Condensed Representation of Adjacency Matrix of simple graph and transform it into an Incidence Matrix

Jatin Goel [1], Shubham Vashishtha [2]

[1]IT Department,Indian Institute of Information Technology,Allahabad
Deoghat,Jhalwa-211015 India
icm2014503@iiita.ac.in
ihm2014007@iiita.ac.in

Abstract—*This report studies about the various method to represent the Adjacency Matrix representation of simple graph (undirected ,unweighted ,no parallel edges and no self loops) in condensed form. We categorized our graph into dense and sparse graph.We used Adjacency List representation for sparse graph and decimal representation for dense graph.We then used bitmasking based technique to represent the dense graph which gives us better compression ratio.After condensing the graph we transformed condensed graph to Incidence Matrix representation of Graph.*
Later part this report have detailed discussion on memory space compression and time complexity analysis of various methods used.

Keywords — *Graph Representation , Bitmasking, Compression of Graph*

1. Motivation

The main motivation behind the graph condensation is limitation of main memory in our computer system.Suppose we have billion of nodes in graph and we have to apply some graph algorithm on that graph.First of all for execution of any algorithm the graph should be in main memory,as we know we can not store such a big graph in our main memory so we have two options now ,one is to increase main memory which is costly or other is to come up with an idea to store the same graph in some compressed form,so that we can apply our algorithm to get desired result from that graph.We can see one such application in social networking sites where large number of users are connected in the form of graph.

2. Problem Description

We have a Graph G(V,E) in the form of Adjacency Matrix A of size [VxV], Our main objectives are :

- To give different condensed representation of the graph so that memory can be utilized efficiently.
- To transform condensed representation of graph to Incidence Matrix I of size [VxE] directly.
- Memory Analysis in condensed form representation and Adjacency Matrix Representation versus input size.
- Time Complexity Analysis of a graph Algorithm in Adjacency Matrix Representation and in condensed form versus input size.

3. Approach

We have categorized Graph into two kind of graphs based on the density of edges between vertices , we divided our approach based on the type of graph :

Sparse Graph : Graph having lesser number of edges compare to the maximum number of edges possible.

Adjacency List Representation : A vector of lists is used. Size of the vector of lists is equal to number of vertices. Let the vector be G. An entry G[i] represents the list of vertices adjacent to the *i*th vertex.
Suppose we have following Adjacency Matrix of the Graph :

V	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	0	0
3	0	1	0	1	0
4	0	0	1	0	1
5	1	0	0	1	0

Corresponding Adjacency List :

$1 \rightarrow 2 \rightarrow 5$
$2 \rightarrow 1 \rightarrow 3$
$3 \rightarrow 2 \rightarrow 4$
$4 \rightarrow 3 \rightarrow 5$
$5 \rightarrow 1 \rightarrow 4$

Algorithm : To convert Adjacency Matrix to Adjacency List

Input : Adjacency Matrix
Output : Adjacency List

Pseudo Code :

```
For i=1 to V
{
        For j=1 to V
        {
                if(A[i][j])
                {
                        Adj_List[i].add(j);
                        Adj_List[j].add(i);
                }
        }
}
```

Row-wise Decimal Representation : We can represent the graph in row-wise decimal form , we have rows as a binary string and we can convert each row as 32 bit decimal number and store only that number instead of complete Array of integers.But this is method is limited to 32/64 vertices based on integer data type we chosen to convert in.
`Int` - 32 Vertices
`long long Int` - 64 Vertices

Suppose we have following Adjacency Matrix of the Graph :

V	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	0	0
3	0	1	0	1	0
4	0	0	1	0	1
5	1	0	0	1	0

Decimal Row-wise Representation of above graph :

D_Val = Decimal Value of Row

V	D_val
1	9
2	20
3	10
4	5
5	18

ALGORITHM : TO CONVERT ADJACENCY MATRIX TO ROW-WISE DECIMAL REPRESENTATION
INPUT : ADJACENCY MATRIX
OUTPUT : ROW-WISE DECIMAL REPRESENTATION

Pseudo Code :

```
For i=1 to V
{
        d = convert_decimal(i_row);
        Dec_Graph[i] = d;
}
```

Dense Graph : Dense graph is a graph in which the number of edges is close to the maximal number of edges.

Bit-masking based Representation : To overcome the decimal based limitation ,we can use bit masking based technique in which we can concatenate integers and use their bit positions using modular arithmetic.We can set the particular bit of an integer using bit masking.
Suppose we have a graph of 75 vertices , we can not represent this graph in decimal integer of 32/64 bit Integer.
So we can concatenate three 32 bit integer per vertex.
Total bits = 32*3 = 96
Now we can represent 96 binary values in only three integers instead of using 75 integers.

Suppose we have a graph of 75 vertices :
For Adjacency Matrix Representation :
Space Required - `bool` A[75][75] = 75*75*1=5625 `bytes`

Bit-masking based Representation :
Space Required - `int` BM[75][3] = 75*3*4 = 900 `bytes`

Suppose we have edge :
$x \rightarrow y$
k = (y-1)/32 + 1;
j = (y-1)%32 + 1;

Now set jth bit in B[x][k].

For Example :

2 → 67 : Set 3th bit of BM[2][3]
63 → 74 : Set 10th bit BM[63][3]
34 → 45 : Set 13th bit BM[34][2]

ALGORITHM : TO CONVERT ADJACENCY MATRIX TO BIT MASKING BASED REPRESENTATION

INPUT : ADJACENCY MATRIX
OUTPUT : BIT MASKING BASED REPRESENTATION

Pseudo Code :

```
Input  :  A[V][V]
V = No. of vertices in Graph
k = Ceil(V/32);
Output : BMG[V][k]
For i=1 to V
{
       For j=1 to V
       {
              if(A[i][j])
              {
                     p = (j-1)/32 + 1;
                     q = (j-1)%32 + 1;
                     x = (i-1)/32 + 1;
                     y = (i-1)%32 + 1;

                     Set(BMG[i][p],q,true);
                     Set(BMG[j][x],y,true);

              }
       }
}
Set Method : To set a particular bit in integer.
Set (data ,pos)
{
   mask = 1 << pos;
   data = data | mask;
}
```

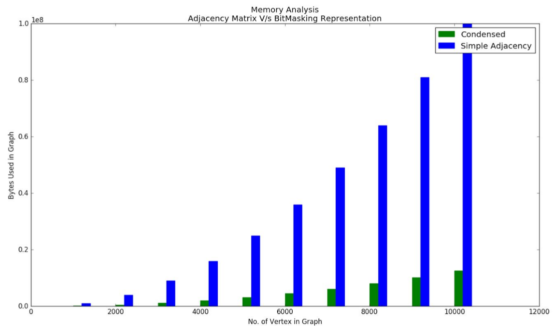
4. Memory Analysis

[1] Bit masking Matrix V/S Adjacency Matrix

V = No. of Vertices in graph
AM = Memory bytes in Adj. Matrix (In 1k bytes)
BM = Memory bytes in Bit masked Matrix (In 1k bytes)

V	BM	AM
1000	128	1000
2000	504	4000
3000	1128	9000
4000	2000	16000
5000	3140	25000
6000	4512	36000
7000	6132	49000
8000	8000	64000
9000	10152	81000
10000	12520	100000

Graphical Representation of Above Data
[Image - 1]



Avg Condensation Ratio = 87.5%

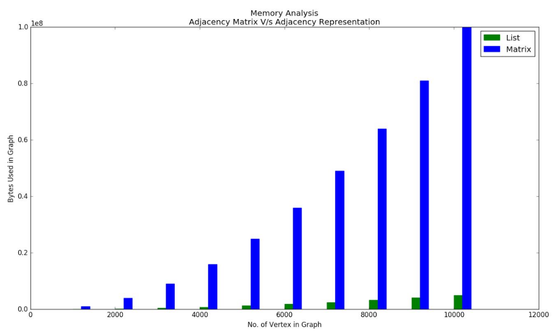
[2] Adjacency List V/S Adjacency Matrix

V = No. of Vertices in graph
E = 5% of V*(V-1)/2
AM = Memory bytes in Adj. Matrix (In 1k bytes)
AL = Memory bytes in Bit masked Matrix (In 1k bytes)

V	AL	AM
1000	50.95	1000
2000	201.90	4000
3000	452.85	9000
4000	803.80	16000
5000	1254.75	25000
6000	1805.70	36000
7000	2456.65	49000
8000	3207.60	64000
9000	4058.55	81000
10000	5009.50	100000

Avg Condensation Ratio = 95%

Graphical Representation of Above Data
[Image - 2]



5. Transform Condensed Form into Incidence Matrix

Transforming Adjacency List to Incidence Matrix: We iterate through each vertex's adjacency list. Whenever we encounter a new edge we add the corresponding edge array to a list. Finally we take the transpose of the list of arrays to obtain the incidence matrix.

Algorithm : To convert Adjacency List to Incidence Matrix

Input : Adjacency List
Output : Incidence Matrix
Pseudo Code :

```
Input  :  A[V]
V = No. of vertices in Graph
A[i] = Adjacency List of vertex i
Output : I[V][E]
I : an empty list
For i=1 to V
{
        For j=1 to deg(i)
        {
                if(i < A[i][j])
                {
                        edge : an array of size V
                        initialized by zero
                        edge[i] = 1
                        edge[A[i][j]] = 1
                        Append edge to I
                }
        }
}
return transpose(I)
```

Suppose we have following Adjacency List :
1 → 2 → 5
2 → 1 → 3
3 → 2 → 4
4 → 3 → 5
5 → 1 → 4

Corresponding Incidence Matrix:

V/E	1	2	3	4	5
1	1	1	0	0	0
2	1	0	1	0	0
3	0	0	1	1	0
4	0	0	0	1	1
5	0	1	0	0	1

Transforming Row-wise Decimal Representation to Incidence Matrix: To check whether edge exists b/w vertex i and j use bit mask operations. Edge exists if the corresponding bit is set.

Algorithm : To convert Graph Decimal Representation to Incidence Matrix

Input : Graph Decimal Representation
Output : Incidence Matrix
Pseudo Code :

```
Input  :  DEC_G[V]
V = No. of vertices in Graph
Output : I[V][E]
I : an empty list
For i=1 to V
{
        bit_pos = V-1
        For j=i+1 to V
        {
                if(DEC_G[j] & (1<<bit_pos))
                {
                        edge : an array of size V
                        initialized by zero
                        edge[i] = 1
                        edge[j] = 1
                        Append edge to I
                }
        }
        bit_pos = bit_pos - 1
}
return transpose(I)
```

Suppose we have following Decimal Row-wise Representation :

D_Val = Decimal Value of Row

V	D_val
1	9
2	20
3	10
4	5
5	18

Corresponding Incidence Matrix:

V/E	1	2	3	4	5
1	1	1	0	0	0
2	1	0	1	0	0
3	0	0	1	1	0
4	0	0	0	1	1
5	0	1	0	0	1

Transforming Bit-masking based Representation to Incidence Matrix:

Algorithm : To convert Bit-masking based Representation to Incidence Matrix
Input : Bit-masking based Representation
Output : Incidence Matrix
Pseudo Code :

```
Input : BMG[V][k]
V = No. of vertices in Graph
k = Ceil(V/32);
Output : I[V][E]
I : an empty list
For i=1 to V
{
        For j=i+1 to V
        {
                p = (j-1)/32 + 1;
                q = (j-1)%32 + 1;
                x = (i-1)/32 + 1;
                y = (i-1)%32 + 1;
                if(isSet(BMG[i][p], q)       {
                        edge : an array of size V
                        initialized by zero
                        edge[i] = 1
                        edge[j] = 1
                        Append edge to I
                }
        }
}
return transpose(I)
```

6. Time Complexity Analysis

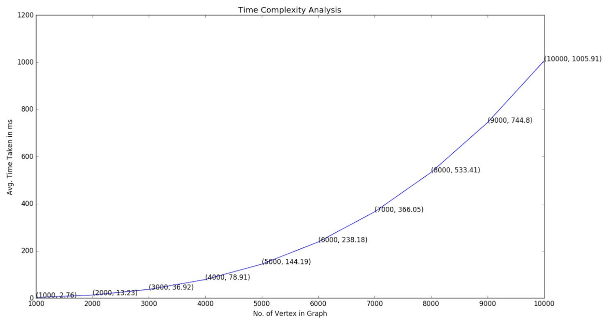
From Adjacency Matrix to Condensed Form : For all the three cases :

- Adjacency List
- Decimal Representation
- Bit masking based Representation

Time Complexity = O(V*V)
Where V = No. of vertex in Graph
Time_Taken = Average Time taken by Algorithm on 50 different graph of V vertices(in milliseconds).

V	Time_Taken
1000	2.76
2000	13.23
3000	36.92
4000	78.91
5000	144.19
6000	238.18
7000	366.05
8000	533.41
9000	744.80
10000	1005.91

Graphical Representation of Above Data
[Image - 3]



From Condensed Form to Incidence Matrix : For following cases :

- Bit masked Representation to Incidence Matrix

Time Complexity = O(V*V) [Analyzed Earlier]

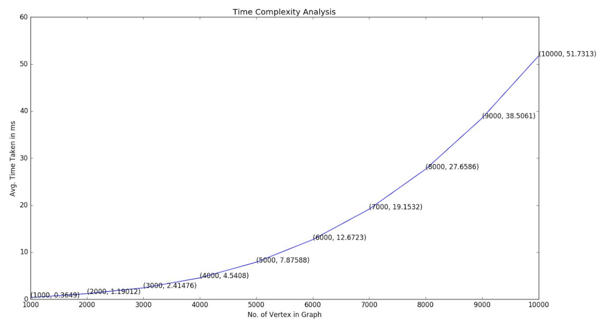
- Adjacency List to Incidence Matrix

Time Complexity = O(V + 2E)

Where V = No. of vertex in Graph
Where E = 5% of V*(V-1)/2
Time_Taken = Average Time taken by Algorithm on 50 different graph of V vertices and E edges(in milliseconds).

V	Time_Taken
1000	0.36
2000	1.19
3000	2.41
4000	4.54
5000	7.87
6000	12.67
7000	19.15
8000	27.65
9000	38.50
10000	51.73

Graphical Representation of Above Data
[Image - 3]



7. Conclusion

We conclude that for sparse graph (i.e where edges are only 5% of the maximum possible edges) adjacency list is the best condensed form whereas for dense graphs bitmask representation is much more compact.
All of the condensed forms proposed can be directly converted to incidence matrix without reverting back to adjacency matrix.