Condensed Representation of Incidence Matrix for Directed Graph with self-loops

***Abstract*—Proposed are three methodolgies which will condense any Incidence Matrix for directed graph (with self-loops) . The three methodolgies aim to reduce the space required to store the graph. At first, the base representation of incidence matrix for a directed graph with self-loops is discussed. Then, three ways of condensing the base representation are presented.**

***Index Terms*—**

I. Introduction

Graphs are mathematical equivalent of representing relationships between two objects. Various methods can be used to store graphs. Incidence matrix is a method in which vertices of a graph are denoted in rows and edges in column. Adjacency matrix is an another method of representing graphs where both rows and columns are vertices.

An incidence matrix [1] with N vertices and E edges will take N x E space for storage. We aim to reduce this storage space. Base representation will represent incidence matrix for directed graph in simplest form. Bit encoding helps in reducing matrix size to 1-D matrix with dimension N. Vector ¡pair>representation will help in reducing the space further storing the edges data only reducing the redundant data stored in the base representation.

II. Algorithms

A. Base representation

For representing incidence matrix for directed graph (with self-loops), we will use the matrix 'A' of size N x E, where N is the number of vertices in graph, E is the total number edges in graph. The matrix is intialized with all values to be 0.

Starting vertices are denoted by 1, ending vertices are denoted by 2 and for self loops the vertex is denoted by 3. Therefore, for every edge 'e' starting from vertex 'v1' to 'v2', values assigned such that, A[v1][e] = 1, A[v2][e] = 2. In case of self-loops at any vertex 'v', value is assigned such that, A[v][e] = 3.

B. Base representation with destination

For representing incidence matrix for directed graph (with self-loops), we will use the matrix 'A' of size N x E, where N is the number of vertices in graph, E is the total number edges in graph. The matrix is intialized with all values to be -1.

For each edge, the value corresponding to the starting vertex denotes the final vertex. Therefore, for every edge 'e' starting from vertex 'v1' to 'v2', values assigned such that, A[v1][e] = v2.

In case of self-loops at any vertex 'v', value is assigned such that, A[v][e] = v.

C. Bit encoding

Given an incidence matrix 'A' in the base form given in Algorithm A, we can compress it to another matrix 'B' of dimension N, where N = number of vertices in graphFor every vertex 'v', value assigned to B[v] = conversion of Quaternary number to decimal number. Quaternary numbers for every vertext 'v' can be obtained by appending values of every column in A[v] row.

D. vector ¡pair>representation

Given an incidence matrix 'A' in the base form given in Algorithm A, we will define a data structure of the form vector¡pair¡int, int>>I. While traversing every edge of incidence matrix, we will push pair¡int, int>in the defined vector I as follows:

For every edge 'e' between 'v1' and 'v2', if:

- A[v][e] = 1 ->v1 = v;
- A[v][e] = 2 ->v2 = v;
- A[v][e] = 3 ->v1 = v, v2 = v;

III. Results

From given figures, it can be clearly seen that:

1) For variable value of N and E, there is a linear increase in amount of reduction of memory used as the size of the input matrix increases.
2) For fixed value of N and increasing value of E, memory used in less in case of small V and large E in case of input matrix and vice-versa

IV. Conclusion

In this paper, we proposed two basic representation of incidence matrix for directed graph(with self-loops) and two approaches for compressing an incidence matrix. From all the approaches, it can be deduced that:

- For comparable values of N and E, vector representation was more optimal in terms of time and space.
- If E is greater and not comparable to N, base representation was more optimal in terms of time and space.
- For higher values of edges, Bit encoding algorithm is not efficient.

References

[1] N. Deo, *Graph Theory with Application to Engineering and Computer Science*. Englewood Cliffs, N.J.: Prentice-Hall, 1974.

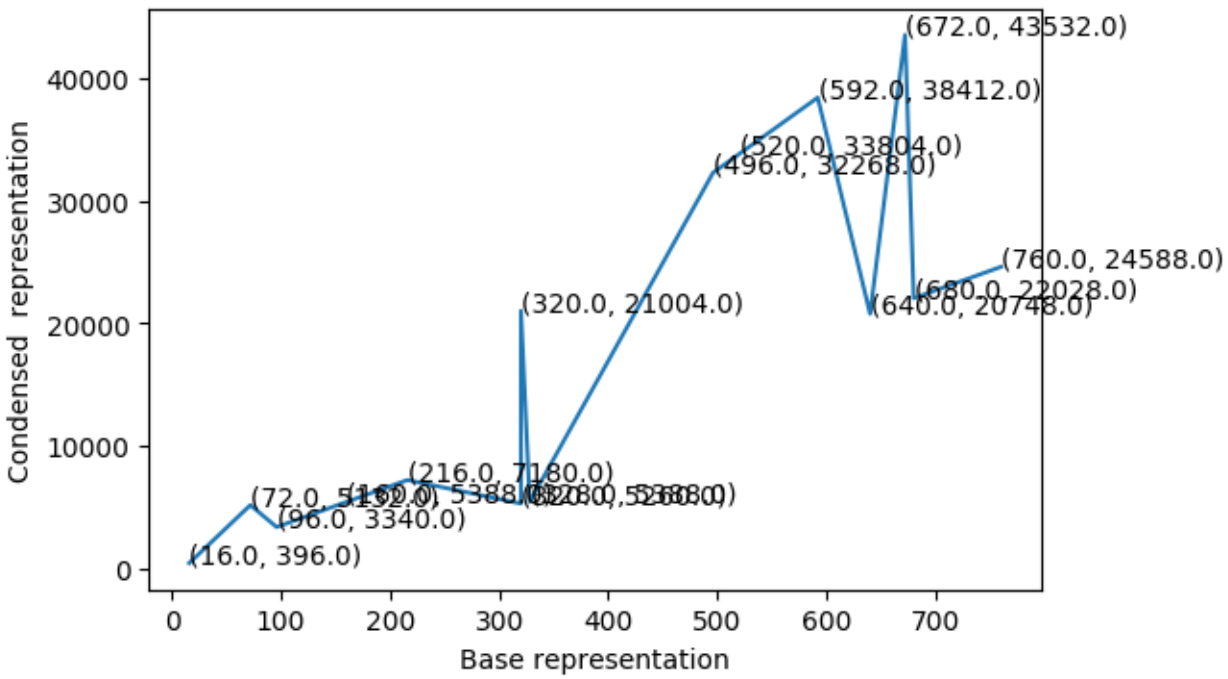

Fig. 1. **Memory analysis for variable N and E**

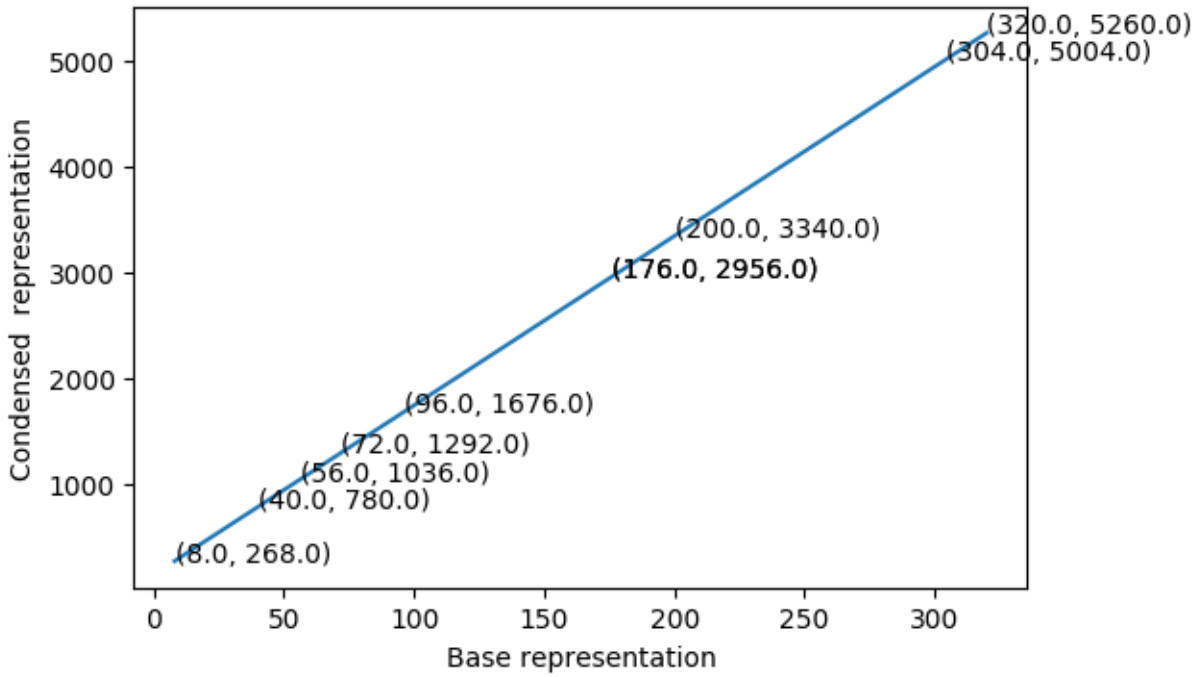

Fig. 2. **Memory analysis for fixed N and increasing E**