Algorithm to Identify and Count the Disconnected Subgraphs from Incidence Matrix

Abhinav Mishra
Indian Institute of
Information Technology
iim2014003@iiita.ac.in

Neelanjana Jaiswal
Indian Institute of
Information Technology
ism2014005@iiita.ac.in

Sannihith Kavala
Indian Institute of
Information Technology
ihm2014004@iiita.ac.in

***Abstract*—The problem of identifying disconnected components in a graph is a fundamental problem in computer science. Given a graph G = (V, E), the problem is to partition the vertex set V into $V_1, V_2, , V_h$, where each V_i is maximized, such that for any two vertices x in V_i and y in V_j where $i \neq j$, there is no path connecting them. In this paper, we present three algorithms to solve this problem. The algorithm preprocesses the input graph in the form of incidence matrix. We try to propose three algorithms which tries to identify and count the subgraphs of the bigger graphs having no paths among themselves. The input graph can be a directed or undirected, simple graph.**

***Keywords*—Disconnected, Subgraph, Incidence Matrix.**

I. Introduction

Graph connectivity is a fundamental problem in computer science, which has many background applications in the real world. For example, reliability is one of the major concerns in communications networks: if a network is reliable, the network would still work when some nodes or edges fail. Reliability in communication networks can be represented by the connectivity between each pair of nodes. In social networks, computing the closeness among people is a very important problem, which also relates to the connectivity of the networks. There are many other applications which are related to the connectivity of networks, e.g., finding web pages of high commonality in internet searching; finding protein complexes and gene clusters in computational biology, etc.

In theoretical computer science, graph connectivity has been well studied for more than forty years. It has a strong relationship with the problems of maximal network flow and minimal cut.

Given an undirected graph G = (V, E), where V is the vertex set and E is the edge set. The graph contains disconnected components of disconnected subgraphs iff its vertex set V can be partitioned into atleast two non-empty, disjoint subsets V_1 and V_2 such that there exists no edge in edge set E which connects vertex from V_1 to vertex from V_2. Suppose that such a partitioning exists. Consider two arbitrary vertices a and b of G, such that $a \in V1$ and $b \in V2$: No path can exist between vertices a and b; otherwise there would be at least one edge whose one end vertex would be in $V1$ and the other in $V2$. Hence, if a partition exists, G is not connected.

II. Motivation

We are finding the number of connected components in a graph given in form of incidence matrix. We try to develop algorithm which will be able to understand graph using incidence matrix and count the number of connected components in optimal time and space. The number of connected components is an important topological invariant of a graph. In topological graph theory it can be interpreted as the zeroth Betti number of the graph. In algebraic graph theory it equals the multiplicity of 0 as an eigenvalue of the Laplacian matrix of the graph. It is also the index of the first nonzero coefficient of the chromatic polynomial of a graph. Numbers of connected components play a key role in the Tutte theorem characterizing graphs that have perfect matchings, and in the definition of graph toughness.

III. Algorithm

We have developed three algorithms for tackling the given task. Following are the algorithm.

A. Algorithm 1

1) Algorithm Description: We have to identify and count the number of subgraphs in the given graph in the form of incidence matrix. An incidence matrix is a matrix that shows the relationship between two classes of objects. If the first class is X and the second is Y, the matrix has one row for each element of X and one column for each element of Y. The entry in row x and column y is 1 if x and y are related (called incident in this context) and 0 if they are not related or connected. We first convert the incidence matrix to adjacency list at the time of input . On receiving a graph $G = (V, E)$, a vertex s (the source) is chosen. The algorithm randomly picks a vertex from vertex set V, and runs the Depth First Searching algorithm to determine and mark all the connected vertices to it. Every Depth First Search represents a disconnected component.

Algorithm

- Convert Incidence Matrix to Adjacency List.
- For each vertex $s \in V$ and $S \notin Visited$ where $Visited$ is the set of vertices already traversed
  DFS-iterative(G,s) , NumberOfSubgraphs++
- ```
  DFS-iterative (G, s):
      let S be stack
      S.push( s )
      mark s as visited.
      while ( S is not empty):
         v  =  S.top( )
  ```
  ```

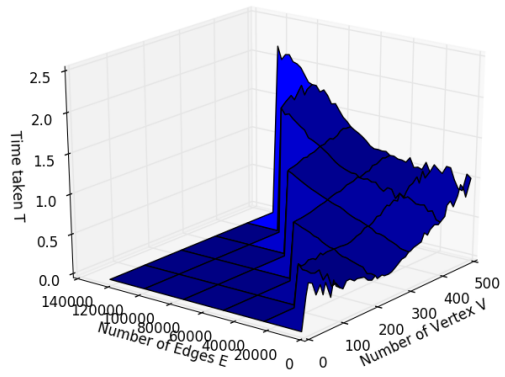

Fig. 1. Number of Nodes and Edges vs Time graph

```
    S.pop( )
for all neighbours w of v in G:
    if w is not visited :
            S.push( w )
            mark w as visited
```

2) Correctness of Algorithm:
Claim 1: DFS is called exactly once for each vertex in the graph.
Proof: Clearly DFS(x) is called for a vertex x only if visited(x)==0. The moment it's called, visited(x) is set to 1. Therefore the DFS(x) cannot be called more than once for any vertex x. Furthermore, the loop "for all v...DFS(v)" ensures that it will be called for every vertex at least once. QED.

Claim 2: The body of the "for all w" loop is executed exactly once for each edge (v,w) in the graph.
Proof: DFS(v) is called exactly once for each vertex v (Claim 1). And the body of the loop is executed once for all the edges out of v. QED.

Therefore the running time of the DFS algorithm is O(E+V).

3) Complexity Analysis of the Algorithm:
The time complexity of the algorithm is determined by two steps in the algorithm. First step is the conversion of Incidence Matrix of order V*E to adjaceny list. The time complexity of this procedure is of order O(E*V). Second step of the algorithm involves searching using Depth First Searching algorithm and it works in O(V+E).

Space Complexity of the algorithm is of order O(V*E) for containing the incidence matrix.

4) Algorithm for graphs of various sizes:
Image (Figure 1) represents V and E vs T plot(Time taken by the algorithm as the nodes and edges vary) where T(Time) is in seconds.

B. Algorithm 2

1) Algorithm Description: We have to identify and count the number of subgraphs in the given graph in the form of incidence matrix. The algorithm runs Breadth First Search algorithm to determine and mark all the connected vertices to it. We initially mark all vertices ass unvisited.It chooses one of the unvisited vertices marks it as visited,insert it in a queue.We start by popping an element from the queue then find all the vertices connected to it using edges ,mark them as visited,insert them in the queue if not visited earlier and the same process is repeated until queue is empty . Every Breadth First Search represents a disconnected component.

Algorithm

- Mark all the vertices as unvisited
- Count = 0
- Loop through all the vertices
-
```
    If unvisited call BFS
        Count++
```

-
```
BFS(G,s) :
    Mark s as visited
    Insert s in queue
    While queue is not empty :
      Pop an element u from the queue
      Loop through all the edges :
        If u is connected to that edge :
        Loop through all the vertices to
        find the vertex 'v' connected to it
            If v is unvisited :
                Mark v as visited
                Insert it in the queue
```

2) Correctness of Algorithm:
Claim 1: BFS is called exactly once for each vertex in the graph.
Proof: Clearly BFS(x) is called for a vertex x only if visited(x)==0. The moment its called, visited(x) is set to 1. Therefore the BFS(x) cannot be called more than once for any vertex x. Furthermore, the loop for all v...BFS(v) ensures that it will be called for every vertex at least once. QED.

Claim 2: The body of the Loop through all the edges loop is executed exactly once for each edge (v,w) in the graph.
Proof: BFS(v) is called exactly once for each vertex v (Claim 1). And the body of the loop is executed once for all the edges out of v. QED

3) Complexity Analysis of the Algorithm:
The worst case running time complexity of the algorithm is O(E*V).For each vertex we loop through all the edges (E) and for each edge connected to it the inner loop is atmost called V times to find the other connecting vertex.The inner loop is called one time for each edge. Hence the time complexity becomes O(E*V).

Space Complexity of the algorithm is of order O(V) for containing the visited array and the queue of vertices.

C. Algorithm 3

1) Algorithm Description: Let the number of nodes in the given incidence matrix be V and the number of edges be E.We have to identify all the disconnected components.Let us consider all the nodes as disconnected.All the nodes now belong to different component.consider every edge one by one ,If an edge is present between two disconnected components then then we merge both the disconnected components into one component.We keep on doing this for all the edges.finally the components which are remaining are disconnected with each other.

Algorithm

```
int func(int idx)
        while(i!=idx)
                parent[i]=parent[parent[i]];
                i=par[i];
        return i;

int main()
        for i in range 1 to n
                parent[i]=i,size[i]=1;

        for i in range 1 to m
                scan endpoints of edge as x and y;
                int u=func(x);
                int v=func(y);
                if(u==v)continue;
                if size[u] less than size[v]
                        parent[u]=v;
                        size[v]+=size[u];
                else
                        parent[v]=u;
                        size[u]+=size[v]

        for i in range 1 to n
                vector[func(i)].push-back(i);

        for i in range 1 to n
                if(vector[i].size() greater than 0)
                        print-nodes-in-vector[i];
```

2) Correctness of Algorithm:
Claim:Since All the components which have an edge between them are being connected,All the nodes part of these components are connected together.Those components that do not have any edge between them at the end remain disconnected.

3) Complexity Analysis of the Algorithm:
The time complexity of the algorithm is determined by time taken to find root parent of the node.Root parent is the node that represents the connected component.It takes at most log(N) time to find root parent.Even though it seems more than log(N) the union by rank and path compression ensures that the time complexity for finding root parent to be log(N).We find root parent for all the end points of the edges.That is 2*E times.So the overall complexity is 2*E*log(V) which is Elog(V)

Space Complexity of the algorithm is same as space complexity required for incidence matrix

IV. Conclusion

We have given three different algorithm for finding the disconnected subgraphs for a given incidence matrix. The worst case time complexity of first two algorithms is same , however they differ in terms of space complexity. The extra space required for the first algorithm is O(V*E) whereas of the second is O(V).The time complexity of the third algorithm is O(E*LogV).Hence the third algorithm is better than the other two in terms of time complexity.