be E and the number of vertices to be V,the incidence matrix has a space complexity of **O(V*E)**.The efficient representation has a space complexity of $O(\sum_{n=1}^{V} degree(in) + degree(out))$.

In the efficient representation,each node in the graph is represented as a structure as shown in the structure above. All the nodes are stored in a map which contains a mapping of the node number to the node pointer.

The structure used contains the following data -:

1) The vertex number(int value)

2) A structure representing incoming links from the node(struct IncomingLink)

This structure contains the following data

a) Weight of the incoming link(int weight).

b) The node that the link originates from(Node* from).

3) A structure representing the outgoing links from the node (struct OutgoingLink).

This structure contains the following data

a) Weight of the incoming link(int weight).

b) The node that the link goes to. (Node * to).

4) A vector containing all the outgoing links originating from the node(vector <OutgoingLink >)

5) A vector containing all the incoming links entering the node.(vector <IncomingLink >)

IV. IMPLEMENTATION

We have used C++ to convert the graph into the adjacency matrix as well as the efficient data structure.The graph is taken as an input from the user in the form of edge lists.For each edge the user is required to enter 3 space separated values representing starting vertex, ending vertex and the weight of the edge.In order to stop entering edges,the user needs to enter a negative number.

V. RESULTS

Reconstructing the graph using the efficient representation is much easier as the time complexity to traverse the incidence matrix is O(V*E), whereas the time complexity for reconstruction using the modified representation is O(V* max(degree(in)+degree(out))). The incidence matrix generated is shown in Fig 3

Github Repository Link : `https://github.com/piano-man/graph_theory.git`

VI. DISCUSSION

In this paper we have shown that how the efficient data structure we used is more powerful than the modified incidence matrix. The traditional incidence matrix is incapable of representing a graph having self loops and weighted parallel

Enter 1 for directed and 0 for undirected
1

Enter the edges in this order : Start , End, Weight (Enter - 1 if you want to stop):
12 4 5
4 12 6
32 42 5
42 42 7
6 7 8
-1

Properties of the Graph are:
Directed
Weighted
Contains Self Loop
Doesn't contain Parallel Edges
Not a Simple Graph

Vertices/ Edges	4	6	7	12	32	42
1	-5	0	0	5	0	0
2	6	0	0	-6	0	0
3	0	0	0	0	5	-5
4	0	0	0	0	0	7
5	0	8	-8	0	0	0

Fig. 3. Generated Output

edges. We have compared both the ways of representation on the basis of space complexity as well as time complexity(time required to regenerate a graph using the data structures described).The regeneration time is lesser when we use the efficient data structure because in the incidence matrix,for every vertex,all the edges in a graph are explored whereas in the efficient structure only the edges incident on the vertex and starting from the vertex are explored.

VII. CONCLUSIONS

A generic graph having self loops can be represented using the modified incidence matrix suggested in this paper and can also be represented more efficiently using the data structure described in the paper.

REFERENCES

[1] Narsingh Deo, Graph Theory with Applications to Engineering and Computer Science.

[2] https://stackoverflow.com/questions/14022701/good-data-structure-for-representing-multigraph-c

[3] http://mathworld.wolfram.com/IncidenceMatrix.html