Finding Parallel Edges using Incidence Matrix of a Directed Graph

Kritika Sharma*, Saurabh Singh†, Anujraj Goel‡
Indian Institute of Information Technology, Allahabad
*icm2014502@iiita.ac.in
†iwm2014004@iiita.ac.in
†iim2014002@iiita.ac.in

***Abstract*—The problem of finding the parallel edges is a well known problem in the field of graph theory. Given a graph G = (V, E), the problem is to find the representation for incidence matrix of a directed graph and identify all parallel edges when the graph is seen as a directed graph and when it is seen as undirected graph. In this paper, the given graph G is first represented in the form of incidence matrix. An algorithm is introduced to find all the parallel edges. This algorithm takes each possible pair of edges and then traverse through all the vertices to check if the two edges satisfy the condition of parallel edges. The algorithm works with average time complexity $O(NM^2)$ and O(MN) auxiliary space where N is the number of vertices and M is the number of edges in the given graph.**

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 (I_{N*M}) [3] where N is number of vertices and M is number of edges. In an incidence matrix, each column stands for an edge and each row for a node and hence matrix entries marks the nodes which are connected by the correspoding edges. Basically, an incidence matrix for a graph G = (V=$\{v_1,v_2,....,v_N\}$, E=$\{e_1,e_2,...,e_M\}$) as an N*M matrix naming I, where :

I_{ij} = 1 if edge e_j is coming out of the node v_i
I_{ij} = -1 if edge e_j is going into the node v_i
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 :

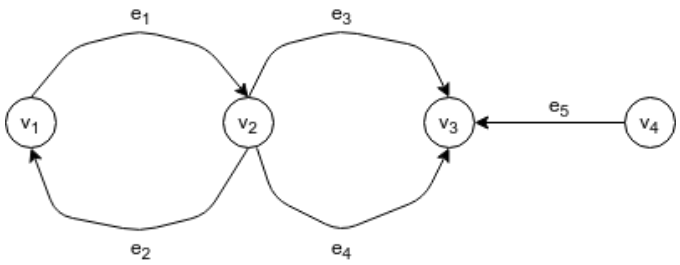

Fig. 1. **Parallel edges in directed graph**

The incidence matrix in table.1 is created as follows : edge e_1 is directed from v_1 to v_2. So, the entry corresponding to

TABLE I
(Incidence matrix for the graph given in fig.1)

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

$I[v_1][e_1]$ 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.

An incidence matrix requires N*M entries for storage which is comparatively more than that required in adjacency matrix which requires N*N entries. This is because number of edges are generally more than the number of vertices. In spite of this incidence matrix is used in many fields. Use of incidence matrix is specially preferred in the electrical applications and switching application too.

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 [4]. 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 [1]. If two edges are starting from same vertex and ending at the same vertex then the two edges are simply parallel. If one edges starts a vertex where other ends and also if this vertex ends where the second vertex starts, then the two edges involved are said to be anti-parallel. In figure 1, edges e_3 and e_4 are parallel edges as both are starting at the same vertex and also ending at the same vertex while the edges e_1 and e_2 are anti parallel because edge e_1 starts at vertex v_1 where e_2 ends and e_2 starts at vertex v_2 where the edge e_1 ends.

Some other examples of parallel edges are given in fig. 2 (for undirected graph) and fig. 3 (for directed graph). In fig.2, edges e1, e2, e3 are parallel as they are starting and ending on the same vertices. In fig. 3, edges e1 and e3 are anti-parallel.

This paper presents the improvised algorithm to find the parallel edges in a graph by just using the incidence matrix for that graph. The algorithm takes each unordered pair of edges (e_i,e_j). For each of those pairs, the algorithm then traverses all the vertices. Taking each vertex v_k, we will check values

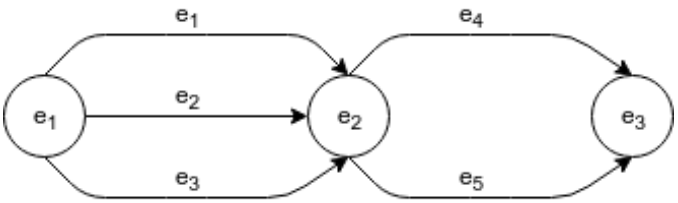

Fig. 2. **Parallel edges in directed graph**

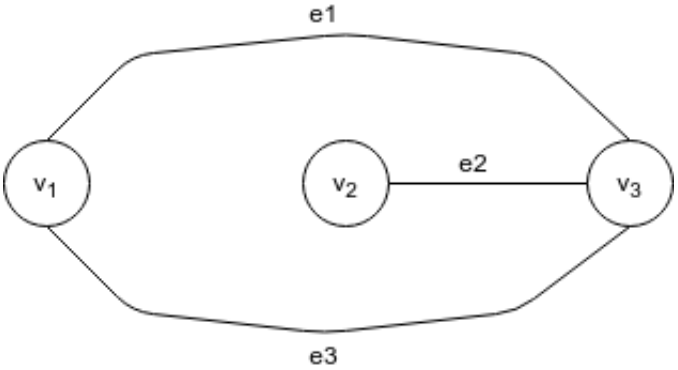

Fig. 3. **Parallel edges in a undirected graph**

for $I[v_k][v_i]$ and $I[v_k][v_j]$. If for any vertex, the value of one of these two is non-zero and other is zero, then the taken pair of edges cannot be parallel. This algorithm works with time complexity $O(NM^2)$ and takes an extra space of O(N*M) for the storage of the incidence matrix.

II. Motivation

The graphs where parallel edges are present are called as Multigraphs [5]. The edges can simple be parallel or anti-parallel. There are many applications where we parallel edges are involved. For eg. In map for the construction of roads, parallel edges can be found using the algorithm proposed and if any of them is useless i.e. if there is a parallel alternative to this then it can be eliminated. Also, paralle edges can be found in the electrical networks for the proper understanding of current flow. A very famous use of multigraphs is in the field of social networking. Sometimes, a simple analysis of these social networks is required which can be accomplished by first identifying the parallel edges in the complex graphs and then these edges are collapsed into single ones which will result into simple graphs and hence, study on these can be carried out easily[2]. All these applications lead for the design of an efficient algorithm for the the identification of parallel edges in a graph.

III. Methods and Descriptions

In this paper, firstly a brute force algorithm is introduced and then an optimised algorithm is designed to find all the parallel edges in a given graph using the incidence matrix. Pseudo codes for the algorithm to find parallel edges from incidence matrix of a directed graph when seen as an undirected graph are given below. Description for the case when the graph is seen as directed is given later.

Methodologies for the suggested algorithm :

(i) Brute force algorithm :

Explanation:

We begin by choosing a pair of vertices and subsequently a pair of edges for the respective pair of vertices. The set of parallel edges is established for the pair of edges which satisfy one of the following conditions:-

1. grid[i][a] == -1 && grid[j][a] == 1 && grid[i][b] == -1 && grid[j][b] == 1
2. grid[i][a] == -1 && grid[j][a] == 1 && grid[i][b] == 1 && grid[j][b] == -1
3. grid[i][a] == 1 && grid[j][a] == -1 && grid[i][b] == -1 && grid[j][b] == 1
4. grid[i][a] == 1 && grid[j][a] == -1 && grid[i][b] == 1 && grid[j][b] == -1

Pseudocode:

Input: Incidence Matrix for a directed graph

Output: Set S containing pairs of parallel edges

E = Number of edges, V = Number of vertices

```
parallel_edges(grid, E, V){
    for(i=0 to E-1):
        for(j=0 to E-1):
            for(a=0 to V-1):
                for(b=0 to V-1):
                    if( check( grid, i , j , a, b) ):
                        S = S U {i,j}
}

bool check(grid, i , j , a ,b ){
    if(  grid[i][a] == -1 && grid[j][a] ==  1 && grid[i][b] == -1 && grid[j][b] ==  1 )
        return true
    if ( grid[i][a] == -1 && grid[j][a] ==  1 && grid[i][b] ==  1 && grid[j][b] == -1 )
        return true
    if ( grid[i][a] ==  1 && grid[j][a] == -1 && grid[i][b] == -1 && grid[j][b] ==  1 )
        return true
    if ( grid[i][a] ==  1 && grid[j][a] == -1 && grid[i][b] ==  1 && grid[j][b] == -1 )
        return true
    return false;
}
```

Complexity analysis : The above algorithm takes space complexity of O(V*E) and time complexities as given below :

Worst case : $O(E^2*V^2)$

Avg Case: $O(E^2*V^2)$

Best Case: $O(E^2*V^2)$

(ii) Improvised Algorithm :

Explanation:

We begin by choosing a pair of edges, for the set of vertices belonging to this pair of edges, we check the following the condition:-

1. grid[i][v] != 0 && grid[j][v]== 0

2. grid[i][v] == 0 && grid[j][v]!= 0
The required set consists of edges NOT satisying this condition.

Pseudocode:
Input: Incidence Matrix for a directed graph
Output: Set S containing pairs of parallel edges
E = Number of edges, V = Number of vertices

```
parallel_edges(grid, E, V){
    for(i=0 to E-1):
        for(j=0 to E-1):
            if( check( grid, i , j , V) ):
                S = S U {i,j}
}

bool check(grid, i , j , V){
    for(k=0 to V-1):
        if( (grid[i][k] != 0 && grid[j][k]==0) || (grid[i][k] == 0 && grid[j][k] !=0)):
            return false
    return true
}
```

Complexity analysis : The above algorithm takes space complexity of O(V*E) and time complexities as given below :
Worst Case : $O(E^2*V)$
Avg Case : $O(E^2*V)$
Best Case : $O(E^2*V)$

IV. Implementation and Results

Given graph G = (V,E) with N vertices and M edges, we can simply find the incidence matrix as explained in the algorithm earlier. This representation is for directed graph. Now, we implemented four algorithms. Firstly we implemented brute force algorithm for both types of graphs and then an optimised algorithm for the same.

(i) Brute force(Undirected): In this implementation we simply took each unordered pair of edges(e_i,e_j). For each edge we initialise a count variable to zero. For each of these pairs, we traversed through each unordered pair of vertices to check if this pair of vertices are the two end points of the taken edges. If this is the case.i.e. the vertices are the end points of both the edges then we increase the count variable. For checking if a pair of vertices(v_k,v_l) is end points of both the edges, we check if :
(a) ($I[v_k][e_i]$ is 1 and $I[v_l][e_i]$ is -1) or (I[vk][ei] is -1 and I[vl][ei] is 1) and
(b) ($I[v_k][e_j]$ is 1 and I[vl][ej] is -1) or (I[vk][ej] is -1 and I[vl][ej] is 1)
If after traversing all pair of vertices, this count comes to be 1, that means these two edges have the same pair of vertices as end points. Hence these two edges are parallel. This process we do for all pair of edges and hence, after running the algorithms, we will have pair of parallel edges with us. This takes space O(M*N) and time complexity of $O(N^2M^2)$. The time vs input size graph is as given in fig. 4. Input size is basically N*M.

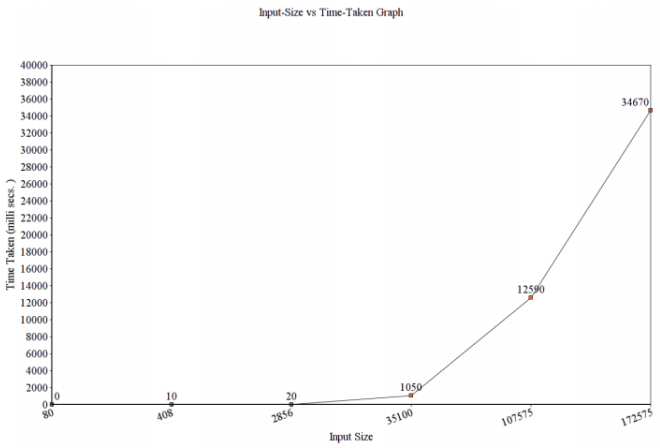

Fig. 4. **Time vs input size graph for brute force algorithm (undirected)**

(ii) Brute force(Directed): In this implementation also, we simply took each unordered pair of edges(e_i,e_j). For each edge we initialise a count variable to zero. For each of these pairs, we traversed through each unordered pair of vertices to check if this pair of vertices are the two end points of the takes edges. If this is the case.i.e. the vertices are the end points of both the edges then we increase the count variable. For checking if a pair of vertices(v_k,v_l) is end points of both the edges, we check if :
(a) $I[v_k][e_i]$ is non-zero and $I[v_l][e_i]$ is non-zero and
(b) $I[v_k][e_j]$ is nonzero and $I[v_l][e_j]$ is non-zero.
If after traversing all pair of vertices, this count comes to be 1, that means these two edges have the same pair of vertices as end points. Hence these two edges are parallel. This process we do for all pair of edges and hence, after running the algorithms, we will have pair of parallel edges with us. This takes space O(MN) and time complexity of $O(N^2M^2)$.

(iii) Optimised algorithm (Undirected) : In this algorithm, we takes each unordered pair of edges (e_i,e_j). For each of those pairs, the algorithm then traverses all the vertices. Taking each vertex vk, we will check values for $I[v_k][v_i]$ and $I[v_k][v_j]$. If for any vertex, the value of one of these two is non-zero and other is zero, then the taken pair of edges cannot be parallel. If both of them are non-zero then the pair of edges taken are parallel. This process is done for each pair of edges. At last we will have all the parallel edges. This algorithm works with time complexity $O(NM^2)$ and takes an extra space of O(N*M) for the storage of the incidence matrix. The time vs input size graph is as given in fig. 5.

(iv) Optimised algorithm (Directed) : In this algorithm, we takes each unordered pair of edges (e_i,e_j). For each of those pairs, the algorithm then traverses all the vertices. Taking each vertex vk, we will check values for $I[v_k][v_i]$ and $I[v_k][v_j]$. If for any vertex, the value of one of these two is 1 or -1 and other is zero, then the taken pair of edges cannot be parallel. If both values are 1 or -1, then the edges are parallel. This

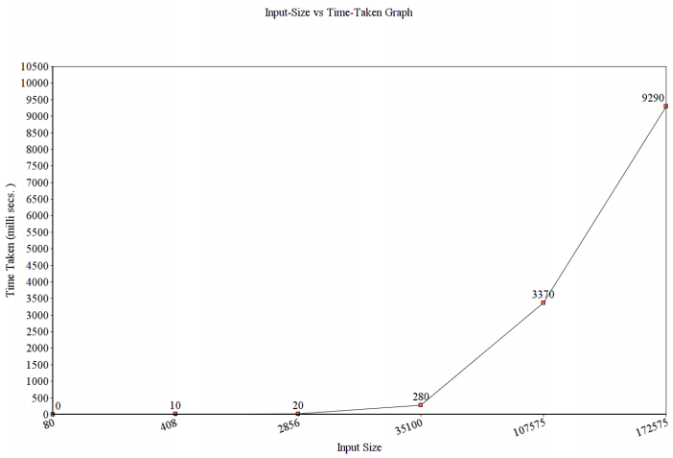

Fig. 5. **Time vs input size graph for optimised algorithm (undirected)**

process is done for each pair of edges. At last we will have all the parallel edges. This algorithm works with time complexity $O(NM^2)$ and takes an extra space of O(N*M) for the storage of the incidence matrix.

In the above four proposed algorithms, worst, best and average time complexities are all same which is $O(N^2M^2)$ in case of brute force and $O(NM^2)$ in case of optimised algorithm.

Hence, for an undirected graph as well as directed graph incidence matrix is constructed and using which an unordered pair of edges (e_i, e_j) are checked whether they are parallel or not using the above proposed algorithm. Hence the parallel edges are found if exists any.

V. Conclusion

In this Paper we have shown that how the incidence matrix for the graph is constructed when it is seen as an undirected graph and when it is seen as an directed graph. For an Incidence matrix, in the directed graph we check the unordered pair of edges (e_i, e_j) and check if there are parallel edges between any two end-points as well as the anti-parallel edges between the two nodes or not. For the undirected graph, in the incidence matrix we see if a pair of edges (e_i and e_j) are parallel edges or not using the above algorithm. The algorithm works with time complexity $O(NM^2)$ and space complexity of O(N*M).

VI. References

[1] Ayan Das(CS1414) and Sayandeep Mitra(CS1413), Graph Theory (August 31, 2014).

[2] Termeh Shafie, Department of Computer and Information Science, University of Konstanz, A multigraph approach to social network analysis (2007-2013)

[3] https://en.wikipedia.org/wiki/Incidence_matrix [Last accessed on August 12, 2017]

[4] http://mathworld.wolfram.com/Multigraph.html [Last accessed on August 12, 2017]

[5] https://en.wikipedia.org/wiki/Multigraph [Last accessed on August 13, 2017]